Evaluation and Improvement of Semantically-Enhanced Tagging System

PhD Thesis

Majdah Hussain Alsharif

Software Technology Research Laboratory
Faculty of Technology
De Montfort University
England

December 2013

Dedication

I dedicate this work to Almighty God, who has made this work possible.

I also dedicate this thesis to a number of people without whom this thesis might not have been written, and to whom I am greatly indebted:

To my parents, for their unconditional support in every way possible since the start of this journey.

To my husband, you are the best companion I could ever wish for. Thank you for the encouragement and support.

To my children Ghada, Abdulrahman, Shaker, and Ghazi thank you for becoming adults just to help me and I love you all.

To all my friends who I missed, thanks for your prayers and kind wishes.

Abstract

The Social Web or 'Web 2.0' is focused on the interaction and collaboration between web sites users. It is credited for the existence of tagging systems, amongst other things such as blogs and Wikis. Tagging systems like YouTube and Flickr offer their users the simplicity and freedom in creating and sharing their own contents and thus folksonomy is a very active research area where many improvements are presented to overcome existing disadvantages such as the lack of semantic meaning, ambiguity, and inconsistency.

TE is a tagging system proposing solutions to the problems of multilingualism, lack of semantic meaning and shorthand writing (which is very common in the social web) through the aid of semantic and social resources.

The current research is presenting an addition to the TE system in the form of an embedded stemming component to provide a solution to the different lexical form problems. Prior to this, the TE system had to be explored thoroughly and then its efficiency had to be determined in order to decide on the practicality of embedding any additional components as enhancements to the performance. Deciding on this involved analysing the algorithm efficiency using an analytical approach to determine its time and space complexity.

The TE had a time growth rate of O (N^2) which is polynomial, thus the algorithm is considered efficient. Nonetheless, recommended modifications like patch SQL execution can improve this. Regarding space complexity, the number of tags per photo represents the problem size which, if it grows, will increase linearly the required memory space.

Based on the findings above, the TE system is re-implemented on Flickr instead of YouTube, because of a recent YouTube restriction, which is of greater benefit in multi languages tagging system since the language barrier is meaningless in this case. The re-implementation is achieved using 'flickrj' (Java Interface for Flickr APIs). Next, the stemming component is added to perform tags normalisation prior to the ontologies querying. The component is embedded using the Java encoding of the porter 2 stemmer which support many languages including Italian.

The impact of the stemming component on the performance of the TE system in terms of the size of the index table and the number of retrieved results is investigated using an experiment that showed a reduction of 48% in the size of the index table. This also means that search queries have less system tags to compare them against the search keywords and this can speed up the search. Furthermore, the experiment runs similar search trails on two versions of the TE systems one without the stemming component and the other with the stemming component and found out that the latter produced more results on the conditions of working with valid words and valid stems.

The embedding of the stemming component in the new TE system has lessened the effect of the storage overhead needed for the generated system tags by their reduction for the size of the index table which make the system suited for many applications such as text classification, summarization, email filtering, machine translation…etc.

Declaration

I declare that the work described within this thesis was originally taken by me between January 2008 and December 2013 except where otherwise acknowledged. It is submitted in partial fulfilment of the requirements of the degree of Doctor of Philosophy at De Montfort University and has not been previously submitted for any other award.

Acknowledgement

First and foremost, I thank my academic advisors, Dr Bella and Professor Zedan for their patience, guidance, and support.

My gratitude goes to Dr. M Magableh for sharing his knowledge.

Also, I would like to acknowledge the STRL, and the Graduate School. My graduate experience benefitted greatly from the courses I took, and the opportunities I had there.

Finally, I would like to thank all the people who contributed in any way to the work described in this thesis.

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
DBMS	Database Management System
HTML	Hyper Text Markup Language
IR	Information Retrieval
JAWS	Java API for WordNet Searching
flickrj	Java Interface to Flickr API
MWN	MultiWordNet
POS	Part Of Speech
PWN	Princeton WordNet
URL	Uniform Resource Locator
JVM	Java Virtual Machine
TE	Tag Enhancer
SNS	Social Network Site

Table of Contents

1 Introduction

1.1 Background

The commercialization of internet access in the late 1980's attracted to it people from outside the academic circles and, since then, the internet has carried on gaining momentum. With more advances in technology, cheaper prices, and fast speed, the number of internet users is growing at a very rapid rate. According to [53], the years 2000-2012 had a global growth rate of 566.4 % and on June 2012, the number of users worldwide is over 2.4 billions. The statistical facts above show that people are getting more dependent on the internet in many aspects of their daily life.

The internet gives access to vast amount of information. For many organizations, their information is as valuable as their assets, and reputation. They use information as weapon for gaining and sustaining competitive advantage when used in decision making and supporting critical processes [40] whilst it is found accurate and within the shortest time. Thus, information needs to be managed: preserved, sorted, maintained up-to-date, and delivered to the right people at the right time to avoid many problems such as financial loss, lost opportunities, damaged reputation…etc. Therefore having a good information management and retrieval systems is essential for the success of many services and businesses nowadays. For example, search engines need to offer up-to-date information, locate individuals and organizations, and summarize news. Local search services need to guide consumers to retailers. Large companies need to have access controlled repositories of e-mail, memos, reports, and other documents for proper decision making [22].

With the popularity of the social web (web 2.0) and since their introduction, social network sites (SNSs) like facebook, twitter, and Flicker have attracted millions of users, many of whom have integrated these sites into their daily practices [17]. In these sites, a user creates a profile, and builds a list of friends to share and exchange contents with them. Users have the freedom to categorise their contents as they see fit using tags.

Moreover, SNS is considered an important marketing tool [123] since it allow users to participate in the business production and promotion through sharing their personal experiences like recommendations, reviews, and ranking.

In 2013, the results of a survey about the social commerce on Facebook, Twitter and Pinterest were published as follows [11]:

- Social media drives roughly equal amounts of online and in-store sales
- Nearly 4 in 10 Facebook users report that they have at some point gone from liking, sharing or commenting on an item to actually buying it
- 43% of social media users have purchased a product after sharing or favoriting it on Pinterest, Facebook or twitter.

Tagging is one of the main applications of the semantic web (web 2.0). It is a simple way for indexing information but it lacks standards and because it's a subjective process, it can generate inconsistent and ambiguous classification [72]. Another drawback in tagging systems is the lack of semantics among tags but with the birth of semantic web, its tools, and technologies, many studies are investigating how to invest this to enhance the tagging experience. Different tagging approaches can address few key tagging problems as listed below [69]:

- Formal taxonomy or ontology approaches: formal taxonomy derives tags through data mining whereas, the ontology approach uses seeding and this requires undesired additional user contributions. However, both approaches give the tags a frame of reference which reduces inconsistency and ambiguity.
- Statistical and pattern analysis approaches: they are very popular because they work well with web applications such as Google's PageRank. Common factors used in these approaches are tag use frequency, popularity, and ranking.
- Social networking and visualization approaches: in the social approach, researchers use the social network to validate tags whilst, another visual approach uses information and tags to improve user behaviour.

Approaches	Pros	Cons
Formal taxonomy and ontology methods	Directly address the issues of ambiguity, inconsistency, and haziness	Too restrictive and can conflict with the informality of folksonomy
Statistical and pattern analysis methods	Have shown promising results and don't require additional user contribution	Don't directly address ambiguity, inconsistency, and haziness
Social network methods	Have shown promising results in generating desired user tagging behavior	Don't directly address ambiguity, inconsistency, and haziness
Visualization methods	Have shown promising results in generating desired user tagging behaviors	Don't directly address ambiguity, inconsistency, and haziness

Figure 1.1: Collaborative Tagging Approaches [69]

The World Wide Web extension called "Semantic Web" or "Web 3.0 enables people to share content beyond the boundaries of applications and websites. It has been described as a web of data [103]. The concept is to form a Web that links documents to each other and recognizes the meaning of the information in them, in other words, to transform the current Web from a series of interconnected, but ultimately semantically isolated data islands into one gigantic, personal information storage, manipulation and retrieval database [13, 61].

1.2 Problem Statement

'Web 2.0' or the 'Social Web' is about discarding static web pages and changing the way web pages are designed and used, allowing more interaction and collaboration between users [83]. With its user-friendly services, Web 2.0 is behind the popularity of social sites such as blogs, Wikis, and tagging systems. The problem is that these sites generate huge amount of metadata. For example, in tagging systems, users freely tag their contents and the result is that some of these tags are inconsistent and ambiguous making the retrieval process inaccurate.

'Web 3.0' or 'Semantic Web' is a web of linked data [77, 115]. It will allow internet users to control data in many ways such as: creating data stores, building vocabularies and establishing rules for managing data [115]. It is about providing users with higher levels of social sharing and participation [77].

Utilizing the benefits of the Semantic and Social Web can provide solutions to improve the accuracy rate in tagging systems and many studies are investigating in this direction.

TE 'Tags Enhancer' is a prototype that uses web tools: Princeton WordNet (PWN), MultiWordNet (MWN), and clustering to generate new tags to improve the quality of the original tags which can decline for reasons such as a lack of semantic, language constraint, and the use of shorthand writing vocabulary. Within TE, a user tag is subjected to some of the mentioned tools or all of them as needed. PWN provides synonyms and/or hypernyms of the user tag thus increasing the semantic value of results which is retrieved after performing the search using both the user and system tags. MWN provides the English translation of the user tag in case it is an Italian word. The last tool is Flickr clustering which can sometimes produce a meaningful word from the shorthanded written user tag.

TE has previously been tested and shown to deliver relative search results for a wider coverage of semantically related results than existing solutions [72].

In most IR system, the user asks for information using a query which contains one or more search terms. These terms are compared against the index terms (important words or phrases) of the IR contents for a match. Both the query terms and the index terms often have many morphological variants.

In the case of TE, although the tag sample data is not large compared to other IR systems, the statistics showed that the semantic component can sometimes generate more than 80 system tags from a single user tag due to the fact that its semantically rich and the use of more than one language. The number of system tags can grow significantly in a larger IR system especially with the addition of more languages, in other words; the sizes of the database and the index tables will increase. Furthermore, the number of search terms inside the query will increase and this can slow the search process.

stemming is used by search engines in IR systems to increase their effectiveness [82]. Experiments in [60, 94, 96, 101] show stemming is beneficial for highly inflected languages. It makes the search broader, in other words; it ensures that the greatest number of relevant matches is included in search results [82]. Some studies claim that stemming can increase the average recall [39] [41]. Moreover, since stemming is about mapping morphological variants to a single stem, this will reduce the number of system tags and lead to the size reduction of search terms, index tables and the database. The work of Lennon et al. (1981) [35, 66] on various stemmers and databases reported the following compression percentages in the size of files, sometimes as much as 50 percent.

Stemmer	Cranfield	National Physical Laboratory	INSPEC	Brown Corpus
INSPEC	32.1	40.7	40.5	47.5
Lovins	30.9	39.2	39.5	45.8
RADCOL	32.1	41.8	41.8	49.1
Porter	26.2	34.6	33.8	38.8
Frequency	30.7	39.9	40.1	50.5

Figure 1.2: Index Compression Percentages from Stemming [35]

1.3 Research Objectives and Questions

The proposal of this research is to evaluate and improve the TE system. In particular, the research studies the effects of adding new tags to the system on the time needed to generate them and on their allocated space. Furthermore, the research proposes modifying TE by using stemming on the user tags prior to querying the semantic resources as a normalisation step for better quality system tags.

The research will investigate the questions below:

Q1: What are the effects on performance of embedding the stemming component to the TE system?

Q2: What is the time complexity of the TE algorithm?

Q3: What is the space complexity of the TE algorithm?

Q4: Is the database design optimised for the TE ER Model?

1.4 Success Criteria

Indicating or rejecting the claim that embedding the stemming component has effects on the performance of the TE system, in terms of index size and the number of search results, needs an experiment to record both the size of the index and the number of search results retrieved from the TE system without stemming and the TE system with stemming and compare the statistics to reach a decision.

Regarding the index size, both versions of the TE system will work with the same dataset of user tags. In the old TE, user tags are directly subjected to the semantic and clustering components in the original TE to generate system tags whereas in the new TE, user tags are submitted to the stemming component to generate stems and then stems use the other components to generate system tags. System tags are used by the search process to be compared against a search term. Thus, they represent the index table in this case. If the number of system tags in the new TE is less than the one in the old TE then stemming is responsible for this reduction.

A sample dataset of 30 words is used to perform search trials on the old and new TE systems and record the number of retrieved results from both systems. Comparing these numbers will show if the new TE system is able to retrieve more results than the

old TE or not because of the stemming component. If the new TE retrieved more results then we can support the claim that stemming is behind this results increase.

1.5 Thesis Structure

The thesis is divided into seven chapters. Below is a short description of the contents of each chapter as follows:

- Chapter 1 (Introduction): this chapter gives the reader a general idea about the research and what it is about. It presents the research problem statement, objectives, and the questions.
- Chapter 2 (The Literature Review): this chapter includes a thorough literature review on the research areas involved in the current research. It covers topics such as tagging systems, stemming algorithms and algorithm complexity theory.
- Chapter 3 (The TE System): this chapter contains a comprehensive summary of the TE (Tag Enhancer) system. It covers related topics necessary for the current research work such as its scope, components, implementation etc.
- Chapter 4 (The Methodology): this chapter includes a detailed account of the tools and methods utilised by the research to achieve the planned research objectives.
- Chapter 5 (Database Optimisation and Algorithm Complexity Analysis): this chapter examines the TE system with respect to database optimisation, time complexity and space complexity. A discussion of the findings is also included.
- Chapter 6 (Stemming Component embedding): this chapter is a translation of the methodology layout in chapter 4. It gives details on the embedding process with its challenges and approaches.
- Chapter 7 (Conclusion and Future Work): this chapter summarizes the research and draws conclusions based upon the findings of the research to give answers to the research questions mentioned in chapter 1.

2 The Literature Review

2.1 Introduction

This chapter is looking into tagging systems taxonomy, their principles, the benefits and problems facing them. Next, the literature will look into two semantic tools used in the TE system which are the Princeton WordNet (PWN) ontology and the MultiWordNet (MWN) ontology, giving a background on both ontologies and their mechanism.

Moving forward, stemming algorithms will be examined from different aspects such as their definition, background history, wide range of techniques and main types of stemming algorithms, with a comparison summary of the benefits and drawbacks of each algorithm mentioned.

Since the research is analysing the cost in time/space of the TE system, the last segment in the literature review is about algorithm analysis and complexity, highlighting the important approaches with their advantages and disadvantages and detailing the sequence followed in each approach. The asymptotic notations used in expressing growth rates are explained with a detailed section on best and worst cases of complexity.

2.2 Methodology of the Literature Review

Define the research topic

The intended focus area of the research must be decided. The researcher should be guided by what interest him the most and should be familiar enough with the chosen area to judge whether it is 'researchable' topic or not [29].

The researcher is interested in databases and data retrieval systems and while reading for specific focus area, he came across the thesis “A Generic Architecture for Semantic Enhanced Tagging Systems” that presented a retrieval prototype (it’ll be called TE in

the literature) and decided to enhance the TE prototype by adding stemming capability.

Define the main concepts and keywords in the topic

Main concepts are defined while building a list of alternatives and synonyms to the keywords. The list is used when searching for materials. In addition, looking at previous work will identify the underlying theories and the ground materials of the research topic which mostly are the frequently cited [29].

By performing this step, the researcher gathered essential information about folksonomies like the broad and narrow types, the uncontrolled vocabulary, Princeton WordNet, MultiWordNet, semantic relations, folksonomies advantages and disadvantages. Furthermore, regarding stemming, there are suffix and affix stripping algorithms, statistical stripping algorithms (n-grams, HMM), the snowball framework…etc.

From the literature material, the researcher highlighted some notable researchers and professors with ground knowledge on their fields such as Gruber, Levitiin, Pianta, Sinclair, Miller, Frakes, D. Harman. In addition, many stemming algorithms have the name of distinguished researchers in the field like Porter, Lovins, Krovetz, Paise, and Husk.

The keywords list included these terms: tagging systems, metadata, social classification, ontologies, web 2.0, collaborative tagging, semantic web, clustering, cluster analysis, stemming, stemmer, stemming algorithm, root extraction, root word, keyword stripping, suffix removal, inflectional language, and conflation.

Select research tools

Aided with tools afforded usually by the university, the search is performed on the university library catalogue and its subscribed databases which cover a wide range of research areas.

Databases offer the most recent academically authoritative text like journals, research papers, theses, conference papers ...etc. ACM Digital Library, CiteseerX, OPAC, DBLP Computer Science Bibliography, IEEE Xplore, Elsevier, ScienceDirect, INSPEC, JISC, ETHOS, Springer , Google Scholar are frequent databases and websites visited by the researcher during the search stage.

Do the search

Initially the search is based on the information gathered in step 2. Each yielded result is processed for additional materials as follows ([29, 68, 84, 109]):

- The references are reviewed giving the researcher more insight into the study. Also, reviewing additional materials that have cited the resulted article gave the researcher information about any developments in the area of the study.
- The early work of authors which relates to the study is reviewed for useful information. In addition, following up on later publication by the same author gave information about what is new or changed since his prior work.
- From the resulted materials, their keywords were used to search for further materials.

Manage references

A reference management tool such as EndNote should be used in this stage to record, utilise, and prevent duplication of references. In this thesis, the researcher used EndNote.

Analyse the materials

The researcher scanned the collected materials by reading the summary or the abstract to be able to decide whether an article is worthy of further reading or inclusion. During the first read, the researcher started note-taking, and grouping of similar materials.

Writing the literature review

Once the initial overview has been completed it is necessary to return to the articles to undertake a more systematic and critical review of the content.

The researcher needs to demonstrate his knowledge in the writing by comparing, contrasting, critically evaluating, and interpreting the literature review contents.

2.3 Tagging Systems and the Semantic Web

2.3.1 Tagging Systems (folksonomies)

Background and Definitions

In 2004, the term folksonomy appeared in Wal's information architecture blog and he later on defined it as *"the result of personal free tagging of information and objects (anything with a URL) for one's own retrieval by the person consuming the information"*. The tagging process occurs within a social environment that is usually shared and open [117].

Another definition of folksonomies is that *"they consist of freely selectable keywords, or tags, which can be liberally attached to any information resource"* [91].

In [81], folksonomy is defined as classification system generated by users to tag (using their selected words or sentences) retrieve and categorize web contents such as online photographs, web resources and web links. It is also defined as the act of adding keywords (metadata) to shared content by many users [91].

The term 'folksonomy' is a combination of two words 'folk' meaning people and 'taxonomy' which comes from two Greek words: taxis, meaning arrangement or order, and nomos, meaning law or science, thus it simply means 'a taxonomy created by the people' [91] although there is no taxonomy involved [91].

In information management, taxonomy is a hierarchical classification in the narrow sense and in the broad sense, it is any means of organizing concepts of knowledge [50].

In folksonomy, the authors of the labelling systems are regarded as the key users (or occasionally the creators) of the contents linked to the labels applied. This is an important difference from taxonomy [81].

The folksonomy term has other synonyms such as collaborative tagging, democratic indexing, social classification system, user-generated metadata, tagging, social tagging etc [91]. Some of these terms are debatable such as collaborative tagging which Wal disagrees with, instead referring to collective tagging [117]. Others disapprove of describing folksonomy as classification arguing that it is a post-hoc categorisation and not pre-optimised classification since it has no notations nor relations [91]. A vital aspect of folksonomy is that it is a flat-based namespace in that it has no hierarchy and no direct relationship between the terms used in it [74].

Metadata role on the web is important; it contains description of the contents of a web page along with keywords usually in metatags. Search engines use metatags to index a page for matching it to similar search keywords [33].

An important part of folksonomy is the tag [117]. The role of the tags is that they help improve the effectiveness of search engines since in most cases the content is identified using a shared vocabulary that is easily accessible and popular [81]. Some of the popular folksonomy-based systems nowadays are:

- CiteULike: www.citeulike.org
- Flickr: www.flickr.com
- YouTube: www.youtube.com

Wal indicated two types of folksonomy as follows [116]:

- Broad folksonomy: many people tag the same object and every person can tag the object with their own tags in their own vocabulary such as in ‘del.icio.us’. With this type of folksonomy tagging trends can be spotted using graphic tools such as the power law curve.

- Narrow folksonomy: one or a few people provide tags that are used by a person to search for information. Here, tags are directly associated with the object. In contrast to broad folksonomy, finding emerging vocabulary or descriptions is harder. The practise of grouping tags is visible whereas it's not so in broad folksonomies. Flickr is an example of narrow folksonomy.

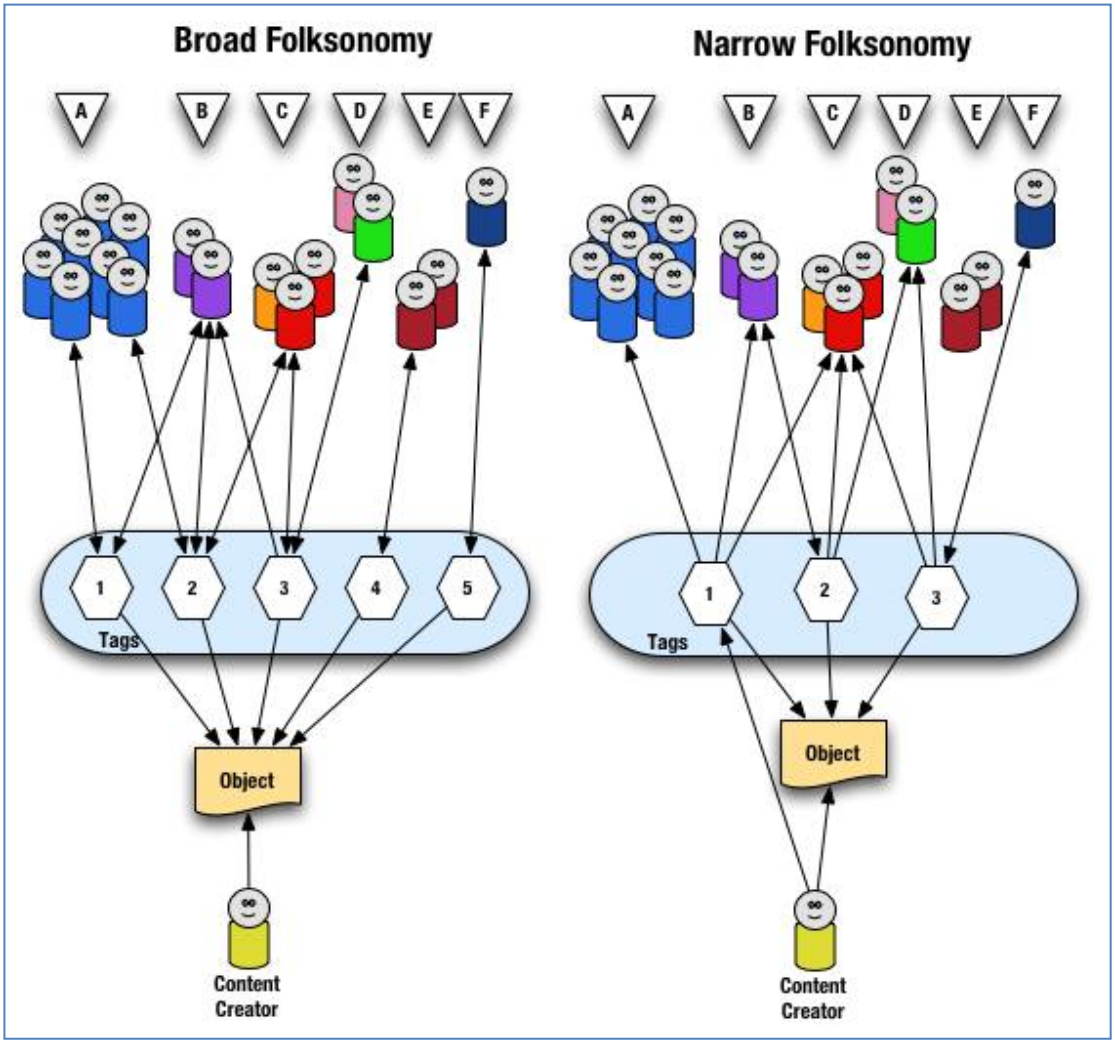

Figure 2.1: Broad and Narrow folksonomies [116]

Delicious are usually tagged by a larger group of users (e.g. by everybody who has bookmarked the web page cnn.com), photos on Flickr are usually tagged just by a single user (e.g. just by the user who has uploaded the photo).

A folksonomy-based system uses at least two basic vocabularies; the searchers' vocabulary and the users' vocabulary. This might lead to the system mismatching entries in both vocabularies. The concept of 'preferred terms' was introduced to allow the folksonomy-based system to control the use of synonyms and homonyms. The system should be able to relate synonyms and suggest the popular synonym as a preferred option [81].

Advantages of folksonomy

- Although finding relevant documents of a direct search may be limited by controlled vocabulary challenges, browsing the whole system including the related interlinked tags will reveal unexpected material from all areas in general [74].
- Usually, in an information retrieval system, we find two or more vocabularies corresponding to the user, the designer, the author of the content, the creators of the classification scheme etc. In this case it might be highly difficult to translate between these vocabularies and this represents an issue in information systems. folksonomy reflects in a direct manner the vocabulary of users, it shifts the focus from the professionals to the users deriving from their preferences in diction, terminology, and precision [57]. According to [76], folksonomy can discover the digital equivalent of 'desire lines' which are foot-worn paths that sometimes appear in a landscape over time and can be later paved to become walkways. Similarly, a system can build a controlled vocabulary using the users' most common tags. The problem with that is the users' vocabulary may be inadequate to do the job because it is very different from the others. The reason is that language is not precise; a word can have different meanings and many synonyms and since users freely adds to their vocabularies, then it is expected to have different ones. Another problem is the short life span of tags in fast developing fields of knowledge meaning what is considered the buzz word today may not be in the near future.

- folksonomy reduces the barriers of entry to the system which was limited to professionals to include users. The reason behind that is that there has been a shift from categorization and classification schemes, that are professionally designed and clearly defined, to ad-hoc set of keywords allowing users without any training or previous knowledge to participate in the system with less cost in terms of time, effort and cognitive costs [74].
- In [113], Udell argues that feedback is the fundamental difference between folksonomy and taxonomy. Within folksonomy, feedback is received in an instant in that once a tag has been assigned to an item; a person is immediately able to see a cluster of items with the same tag. If that view is not as expected, changing or adding tags is allowed. Expanding the scope to include all items with matching tags from all users is very powerful and similarly the view might be different from the expectation. Solutions may include adapting to the group norm, keeping the tag in a bid to influence the group norm, or both. Users can communicate asymmetrically through metadata as result of the tight feedback loop. The individual choices of tags describing contents in a folksonomy are a representation of the negotiation about the meaning of these tags between users [113].
- Individuals can use a folksonomy such as Flickr to organize contents using their own vocabulary. The individual's organizational behaviour reflects his needs within that context. An example is the use of the tag 'toread' on Delicious [74]. On the other hand, Flickr is a public space to share contents between users and the organizational behaviour of an individual is affected by his relationship to other users, and user groups who they share tag use with him [74, 92]. Similar to what has been discussed previously on participation, a folksonomy lowers the barriers to cooperation. Members of a group do not have to agree on a hierarchy of tags or detailed taxonomy; they only need to generally agree on the meaning of a tag enough to label similar material with terms for there to be cooperation and shared values. In this case some users may optionally alter their vocabulary.

- While folksonomies are regarded as subject categorization systems, some of their tags can be used in unexpected and interesting ways such as acting as a communicative tool in a photographic conversation where participants try to define a term using their own photographs and metadata [74].

Disadvantages of folksonomy

- Ambiguity: with the existence of an uncontrolled and shared vocabulary within folksonomy, tags tend to be ambiguous since their assignment process to contents does not follow explicit systematic guidelines and scope notes. Another source of ambiguity arises from the use of acronyms which cause no problems in environments with controlled vocabularies but in folksonomy, the same acronym can tag contents from completely separate domains and ideas [74, 92].
- Spaces and Multiple Words: some Folksonomies like Flickr seems designed primarily to deal with the single form of words. Delicious prohibits the use of spaces in tag names, whereas Flickr allows them. Users in some cases create a single tag from multiple words without spaces, i.e., `'flickrtravelaward'` on Flickr. Both systems ignore letter case, which minimise the significance of tagging using acronyms [74, 92].
- Synonyms: these are different words with similar or identical meanings [81]. Like its vocabulary, folksonomy does not enforce rules to control the use of synonyms in the system. For example, contents related to Apple Macintosh computers can be tagged using tags such as `'mac'`, `'macintosh'`, and `'apple'` [74].
- Different word forms, plural and singular, exist too [74] and this is a problem because a question aimed at one cannot retrieve the other unless the system is built to perform such replacements [81].
- Polysemy: it is a word with multiple meanings. `'poly'` means many and `'semy'` refers to meanings [81]. For example, the `'apple'` term can be used in tagging the fruit, an Apple retail store, or an Apple computer [64]. Because

tags cannot be semantically distinct and there are no rules for selecting them, inappropriate connections between items can exist resulting in using a tag to describe different concepts [64] and this leads to a decrease in the quality of the retrieval results.

In general, the use of controlled vocabularies can resolve some of these disadvantages but due to the nature of some of the tagging systems, including a controlled vocabulary in them would be impossible [74].

Clustering

Data clustering is used for statistical data analysis that partitions a dataset into subsets of similar objects or data clusters [12].

The clustering technique is applied to a wide range of topics and areas such as pattern recognition, compression, and classification [37]. It is used in folksonomies to improve search and navigation by addressing problems like annotating tags using shorthand writing, having tags with high diversity, redundant tags, and tag ambiguity since the uncertainty of a single tag in a cluster can be overwhelmed by the additive effects of the rest of the tags [43].

In Flickr, clustering discriminates between different meanings of a user query. For example, searching with the tag “apple” will retrieve several groups of pictures. The groups represent the apple fruit, apple products such as iPods, iMacs, and New York city. The user may interactively disambiguate his query by selecting the appropriate group [43].

2.3.2 Princeton WordNet (PWN) Ontology

Overview

This is an online lexical reference system. In this case, English verbs, nouns, adverbs, and adjectives are organised into synonym sets which individually represent one underlying lexical concept [79].

Organizing lexical information, based on the standard alphabetical procedures, entails the gathering of words with similar spellings and the scattering of similar or related words throughout the list in a haphazard manner [79]. Regrettably, the main pitfall of this is that there is no obvious or simple alternative allowing lexicographers to keep track of the activities that have been carried out and helping readers with finding their target words [79].

Whilst users can easily find these words in the dictionary list, this process can be tedious and time consuming. That is why many people prefer to ignore the use of the dictionary. This is because finding the information they require would result in interrupting their work and breaking their line of thought [79].

With all the technological advancement in modern society, there is a solution that has been put forward to resolve the complaint. The first obvious remedy is the use of on-line dictionaries [79]. These are forms of lexical databases that are readable by the computer. Here, computers are used to search for words throughout the alphabetical list since the machinery is much faster than any human [79]. As soon as the user keys in or selects the specified word, a dictionary entry is made available for him to use [79]. Furthermore, since dictionaries are printed from tapes readable to computers, it is relatively simple to convert such prints into the appropriate form of lexical database [79]. Since it is relatively inefficient to limit the utilization of powerful machinery to rapid page turners, Princeton WordNet (PWN) is a proposal for a more effective combination of modern high speed computation and traditional lexicographic information [79].

In 1985, at Princeton University, a gathering of psychologists and linguists worked on developing a lexical database with the initial idea of offering the feature of dictionary searching in a conceptual manner rather than the alphabetical one [79]. This feature was supposed to work in close conjunction with an on-line dictionary. The progress of the work, forced the original plan to evolve into a more ambitious one with

reformulated principles and goals and the final product was Princeton WordNet (PWN) [79].

Number of words, synsets, and senses

POS	Unique Strings	Synsets	Total Word-Sense Pairs
Noun	117798	82115	146312
Verb	11529	13767	25047
Adjective	21479	18156	30002
Adverb	4481	3621	5580
Totals	155287	117659	206941

Table 2.1: Princeton WordNet (PWN) v3.0 Database Statistics 2006 [120]

As shown in the table above, current Princeton WordNet (PWN) (version 3.0) contains 155,287 lexical entries that are organized into 117,659 synsets which are sets containing grouped synonyms and linked to each other by conceptual relations.

Different from a standard dictionary, Princeton WordNet (PWN) divides the lexicon into the categories: nouns, verbs, adjectives, and adverbs [79].

The most ambitious feature of Princeton is that it has attempted to organize lexical information in terms of word meanings rather than word forms [79].

The Semantic Relations in Princeton WordNet (PWN)

Semantic Relation	Syntactic Category	Examples
Synonymy (similar)	N, V, Aj, Av	pipe, tube rise, ascend sad, unhappy rapidly, speedily
Antonymy (opposite)	Aj, Av, (N, V)	wet, dry powerful, powerless friendly, unfriendly rapidly, slowly
Hyponymy (subordinate)	N	sugar maple, maple maple, tree tree, plant
Meronymy (part)	N	brim, hat gin, martini ship, fleet
Troponomy (manner)	V	march, walk whisper, speak
Entailment	V	drive, ride divorce, marry

Note: ***N*** = *Nouns* ***Aj*** = *Adjectives* ***V*** = *Verbs* ***Av*** = *Adverbs*

Table 2.2: Semantic Relations in Princeton WordNet (PWN) [78]

The table above lists few semantic relations in Princeton WordNet (PWN). These relations were selected from a wide range of semantic relations which can be established between words and word senses. The reasons for selecting these semantic relations to be included in Princeton WordNet (PWN) are [78]:

- They are familiar in concept thus users do not need any advanced training in linguistics.
- They can be applied broadly throughout English.

The semantic relation is formed using pointers connecting word forms or synsets [78]. The semantic relations in Princeton WordNet (PWN) are:

1. Synonymy: (syn = same , onyma = name) similarity is the most important relation in Princeton WordNet (PWN) [79] because PWN represents word senses using sets of synonyms (synsets) [78]. A definition of synonymy considers two expressions as synonymous in a linguistic context if substituting one with the other in the same context will not alter the meaning [79]. Synonymy is a symmetric relation between word forms [78].
2. Antonymy (opposing-name): it is a symmetric relation too between word forms that is especially important when organizing the meanings of adjectives and adverbs [78].
3. Hyponymy (sub-name) and Hypernymy (super-name): hypernymy is the inverse of hyponymy. Hyponymy/hypernymy is a semantic relation between word meanings. Both relations are transitive between synsets. Hypernymy is responsible for hierarchically organizing the meanings of nouns because normally only one hypernym exist [78]. Inside the hierarchy, hyponym is placed below hypernym. This hierarchical representation is used in the construction of information retrieval systems [79].
4. Meronymy (part-name) and Holonymy (whole-name): Holonymy is the inverse of Meronymy and both are complex semantic relations [78].
5. Troponymy (manner-name): for verbs, this relation represents what hyponymy represents for nouns but it has shallower hierarchies [78].
6. Entailment: it is a relation between verbs in Princeton WordNet (PWN), for example the verb to divorce is entailed by to marry [78].

2.3.3 MultiWordNet (MWN) Ontology

Overview

MultiWordNet refers to a project which has the aim of developing an Italian WordNet that is in strict alignment with Princeton WordNet (PWN). According to [93], the first

version of the MultiWordNet has an estimate of around 37000 Italian words which are organized into 28000 synsets with information that is related to the correspondence between English and Italian Princeton WordNet synsets.

Moreover, MultiWordNet (MWN) is perceived to adopt a methodological framework that is highly different from Euro WordNet which is a multilingual database with independent WordNets for several European languages and correspondences between them.

The model adopted by MultiWordNet (MWN) builds WordNets in many languages while trying to retain the semantic relations in the Princeton WordNet (PWN) whenever possible. This is achieved by creating the new synsets in correspondence with the Princeton WordNet (PWN) synsets whenever that can be possible. Looking into these English synsets, any semantic relation that exists between them can be imported. This simply means that any relation between two synsets in Princeton WordNet (PWN) must also exist between the corresponding synsets in the new language.

The MultiWordNet model is perceived to be less complex and it ensures the highest level of compatibility among different WordNets. It follows in a strict manner the building criteria and subjective choices of Princeton WordNet (PWN), however the MultiWordNet (MWN) model is believed to have some shortcomings. The most notable one is that MultiWordNet (MWN) is extremely dependent on the lexical and conceptual structure of one of the languages involved, yet this can be lessened by letting the new WordNet branch from the Princeton WordNet (PWN) in situations where that might be considered necessary.

In MultiWordNet (MWN), automatic procedures can be derived in an aim to speed up both the divergence detection, between the WordNet being developed and the Princeton WordNet (PWN), and the building of corresponding synsets. Princeton WordNet (PWN) can be a good resource to use by these procedures [93].

The first instantiation of the MultiWordNet (MWN) model was the Italian WordNet which is based on two basic automatic procedures [93]:

- Assign procedure: when assigning an Italian word sense, the procedure puts together a weighted list of the most likely Princeton WordNet (PWN) synsets correspondences.
- (Lexical Gaps) LG procedure: it allows lexical gaps to be detected. These gaps often exist when a lexical concept of a given language is represented using a free combination of words in a different language [93].

Both of these procedures apply the Collins bilingual dictionary, the electronic version, as a vital linguistic resource. The Collins bilingual dictionary is of medium size. Its English section contains 40,959 headwords and 60,901 translation groups whereas the Italian section includes 32,602 headwords and 46,565 translation groups [93]. Translation group (TGR) refers to a set of translation equivalents. The job of this group is to translate one of the senses of a source language word [93].

The Assign-procedure

Adopting the MultiWordNet (MWN) model is all about generating Italian synsets which are considered to be synonymous (semantically correspondent) of the Princeton WordNet (PWN) synsets whenever possible. If this can't be achieved then it is a case of English-to-Italian or an Italian-to-English lexical idiosyncrasy [93].

Italian synonymous synsets can be constructed using two strategies as follows [93]:

- English-to-Italian translation equivalents are used in the first strategy. If there exist 'S' (a PWN synset), then the strategy is looking for all Italian translation equivalents which are cross-linguistic synonyms of the English words of 'S'. The retrieved translation equivalents represent the Italian synonymous synset of 'S'. English-to-Italian lexical idiosyncrasy occurs if no translation equivalents are retrieved.

- Italian to English translation equivalents are used in the second strategy. If there exist 'I' (an Italian word sense), the strategy is looking for 'S' (a PWN synset) including at least one English translation equivalent of 'I'. The strategy then creates a link between 'I' and 'S'. Italian-to-English lexical idiosyncrasy occurs when a set of Italian synonyms have no PWN synonymous synset.

The Lexical Gaps-procedure

Based on contrastive analysis literature, a lexical level can have different types of idiosyncrasies whenever a source and a target language exist. However, just a few of these idiosyncrasies are relevant to the coded information inside MultiWordNet (MWN) which is strictly aligned with the Princeton WordNet (PWN) building criteria [93]. In other words, within MultiWordNet (MWN), let's assume there is 'L1' (a synset of language#1) containing lexical units 'w_1, ..., w_n'. 'L1' will only have a correspondent in language#2 'L2' if there is at least one or more lexical units in 'L2' which are cross-language synonyms of 'w_1, ..., w_n'. This result in having only two types of idiosyncrasies implying the lack of cross-language correspondence in MWN [93].

- Lexical gaps: occur whenever a language expresses a concept through a lexical unit while the other language expresses it with a free combination of words. Following the MultiWordNet (MWN) building criteria only idioms and restricted collocations are considered lexical units and thus can be synonymous with simple or compound words. On the contrary, a free combination of words is not a lexical unit and the elements are not bound specifically to each other and so they occur with other lexical items freely thus implies a missing synset for that language [93].
- Denotation differences: the translation Equivalent of a source language exists but it is more general or more specific. In the former case the translation equivalent is a sort of cross-linguistic hypernym of the source language word and in the latter case it is a cross-linguistic hyponym [93].

Information related to lexical gaps can be used in two ways. Deciding this is dependent on the type of gaps at hand. Are they Italian-to-English gaps? or vice versa [93].

- The Italian-to-English gaps: they point to a set of Italian synsets that will be entered to the Italian WordNet manually. Building these synsets in correspondence with any English synset is not possible and hence they cannot be constructed based on the results of the Assign-procedure [93].
- The English-to-Italian gaps: they point to Princeton WordNet (PWN) synsets which lack any Italian correspondents and they can be excluded from those selected by the Assign-procedure [93].

2.4 Stemming

2.4.1 Background

In linguistics, Morphology studies the internal structure of words. It has two subtypes: derivational and inflectional. The latter subclass is important in stemming [1].

An inflection produces one or more grammatical categories by adding a prefix, suffix or infix, or another internal modification such as a vowel change [20]. Conflation is about reversing the inflection process and within the English Language, it has problems when working with [1]:

- Verbs that do not have a strict inflection pattern and change their stem when changing tenses (e.g. throw, threw, thrown).
- Verbs that are completely irregular (e.g. be, was, been)

These problems cause stemming errors where unrelated words are conflated together and unrelated terms are matched. To overcome these errors and have an efficient and effective conflation, affix removal conflation techniques were established and they are

referred to as 'stemming algorithms or stemmers'. While ignoring the occurrence of occasional errors, they attribute to performance improvements [1].

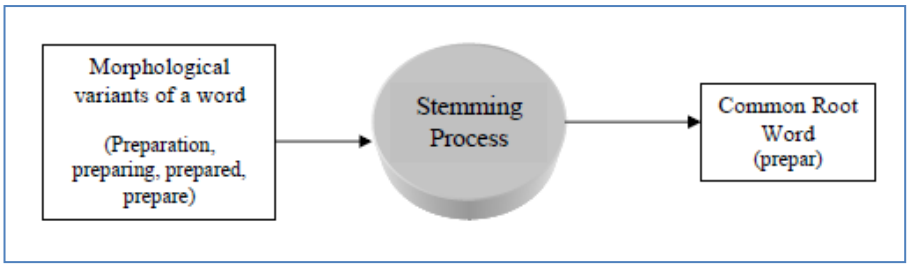

Figure 2.2: The Stemming Process [106]

Stemming is a popular tool for word standardisation that matches morphologically related terms. Its construction is a language specific process [25] which is harder in languages that are considered morphologically complex or known to have many irregularities [63].

Most studies have been focused on the development of stemming algorithms in English, and similar languages such as Slovene and French [105]. in English, a word consists of a stem, which refers to some meaning, and affixes to modify that meaning and/or to fit the word for its syntactic role [88]. It is used in the fields of data mining and information retrieval systems (IR) to enhance the quality of the results and cut down on the storage requirements for the processed information [25, 118].

According to [106], stemming provides two basic advantages:

- Increased recall rate of the information retrieval. The recall rate represents the number of relevant documents retrieved divided by the total number of documents retrieved.
- Memory saving via reducing the entries in the index table, thus reducing its size. Replacing full terms with their corresponding stem can achieve a 50% compression [106].

2.4.2 Definitions

- Stemming is an automated process to extract the base form of a given word of a language [99].
- Stemming is the process where affixes (prefixes, infixes, or/and suffixes) are removed from words to reduce them to their stems or roots [16, 46].
- Stemming is the process of reducing inflected words to their stem by removing any attached affixes from a word [25].

2.4.3 Techniques

As mentioned in the definition, stemming is about removing affixes from words and stemming algorithms can be categorized based upon this to:

- Affix Stemming Algorithms: in addition to removing suffixes, common prefixes are removed using several approaches [25].
- Suffix-Stripping Algorithms: these depend on a list of stored rules to guide the stemming process [25]. This is commonly accepted as a good idea.
- Prefix Stripping Algorithm: This is not widely practised and not generally felt to be helpful except in some subject domains such as chemistry [87].

When developing a stemming algorithm, certain issues must be considered such as iteration and context awareness. Suffices are attached to a word in a certain order that can be put in a set of order-classes and starting from the end of the word, the stemming algorithm will iteratively remove suffices one at a time [2]. Regarding context, a stemming algorithm can be one of the following [2]:

- Context-sensitive Algorithm: it involves a number of qualitative contextual restrictions preventing the removal of endings that may produce wrong stems.
- Context free Algorithm: it removes endings without any restrictions.

In addition, stemming algorithms can be labelled as:

- Morphological Stemming algorithms (such as Porter Stemming algorithm): this is based on morphological issues that are completely independent from the syntactic and semantic structure of the sentence. Both inflections and derivational affixes are removed [86].
- Syntactic Stemming algorithm (such as Stanford Stemming algorithm): this is performed during the syntactic analysis of the sentence where only inflections are removed. Thus the word 'arrivals' in Stanford stemming algorithm is stemmed as 'arrival' whereas it is stemmed as 'arrive' in a Porter stemming algorithm [86].

Moreover, stemming algorithms can be categorized according to their strength as follows [16]:

- Light stemming algorithm: it adopts understemming, meaning it does not conflate words of the same concept resulting in a reduced recall where fewer relevant results are returned by a Text Retrieval system. It strips suffixes based on regular expressions such as 'ing', 's', 'e' [16, 88].
- Heavy stemming algorithm: It adopts overstemming, meaning the conflation of words from different concepts resulting in reduced precision caused by the return of irrelevant results [16, 88].

Below are other existing types of stemming algorithms which vary in their performance and accuracy and methods [25]:

- Brute-Force Algorithm: it queries a lookup table, containing relations between root forms and inflected forms. It looks for a matching inflection and then the associated root of the match is returned if found [25].
- Lemmatisation Algorithm: initially it identifies each part of speech (POS) of a word and then normalises them using specific rules for each part of speech. Determining the correct POS is essential in this type [25].
- Stochastic Algorithm: it uses probability to find the root of a word. By training the algorithm on a table of root to inflected form relations, a probabilistic model is developed [25].

2.4.4 Types

Stemming algorithms can be classified into three groups depending on the method used to produce stems as shown in the figure below:

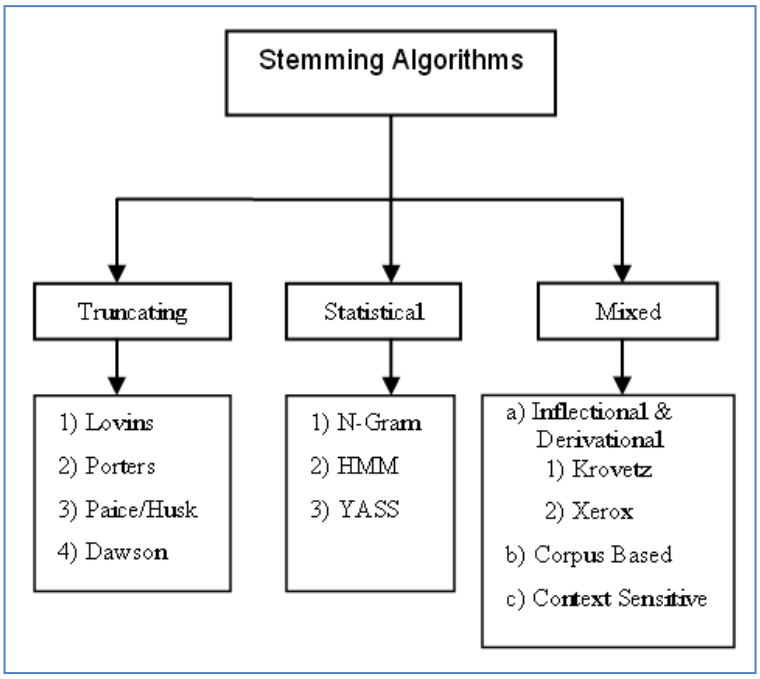

Figure 2.3: Types of Stemming Algorithms [55]

Truncating Method (Affix Removal)

It involves removing the affixes of a word and below are some adopting stemming algorithms [55]:

- Lovins Stemming Algorithm [58]: proposed by Lovins in 1968, it uses two tables. The 1st table stores 294 endings, 29 conditions, and 35 transformation rules arranged on a 'longest match' principle [55] and the 2nd table stores some rules dealing with double consonants and handling other adjustments. Based on the first table, the algorithm removes only the longest suffix from a word and then recodes it using the second table which performs some adjustments on the stem converting it into a valid word [55, 87]. The advantages of this stemming algorithm are its fast speed since it is a single pass algorithm and it has the ability to cope with special cases such as double constants and irregular plurals [55]. The drawbacks include its consumption of

time and data, the unavailability of many suffixes in the first table can sometimes be unreliable as it fails to construct words from the stems or to match stems to like-meaning words [55].

- The Porter Stemming Algorithm [16, 25, 58, 86, 88, 99]: proposed by Porter in 1980. This has since undergone many modifications [55]. It is considered as a light suffix-stripping stemming algorithm. It works by removing common suffixes iteratively using a 5-step sequence with a different lookup table being used in each step [87]. Porter designed a framework called 'snowball' to help others adopt the algorithm to the language of their choice [55]. The porter algorithm is fast, efficient, and simple thus it is commonly used in TR systems [99]. Its simplicity is regarded as a disadvantage too since it causes the stemming algorithm to produce incorrect stems in many cases (e.g. it does not conflate the words 'add' and 'adding') [16]. The focus on developing stemming algorithms was made mainly on the English language with scattered but great efforts made in other more complex natural languages like Arabic or Turkish. The implementation of a stemming algorithm involves encoding it to a programming language such as C or Java for example [97]. The lack of unambiguous stemming algorithms makes the implementation process difficult and leads to a shortage of readily available stemming algorithms in non-English language. This was the driving force for Porter to develop Snowball [97]. Snowball is a language to develop stemming algorithms. It is quite small and for experienced programmers, it can be understood in hours [97]. It has its own

complier and script. The complier translates the Snowball script (.sbl file) into an equivalent program of one of two formats below but In the end, each stemming algorithm will have its standard vocabulary of words and their stemmed equivalents [97].

  - An ANSI C program: the result is a program file and corresponding header file.
  - A Java program

- Paice/Husk Stemming Algorithm [58]: this was developed in the late 1980s in Lancaster University by Chris Paice and Gareth Husk [2]. Initial implementation was in Pascal but was followed with other versions in Java, C , and Perl [2]. It is a simple heavy iterative stemming algorithm [87, 88] which uses 120 rules stored in one table and indexed by the last letter of a suffix for quick access [55, 87]. Each iteration involves looking up a rule based on the last character in the word. If a match is found, the rule decides whether to delete or replace the ending and then the process repeats itself, otherwise the algorithm terminates. The algorithm is designed to terminate in other situations too, such as if a word starts with a vowel and there are only two letters left or if a word starts with a consonant and there are only three characters left [55, 87]. In this stemming algorithm, the rules lead to heavy stemming that is considered extremely advantageous for index compression but tends to produce many overstemming errors [55, 87]
- Dawson Algorithm: Considered as an extension of the Lovins algorithm, it is similarly fast but it uses a list that is much larger and comprehensive with about 1200 suffixes stored in reverse order, indexed by their length and last letter, and organized as a set of branched character trees for rapid access. This

stemming algorithm is very complex and lacks a standard reusable implementation [55].

Paice concluded that the Porter stemming algorithm has a smaller stemming-error rate than the Lovins stemming algorithm which was noted to have better data reduction [55]. The large suffix set in the Lovins algorithm made it much bigger than the Porter algorithm but gave it the advantage of fast speed because it is implemented using two major steps [55].

Statistical Method

Implementing this method means that the stemming algorithm must perform a statistical procedure before removing the affixes [55].

- N-Gram Stemming Algorithm [58]: it is language independent stemming algorithm which represents a set of 'n' consecutive characters extracted from a word. The concept here is that similar words will have a high proportion of n-grams in common. If 'n' equals to 2 or 3, the extracted words are called digrams or trigrams, respectively [55].

Mixed Methods: Inflectional and Derivational Methods

- Krovetz stemming Algorithm: it was developed in 1993 by Robert Krovetz [55] and it utilizes the internal structure of a word (morphology), a dictionary, and a list of exceptions [16]. The process starts by removing the suffix and then looking up the dictionary for recoding the stem to a spell-checked meaningful word [55]. Depending on a dictionary has its own problems. First the dictionary must be created manually in advance which is labour intensive and this leads to the next problem when the stemming algorithm is unable to deal with a word because it is not listed in the dictionary [55]. Using the inflectional and the derivational morphology analysis [55] made this algorithm complex but

also accurate as it generates morphologically correct stems, handles exceptions, and process prefixes too [55]. Compared to the Porter stemming algorithm, it is slower with large size input documents [16, 55] and becomes weaker and less effective [55]. In general, it is considered as an effective, light (lighter than Porter and Paice/Husk) and accurate algorithm [55] which is why Krovetz recommended its use as a pre-processing step when working with a heavy stemming algorithm to increase speed, effectiveness [55] and to reduce common errors [16].

2.4.5 Non-English Stemmers

The internet has made a large volume of information, in multiple languages, available online. The need to access specific information increased the felt demand for a multi-lingual text retrieval system [89]. For example, search engines are getting more sophisticated using advanced search parameters, and classification tools [82].

The early research was mainly on English language, and then major European languages followed. These languages have few Standard stemmers available. Other languages such as languages from the Indian sub-continent are making progress but the scarce availability of tools and other lexical resources are slowing the process [89].

According to [42], spoken languages have a rough classification as follows:

- Inflective languages: words consist of a stem and a fixed number of suffixes and/or prefixes, thus the number of combinations is fixed. Most European languages fall into this class.
- Agglutinative languages: words consist of a stem and a potentially infinite number of suffixes. Hungarian, Turkish, and Korean are examples of such languages.
- Isolating languages: words are fixed thus each word is also the stem. Examples of such languages are Chinese or Vietnamese.

- Intraflective languages: here, the word expresses its root meaning with consonants, and its grammatical variations with vowels intermixed with the consonants. Examples of such languages are Arabic and Hebrew.
- Incorporating languages: words consist of many stems glued together by complicated rules. Examples of such languages are some North American native languages.

In the following, the research in stemming done on Arabic and Indian languages is summarised to highlight the progress done in languages other than English and European.

Arabic

The work of Khoja attempts to find roots for Arabic words by stripping the prefixes and suffixes and comparing that against a dictionary of root words.

In 2002, Larkey found that stemming has a large effect on Arabic information retrieval, at least in part due to the highly inflected nature of the language [62]. At the same time, Darwish presented a rapid method of developing a shallow Arabic morphological analyzer based on automatically derived rules and statistic [32]. Recent work done by Sembok developed an Arabic stemmer with the rule-based approach plus a dictionary of root words to verify the validity of the root candidates [104].

Indian

For Indian language, early and noted research includes the work of Larkey and others in 2003 which presented a light stemmer in conjunction with list of common suffixes. Another stemmer with a similar approach was developed in 2003 by Ramanathan and Rao which used a hand crafted suffix list and performed longest match stripping [45]. For their stemmer, Chen and Gey opted for a statistical method. In 2007, a Bangali stemmer was presented by Dasgupta and Ng [89] while Islam et al. proposed a light weight stemmer for Bengali which strips the suffixes using a predetermined suffix list.

YASS stemmer was developed by Majumder et al. (2007) based on statistical approach using string distance measure[73]. For Gujarati , Suba et al. (2011) developed two stemmers. The first one is a lightweight stemmer based on a hybrid approach and the other one is a heavyweight stemmer based on a rule-based approach [110]. Gupta and Lehal (2011) had their stemmer for Punjabi which obtains the stem and then checks it against Punjabi noun morph and proper names list [44].

2.4.6 Applications

Stemming has some applications in machine translation. For example, the work done by Lee presented a morphological analysis technique to improve statistical machine translation qualities. The technique improves Arabic-to-English translation qualities significantly [65]. The experiments by Popovic and Ney regarding statistical machine translation from inflected languages into English showed that the use of word morphemes improves the translation quality [95]. The model proposed by Yang translated unseen word forms in German-English and Finnish-English text by hierarchical morphological abstractions at the word and the phrase level and showed improvements over state-of-the-art phrase-based models [121].

Stemming is used also in the area of document summarization [31, 85]. The XDoX summarizer designed by Hardy and others used stemming for data processing. The XDoX produced readable, coherent and well organized summarizes. In most cases the system successfully presented main points, skipped over minor details, and avoided redundancy [48]. Mixed models are used by Arora and Ravindran to capture topics and pick up sentences and then evaluate the generated summary using the Porter Stemmer through the ROUGE evaluator [9].

Text classification is about the automatic pre-defined label placement on previously unseen documents. It is used in document indexing, e-mail filtering, web browsing, and personal information agents. Stemming is used in many text classification experiments [102]. The role of stemming in text classification [41, 82] is getting different point of

views . The research done by Riloff in 1995 concludes that stemming algorithms are appropriate for some terms and that having all morphological variants is more beneficiary [41]. In 2000, Busemann had shown that morphological analysis increases performance for a series of classification algorithms in German [21]. It is also used in text mining and information extraction [38].

2.4.7 Discussion

The next table summarizes the key features of all mentioned algorithms [55].

Advantages	Limitations
Truncating (Affix Removal) Methods	
Lovins Stemming algorithm	
• Fast, single pass algorithm. • Handles removal of double letters. • Handles many irregular plurals.	• Time consuming. • Missing some suffixes. • Not very reliable and frequently fails to form words from the stems. • Dependent on the author's technical vocabulary.
Porters Stemming algorithm	
• Compared to Lovins it's a light stemming algorithm. • Has the best output compared to other stemming algorithms with lower error rate. • Snowball is language independent.	• Some produced stems are not real words. • Time consuming because of its 5 steps and 60 rules.
Paice / Husk Stemming algorithm	
• Simple form with each iteration doing	• Heavy algorithm and over stemming can

deletion and replacement.	happen.
Dawson Stemming algorithm	
• Covers more suffixes than Lovins. • Fast execution.	• Very complex. • Lacks a standard Implementation.
Statistical Methods	
N-Gram Stemming algorithm	
• Language independent.	• Not time efficient. • Needs significant space for creating and indexing the n-grams. • Not a practical method.
Mixed Methods (Inflectional & Derivational Methods)	
Krovetz Stemming algorithm	
• A light stemming algorithm. • Can be used as a pre-stemmer for other stemming algorithms.	• For large documents, it is not efficient. • Can't cope with words outside the lexicon. • Not consistent in producing good recall and precision. • Lexicon needs to be created in advance.

Table 2.3: Comparative Summary of some Stemming algorithms [55]

In [36], the strengths of four stemming algorithms were evaluated using six metrics and they were ranked from strongest to weakest as follow: Paice, Lovins, Porter, and SRemoval.

Correctness, retrieval effectiveness, and compression performance are several criteria for judging stemming algorithms [106].

The effects of stemming on retrieval performance have been targets of several investigations which found that stemming improves retrieval's performance and that there were no consistent differences in performance between different stemming

algorithms [88]. This method does not provide any insights on stemming algorithm optimisation [63, 88].

All previously discussed stemming algorithms do not function 100% meaning they outperform in some areas but can be a let-down in others. Still, they are good enough to be applied to the text mining, NLP or IR applications [55]. Stemming algorithms have many similarities but the main difference is their approach. The rule-based approach stemming algorithm does not guarantee a correct output every time and the produced stems are not always correct words, whereas the linguistic approach does not properly stem words outside the lexicon which must be exhaustive. Moreover, the statistical approach is language independent but does not always give reliable and correct stems. In many existing stemming algorithms there is a trade-off between overstemming and understemming [88]. A perfect stemming algorithm should not overstem or understem and this can be achieved if it takes into considerations words syntax, semantics, and their POS. Also, including a lookup dictionary will be beneficial in reducing errors and converting stems to words [55].

2.5 Summary

The chapter discussed many topics related to the thesis. It began by giving definitions and a background literature to Folksonomies. Some advantages and shortcomings of Folksonomies were outlined. Next, an overview of the Princeton WordNet and MultiWordNet ontologies was included because they are used in the TE system with tags to enhance their quality. Adding a stemming component to the TE system required reading into their background, definitions, techniques, and types to decide on the best algorithm to use. The chapter concluded by discussing algorithm analysis and complexity to evaluate the algorithm of the TE system which is discussed in the next chapter to highlight its scope, components and implementation methods.

3 The TE (Tag Enhancer) System

3.1 Introduction

This chapter will address the area of study that has been chosen by this research for the critical algorithm efficiency analysis and modification. The TE system has three main sections: the proposed system architecture, the prototype, and the experiment. This chapter will discuss only the prototype section, highlighting the decisions that were made throughout the process with respect to scope, space and time. Taking note of these decisions is very important when doing the algorithm analysis because they will be the focus of the critical discussion later on.

3.2 Overview

In the TE system, whenever the user provides a tag ‘user tags’, a new set of tags ‘system tags’ are added by the system database for the sake of overcoming the lack of semantics in the user tags. System tags are extracted from different resources depending on the user tag as follow [72]:

- User tag is IN the vocabulary: its related system tags will be added from Princeton WordNet (PWN) and MultiWordNet ontologies (semantic resource).
- User tag is NOT in the vocabulary: its related system tags will be extracted using social tag-based system clustering (social resource).

The study claims that the new added system tags along with the raw user tag will improve the search process by providing more accurate results [72].

3.3 The Scope of TE

In [72], a generic architecture for a Tag-Based System is presented with the components: tagging component, search component, semantic component, clustering component and database component.

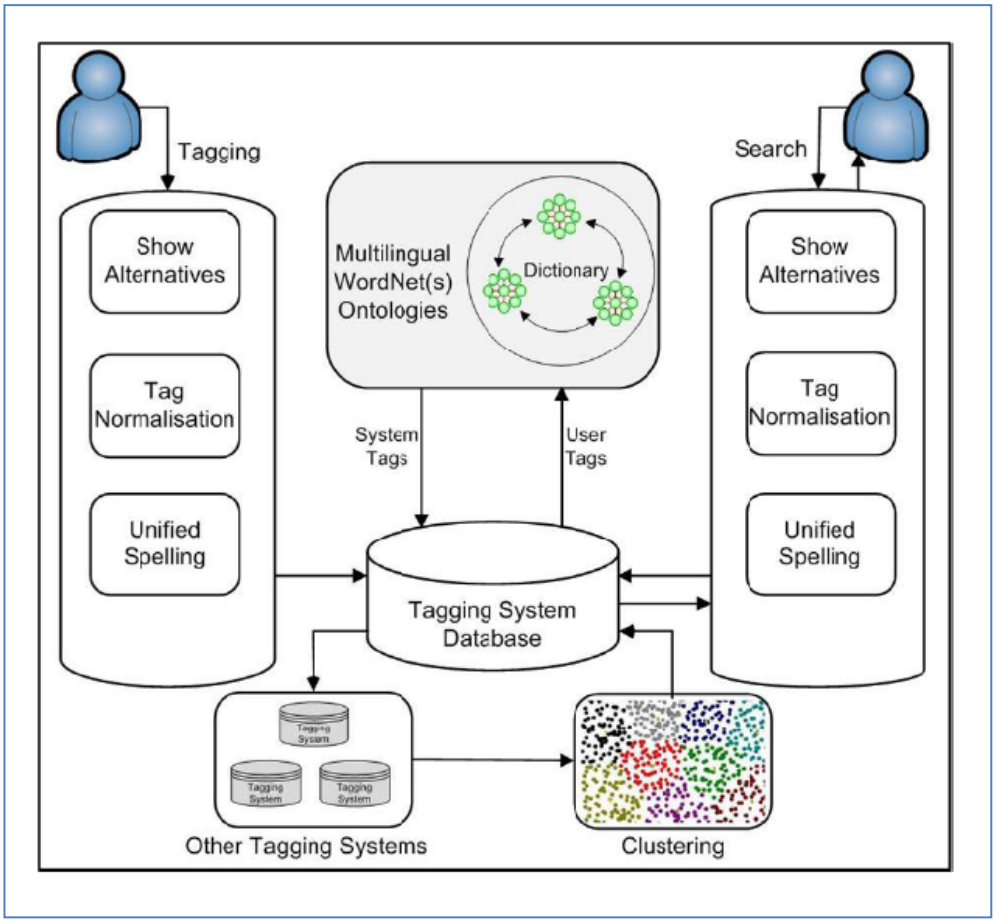

Figure 3.1: The Proposed Generic Architecture for Tag-Based Systems [72]

Before implementing the TE algorithm, the scope of the proposed generic architecture was limited to only some of the components listed previously and they are the semantic component and the clustering component. The TE is only dealing with the following tagging problems: semantic relations, multilingualism and shorthand tags. TE mainly proposes to improve the following aspects [72]:

- The semantic aspect via the semantic component (semantic resource)
- The multilingualism aspect via the semantic component (semantic resource)
- The clustering aspect via the clustering component (social resource)

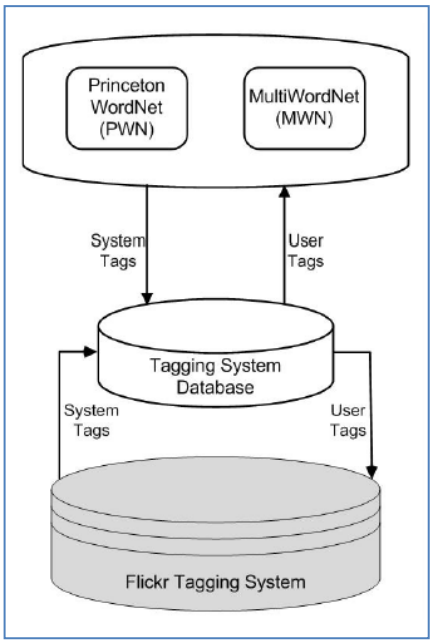

Figure 3.2: The Scope of TE [72]

3.3.1 The Semantic Component

This component deals with the following problems [72]:

- Word Synonyms & Semantic Relations: the interaction here is between the tagging system database and Princeton WordNet (PWN). It raises a query to retrieve a set of words that are relevant to the user tag as they are either synonyms or hypernyms of it. In order to address this task, the Princeton WordNet (PWN) ontology is used. In Princeton WordNet (PWN) and other ontologies which are based on the Princeton WordNet (PWN) structure, words have relations between them and each word has many senses which are different meanings for the same word. Senses in Princeton WordNet (PWN) and similar ontologies are generally ordered from most to least frequently used, with the most common sense listed first and so forth. In the study, relations deals only with the first sense.
- Multilingualism: Most tag-based systems do not force users to use specific languages during the processes of tagging or searching. This implies that unless the search keyword and the user tag use the same language, no results will be

found. Multilingual lexical ontologies can be used as translators since they store a few languages, (usually in a database) with a cross language link among the word translations in different languages. EuroWordNet ontology contains seven European languages (Dutch, Italian, Spanish, German, French, Czech and Estonian) whereas MultiWordNet (MWN) covers only English and Italian. MultiWordNet (MWN) is free for researchers thus it is used in this study by the semantic component which queries it using the user tag to retrieve relevant words in English or Italian.

When a new tag is submitted, the TE system queries three resources in worst case scenario. For example, submitting the tag “btw”, which is shorthand writing for “by the way”, to the TE system will require querying the Princeton WordNet (PWN), the MultiordNet (MWN), and finally the clustering component. Moreover, a tag can be semantically rich and yield over 80 system tags (see table 5.5 and table 5.11) after querying the semantic component.

[72] considered the critical factors of time and space and decided on saving time during the search process and generating the system tags when submitting new tags. [72] claims that time is more important than space especially during the search process because a response time is involved, whereas it is not noticeable at the tagging process. Furthermore, the study points out that since all the data used are textual, the space factor is less significant since they consume small space due to their nature. The thesis discusses this point thoroughly in chapter 5.

3.3.2 The Clustering Component

This component handles the problem of shorthand writing. It interacts only with its tagging system database and also with at least one external tagging system database. It is an additional source along with Princeton WordNet (PWN) to add semantic to tags when Princeton WordNet (PWN) fails to do so, particularly in the case where tags are shorthands, colloquial words, or specialised technical terms. To save time on clustering, the architecture can use the Flickr tagging system to retrieve tag clusters

using APIs provided by Flickr. The number of clusters varies from one tag to another and the same can be said about the tags inside each cluster. When a tag is submitted to the Flickr tagging system database, a variable number of clusters will be retrieved with a different number of tags in each cluster. According to [72], most tags retrieve one cluster only. Furthermore, it was found that the Top-N tags in the 1st cluster are the most related tags [72]. Therefore, the TE system decided to add the Top-3 tags from the 1st cluster as system tags. The actual clustering algorithm used in Flickr has not been officially released. Revealing the clustering algorithm will help in automating the process of judging the clusters relatedness. This is important since the Top-N tags in the most related cluster can be used as system tags to provide a better context. The TE system is limited to retrieving only the Top-10 tags in each cluster and suggests running these procedures periodically to keep up-to-date with the social vocabulary.

3.3.3 The Database Component

The database of the TE system stores information about tagged objects, clusters, etc. It interacts with the components below as follows:

- Semantic Component: to store system tags in the database
- Clustering Component: to store system tags in the database

The design of the database component is illustrated below. It is built using MySQL Database Management System.

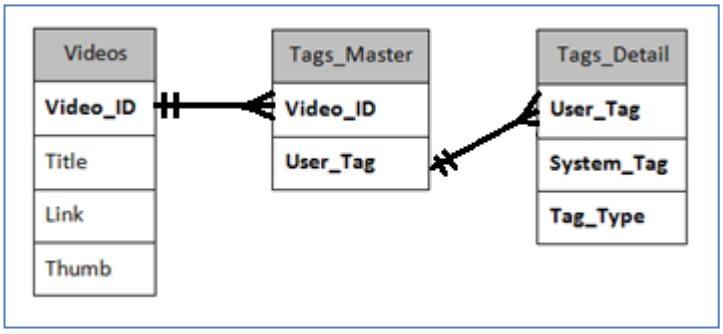

Figure 3.3: The ER Model of the TE Database

Table	Key	Attribute	Description
Videos	primary	Video_ID	Unique ID for identifying each video
	-	Title	Video title as it appears in YouTube (imported from YT). In the experiment it will be a hyperlink to the video on YouTube
	-	Link	video URL on YouTube (imported from YT)
	-	Thumb	Video thumbnail as it appears in YouTube (imported from YT)
Tags_Master	primary	Video_ID	Unique ID for identifying each video
	primary	User_Tag	Raw tag used to annotate the video
Tags_Detail	primary	User_Tag	Raw tag used to annotate the video
	primary	System_Tag	Added tag from the semantic component or the clustering component
	primary	Tag_Type	The resource of the System Tag (semantic or social)

Table 3.1: Breakdown of the Database Tables

3.4 The TE System

3.4.1 The Data

The database is populated with data from different resources as follows [72]:

- Initially, a set of English and Italian keywords stored in a String Array is used to query YouTube looking for matching videos. All videos retrieved from the YouTube site are saved in one list called 'video list' and then certain information about every video in this list is saved inside the TE database. For the TE system, the most important piece of information is the tags attached to each retrieved video.
- The TE system needs sample data (i.e. tags sample) because it is not operating a real tagging system. The tags sample is imported using YouTube's own Java Data API and stored in the TE database where the algorithm is applied on them later on.

- System tags from semantic ontologies (Princeton WordNet (PWN) and MultiWordNet (MWN)) via the semantic component.
- System tags from Flickr using its clusters via the clustering component.

3.4.2 The TE Implementation

The Programming Languages

The algorithm is implemented using Java. This is because it has a vast amount of APIs available to interact with all the resources needed for the TE system's implementation as shown below:

- YouTube APIs: which are called 'YouTube Data API'. See available documentation at: https://developers.google.com/youtube/getting_started
- Princeton WordNet (PWN) APIs: which are called 'Java API for WordNet Searching (JAW)'. See available documentation at: http://lyle.smu.edu/~tspell/jaws/index.html
- Flickr APIs: which are called 'Flickr Java API' (flickrj)'. See available documentation at: http://flickrj.sourceforge.net/

Querying the Semantic Resources

Princeton WordNet (PWN) contains only English language whilst the MultiWordNet (MWN) contains English language and Italian language. This means that the TE system can obtain English system tags from two resources whereas it only has one resource to obtain Italian system tags [72].

The TE system queries only Princeton WordNet (PWN) to get the English system tags since its WordNet version is more recent than MultiWordNet (MWN). This leaves MultiWordNet (MWN) responsible for retrieving the Italian system tags, and finding the corresponding translation for the user tags [72].

3.5 Summary

The chapter discussed the TE system starting with an overview and then it investigated its scope by detailing the components included. The role of each one of these components was explained in addition to looking at the methods used by them to perform their designated objectives. The last part of this chapter outlined the general implementation plan and its programming environment which involved the services of many APIs for manipulating the different semantic and social sources. In the next chapter, the thesis explains the methodology used to evaluate the TE system and embed the stemming component.

4 The Methodology

4.1 The Efficiency of Algorithms (The Performance)

Computational complexity theory is concerned with looking at what computational resources are required to solve a given task [10]. The questions it studies include the following:

- Many computational tasks involve searching for a solution across a vast space of possibilities. Is there an efficient search algorithm for all such tasks, or do some tasks inherently require an exhaustive search?
- Can algorithms use randomness to speed up computation?
- Can hard problems be solved more quickly if we allow the algorithms to err on a small number of inputs, or to only compute an approximate solution?
- Is there any use for computationally hard problems?
- Can we use the counterintuitive quantum mechanical properties of our universe to solve hard problems faster?
- Can we generate mathematical proofs automatically? Can we check a mathematical proof by only reading three probabilistically chosen letters from it?

The efficiency of an algorithm is considered more important than the execution technology. It measures the amount of memory and time needed by an algorithm to run [51]. Choosing the best algorithm to solve a problem is essential and there are a few approaches that guide programmers in this process as listed below [19, 52]:

1. Empirical (Performance Measurement): this method is machine dependent because the algorithm is implemented, executed and time is recorded.
2. Analytical (Performance Analysis): this method considers high-level descriptions of the algorithm. For each proposed algorithm, several factors must be determined mathematically. The factors include execution time, memory, space etc. This approach has several benefits:

- It is independent of the computer used, the programming language or the programmer's skills.
- It saves programming and testing time for inefficient algorithms.
- It tests instances of any size.

3. The hybrid approach: this uses the previous approaches referred to above. It does this by determining theoretically the form of the function describing the algorithm's efficiency and then empirically determines any required numerical parameters.

Analytical	Empirical
Inputs of all possible sizes are accounted for.	Limited set of inputs.
Comparing run times of 2 algorithms is machine independent.	Comparing run times of 2 algorithms is machine dependent. Identical environment (software and hardware) must be used.
Algorithm implementation is not required.	Algorithm must be implemented.

Table 4.1: Algorithm Performance Methods [52]

4.1.1 Empirical (Performance Measurement)

This method involves more work than the performance analysis, thus there are several steps to follow to insure better results when implementing the testing experiment. The steps are as follows [67]:

1. Setting the purpose of the experiment.
2. Deciding on the efficiency metric and measurement unit.
3. Specifying the input sample in terms of range, size etc.
4. Implementing the algorithm.
5. Generating an input sample.
6. Running the algorithm implementation using the sample.
7. Recording the observed data.
8. Analysing the results.

The implementation should be designed to provide means to record data. The first method is to use counter(s) to calculate the number of times the algorithm's basic operation is executed. An alternative method is to time certain parts of the implementation [67]. In Java, the method `'currentTimeMillis( )'` in the System class can be used. Consider the following when using the latter method [67]:

- The system's time is typically not very accurate and you can get different times while running the same code and input. Taking the average of several trials is a much better option.
- High-speed computers can report the run time as zero. The solution is to add an extra loop and go through it 'n' times, measure the total time, and then divide it by 'n' to get the time for one loop.
- Computers with a time-sharing system can return time results that include time spent on other programs. Thus, they request only user time from the system.

Test conducted by running Selection Sort program on IBM compatible PC with: Intel 80386 processor with 80387 numeric coprocessor + turbo accelerator + Borland's Turbo C compiler	n	Time (seconds)
	30...100	.00
	200	.11
	300	.22
	400	.38
	500	.60
	600	.82
	700	1.15
	800	1.48
	900	1.86
	1000	2.31
	1100	2.80
	1200	3.35
	1300	3.90
	1400	4.54
	1500	5.22
	1600	5.93

Table 4.2: Empirical Test Example(Adapted) [5]

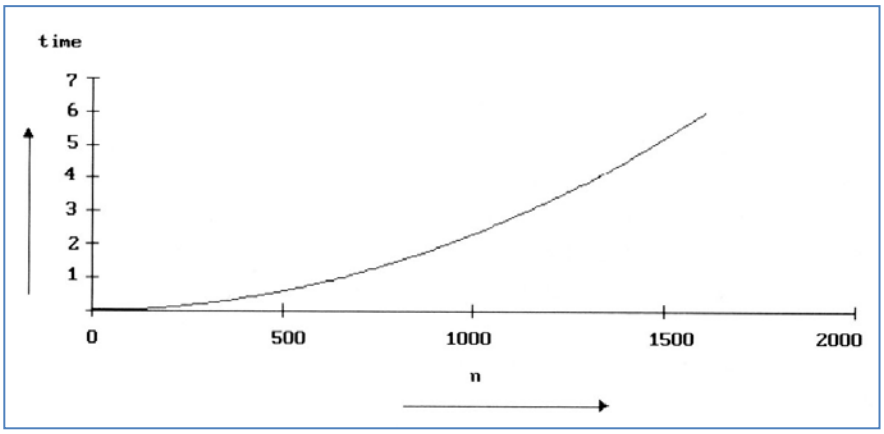

Figure 4.1: Empirical Test Example's Graph [5]

4.1.2 Analytical (Performance Analysis)

Performance analysis of algorithms is useful in [51]:

- Determining if the algorithm is practical.
- Predicting run time for large inputs.
- Comparing two algorithms with different asymptotic complexity functions.

Two criteria are used to judge the performance of an algorithm [6, 51]:

- Space complexity (storage requirement): this is the amount of memory it needs to run to completion.
- Time complexity (computing time): this is the amount of CPU time it needs to run to completion.

The memory's hierarchy is divided into levels, each with its own unique response time. The performance analysis discards the differences in those response times [51]. The actual space and time requirements of a program are dependent on [18, 67]:

- The compiler generating the machine code.
- The quality of the implementation program.
- The speed of the computer.

In algorithm design, time and space can co-exist without competing with each other to find an algorithm with minimum time and space costs [67].

Nevertheless, there are some trade-offs between the above two factors. Trading space for time is the most common. In some cases, the problem's input is pre-processed wholly or partially and then the resulting data is stored. This is called 'input enhancement' and it makes solving the problem later on much faster. Here the time is more important [67]. Another case where time takes precedence over space is 'pre-structuring' where extra space is used to provide faster and/or more flexible data access such as in hashing and B-Trees indexing [67].

Another case that deserves a mention is 'dynamic programming' where solutions to overlapping sub problems of a problem are stored in a table from which a solution to the original problem is then obtained [67].

To proceed analysing the performance of non-recursive algorithms, the next steps must be followed:

1. Decide the input's size.
2. Identify the algorithm's basic operation and whether its repetition is solely dependent on the input's size or other extra factors.
3. Identify the worst-case and average-case efficiencies. Measure best-case efficiency whenever needed.
4. Sum up the execution times of the algorithm's basic operation.
5. Calculate the sum's order of growth.

The following factors can measure the efficiency of an algorithm:

- Memory space (Space Complexity): this is measured by several factors such as the number of variables and the number and sizes of the data structures used in the algorithm [49].

- Execution time (Time Complexity): this is measured by the number of elementary actions performed by the processor in such an execution. In other words it calculates the amount of time required to execute an algorithm [49].

The performance of the above factors and the algorithm in general varies from input to input.

4.2 Time Complexity

Time Complexity describes the relationship between the size of the input and the execution time of the algorithm and it is mostly expressed as a proportionality [18]

As an example, sorting large lists takes more time than short lists and performing multiplication on huge matrices is slower than on small ones [47].

Time Complexity indicates the run speed of an algorithm [67]. The equation: T(P)= C+TP(I) defines the time required 'T(P)' to run a program 'P' where [5, 6]:

- 'C' (fixed time requirements): compile time independent of instance characteristics.
- 'TP(I)' (variable time requirements): execution time.

Measuring 'T(P)' is done using one of these methods [6]:

- Conducting an experiment using a 'stop watch' (usually time is in seconds or microseconds).
- Counting program steps.

Measuring the theoretical efficiency of an algorithm can be explained by the principle of invariance, according to which two different implementations of the same algorithm will not differ in efficiency by more than some multiplicative constant [19]. To explain more, if two implementations take '$T_1(n)$' and '$T_2(n)$' seconds respectively to solve an instance of size 'n', then there always exists a positive constant 'c' such that $T_1(n) <= cT_2(n)$ whenever 'n' is sufficiently large [19].

This principle is valid regardless of the programming language, the programmer's skills (unless it modifies the algorithm), and the computer used (of conventional design). Although changing the machine may speed up solving a problem by 10 or 100 times, still the change of algorithm will give improvements that gets more and more marked as the size of the instances being solved increases [19].

Expressing the theoretical efficiency of an algorithm is only done within a multiplicative constant. Thus, if an algorithm takes a time in the order of 'T(n)' for a given function 'T', there exists a positive constant 'c' and an implementation of the algorithm capable of solving every instance of the problem in a time bounded above by cT(n) seconds, where 'n' is the size of the instance considered.

Other units can replace seconds in the above definition by changing the constant to bound the time by aT(n) years or bT(n) microseconds [19].

Time complexity T(n) is measured in the order of a function O(f(n)). For any 'n' that is sufficiently large, this determines the upper and lower bounds on the amount of work done [18].

A computation that runs in linear or quadratic time is efficient [10]. In the analysis of algorithms, the logarithms to the base 2 are so frequently used and have their own notation 'lg n' (short for $\log_2 n$).

The objectives of the analysis of time complexity are [18]:

- To determine the feasibility of an algorithm by estimating the upper bounds of the performed work.
- To compare different algorithms and then decide on the best ones for the implementation.

Sometimes in the analysis, if work takes a constant amount of time independent of the input size, it is ignored. This helps to simplify things and the time complexity is considered constant and is denoted as O(1) [18].

Furthermore, usually the focus is on the differences in performance between algorithms performing the same task [18].

Simplified analysis can be based on the number of [18]:

- Performed arithmetic operations.
- Performed comparisons.
- Times through a critical loop.
- Array elements accessed etc.

Algorithm analysis has different scenario cases as follows [18]:

- Average Case: determines the average performance.
- Worst Case: produces an upper bound on the algorithm performance for large problems (large 'n') and it is simpler to work out. It is expressed as T(n) = n where T(n) is the maximum number of steps in any execution of the algorithm with 'n' inputs [47].

Within this context, the terms above are defined as follow [47]:

- Input size: defining the size as the input's required storage in bits is too low-level and not useful. Instead, it is problem-dependent. For example, if the algorithm is about sorting elements then the number of elements is the input size.
- Step: anything a computer does in a fixed amount of time.

4.3 Space Complexity

The space complexity for a given input is the number of elementary objects that the algorithm needs to store whilst executing [108]. It is the amount of memory space needed by an algorithm plus the space needed for its input and output [67].

Space complexity S(P) is calculated by the rule: S(P)=C+SP(I), where [5, 6]:

- 'C' (fixed space requirements): independent of the characteristics of the inputs and outputs. Examples are spaces for instruction, simple variables, fixed-size structured variable and constants
- SP(I)(variable space requirements): depend on the instance characteristic 'I'
  - number, size, values of inputs and outputs associated with 'I'
  - recursive stack space, formal parameters, local variables, return address

In computational complexity theory, some computational models use resources to solve computational problems. DSPACE is one of these computational resources specialising in memory space. It represents the total amount of memory space that a computer needs in order to solve a given computational problem with a given algorithm [100].

For an algorithm T and an input x, DSPACE(T, x) denotes the number of cells used during the (deterministic) computation T(x).

We will note DSPACE(T) = O(f (n)) if DSPACE(T, x) = O(f (n)) with n = |x | (length of x).

Note: if T(x) does not stop then, DSPACE(T) is undefined.

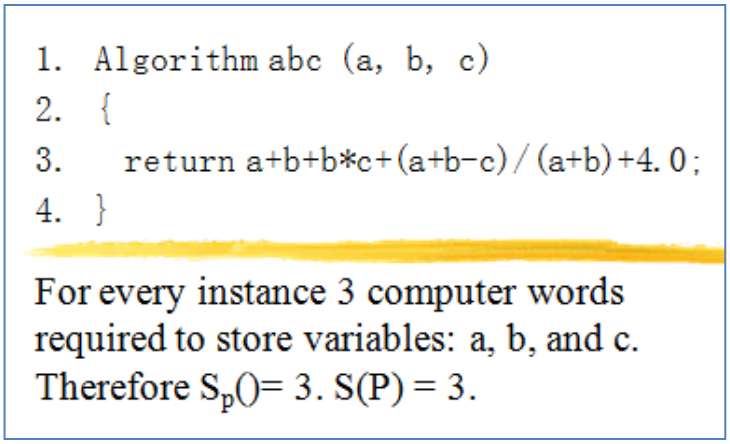

Figure 4.2: Example (1) of Space Complexity [7]

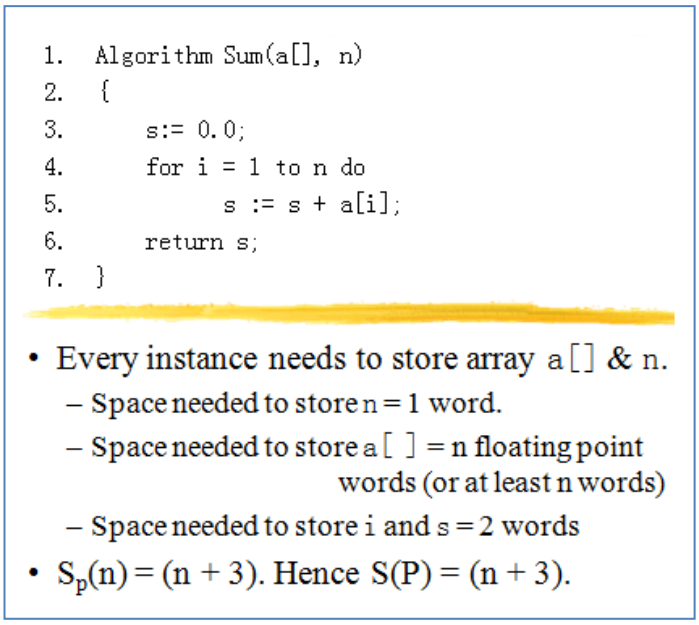

Figure 4.3: Example (2) of Space Complexity [7]

Memory space can be estimated theoretically in a similar way to computing time. Sometimes, both factors can effect each other where using more space results in reduced computing time and conversely.

4.4 Cases of Complexity

- The worst-case efficiency: is the efficiency for the worst-case input of size 'n' for which the algorithm runs the longest among all possible inputs of that size. It can be determined by analysing the algorithm to see what kind of inputs yield the largest value of the basic operation's count C(n) among all possible inputs of size 'n' and then computing this worst-case value Cworst(n) [67].
- The best-case efficiency: is the efficiency for the best-case input of size 'n' for which the algorithm runs the fastest among all possible inputs of that size. This case can be determined by determining the kind of inputs for which the count C(n) will be the smallest among all possible inputs of size 'n'. (Note that the best case does not mean the smallest input; it means the input of size 'n' for which the algorithm runs the fastest) [67]. The analysis of the best-case efficiency is not nearly as important as that of the worst-case efficiency [67].

- The average-case efficiency can provide insight on how the algorithm behaves with a typical or random input, which cannot be yielded from either the worst-case or the best-case analysis. Deciding on this case requires making assumptions about possible inputs of size 'n' [67]. It is not equal to the average of the worst-case and the best-case efficiencies although they occasionally match [67]. This case can draw attention to an important algorithm with an average case efficiency better than its overly pessimistic worst-case efficiency [67].

4.5 Asymptotic Notation Functions

For comparing, 'rate of growth' for time and space, functions are used to map the input size to run time or space cost. Asymptotic notation can describe functions with similar asymptotic behaviour ignoring small input sizes, constants, etc. [34].

4.5.1 Big-O Notation (Upper Bound of the Growth Rate)

Big O of a function gives a 'rate of growth' of the step count function f(n), in terms of a simple function g(n), which is easy to compare [6].

Given two functions f(n) and g(n), f(n) = O(g(n)) if there exists positive constants c and n_0 such that |f(n)| <= c|g(n)| for all n, n>=n_0. f(n)=O(g(n)) if f(n) grows no faster than g(n) [4, 5].

Example (1): for an algorithm, time complexity is calculated: $T(n) = 5n^2 + 17 \log n$

The constant 5 can be ignored. The 'low-order' term (in this example it is 17 log n) should also be dropped [47].

To mathematically explain the rules about constants and low-order terms, the Big O notation was developed. The notation characterizes functions according to their growth rates. Different functions with the same growth rate may be represented using the same O notation [71].

If a function f(n) can be written as a finite sum of other functions, then the fastest growing one determines the order of f(n). Furthermore, if a function is a polynomial in **'n'**, then as **'n'** tends to infinity, the lower-order terms of the polynomial can be discarded. In other words:

- If f(x) is a sum of several terms, the one with the largest growth rate is kept, and all others omitted.
- If f(x) is a product of several factors, any constants are omitted.

Growth Rate Functions

In respect to time efficiency, below are a few growth rate functions [8]:

- O(I) - constant time: This means that the algorithm requires the same fixed number of steps irrespective of the size of the task. Example: 'n' Stack Push and Pop operations.
- O(n) - linear time: This means that the algorithm requires a number of steps proportional to the size of the task. Example: search in an unsorted 'n' list.
- $O(n^2)$ - quadratic time: The number of operations is proportional to the size of the task squared. Example: selection sort of 'n' elements.
- O(log n) - logarithmic time: Example: binary search in a sorted list of 'n' elements.
- O(n log n) – 'n log n ' time: Example: quick sort
- $O(a^n)$ (where a > 1) - exponential time: Example: recursive Fibonacci.

Polynomial growth (linear, quadratic, cubic, etc.) is considered manageable as compared to exponential growth [8] and the smaller, the better [90]. If an algorithm has its 'order of growth' function made of a sum of several terms, then the order of growth is determined by the fastest growing term [8]. Taking $O(n^c)$ and $O(c^n)$, If c > 1 then $O(c^n)$ grows much faster. A superpolynomial function grows faster than n^c for any c whereas subexponential function grows more slowly than any exponential function of the form c^n.

Instance Characteristic n								
Time	Name	1	2	4	8	16	32	Growth
1	Constant	1	1	1	1	1	1	Slowest
log n	Logarithmic	0	1	2	3	4	5	(Best)
n	Linear	1	2	4	8	16	32	
n log n	Long Linear	0	2	8	24	64	160	
n^2	Quadratic	1	4	16	64	256	1024	
n^3	Cubic	1	8	64	512	4096	32768	
2^n	Exponential	2	4	16	256	65536	4294967296	Fastest
n!	Factorial	1	2	24	40326	20922789888000	$26313x10^{33}$	(Worst)

Table 4.3: Time Growth Classes

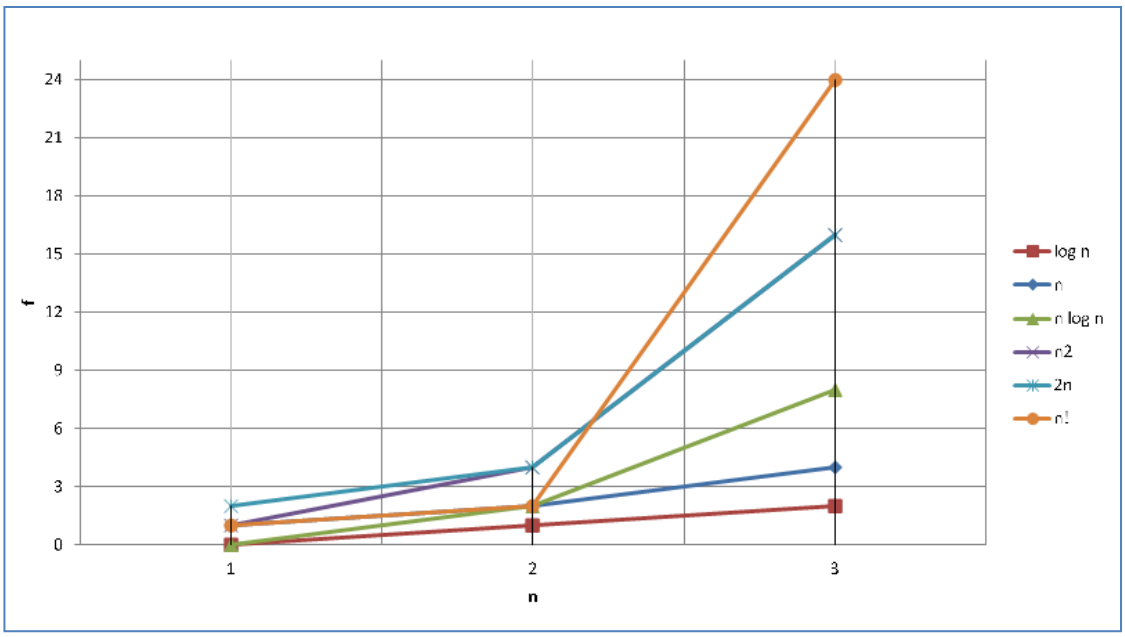

Figure 4.4: Time Growth Classes Plot (based on table4.3)

	Time for $f(n)$ instructions on a 10^9 instr/sec computer						
n	$f(n)=n$	$f(n)=\log_2 n$	$f(n)=n^2$	$f(n)=n^3$	$f(n)=n^4$	$f(n)=n^{10}$	$f(n)=2^n$
10	.01µs	.03µs	.1µs	1µs	10µs	10sec	1µs
20	.02µs	.09µs	.4µs	8µs	160µs	2.84hr	1ms
30	.03µs	.15µs	.9µs	27µs	810µs	6.83d	1sec
40	.04µs	.21µs	1.6µs	64µs	2.56ms	121.36d	18.3min
50	.05µs	.28µs	2.5µs	125µs	6.25ms	3.1yr	13d
100	.10µs	.66µs	10µs	1ms	100ms	3171yr	$4*10^{13}$yr
1,000	1.00µs	9.96µs	1ms	1sec	16.67min	$3.17*10^{13}$yr	$32*10^{283}$yr
10,000	10.00µs	130.03µs	100ms	16.67min	115.7d	$3.17*10^{23}$yr	
100,000	100.00µs	1.66ms	10sec	11.57d	3171yr	$3.17*10^{33}$yr	
1,000,000	1.00ms	19.92ms	16.67min	31.71yr	$3.17*10^{7}$yr	$3.17*10^{43}$yr	

Table 4.4: Execution Times of Different Time Complexity [5]

The Big-O has its limitations. It is most useful on large problems with very large input size. Furthermore, in some algorithms, the omitted constant can have a serious effect on the growth rate. For example, algorithm A's growth rate is $T_a(n) = 1000n = O(n)$ and

algorithm B's growth rate is $T_b(n)= n^2 = O(n^2)$. According to Big-O, algorithm A is faster than B but when n<1000, the omitted constant (1000) slows down A considerably [51].

4.5.2 Omega Notation (Lower Bound of the Growth Rate)

Big Omega notation is used to describe the best case (lower bound) running time for a given algorithm.

Given two functions f(n) and g(n), $f(n) = \Omega(g(n))$ if there exists positive constants c and n_0 such that $|f(n)| >= c|g(n)|$ for all n, $n>=n_0$. $f(n)= \Omega (g(n))$ if f(n) grows no slower than g(n) [4, 5].

4.5.3 Theta Notation (Between Lower and Upper Bound)

Theta notation defines the upper and lower bounds of a function in an exact asymptotic behaviour. It is typically used for comparing running times or growth rates between two growth functions.

Given two functions f(n) and g(n), $f(n)= \Theta(g(n))$ if there exists positive constants c_1, c_2 and n_0 such that $c_1|g(n)| <= |f(n)| <= c_2|g(n)|$ for all n, $n>=n_0$. $f(n)=\Theta(g(n))$ if f(n) and g(n) grow at the same rate [4, 5].

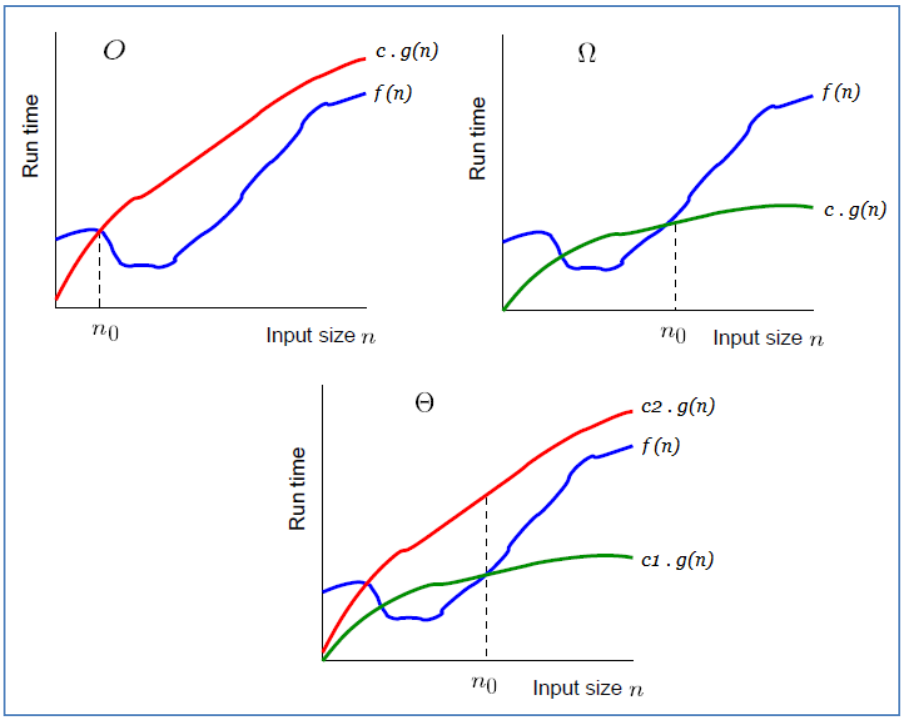

Figure 4.5: Asymptotic Notation Functions [56]

4.6 The Research Adopted Methodology

4.6.1 Research Background

After deciding on the research objectives, a thorough research and literature review was conducted on the related areas such as Folksonomies (Tagging Systems), theory of computation, Princeton WordNet (PWN), MultiWordNet (MWN), Stemming Algorithms, and finally the TE System.

4.6.2 The Algorithm Efficiency Analysis

The researcher opted for an analytical approach rather than an empirical one for reasons previously mentioned such as:

- Execution time and memory space can be determined mathematically in an independent manner regardless of some factors related to the developments environment such as computer specifications, programming language or skills.
- The algorithm can be tested with any input size.

- The algorithm implementation is not necessary.

For time and space complexity, the rate of growth is measured using asymptotic notation functions that can map the input size to run time or space cost. This method is used when comparing two algorithms with different asymptotic complexity functions to determine the more efficient of the two. To determine the feasibility of the algorithm, the Big-O notation is used for characterising functions according to their upper bound growth rates.

4.6.3 The Stemming Component

Data

As previously mentioned in chapter 3, the TE system imported its tags sample from YouTube to avoid building a tagging system from scratch. The tags sample is stored in the database of the TE system ready for the semantic enhancements.

At the start of developing the stemming component, running the original TE algorithm ended in unexpected results. The database was populated with details of each video such as title, link, owner, etc but it was missing the most important piece of information and that is the video's tags. After some research into YouTube developers' website, the problem became clear and it is explained in this YouTube announcement: http://apiblog.youtube.com/2012/08/video-tags-just-for-uploaders.html. Basically, prior to 28th of August 2012, YouTube APIs methods used to allow developers to retrieve video's tags via the <media:keywords/> element which contain the video's keywords (tags). After this announcement, any API method retrieving a video entry will have an empty <media:keywords/> element unless the developer is authenticated as the owner of the video. Faced with this drawback, YouTube had to be replaced with another source that is able to act as the supplier of the tags sample. In general, Flickr was an obvious replacement but it had to be checked whether it enables tags to be accessed from the outside. Researching Flickr APIs for similar functionalities as the ones used earlier with YouTube was productive

and when these APIs were put to the test, they generated the expected result and the TE system had its tags sample ready again. Since the contents in Flickr are exclusively photos and not videos like in YouTube, the text from here forward will use the word photo(s) in any context the previously involved videos although the programme code and the database kept the mention of video to avoid any complications or unnoticeable errors that can occur during the renaming process thus in the code, the word video means actually photo.

The Selection of the Stemming Algorithm

To overcome the problem of different lexical forms in Folksonomies (tagging systems), a stemming algorithm can be applied thus reducing the lexical forms to only one form called 'root' or 'stem'.

A stemming component is added to the TE system for the same purpose indicated above. It will operate in a totally hidden manner (behind the scene) from the user. Inside it, the user tag is normalised (i.e. stemmed) meaning it is reversed back to the original lexical form (stem) then the TE system will store the stem as a user tag in the database of the TE system, so if a photo returns the following tags: 'abstract', 'abstracted', 'abstractedly', 'abstraction', 'abstracts' then the stemming component will normalise them and produce one word 'abstract' and that is a reduction in the number of user tags that will be saved in the database of the TE system and used to query the semantic and social sources. This step will save on database space and effects the algorithm time as discussed later on chapter 6.

Many stemming algorithms exist and can be used in the tagging system (e.g. Krovetz algorithm, Dawson algorithm, Porter algorithm, etc).

Based on the Literature review conducted on stemming algorithms, the research opted for using the English (porter2) stemming algorithm for reasons such as:

- The availability of the source code: Porter had developed a language called Snowball that enables algorithm developers to express their stemming rules in

a natural way regardless of the language. The Porter2 stemmer is implemented using Snowball in many languages such as English, Italian, French etc.

- The common language of implementation: a basic demo is available in C or Java and this goes in harmony with our development environment which involves the usage of Java, Java API for WordNet Searching (JAWS), and Flickr Java API (flickrj).

The timing of the stemming process is very important and can have different consequences depending on when it is performed. Stemming can be carried out on any term (tag) before one of two main processes [106]:

- Before term indexing: the advantages here are that term indexing will be efficient, the index file will be compressed, and the whole operation will be seamless. Furthermore, when it is time to search for a term, it will not cost the system resources since the stemming is already done. This approach of performing stemming before the indexing has some drawbacks as follows:
    - The original tag will be lost forever because it is not being saved in the system's database.
    - Some tags will not return any system tags but this will happen for the wrong reason. As discussed previously in chapter 2, the Porter stemming algorithm is a fast but light stemmer which can occasionally generate unreal words as stems. The consequence of this inaccuracy is that querying the semantic ontologies PWN, MWN, and the social source will yield nothing in most cases. To fix this, the TE system will keep hold of the original tag even after the normalisation process. If the sources returned system tags then it will be discarded otherwise the stem is going to be the one discarded and the sources are searched again using the original tag.
- Before term search: the obvious disadvantage is that it is going to be costly in respect to time and resources and the user may experience some wait time.

5 Database Design Optimisation and Algorithm Complexity Analysis

5.1 Database Design Optimisation

5.1.1 Introduction

The storage requirements for table data are dependent on a few factors. Storage engines have different ways for representing data types and storing raw data. Compressing table data either for a column or an entire row can complicate the calculation of storage requirements for a table or column. Moreover, storage engines have various methods for data allocation and storage, according to the method they use for handling the corresponding types [122]

Despite differences in storage layout on disk, the internal MySQL APIs that communicate and exchange information about table rows use a consistent data structure that applies across all storage engines [107].

Other factors such as Character Set and Collation, Data Types, and Indexes selection play a significant part in the efficiency of the database.

5.1.2 The Storage Engine

In MySQL, storage engines are the components that handle the SQL operations for different table types. MySQL offers various storage engines for different use cases. There is no restriction on using more than one storage engine throughout the server or schema [24].

For general use cases, `InnoDB` is the most suited storage engine recommended by Oracle. It has been designed to provide maximum performance when processing large data volumes [107].

The InnoDB storage engine maintains its own buffer pool for caching data and indexes in main memory. By default, with the innodb_file_per_table setting enabled, each new InnoDB table and its associated indexes are stored in a separate file. InnoDB tables can handle large quantities of data, even on operating systems where file size is limited to 2GB [24].

As the default engine of MySQL v5.5.5, the InnoDB engine has many features as listed below [24, 30, 107, 122]:

- Transaction-safe (ACID compliant) because of data protection capabilities such as commit, rollback, and crash-recovery.
- Increased multi-user concurrency and performance attributed to row-level locking and Oracle-style consistent non-locking reads.
- Tables arrange data on disk to optimize queries based on primary keys.
- Data compression: reduce storage and I/O through the significant table compression.
- Minimized expensive disk I/O by using the memory and the processor resources efficiently.
- More efficient storage for large column values: fully off-page storage of long BLOB, TEXT, and VARCHAR columns.
- Support for FOREIGN KEY referential-integrity constraints which maintain data integrity.
- Fast index creation/deletion without copying the data.
- Barracuda file format maintains upward and downward compatibility.
- Performance and scalability enhancements: includes features such as multiple background I/O threads, multiple buffer pools, and group commit.
- Automatic data recover

One of the important features above is the foreign key referential integrity which is about ensuring that the foreign key in a referencing table must always refer to a valid row in the referenced table. It keeps the relationship between the two tables

synchronized during updates and deletes. The next table lists some specific features of the `InnoDB` engine.

Storage limits	64TB	***Transactions***	Yes	***Locking granularity***	Row
MVCC	Yes	***Geospatial data type support***	Yes	***Geospatial indexing support***	No
B-tree indexes	Yes	***Hash indexes***	No [a]	***Full-text search indexes***	Yes [b]
Clustered indexes	Yes	***Data caches***	Yes	***Index caches***	Yes
Compressed data	Yes [c]	***Encrypted data [d]***	Yes	***Cluster database support***	No
Replication support [e]	Yes	***Foreign key support***	Yes	***Backup / point-in-time recovery [f]***	Yes
Query cache support	Yes	***Update statistics for data dictionary***	Yes		

[a] InnoDB utilizes hash indexes internally for its Adaptive Hash Index feature.
[b] InnoDB support for FULLTEXT indexes is available in MySQL 5.6.4 and higher.
[c] Compressed InnoDB tables require the InnoDB Barracuda file format.
[d] Implemented in the server (via encryption functions), rather than in the storage engine.
[e] Implemented in the server, rather than in the storage engine.
[f] Implemented in the server, rather than in the storage engine.

Table 5.1: InnoDB Storage Engine Features [80]

Other engines includes: `MyISAM`, `Memory`, `CSV`, `Archive`, `Blackhole`, `Merge`, `Federated`, and `Example`. The following table compares the main features of some of these engines [80]:

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Storage limits	256TB	RAM	64TB	None	384EB
Transactions	No	No	Yes	No	Yes
Locking granularity	Table	Table	Row	Table	Row
MVCC	No	No	Yes	No	No
Geospatial data type support	Yes	No	Yes	Yes	Yes
Geospatial indexing support	Yes	No	No	No	No
B-tree indexes	Yes	Yes	Yes	No	Yes
Hash indexes	No	Yes	No [a]	No	Yes
Full-text search indexes	Yes	No	Yes [b]	No	No
Clustered indexes	No	No	Yes	No	No
Data caches	No	N/A	Yes	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Compressed data	Yes [c]	No	Yes [d]	Yes	No
Encrypted data [e]	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Replication support [f]	Yes	Yes	Yes	Yes	Yes
Foreign key support	No	No	Yes	No	No
Backup / point-in-time recovery [g]	Yes	Yes	Yes	Yes	Yes
Query cache support	Yes	Yes	Yes	Yes	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

[a] InnoDB utilizes hash indexes internally for its Adaptive Hash Index feature.

[b] InnoDB support for FULLTEXT indexes is available in MySQL 5.6.4 and higher.

[c] Compressed MyISAM tables are supported only when using the compressed row format. Tables using the compressed row format with MyISAM are read only.

[d] Compressed InnoDB tables require the InnoDB Barracuda file format.

[e] Implemented in the server (via encryption functions), rather than in the storage engine.

[f] Implemented in the server, rather than in the storage engine.

[g] Implemented in the server, rather than in the storage engine.

Table 5.2: Storage Engines Features Summary [80]

Individual storage engines might impose additional restrictions that limit table column count. Examples [122]:

- `InnoDB` permits up to 1000 columns.
- `InnoDB` restricts row size to something less than half a database page (approximately 8000 bytes), not including `VARBINARY`, `VARCHAR`, `BLOB`, or `TEXT` columns.

- Different InnoDB storage formats (COMPRESSED, REDUNDANT) use different amounts of page header and trailer data, which affects the amount of storage available for rows.

5.1.3 The Character Set

A character set is a set of symbols and encodings. A collation is a set of rules for comparing characters in a character set. Each character set can have one or more collations. Encoding is the coded value that is paired with each character inside a character set [24].

MySQL can store data using a variety of character sets and perform comparisons according to a variety of collations for the MyISAM, MEMORY, and InnoDB storage engines. The character sets can be specified at any level (server, database, table, and column level). Furthermore, a mix of different character sets or collations can exist in the same server, database or table [24].

MySQL supports 70+ collations for 30+ character sets within groups such as: Unicode, West European, Central European, Asian, etc [24].

Regardless of the platform, program, or language, a Unicode character set assigns each character a unique number [112]. It can be implemented by different character encodings. UTF-8 encoding is one of the most commonly used. Using this encoding means that any ASCII characters will need one byte whereas other characters will require up to four bytes [59].

The idea of UTF-8 is that various Unicode characters are encoded using byte sequences of different lengths [59, 107, 122]:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a 2-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.

- Korean, Chinese, and Japanese ideographs use 3-byte or 4-byte sequences.

5.1.4 The Schema

5.1.4.1 The Data Types

The TE system uses the database described previously. The schema is simple with three tables: videos, tags_detail, and tags_master. There are few foreign keys for referential constraint between tables. The tables' fields are of the data type VARCHAR.

Data Type	Storage Required L represents the actual length in bytes of a given string value.
CHAR(M)	M × w bytes, 0 <= M <= 255, where w is the number of bytes required for the maximum-length character in the character set
BINARY(M)	M bytes, 0 <= M <= 255
VARCHAR(M), VARBINARY(M)	L + 1 bytes if column values require 0 – 255 bytes, L + 2 bytes if values may require more than 255 bytes
TINYBLOB, TINYTEXT	L + 1 bytes, where $L < 2^8$
BLOB, TEXT	L + 2 bytes, where $L < 2^{16}$
MEDIUMBLOB, MEDIUMTEXT	L + 3 bytes, where $L < 2^{24}$
LONGBLOB, LONGTEXT	L + 4 bytes, where $L < 2^{32}$
ENUM('value1','value2',...)	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET('value1','value2',...)	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

Table 5.3: Storage Requirements for String Types [80]

VARCHAR, VARBINARY, BLOB and TEXT types are data types of variable length. These data types determine their storage requirements based on the following factors:

- The actual length of the column value.
- The maximum length of the column.
- The character set of the column since some of these sets contain multi-byte characters.

A row has a maximum size of 65,535 bytes. This restriction affects the maximum amount of bytes that can be saved in a VARCHAR or VARBINARY column (regardless of storage engine), which is shared among all columns [122]. For example, in UTF-8 encoding, characters need a maximum of three bytes per character, so for a CHAR (255) CHARACTER SET utf8 column, the server must allocate 255 × 3 = 765 bytes per value. Consequently, a table cannot contain more than 65,535 / 765 = 85 such columns.

Storage for variable-length columns includes length bytes, which are assessed against the row size. For example, a VARCHAR(255) CHARACTER SET utf8 column takes two bytes for storing the length of the value, so each value can take up to 767 bytes.

5.1.4.2 Indexes

The job of an index is to find rows with specific column values in a speedy manner. If no index exists, MySQL has to search for the target values starting from the first row reading through the entire table to find them. The cost of the search will grow as the table gets larger. Most MySQL indexes (PRIMARY KEY, UNIQUE, INDEX, and FULLTEXT) are stored in B-trees [24].

Regarding queries on small or large tables, if report queries are processing most or all of the rows, then Indexes become less important.

When a query needs to access most of the rows, reading sequentially is faster than working through an index.

Primary Keys

The primary key represents column(s) that are essential in vital queries. To speed up the execution of queries, MySQL associates an index with the primary key. A primary key cannot be NULL. In InnoDB storage engine, the physical organization of data inside tables allows ultra-fast lookups and sorts based on the primary key column(s) [107].

Foreign Keys

When a table and query has many columns, it is beneficial to move its less used data to tables with fewer columns. These can then be cross-referenced back to the main table using its primary key. Tables with fewer columns are able to fit more rows into each data block.

For fast lookups, each split table can assign a primary key and any column combinations can be performed using join queries.

Indexes

Mostly, an index is a single column and it copies that column's values in a B-tree data structure for fast lookups. Depending on the storage engine, the maximum number of table indexes and their length can vary. In general, all engines support at least 16 table indexes with length of at least 256 bytes [24].

MySQL offers other keywords to define various indexes such as KEY, which is a synonym for `INDEX`. Also there is `UNIQUE` which forces all values in the index to be distinct.

5.1.5 Optimisation Procedures

Many Database management systems have their own recommendation for optimal database design and below are some of these general recommendations [24, 107, 122]:

- Use the smallest data types possible.
- If a column can be either strings or numbers, always choose numbers because large numeric values occupy less bytes than strings and the tasks of transferring or comparing them will also take less memory.
- Use VARCHAR in place of CHAR when storing variable-length strings or when having many `NULL` values inside columns. Smaller tables have less I/O and can fit more effectively in the buffer pool.

- When a column is not allowed to have NULL values, state it as NOT NULL at the table creation stage. This helps in: selecting the most effective index for a query, and reducing the overhead cost of checking if every value is NULL.
- Avoid using long PRIMARY KEY (either a single or composite) because it wastes a lot of disk space since it's duplicated in each secondary index.
- Use OPTIMIZE TABLE statement to compact any wasted space when a table grows significantly or data reaches a stable size.
- Use COMPRESSED row format for large or repetitive data tables. Tasks like putting data in buffer pool or scanning full table will require less disk I/O.
- Creating indexes should be very strict for those who will improve query performance because indexes slow down the insert and update operations.
- When searching a table using different columns, it is better to replace them with a single composite index where the 1st part of it is the most used column. However, if this is the norm, then the 1st part of the index must be the column that has the most duplicates to gain better compression of the index.
- Any query will use one index only, that is why there is no need to have a second index in each column.
- Throughout tables, columns with identical information should be of the same data types to speed up joins based on them.
- Choose simple column names to use across different tables for simplified join queries.
- If a column is included in the WHERE clause, setting up indexes on it can make queries faster and for queries referencing different tables and using joins and foreign keys this is very important.
- Reduce the number of full table scans especially if tables are big.
- Keep the optimizer up to date with table statistics using the ANALYZE TABLE statement so that it has the information required for developing an efficient execution plan.

5.1.6 Discussion

The Engine

The TE system requires a simple straight forward database use case, thus the schema consists of three tables connected through two one-to-many relations that use foreign keys. The InnoDB storage engine was used throughout the schema as it had no special requirements, thus it was the obvious choice as a storage engine for the reasons mentioned earlier in this chapter such as performance and efficiency with large data volumes. Its support for foreign keys and fast index creation and deletion were the deciding factors, mainly because other storage engines do not natively support them.

The Character Set and Collation

The TE system involves the use of the English and Italian languages. Both languages fall into the West European Character Sets. As said previously [119], UTF-8 encoding has a variable length. It uses between one to four bytes for the character encoding whereas UTF-16 encoding always uses two or more bytes. When characters with low encoding space (one byte) dominate, the use of UTF-8 becomes more economical than UTF-16 which is more suitable if the application is using many foreign interchange processes. UTF-8 encoding is the most portable in many applications. Regarding data corruption errors, which can occur during transfers between systems, UTF-8 encoding is resilient to them and it is better than UTF-16 and UTF-32 in that regard.

The use of UTF-7 encoding within retrieval systems is not recommended and although UTF-7 encoding is very useful as an interchange format, working with it can be a slow process and that is why it should not be stored as it is. Instead, it has to be converted to UTF-8 on arrival. In conclusion, we found that for all the reasons listed above, UTF-8 encoding has become the preferred encoding and the dominant standard and hence the character set utf8 (UTF-8 encoding) is chosen for all tables. Any characters outside the UTF-8 encoding will be encoded and escaped [14].

The collation was not apparent in the sql script of the TE database but `utf8_general_ci` is faster than `utf8_unicode_ci` and less accurate.

Caution is always advised when mixing different character sets and collations at column level to avoid problems when performing joins or other cross-column operations.

Data Types

In TE, the database is initially populated with a sample of records holding information on YouTube videos. In this thesis, the source of the sample data had to be changed from YouTube to Flickr as explained in section 4.6.3

The database includes one data type only and that is VARCHAR. Since most of the values saved are strings, it is a logical choice especially when the values are of variable lengths.

The `video_id` column represents the video number in YouTube which in the YT documentation is defined as `String` but does not disclose its size limit. All the saved values of `video_id` inside the TE database (TE database has +7000 records in the `videos` table) are 10 characters in length. If `video_id` is defined as `UNSIGNED INT`, it can represent a maximum of 4,294,967,295 (4 bytes) or `UNSIGNED BIGINT` and reach up to 18,446,744,073,709,551,615 (8 bytes). Doing this will entail the parsing of `video_id` from `String` to `int` within the Java code or a type mismatch error will occur.

The `link` and `thumb` columns save URLs and it is recommended that they be TEXT for long URLs otherwise VARCHAR is the most suitable data type. The URLs in the TE database are relatively short; hence both the link and thumb columns are VARCHAR.

Table	Column	VARCHAR Length	Size (Bytes)	Max Length	Min Length	Avg Length	VARCHAR Suggested Length	Size (Bytes)	Size Saving%
videos	video_id	50	151	10	6	9.8	15	46	69.54%
	title	400	1202	255	0	23.3	300	902	24.96%
	link	200	602	87	40	51.5	100	302	49.83%
	thumb	200	602	63	56	62.5	100	302	49.83%
tags_master	video_id	50	151	10	6	9.8	15	46	69.54%
	user_tag	250	752	237	1	9.3	250	752	0.00%
tags_detail	user_tag	250	752	237	1	9.3	250	752	0.00%
	system_tag	250	752	237	1	9	250	752	0.00%
	tag_type	50	151	23	13	16.6	23	70	53.64%

Table 5.4: Database Columns Sizes

According to the table above, the TE sql script for generating the database tables is generous when assigning lengths to the VARCHAR columns. After investigating the maximum and minimum values, it became clear that a size reduction is possible to a great degree and very beneficial given that the character set used is `utf8` which requires up to 3bytes/character (the maximum in MySQL) plus one or two length bytes. The actual and suggested new sizes are shown in the table above, along with the saving percentage. Furthermore, if the column `video_id` is changed to `UNSIGNED INT` or `UNSIGNED BIGINT`, the saving here will be 97.4%, 94.7% respectively with the added benefit of faster operations such as comparison and transferring.

Additionally, the `link` column can be made shorter by 29 characters if the initial fixed part (http://www.flickr.com/photos/) gets truncated before being inserted in the table. In general, it is a good practice to divide the URL into portions such as hostname and protocol and save them in a separate table.

The `tag_type` value is chosen from a set of flags to distinguish the source of the system tags. Its length is between 13 and 23 characters and therefore there is no need to go as far as VARCHAR(50).

The Indexes

The primary keys for the three tables are justified, although it could be argued that using the single auto-generated primary key is less complicated than composite primary keys in tables: tags_master and tags_detail. The answer is, unless the single key is capable of enforcing uniqueness without adding any special constraints on the other columns, the composite primary key must be used [54].

The simple use case of the TE system highlights clearly the columns that are most suited to act as foreign keys between the tables, thereby enforcing referential integrity and normalisation.

In addition to the primary and foreign keys assigned, there is a `UNIQUE` index on the link column. Indexes on URLs are not recommended because the value tends to be long and cannot be indexed in full. Instead, an extra column with a hash value should be created and indexed. In the schema, the `link` column is (200) which is within the limits of the index length inside `InnoDB` engine (255 characters) and also the URLs are considered short. Thus indexing the link column will force values to be distinct although it does not improve any query performance.

5.2 Time Complexity

The main target of the performance analysis for the TE system is to estimate the cost of enhancing the tags before performing the search on them. It will also check out whether the process is efficient, especially when the experiment showed no significant difference between the results retrieved with or without the tag enhancements due to several factors [72]. The analysis will measure the time as the number of tags grow. Thus, the next paragraph will shed some light on the tags size inside some tagging systems.

In September 2010, Flickr reached 5 billion images with an upload rate of 3,000+ images/min. A year later this number increased to 6 billion whereas in YouTube, videos are uploaded with a rate of 72 hours of video per minute with more than 200 million

content ID videos alone. Flickr allows 75 tags per photo as a maximum limit but on average each photo will be annotated using 8.94 tags [70]. On the other hand, every YouTube video has a keywords list of 500 bytes in length (including commas) thus tags are estimated to reach 167 tags maximum.

In the population process of the TE system, a set of 169 English and Italian keywords are used to retrieve seven videos maximum per each keyword. The search resulted in filling the database with 7810 videos. The table below gives essential statistics about tags inside the database of the TE system.

	Min	Max	Average
User Tags/Video	1	75	10
System Tags/ User Tag	1	89	3

Table 5.5: The TE Database Tags Statistics

5.2.1 Time Efficiency Analysis of Nonrecursive Algorithms

The Steps of Time Efficiency Analysis of Nonrecursive Algorithms are as follows [67]:

1. Decide on the parameter(s) of which their input size is the focus.
2. Identify the basic operation.
3. If the basic operation's execution count depends on the input size plus some additional property, investigate the worst-case, average-case, and best-case (if needed) efficiencies separately.
4. Assign a sum to represent the basic operation's execution count.
5. Apply summation formulas and manipulation rules on the count to conclude a closed-form formula or its order of growth.

Input Size

Typically, algorithms with large inputs (e.g. more numbers, lengthy strings, bigger graphs) have longer runtime. Thus, the algorithm's efficiency is formulated by using a function of 'n' (the input size).

The parameter selected is the one of which the size's growth rate is the most important to the analysis objectives.

- Searching/sorting. n=number of items in list.
- String processing. n = length of string(s).
- Matrix operations, n = dimension of matrix. $n \times n$ matrix has n^2 elements.
- Graph processing. n_V = number of vertices and n_E = number of edges.

Figure 5.1: Typical n in Common Algorithms [23]

In general, spotting the parameter is a straight forward matter except in certain cases such as when the dependency of the candidate parameter is compromised because of another parameter.

The Basic Operation

It is the most important operation of the algorithm since it is contributing the most to the total running time [67]. Usually it is the most time-consuming operation inside the innermost loop. Thus, the algorithm's time efficiency can be measured by counting the number of times the algorithm's basic operation is executed on inputs of size 'n' [67].

The Count and Rules

The algorithm analysis is independent of the hardware used to implement or run the code. It uses a model machine which specifies a set of rules to determine how and what operations are to be counted during the analysis, since there are no standard rules.

The counting process can follow one of the methods below:

- Count every program step and calculate their frequency. This is usually done using a tabular form.
- Counting only the actual number of basic operations.
- Counting iterations only.

Although the last two methods are the most common, in some cases they can be insufficient and having an exact count of operations is more beneficial. The following table outlines an estimated time cost of some operations. It is not accurate but it gives an idea on the speed of some of the common operations in relation to each other.

Operation	Time Unit
Assignment	1
Arithmetic / Logical	1
Constructor/ Destructor	1
Procedure Entry / Exit	1
Select Condition	Worst Branch Timing
Loop [75]	(over the number of times the loop is executed) the body time + time for the loop check and update operations + time for the loop setup
Function Calls [75]	1 for setup + the time for any parameter calculations + the execution time of the function body
Database	
Connecting to Server	3
Sending Query to server	2
Parsing Query	2
Inserting row	1 × size of row
Inserting indexes	1 × number of indexes
Closing Server Connection	1

Table 5.6: Operations Estimated Relative Time Cost

5.2.2 Discussion

The analysis determined the complexity of the functions used in the main body and a few of these functions have a constant growth rate as shown below:

Function	Complexity
getSynonyms (String tag)	O(1)
getRelatedWords (String tag)	O(1)
getSimilarWords (String tag)	O(1)
getParentNoun(String tag)	O(1)
tag_clustering (String tag)	O(1)

Table 5.7: Functions with Constant Growth Rate

All functions above except `tag_clustering` have simple statements without iterations and conditions, etc. Regarding `tag_clustering`, there is an implementation limitation in the case of retrieving tag's clusters. The TE system will add the Top-3 tags only from the 1st cluster, if any, as system tags. The outer loop will execute to a maximum once while the inner loop will iterate three times maximum and then quit. Thus, in the worst case, this function will have a constant time O(1).

The time complexity of the remaining functions relies on the number of the system tags produced from within them. These system tags include synonyms, related, similar, hypernyms and translated tags. The notation used in the analysis is as follow:

Tag Type	User Tags	System Tags	Synonyms	Similar	Related	Translated	Hypernyms
Notation	N	M	Sy	S	R	T	H

Table 5.8: TE Analysis Notations

Initially, the complexity equations of the functions have distinguished between the sources of the system tags inside them. For simplicity purposes and for more generalized equations, constants are eliminated and all tag sources are considered system tags as shown in the table below:

Function	Complexity Equation	
	Initial	Final
english_synonyms_related_similar (String tag)	3+11Sy+12R+12S	M
english_hypernyms (String tag)	1+11H	M
english_2_italian_translation_related_similar (String tag)	15+11T+11R+11S	M
italian_synonyms_related_similar (String tag)	15+11Sy+11R+11S	M
italian_hypernyms (String tag)	5+11H	M
italian_2_english_translation_related_similar (String tag)	4+4T+11T.Sy+12T.R+12T.S	M

Table 5.9: The Growth Rate of the TE Functions

The last equation in the above table was generalised to be M^2, but because the count of the tags: translated (T), synonyms (Sy), and related (R) is a small number compared to the similar tags (S), the complexity equation discarded the values of T, Sy, R to finally settle only on M.

Moving to the main body of the algorithm which includes the semantic and social components, we can see three main loops. The first loop iterates through the keywords to query YouTube and retrieve matching videos. Populating the database of the TE system is done through the second loop that iterates through all the retrieved videos recording the essential information about each one. The third and innermost loop iterates through the user tags of each video within the block headed by:

```
For (int j = 0 ; j < keywordStringList.size() ; j++)
```

The body of this loop is responsible for the main tasks of the TE system which are querying the sources: Princeton WordNet (PWN), MultiWordNet (MWN), and Flickr clustering. The code uses the number of user tags associated with each video as a counter. This counter is chosen as the input size parameter and will be referred to as N. Initially, the complexity equation calculated is O(N + N.M), but looking at the statistics in Table 5.5 specifically the Max column, we can assume that in the worst

case scenario, the value of N and M are very close. Thus the complexity equation is modified to $O(N^2)$ which is a polynomial (quadratic) time where the number of operations is proportional to the size of the task squared. Visiting Table 4.4 again, we relist it focusing only on quadratic time.

Time for f(N) instructions on a 109 instr/sec computer [microsecond(µs) = 10^{-6} sec, Millisecond(ms) = 10^{-3} sec]		
N	N^2	$f(N) = N^2$
10	100	0.1 µs
50	2500	2.5 µs
75 (Flickr)	5,625	5.625 µs
100	10,000	10 µs
167 (YouTube)	27,889	27.889 µs
1000	1,000,000	1 ms
10,000	100,000,000	100 ms
100,000	10,000,000,000	10 sec
1,000,000	1,000,000,000,000	16.67 min

Table 5.10: Execution Times of Different Time Complexity Functions

As mentioned in chapter two, an algorithm with a computation that runs in linear or quadratic time is 'efficient' [10] and polynomial growth is considered manageable. Nonetheless, the code can benefit from more adjustments that can decrease its execution time. The code, including the main and the functions, has many SQL select and insert statements and some of them are located inside conditions and iterations. The code executes each statement one by one which is costly since select and insert statements have slow and slower times (respectively) than a normal statement. This situation can be optimized by applying a batch execution instead, which can be guided

using certain flag variables indicating whether a specific select/insert statement is to be executed or not under a certain condition/iteration.

Another concern is the retrieval maximum limit from the different semantic resources (PWN and MWN) which does not exist, whereas it is forced on the retrieval from the social resource (Flickr clustering). The following table shows the minimum and maximum count for all system tag types. Based on the table information, a user tag can yield 157 system tags in worst case scenario. This number is more accurate than 89 (see Table 5.5) because it accounts for the maximum count in each type, regardless of the user tag. However, the information in Table 5.5 represents the maximum count of system tags for each user tag. Retrieving a considerable number of tags related to one tag type, such as in the case of SIMILAR tags, is unnecessary and can be limited to a reasonable number.

Tag Type		Max	Min
SIMILAR	EN_EN_SIMILAR	79	17
	IT_IT_SIMILAR		
	EN_IT_SIMILAR		
	IT_EN_SIMILAR		
SYNONYMS	EN_EN_SYNONYMS	28	14
	IT_IT_SYNONYMS		
TRANSLATION	EN_IT_TRANSLATION	27	4
	IT_EN_TRANSLATION		
HYPERNYM	EN_EN_HYPERNYM	12	7
	IT_IT_HYPERNYM		
RELATED	EN_EN_RELATED	8	4
	IT_EN_RELATED		
CLUSTERING	CLUSTERING_TAGS	3	3
	TOTAL	157	49

Table 5.11:Totals of System Tags

Finally, the coder opted for the tagging process to take place when entering a new tag instead of doing that 'on the fly' when searching for a certain tag. As stated before, this made the execution time less critical, but it still needs to be manageable, because it is invisible to the user who is not waiting for any response from the program at this stage.

5.3 Space Complexity

5.3.1 Java Virtual Machine (JVM) and Data Types

The Java virtual machine has their data types divided into:

- Primitive type: variables of the primitive types hold primitive values which are the actual data.
- Reference type: variables of the reference type hold reference values referring to dynamically created objects.

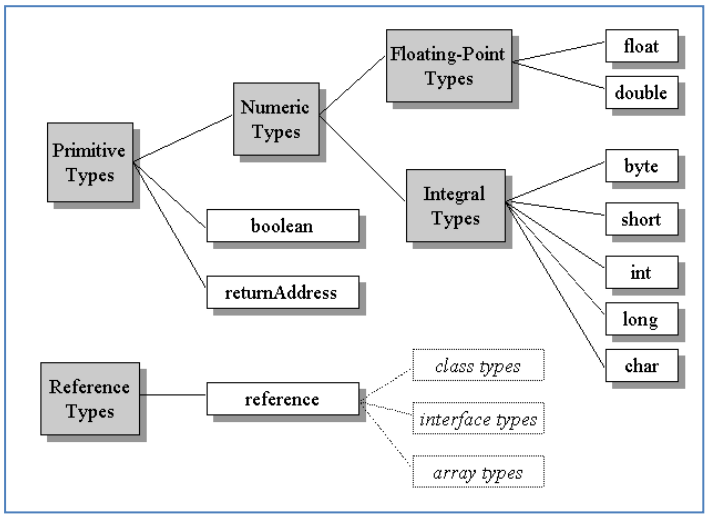

Figure 5.2: Java Virtual Machine's Families of Data Types [114]

The compiler uses `int` or `byte` to represent a Boolean where false is represented by integer zero and true by any non-zero integer whereas arrays of Boolean are accessed as arrays of byte.

The Reference type can be one of the following:

- Class type: values are references to class instances.
- Interface type: values are references to class instances that implement an interface.
- Array type: values are references to arrays.
- Null: variable does not refer to any object.

In Java virtual machine, each data type has a specific range of values (see the table below).

Type	Range
`byte`	8-bit signed two's complement integer (-2^7 to $2^7 - 1$, inclusive)
`short`	16-bit signed two's complement integer (-2^{15} to $2^{15} - 1$, inclusive)
`int`	32-bit signed two's complement integer (-2^{31} to $2^{31} - 1$, inclusive)
`long`	64-bit signed two's complement integer (-2^{63} to $2^{63} - 1$, inclusive)
`char`	16-bit unsigned Unicode character (0 to $2^{16} - 1$, inclusive)
`float`	32-bit IEEE 754 single-precision float
`double`	64-bit IEEE 754 double-precision float
`returnAddress`	address of an opcode within the same method
`reference`	reference to an object on the heap, or `null`

Table 5.12: Ranges of the Java Virtual Machine's Data Types [114]

The JVM specification does not define sizes for their data types. This decision is left to the coder for each individual implementation.

`Word` is the basic unit of size for data values in JVM. It is large enough to store values of `byte`, `short`, `int`, `char`, `float`, `returnAddress`, or `reference`. Two words must be large enough to store values of `long` or `double`.

Based on the above restriction, the coder must choose a word size with at least 32 bits but it can be of any other size as long as it delivers an efficient implementation.

5.3.2 General Formula of Memory Usage

In JVM (specifically HotSpot), the heap is the memory area used by a Java object for dynamic memory allocation. Generally, it consists of [27]:

1. Object Header: includes a few bytes of 'housekeeping' information.
2. Memory for Primitive Types according to their size.
3. Memory for Reference Types: 4 bytes each.
4. Padding: this consists of a few wasted bytes after the object data to make every object start at an address that is a convenient multiple of bytes. This decreases the amount of bits needed to represent a pointer to an object.

5.3.3 'Housekeeping' Information

On the heap, instances of an object take up more memory than their actual fields to save some 'housekeeping' information such as their classes, IDs and status flags (reachable, synchronization-locked etc.) In Hotspot, a normal object needs 8 bytes for housekeeping info whereas an array requires 12 bytes [27].

5.3.4 Memory Usage of Arrays

Single-Dimension Array

This type of array is considered as a single object with the usual header of 8 bytes plus 4 more bytes to accommodate its length. Thus, in total, the array header is 12 bytes [26]. Regarding the actual data inside the array, it is calculated by:

The number of elements X the number of bytes required for one element (based on its type).

For an object reference, one element needs 4 bytes. If the array memory usage summation is not a multiple of 8 bytes, then it is rounded up to the next multiple. A Boolean array requires one byte per element [26].

Memory usage of a two-dimensional array

In Java, a multidimensional array is a set of nested arrays. Every row of a two-dimensional array has the overhead of an object [26].

Multidimensional arrays

As previously mentioned, each row of the outside array creates an array of references to another array holding the actual primitive data or references (if it is an object array) [26].

5.3.5 Memory usage of Strings

A Java `String` is made up of more than a singular object and it contains some extra variables as follows [28]:

- A char array holding the actual characters.
- An integer offset indicating the string start point.
- An integer representing the length of the string.
- An integer for the cached calculation of the hash code.

According to 'Hotspot Java 6 VM', the minimum memory usage of a String is calculated using the formula: 8 * (int) ((((no chars) * 2) + 45) / 8) in the condition that it is 'newly created' string and not created from a substring [28].

Example (1): An empty string

It will need the following: 4 bytes (char array) + 4 bytes*3 (integer fields) + 8 bytes (header) = 24 bytes (multiple of 8).

In addition, the empty 'char' array will need a 12 bytes (header) rounded up to 16 bytes making the total memory allocated for an empty string 40 bytes [28].

Example (2): 17 characters string

Initially we have 4 bytes (char array) + 4 bytes*3 (integer fields) + 8 bytes (header) = 24 bytes (multiple of 8).

Then the char array will need: 12 bytes (header) + 17*2 bytes = 46 bytes rounded up to 48 bytes.

The total memory usage is 24+48 = 72 bytes [28].

5.3.6 Calculating the Space Complexity

1. Identify the parameter(s) that determine the problem size.
2. Calculate the space (memory) needed for a particular size.

3. Calculate the space (memory) needed for double the earlier size.
4. Repeat step 3 many times until you reach a relationship between the size of the problem and its space and that will give the space complexity [15]

5.3.7 Discussion

The parameter that will determine the problem size is the number of user tags per video. For each video, the user tags are stored in the parameter `keywordStringList` of type `List<String>` and thus the main loop in the algorithm which performs the necessary steps to produce the different system tags is using the parameter `keywordStringList.size` as its counter `N`. Most of the parameters are declared as public and they are allocated constant space `SPACE(1)` which is not affected by the growth rate of `N`. The only parameter that is varying in size depending on the growth rate of `N` is `keywordStringList` with `SPACE(N)` which is a linear space.

6 The Stemming Component Embedding

Prior to embedding the stemming algorithm inside the TE code, we had to address the aforesaid YouTube issue in Chapter 4. The database population code segment with its YouTube APIs had to be replaced with a new segment providing the same functionality. The shift to Flickr involved using flickrj which is a java interface to Flickr APIs. The new code segment involved using many Packages, Interfaces, Classes, and methods such as:

Type	Name	Variable
Package	com.aetrion.flickr	-
Class	Flickr	flickr
Method	getPhotosInterface	-
Package	com.aetrion.flickr.tags	-
Class	Tag	tag
Method	getValue	-
Package	com.aetrion.flickr.photos	-
Class	com.aetrion.flickr.photos.**Photo**	photo
Method	getId	-
Method	getTitle	-
Method	getUrl	-
Method	getThumbnailUrl	-
Method	getTags	-
Class	com.aetrion.flickr.photos.**PhotoList**	photoList
Method	size	-
Method	get	-
Class	com.aetrion.flickr.photos.**PhotosInterface**	photosInterface
Method	search	-
Method	getInfo	-

Class	com.aetrion.flickr.photos.**SearchParameters**	searchParams
Method	setSort	-
Method	setText	-
Package	java.util	-
Interface	Collection<E>	tagsList
Method	size	-
Method	iterator	-
Interface	List<E>	keywordStringList
Method	add	-
Class	ArrayList<E>	-
Interface	Iterator<E>	Itea
Method	hasNext	-
Method	next	-

Table 6.1: Some used Components from Java and flickrj

The Porter2 stemming algorithm is encoded in many programming languages by Porter himself or by other trusted programmers [98]. In his web site, Porter lists some of these encodings that he trusts their credibility.

The selected Java encoding for the Porter stemming algorithm was developed by Martin Porter and the last version was released in 2000. The diagram below displays the key steps of the Porter stemming algorithm [3].

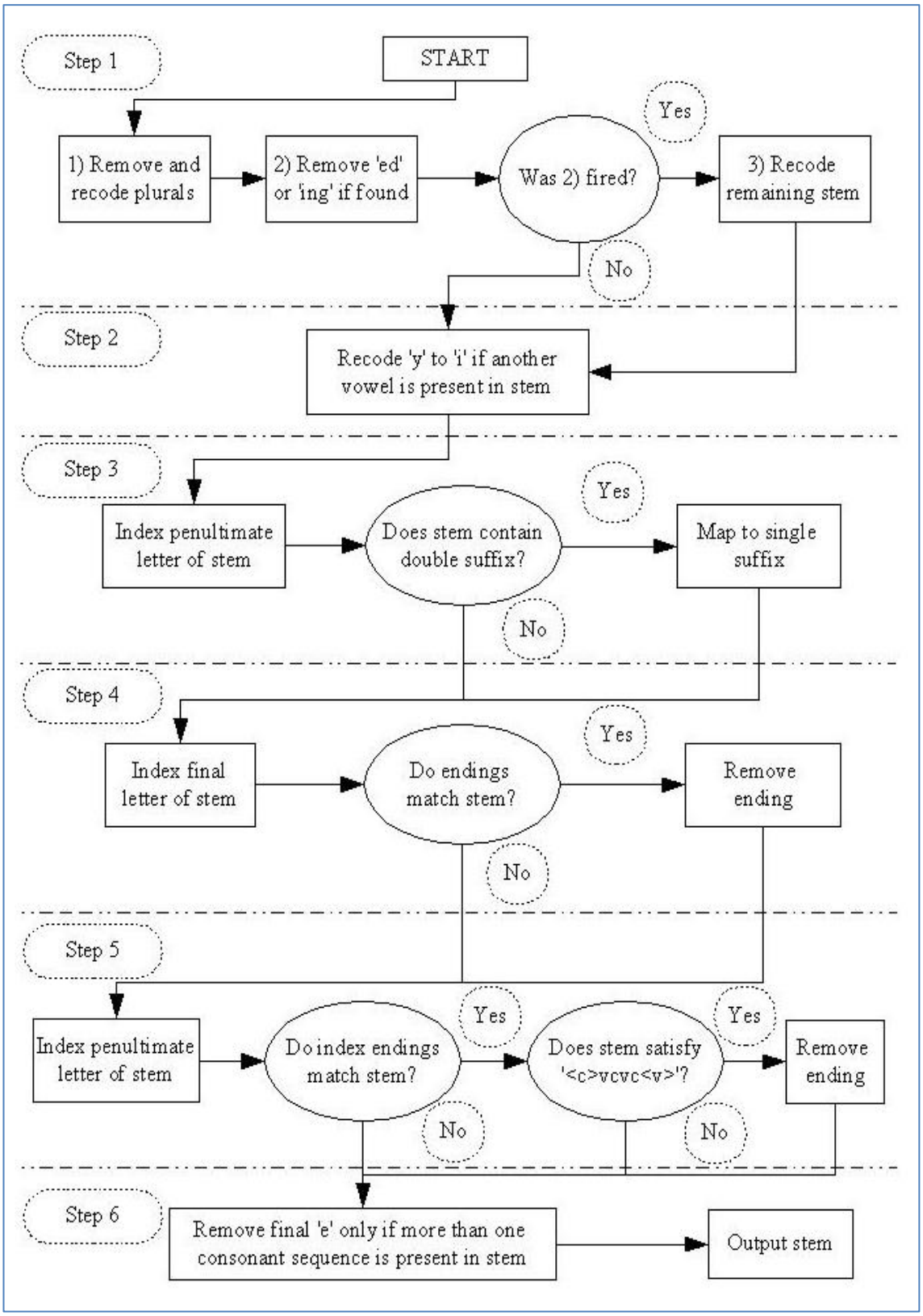

Figure 6.1: The Porter Stemming Algorithm Flowchart [3]

Listed below, are some terms and notations regarding the stemming algorithm [3]:

- consonant: in the English alphabet they are B, C, D, F, G, H, J, K, L, M, N, P, Q, R, S, T, V, X, Z, and usually W and Y.
- Vowel: in the English alphabet they are A, E, I, O, U.
- C: is a consonants list with length greater than or equal to 1.
- V: is a vowels list with length greater than or equal to 1.
- m: is the number of repetitions.
- []: represents the optional presence of the contents inside.

Based on the previously mentioned notations, the formula `[C](VC)m[V]` is a representation of any word. `'m'` is called the measure of a word and its value ranges from zero upwards. It decides on the suffix removal [3].

The formula '(condition) S1 -> S2' is applied for all the rules and it is read as follows: if the remaining letters of suffix S1 will satisfy the condition, replace suffix S1 with suffix S2.

The most important step in the algorithm is step one which handles past participles and plurals and because of the complexity of this task, the step has three parts (1a, 1b and 1c) in the original definition as follows [3]:

1. 1a: this part removes 's' from plurals, for example sses -> ss and recodes.
2. 1b: this part removes 'ed' and 'ing' if found and then transforms the remaining stem.
3. 1c: this part simply transforms a terminal 'y 'to an' i' (in the flowchart above it is shown as step 2)

The steps after these become relatively straightforward and have their rules that cope with different order classes of suffices.

A method was added to the TE code to handle the stemming procedure through interacting with a package containing classes and methods for implementing the

Porter's stemming algorithm. For every photo entry, each tag is stemmed using the new method and the resulting stem will replace the original tag as the user tag that will be used to query PWN, MWN, and Flickr clusters. If querying all the external sources produces no results then, we must consider the possibility that the new user tag is not a real word, due to overstemming errors.

6.1 Summery

The chapter discussed how the stemming component is embedded in the TE system by including a table of Java and flickrj components which are essential in providing the necessary functionality to generate the sample data from Fickr and perform the stemming algorithm on the tags. Furthermore, the steps of the porter stemmer are explained and illustrated using flowchart. The chapter concluded by describing the behaviour of the stemming component inside the TE system.

7 The Evaluation

7.1 Introduction

In the problem statement, the researcher presented his argument about the need for a stemming component in the TE system to lessen the effect of problems such as the large size of the index table, and database.

The effect of adding this component needs to be investigated. Therefore, an experiment is designed to study the original TE system (without the stemming component) and the new TE system (with the stemming component) in terms of the size of the index table and the number of results retrieved from both systems testing the claims that stemming reduces the size of the index table and broaden the search to include more results.

The experiment is performed on a data sample imported from flickr containing 7810 photos using 33760 unique tags but only 8728 tags are generated from the PWN and MWN ontologies and these are the tags that will be used in the new TE system and stemmed with the stemming component. With that said, the experiment will exclude any system tags generated from the clustering component for fairness reason.

New tables are added to the database design to accommodate the addition of the stemming component as follows:

Table name	Role
tags_stems_master	Stores the values of the original user tags and their stems
tags_stems_detail	Stores the values of the stems and their generated system tags and the type of tag
search_runs	This table is added as a log to the search trails. It stores the search keyword and the number of results from the original and new TE systems (not shown in the DB diagram)

Table 7.1: List of the new tables in the Database

The database design is explained by the next relationship diagram

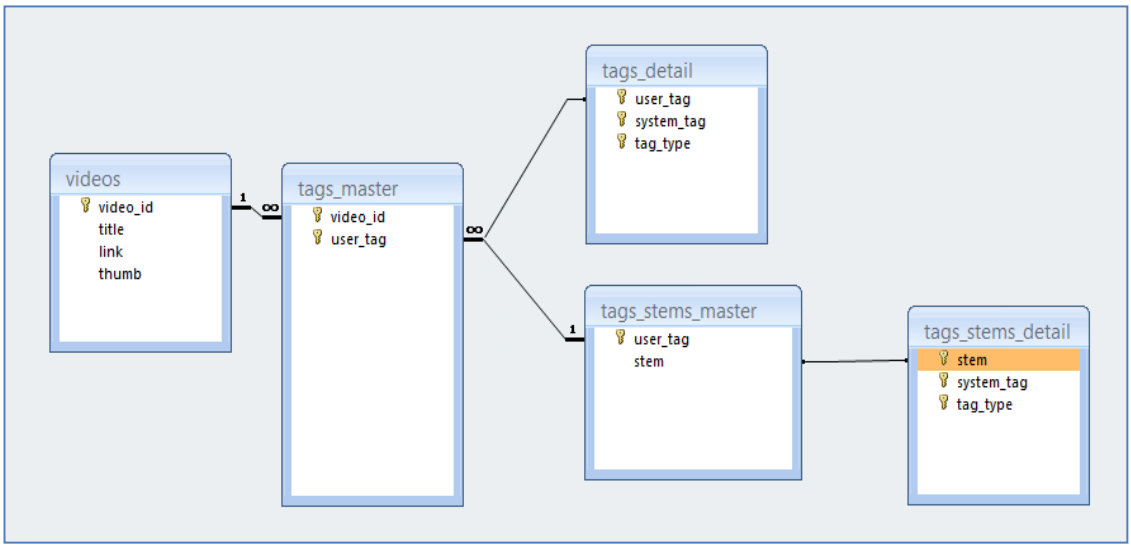

Figure 7.1: TE Database Diagram

7.2 The experiment

7.2.1 Index table

In the original TE system, any search trail is conducted by comparing the search term against the stored system tags as 'system_tag' in table 'tags_detail'. Next, all associated user tags 'user_tag' which generated the user tags from the semantic and clustering components are retrieved from table 'tags_master'. Finally all distinct videos tagged using the collected user tags are retrieved.

In the new TE system, a similar method is applied except that all user tags are subjected to the stemming component to reduce the number of word variants. Thus, system tags here are generated from stemmed user tags instead of the original user tags.

The number of system tags in both TE systems determines the size of the index table. Furthermore, since system tags are generated from the user tags in the original TE system and from stems in the new TE system, then the size of both sources is reflected in the size of the index table of both systems.

In the original TE system, calculating the number of user tags is straightforward. Querying the 'tags_detail' table for distinct 'user_tag' values resulted in 9633 hits but after excluding orphan user tags that didn't yield any system tags the number was reduced to 8728 and this is considered in the experiment as the base.

In the new TE system, the same approach is used to calculate the number of distinct stemmed English/Italian user tags from 'tags_stems_detail' table and the result is 4813 hits reduced to 4478 after eliminating orphan stems.

	user tags in Original TE	stemmed user tags in New TE	difference	ratio
total	9633	4813	4820	
Total - orphans	8728 (baseline)	4478	4250	48.69%

Table 7.2: User Tags & Stems User Tags Statistics

From the previous table we can see that the stemming component reduced the number of user tags by more than 48%.

Furthermore, the percentage of orphan entries from the user tags is 9.4% and 7% from stems thus both percentages are less than 10%. The following table shows the statistics regarding the number of system tags in both systems which is reduced significantly after stemming by more than 48% which is near the reduction percentage of the user tags confirming the effect of this reduction on the number of generated system tags.

system tags in Original TE	system tags in New TE	difference	ratio
34537	17761	16776	48.57%

Table 7.3: System Tags Statistics

The experiment has proved that the stemming component has reduced the size of the index table by more than 48%.

7.2.2 The search results

For the search process, the experiment runs the same sample data of 30 search terms on both TE systems and the results from both are counted as listed below:

no.	search term	results from original TE (A)	results from new TE (B)	difference (B) - (A)
1	abandon	19	19	0
2	abstraction	47	47	0
3	absurd	3	3	0
4	cake	35	10	-25
5	capo	39	44	5
6	care	26	14	-12
7	case	77	76	-1
8	cast	7	10	3
9	cleaned	6	7	1
10	clothing	32	23	-9
11	gentle	5	4	-1
12	instance	38	36	-2
13	instructor	21	20	-1
14	integrated	32	3	-29
15	interview	7	7	0
16	involvement	17	14	-3
17	marriage	4	3	-1
18	measure	28	27	-1
19	meat	15	13	-2
20	personnel	37	21	-16
21	program	25	27	2
22	publicity	54	57	3
23	seeker	4	4	0
24	selection	20	7	-13
25	table	40	22	-18
26	tail	8	10	2
27	workplace	80	83	3
28	world	94	105	11
29	yellowness	64	64	0
30	yield	19	22	3

Table 7.4: Search Results Statistics from both TE Systems

	new TE results > original TE results	**new TE results = original TE results**	**new TE results < original TE results**
count	9	6	15
percentage	30%	20%	50%

Table 7.5: Percentage of Search Results

From the above table, it is clear that in half of the search trails the original TE retrieved more photos and therefore, the stemming did not broaden the search results. This fact is different from the claim and we had to take a closer examination at the search process to invistigate. The search query had been broken down to sub queries to identify the tags used in them as parameters. A few points were noted and can explain the confusion in table 7.5:

1. Stemming errors: stemming some user tags produced overstemmed or understemmed words and sometimes these words cannot produce system tags from PWN and MWN ontologies because they are distorted to even be real English words. Searching for a term means looking up the search term in the 'system_tag' field in the 'tags_detail' table in the old TE system and this will select its associated user tags whereas; it means looking up the same search term in the 'system_tag' field in the 'tags_stems_detail' table in the new TE system and this will select its associated stems and sometimes not all retrieved user tags from 'tags_detail' has their corresponding stems in the retrieved stems from 'tags_stems_detail' table. For example, the search term 'cake' has associated user tags 'cookie' , 'cookies', 'cupcake', 'cupcakes', and 'pancake' that do not have their stems in the retrieved stems from 'tags_stems_detail'. Stemming 'cookie' will result in 'cooki' which is not a real English word thus it will generate no system tags to be saved in the 'tags_stems_detail'.

Search term	Associated User tags	Associated Stems	results from original TE (A)	results from new TE (B)	(B) – (A)
table	Booth	Booth	40	22	-18
	Counters	Counter			
	Desk	Desk			
	Mesa	Mesa			
	Table	-			
cake	Baba	Baba	35	10	-25
	Cake	Cake			
	Cookie	-			
	Cookies	-			
	Cupcake	-			
	Cupcakes	-			
	Gingerbread	Gingerbread			
	Pancake	-			

Table 7.6: Examples of Stemming Errors

2. The quality of user tags: the photos in the data sample were tagged by users and these user tags are saved in the 'user_tag' field in both the 'tags_master', 'tags_stems_master' tables. Some of these tags are not real English words even before stemming them. For example a photo is tagged using the word 'mens' which is not an English word.
3. The language of the sample data: except for one tag, all user tags are English and as mentioned in the literature review, IR researchers had this reasonable assumption that for languages that are more highly than English, stemming will have greater improvements [38].

From the above table, every search that compared the search term against correctly stemmed user tags (avoiding points 1 and 2) yielded more results than what is retrieved by comparing the search term against the original user tags. Thus, using

additional tools to check the correctness of the stems will overcome point1. For example, we can embed a Lemmatization analyzer to validate the generated stem.

Regarding point2, a dictionary of the language used will decide if a user tag is a valid word or not.

Finally, the last point takes the discussion back to the multilingual aspect of the TE system which is implemented to certain degree by supporting the use of Italian words which use MWN ontology and the Italian porter stemmer. The addition of more complex languages to the TE system can make the stemming more beneficial.

8 Conclusion and Future Work

8.1 Research Overview

The topic of tagging systems is a very active research area and many studies had presented various improvements to the existing architecture of the tagging system. Despite the disadvantages of folksonomies, users are willing to overlook lack of semantic meaning, ambiguity, inconsistency…etc (see chapter 2) in order to take advantage of the simplicity and freedom of folksonomies. The advances in the fields of semantic web and social web had a huge impact on the research of folksonomies. The thesis conducted a comprehensive literature review on tagging systems including the history, the reasons behind their popularity, and the drawbacks. Furthermore, the literature review covered the subjects of Princeton WordNet (PWN) and Multi WordNet (MWN) ontologies.

The TE system is the result of research targeting certain problems in tagging systems. It includes two components, a semantic component and a clustering component to address the drawbacks of multilingualism and a lack of semantic and shorthand writing (which is very common in the social web). The TE system is a partial implementation of the proposed architecture presented in [72].

The current research is proposing the embedding of a new component to the TE system, suggested in the original architecture illustrated in Figure 3.1, to the TE system. In order to achieve the proposed objective, the current research had to perform the following tasks:

- The TE system had to be explored thoroughly and then summarised in chapter3.
- The efficiency of the TE algorithm had to be determined in order to decide on the practicality and feasibility of the system before investing time and effort in adding new features to a system that cannot be applied in the real world.

Moreover, the research also involves the database of the TE system. Many design factors in the database architecture have been critically examined to determine if they are set for optimal performance. The examination performed has been mostly guided by the manual provided by MySQL and all findings were presented in chapter 5, along with the algorithm complexity analysis.

Based on these findings, the research went ahead and proposed embedding a stemming component to the TE system for normalisation purpose and for reducing the index table and broadening the search results. The stemming component used the Java encoding of the Porter stemming algorithm. This selection was made after looking into the background of stemming algorithms, their techniques, their types, and the pros and cons of some of the popular stemming algorithms.

In the sample tagging system, user tags are subjected to normalisation using the stemming component, which is embedded inside the original TE code using a method and a package. The generated stem is saved as the new user tag replacing the original user tag and then it is used to query the semantic and social sources instead of the original user tag.

An experiment is designed to measure the effect of stemming on the size of the index table and the scope of the search results by running two versions of the TE systems one without the stemming component and the other one with it using the same search dataset and the findings section below explain the outcomes of the experiment.

8.2 Findings

Q1: What are the effects on performance of embedding the stemming component to the TE system?

Stemming can offer a solution for the problem of word's different lexical forms which is common in tagging systems in addition to reducing the size of the index table and broadening the search results.

In the TE system, the time complexity is dependent on the number of user tags 'N' which is used as a counter for the inner most iteration with the main body of the code thus the smaller this number, the better. Therefore, exposing the user tags to the stemmer before querying any of the semantic or social sources will get rid of some of them and that will reduce the value of 'N' and thus improving the time.

After the normalisation of user tags, there was a drop in the number of retrieved system tags and by looking at the run listing we found that in some instances, the targeted sources were queried using unreal words thus no results could be retrieved. This situation is caused by overstemming which is common when using light stemming algorithms such as the Porter2 stemmer. To fix this, the TE system can use a heavier stemming algorithm or pre-stemmer but as explained before, the Porter2 stemmer was selected because it is fast and language independent and this is excellent for what the TE system needs. Another suggesting is the addition of lemmatization analyzer and/or vocabulary of the language used which will indicate the valid words prior to stemming and then the analyzer will validate the correctness of the stem prior to querying the semantic and clustering components.

The experiment indicates that the stemming component reduced the number of user tags by more than 48%. Furthermore, a similar reduction percentage is noted regarding the number of system tags in both systems which represent the index table.

As part of the experiment, search trails were performed on the old TE and new TE using 30 terms and the retrieved results are recorded for comparison. At first, it looked like the new TE with the stemming component retrieved fewer results in almost half the trails with 30% of the trials having the same number of results from both systems.

Examining the search queries showed that there is an explanation to this. Overstemming or overstemming user tags cannot produce system tags from PWN and MWN ontologies because they are distorted to even be real English words. Another factor is the quality of user tags. If they are they invalid words in the language used then they will not generate valid stems. Moreover, the language used might have

played a role in the statistics above since some researchers point that simple morphological languages like English does not benefit from stemming as other complex languages.

Incorporating other tools can be helpful in overcoming the points mentioned here such as the use of vocabulary, analyzer, and other languages as discussed in the evaluation chapter.

Q2: What is the time complexity of the TE algorithm?

The time complexity analysis conducted shows that the TE system has a time growth rate of $O(N^2)$ which is a polynomial (quadratic) time. As mentioned previously, in the literature review in chapter 2, an algorithm with a computation that runs in linear or quadratic time is considered to be 'efficient' and polynomial growth is considered manageable. The execution time can make use of some modifications to reduce it such as:

- SQL patch execution: the code contains many SELECT and INSERT SQL statements and most of them are located inside conditions and iterations and, as mentioned in chapter 5, executing these statements one by one is time costly especially in the case of SELECT and INSERT statements. Patch execution can perform instead, guided by certain flag variables to indicate whether a specific SELECT/INSERT statement is to be executed or not under a certain condition/iteration.
- Limit the input from the semantic resources: after running the TE code and looking at the database, we found a user tag that generated 157 system tags mainly with similarity relation. This considerable number of system tags has its cost on the database storage, the time needed to query the semantic resources, and the time needed to perform the search. Enforcing a limit on the semantic retrieval input will be beneficial.

Q3: What is the space complexity of the TE algorithm?

The problem size is determined by the number of user tags per photo. Every photo has its user tags stored inside the parameter keywordStringList of type List<String> and thus keywordStringList.size is used as a counter (N) in the main loop of the algorithm. The only parameter that is varying in size according to the growth rate of N is keywordStringList with SPACE(N) which is a linear space. The rest of the parameters are mainly declared as public and are allocated a constant space SPACE(1).

Q4: Is the database design optimised for the TE ER Model?

The selection of InnoDB storage engine is fit for the requirements of the TE system especially for its support for foreign keys and fast index creation/deletion.

Data types can be optimised to reduce the database size as mentioned in chapter 5. Some database columns may need to change their data type while others may only adjust their size. Some database columns can express their contents in a different manner to reduce the database size too as in the columns link and thumb from the videos table. The database schema is very basic, thus the columns needed to be selected as indexes were obvious, so the appointed indexes are simply justified.

8.3 Success criteria revisited

In the evaluation chapter, we conducted an experiment to gather certain information to answer our main research question. The outcomes of the experiment indicated that stemming reduces the size of the index table and increased the number of results retrieved on the condition of working with valid user tags and valid stems.

8.4 Contributions

- The TE system is re-implemented on Flickr instead of YouTube. This has the advantage of overcoming the language barrier when using more than one language to generate system tags since photos are expressed visually and the user does not need to have prior knowledge of any of the supported languages.

- User tags are normalised before the querying of the semantic resources Princeton WordNet (PWN) and MultiWordNet (MWN). This is accomplished by embedding a stemming component which is based on the Java encoding of the Porter's 2 stemmer. This component provides a solution for the problem of different lexical forms which can be found in many tagging systems. Furthermore, it reduces the size of the index table and increases the retrieved results (on the condition of using valid user tags and valid stems). The decrease in the index table means that the search process will compare the search keyword against less system tags and this speed up the search. In addition, the TE system in particular will benefit from the stemming component because of its method of generating extra tags to enhance the semantic of the original user tag. Using the user tags from our sample database, we can see that a user tag can generate up to 157 system tags and there is no limit on this number getting larger with a semantically rich user tag. Thus, having variants of this user tag is a serious problem that can be easily avoided by mapping all variants to their stem and only generate system tags based on it.
- The old TE system is using the WordNet and MultiWordNet ontologies to provide semantic to the user tags in English and Italian only. The embedded stemmer follows on that and it handles English and Italian words.

The new TE system can be generalized across many languages by incorporating more languages from MultiWordNet or adding other multilingual ontologies. A list of ontologies is available from the Global WordNet Association web site [111]. The stemming component can also use stemmers in other languages. The Snowball framework for the porter stemmer offers the algorithm encoding for the stemmer in many languages like French, Spanish, Portuguese, German, Dutch, Swedish …etc [97]. Other than the porter stemmer, there are many stemming algorithms in Non-English languages that can be used but some might be difficult to encode. Enriching the TE system with more languages makes the search boarder and this is useful in tagging system based on photo content where language barrier does not matter.

The researcher claims that the proposed new TE with the stemming component has a balance to a certain degree between the storage needed for the generated system tags and the storage saved by mapping variants to one stem. This means that TE can cover more semantically related results with less concern about the storage overhead.

This model can be used in many applications such as machine translation, document summarization, text classification, e-mail filtering, web browsing, and information extraction.

8.5 Limitations and Future work

The research can be further advanced by exploring the following areas:

- Investigating the effect of adding more tags from different sources and looking at the time needed to perform the search process. This will establish the trade-offs between either having a more versatile search result or performing a faster search.
- Examining the quality of the retrieval results by measuring the precision, and recall rate.

References

1. (2005) *The Lancaster Stemming Algorithm: Background to Stemmming* [WWW] Lancaster University. Available from: http://www.comp.lancs.ac.uk/computing/research/stemming/Links/background.htm [Accessed 20/10/2013].

2. (2005) *The Lancaster Stemming Algorithm: Introduction* [WWW] Lancaster University. Available from: http://www.comp.lancs.ac.uk/computing/research/stemming/Links/introduction.htm [Accessed 20/10/2013].

3. (2005) *The Lancaster Stemming Algorithm: Porter* [WWW] Lancaster University. Available from: http://www.comp.lancs.ac.uk/computing/research/stemming/Links/porter.htm [Accessed 20/10/2013].

4. (n.d.) *Performance Analysis* [WWW] Shanghai Jiao Tong University. Available from: http://ee.sjtu.edu.cn/po/Class-web/data-stucture/csc120ch2.doc [Accessed 15/12/2012].

5. (n.d.) *What is Program* [WWW]. Available from: http://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=0CEMQFjAC&url=http%3A%2F%2Fprogramming.im.ncnu.edu.tw%2FCH1.ppt&ei=gVlOUbzHMeeR0QXy-YG4Bw&usg=AFQjCNF1Rp8MESQwXj5TXX73ynLMyQAGdQ&sig2=tIZg99k3bKafaKR4y765OA&bvm=bv.44158598,d.d2k [Accessed 5/12/2012].

6. AL-SAYED, G. (n.d.) *Data Structures Performance Analysis* [WWW] King Saud University. Available from: faculty.ksu.edu.sa/Gamal/CSC212%20Lectures/Lect2.ppt [Accessed 5/2/2013].

7. AL-SHORBAGY, G. (n.d.) *Lecture 2: Data Structures - Performance Analysis* [WWW] King Saud University. Available from: http://faculty.ksu.edu.sa/Gamal/CSC212%20Lectures/Lect2.ppt [Accessed 24/8/2012].

8. AP COMPUTER SCIENCE (n.d.) *Big-O Notation* [WWW]. Available from: http://www.apcomputerscience.com/apcs/misc/BIG-O.pdf [Accessed 25/2/2013].

9. ARORA, R. and RAVINDRAN, B. Latent Dirichlet Allocation based Multi-Document Summarization. In: *Proceedings of the 2nd workshop on Analytics for Noisy Unstructured Text Data*, Singapore, July 2004: ACM, pp. 91-97.

10. ARORA, S. and BARAK, B. (2009) *Computational Complexity: A Modern Approach.* Cambridge, UK: Cambridge University Press.

11. BARNES, N.G. (2014) Social Commerce emerges as Big Brands position themselves to Turn "Follows", "Likes", and "Pins" into Sales. *American Journal of Management*, 14 (4), pp. 11-18.

12. BEGELMAN, G., KELLER, P., and SMADJA, F. Automated Tag Clustering: Improving Search and Exploration in the Tag Space. In: *Preceedings of the 15th Int. World Wide Web Conference (WWW2006)*, Edinburgh, May 2006, pp. 15-33.

13. BERNERS-LEE, T., HENDLER, J., and LASSILA, O. (2001) The Semantic Web. *Scientific American*, 284 (5), pp. 28-37.

14. BERNERS-LEE, T., MASINTER, L., and MCCAHILL, M. (1994) *Uniform Resource Locators (URL)* [WWW]. Available from: http://www.w3.org/Addressing/rfc1738.txt [Accessed 12/1/2013].

15. BESTAVROS, A. (1995) *CLA CS-101: Introduction to Computers* [WWW] Boston University. Available from: http://www.cs.bu.edu/~best/courses/cs101/F97/lectures/AlgorithmComplexity.html [Accessed 10/4/2013].

16. BIGGERS, L.R. and KRAFT, N.A. (2012) *A Comparison of Stemming Algorithms for Text Retrieval Based Feature Location* [WWW] The University of Alabama. Available from: http://scholar.googleusercontent.com/scholar?q=cache:Eq32lcz7IOEJ:scholar.google.com/+A+Comparison+of+Stemming+Algorithms+for+Text+Retrieval+Based+Feature+Location&hl=en&as_sdt=0,5 [Accessed 6/4/2013].

17. BOYD, D.M. and ELLISON, N.B. (2007) Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication*, 13 (1), pp. 210-230.

18. BOYKOV, Y. (2012) *Analysis of Algorithms* [WWW] The University of Western Ontario. Available from: http://www.csd.uwo.ca/courses/CS1037a/notes/topic13_AnalysisOfAlgs.pdf [Accessed 20/2/2013].

19. BRASSARD, G. and BRATLEY, P. (1988) *Algorithmics: Theory & Practice.* Englewood Cliffs, NJ: Prentice-Hall, Inc.

20. BRINTON, L.J. (2000) *The Structure of Modern English: A Linguistic Introduction.* Amsterdam: John Benjamins.

21. BUSEMANN, S., SCHMEIER, S., and ARENS, R.G. Message Classification in the Call Center. In: *Proceedings of the 6th Conference on Applied Natural Language Processing*, Seattle, May 2000: Association for Computational Linguistics, pp. 158-165.

22. BUTTCHER, S., CLARKE, C.L., and CORMACK, G.V. (2010) *Information Retrieval: Implementing and Evaluating Search Engines.* Mit Press.

23. BYLANDER, T. (2006) *Chapter 2: Fundamentals of the Analysis of Algorithm Efficiency* [WWW] The University of Texas at San Antonio. Available from: http://www.cs.utsa.edu/~bylander/cs3343/chapter2handout.pdf [Accessed 20/3/2013].

24. CABRAL, S.K. and MURPHY, K. (2011) *MySQL Administrator's Bible.* Indianapolis: Wiley.

25. CHAUPATTNAIK, S., NANDA, S.S., and MOHANTY, S. (2012) A Suffix Stripping Algorithm for Odia Stemmer. *International Journal of Computational Linguistics and Natural Language Processing*, 1 (1), pp. 1-5.

26. COFFEY, N. (2011) *How to calculate the Memory Usage of a Java Array* [WWW] JAVAMEX. Available from: http://www.javamex.com/tutorials/memory/array_memory_usage.shtml [Accessed 7/5/2013].

27. COFFEY, N. (2011) *Memory Usage of Java Objects: General Guide* [WWW] JAVAMEX. Available from: www.javamex.com/tutorials/memory/object_memory_usage.shtml [Accessed 7/5/2013].

28. COFFEY, N. (2011) *Memory Usage of Java Strings and String-Related Objects* [WWW]. Available from: http://www.javamex.com/tutorials/memory/string_memory_usage.shtml [Accessed 7/5/2013].

29. CRONIN, P., RYAN, F., and COUGHLAN, M. (2008) Undertaking a Literature Review: a Step-by-Step Approach. *British Journal of Nursing*, 17 (1), pp. 38.

30. CURIOSO, A., BRADFORD, R., and GALBRAITH, P. (2010) *Expert PHP and MySQL.* Indianapolis: Wiley.

31. DALIANIS, H. (2000) *SweSum - A Text Summarizer for Swedish* [WWW]. Available from: http://www.dsv.su.se/~hercules/papers/Textsumsummary.html [Accessed 12/6/2014].

32. DARWISH, K. Building a Shallow Arabic Morphological Analyzer in One Day. In: *Proceedings of the ACL-02 Workshop on Computational Approaches to Semitic Languages*, Pennsylvania: Association for Computational Linguistics, pp. 1-8.

33. DRAKE, M. (2003) *Encyclopedia of Library and Information Science.* 2nd Edition. Taylor & Francis.

34. DUNHAM, D. (2010) *Growth of Functions: Asymptotic Notation* [WWW]. Available from: http://www.d.umn.edu/~ddunham/cs3512s10/notes/l12.pdf [Accessed 7/1/2013].

35. FRAKES, W.B. and BAEZA-YATES, R. (1992) *Information Retrieval: Data Structures & Algorithms.* Englewood Cliffs: Prentice Hall.

36. FRAKES, W.B. and FOX, C.J. (2003) Strength and Similarity of Affix Removal Stemming Algorithms. In: *ACM SIGIR Forum*, Toronto, July 2003: ACM, pp. 26-30.

37. FUNG, G. (2001) A Comprehensive Overview of Basic Clustering Algorithms.

38. GALVEZ, C. and DE MOYA-ANEGON, F. (2006) An Evaluation of Conflation Accuracy using Finite-state Transducers. *Journal of Documentation*, 62 (3), pp. 328-349.

39. GALVEZ, C., DE MOYA-ANEGON, F., and SOLANA, V. (2005) Term Conflation Methods in Information Retrieval: Non-Linguistic and Linguistic Approaches. *Journal of Documentation*, 61 (4), pp. 520-547.

40. GATLING, G. (2012) *The Need for Information Management* [WWW] SAP. Available from: http://blogs.sap.com/innovation/analytics/information-management-01494 [Accessed 24/6/2014].

41. GAUSTAD, T. and BOUMA, G. (2002) Accurate Stemming of Dutch for Text Classification. *Language and Computers*, 45 (1), pp. 104-117.

42. GELBUKH, A., ALEXANDROV, M., and HAN, S.-Y. (2004). Detecting Inflection Patterns in Natural Language by Minimization of Morphological Model. Progress in Pattern Recognition, Image Analysis and Applications, Springer: 432-438.

43. GEMMELL, J., SHEPITSEN, A., MOBASHER, M., and BURKE, R. Personalization in Folksonomies based on Tag Clustering. In: *Proceedings of the 6th Workshop on Intelligent Techniques for Web Personalization and Recommender Systems (AAAI 2008)*, Chicago, July 2008, pp. 42.

44. GUPTA, V. and LEHAL, G.S. Punjabi Language Stemmer for Nouns and Proper Names. In: *Proceedings of the 2nd Workshop on South and Southeast Asian Natural Language Processing (WSSANLP)*, Hyderabad: Citeseer, pp. 35-39.

45. GUPTA, V. and LEHAL, G.S. (2013) A Survey of Common Stemming Techniques and Existing Stemmers for Indian Languages. *Journal of Emerging Technologies in Web Intelligence*, 5 (2), pp. 157-161.

46. HADNI, M., LACHKAR, A., and OUATIK, S.A. (2012) A New and Efficient Stemming Technique for Arabic Text Categorization. In: *Int. Conference on Multimedia Computing and Systems (ICMCS)*, Tangier-Morocco, May 2012: IEEE, pp. 791-796.

47. HADZILACOS, V. *TIME COMPLEXITY OF ALGORITHMS* [WWW] University of Toronto. Available from: http://www.cs.toronto.edu/~vassos/teaching/c73/handouts/brief-complexity.pdf [Accessed 15/9/2012].

48. HARDY, H., SHIMIZU, N., STRZALKOWSKI, T., TING, L., ZHANG, X., and WISE, G.B. Cross-Document Summarization by Concept Classification. In: *Proceedings of the 25th Annual Int. ACM SIGIR Conference on Research and Development in Information Retrieval*, Finland, August 2002: ACM, pp. 121-128.

49. HAREL, D. and ROSNER, R. (1992) *Algorithmics: The Spirit of Computing.* 2nd. Addison-Wesley.

50. HEDDEN, H. (2010) *The Accidental Taxonomist.* Information Today.

51. HONG, J. (2002) *Performance Measurement* [WWW] Pohang University of Science and Technology. Available from: http://www.google.co.uk/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0CDkQFjAA&url=http%3A%2F%2Fdpnm.postech.ac.kr%2Fcs233%2Flecture%2Flecture2-perf-measurement.ppt&ei=j1ZOUeyDL-b80QXI44DwDA&usg=AFQjCNFZjZNWhXtfHRe3usVexBgRp2fRFg&sig2=AEenzgp0kap58qu3Yy9mUw&bvm=bv.44158598,d.d2k [Accessed 10/1/2013].

52. HUSSAIN, M. (n.d.) *Performance Analysis* [WWW] King Saud University. Available from: http://faculty.ksu.edu.sa/mhussain/CSC212/Lecture%20-%20Performance%20Analysis.pdf [Accessed 10/2/2013].

53. INTERNET WORLD STATS (2012) *Internet Usage Stats and Population Statistics* [WWW]. Available from: www.internetworldstats.com/stats.htm [Accessed 1/7/2013].

54. JEFF'S SQL SERVER BLOG (2007) *Composite Primary Keys* [WWW]. Available from: http://weblogs.sqlteam.com/jeffs/archive/2007/08/23/composite_primary_keys.aspx [Accessed 21/12/2012].

55. JIVANI, A.G. (2011) A Comparative Study of Stemming Algorithms. *Int. Journal of Computer Technology and Applications*, 2 (6), pp. 1930-1938.

56. KATOEN, J. (2002) *Introduction to Algorithm Analysis: Lecture #1 of Algorithms, Data Structures and Complexity* [WWW]. Available from: http://jriera.webs.ull.es/Docencia/lec1.pdf [Accessed 28/9/2012].

57. KIM, H.L., SCERRI, S., BRESLIN, J.G., DECKER, S., and KIM, H.G. The State of the Art in Tag Ontologies: a Semantic Model for Tagging and Folksonomies. In: *International Conference on Dublin Core and Metadata Applications*, Berlin, September 2008, pp. 128-137.

58. KODIMALA, S. (2010) *Study of Stemming Algorithms.* Msc, University of Nevada.

59. KORPELA, J. (2006) *Unicode Explained.* Sebastopol: O'Reilly.

60. KRAAIJ, W. and POHLMANN, R. (1995) Evaluation of a Dutch Stemming Algorithm. *The New Review of Document and Text Management*, 1, pp. 25-43.

61. KUCK, G. (2004) Tim Berners-Lee's Semantic Web. *SA Journal of Information Management*, 6 (1).

62. LARKEY, L.S., BALLESTEROS, L., and CONNELL, M.E. Improving Stemming for Arabic Information Retrieval: Light Stemming and Co-Occurrence Analysis. In: *Proceedings of the 25th Annual Int. ACM SIGIR Conference on Research and Development in Information Retrieval*, Tampere, August 2002: ACM, pp. 275-282.

63. LAZARINIS, F., VILARES, J., TAIT, J., and EFTHIMIADIS, E.N. (2009) Current Research Issues and Trends in Non-English Web Searching. *Information Retrieval*, 12 (3), pp. 230-250.

64. LEE, S.-S. and YONG, H.-S. Component based Approach to handle Synonym and Polysemy in Folksonomy. In: *Proceedings of the 7th IEEE Int. Conference on Computer and Information Technology (CIT)*, Fukushima, October 2007, pp. 200-205.

65. LEE, Y.-S. (2004). Morphological Analysis for Statistical Machine Translation. Proceedings of HLT-NAACL 2004. Boston, Association for Computational Linguistics.

66. LENNON, M., PEIRCE, D.S., TARRY, B.D., and WILLETT, P. (1981) An Evaluation of some Conflation Algorithms for Information Retrieval. *Journal of Information Science*, 3 (4), pp. 177-183.

67. LEVITIN, A. (2012) *Introduction to the Design & Analysis of Algorithms.* 3rd. Boston: Pearson.

68. LEVY, Y. and ELLIS, T.J. (2006) A Systems Approach to conduct an Effective Literature Review in Support of Information Systems Research. *Informing Science: International Journal of an Emerging Transdiscipline*, 9 (1), pp. 181-212.

69. LI, Q. (2008) Collaborative Tagging Applications and Approaches. *IEEE MultiMedia*, 15 (3), pp. 14-21.

70. LIACS MEDIALAB (2008) *The MIRFLICKR Retrieval Evaluation* [WWW] Leiden University. Available from: http://press.liacs.nl/mirflickr/ [Accessed 30/3/2013].

71. LIN, J.Y., YANG, C.H., TSENG, S.C., and HUANG, C.R. The Structure of Polysemy: A study of multi-sense words based on WordNet. pp. 320-329.

72. MAGABLEH, M. (2011) *A Generic Architecture for Semantic Enhanced Tagging Systems.* PhD, De Montfort University.

73. MAJUMDER, P., MITRA, M., PARUI, S.K., KOLE, G., MITRA, P., and DATTA, K. (2007) YASS: Yet Another Suffix Stripper. *ACM Transactions on Information Systems (TOIS)*, 25 (4), pp. 18.

74. MATHES, A. (2004) Folksonomies-cooperative classification and communication through shared metadata. *Computer Mediated Communication*, 47 (10).

75. MCPHERSON, D. (2005) *Analysis* [WWW] Virginia Tech. Available from: http://courses.cs.vt.edu/cs2604/spring05/mcpherson/note/Analysis.pdf [Accessed 5/3/2013].

76. MERHOLZ, P. (2004) *Metadata for the Masses* [WWW] Adaptive Path. Available from: http://www.adaptivepath.com/ideas/e000361/ [Accessed 15/12/2012].

77. MIKA, P. and GREAVES, M. (2008) *Editorial: Semantic Web & Web 2.0. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6 (1), pp. 1-3.

78. MILLER, G.A. (1995) WordNet: a lexical database for English. *Communications of the ACM*, 38 (11), pp. 39-41.

79. MILLER, G.A., BECKWITH, R., FELLBAUM, C., GROSS, D., and MILLER, K.J. (1990) Introduction to wordnet: An on-line lexical database*. *International journal of lexicography*, 3 (4), pp. 235-244.

80. MYSQL (n.d.) *MySQL 5.6 Reference Manual* [WWW]. Available from: http://dev.mysql.com/doc/refman/5.6 [Accessed 20/12/12].

81. NORUZI, A. (2007) Folksonomies-Why do we need controlled vocabulary? *Webology*.

82. NTAIS, G. (2006) *Development of a Stemmer for the Greek Language.* Msc, Stockholm University.

83. O'REILLY (2005) **What Is Web 2.0* [WWW]. Available from: http://oreilly.com/pub/a/web2/archive/what-is-web-20.html?page=2 [Accessed 25/4/2012].

84. OBENZINGER, H. (2005) *What can a Literature Review do for me? How to research, write, and survive a Literature Review* [WWW] Stanford University. Available from: http://ce.uoregon.edu/aim/Capstone07/LiteratureReviewHandout.pdf [Accessed 24/9/2014].

85. ORASAN, C., PEKAR, V., and HASLER, L. A Comparison of Summarisation Methods based on Term Specificity Estimation. In: *Proceedings of the 4th Int. Conference on Language Resources and Evaluation (LREC2004)*, Lisbon, May 2004, pp. 1037-1041.

86. OZGUR, L. and GUNGOR, T. Analysis of Stemming Alternatives and Dependency Pattern Support in Text Classification. In: *Proceedings of the 10th Int. Conference on Intelligent Text Processing and Computational Linguistics (CICLing), Research in Computing Science Vol (41)*, Mexico City, pp. 195-206.

87. PAICE, C.D. (1990) Another Stemmer. *SIGIR Forum*, 24 (3), pp. 56-61.

88. PAICE, C.D. (1994) An Evaluation Method for Stemming Algorithms. In: *Proceedings of the 17th Annual International ACM SIGIR Conference on Research & Development on Information Retrieval (SIGIR)*, Dublin, July 1994: Springer-Verlag, pp. 42-50.

89. PANDEY, A.K. and SIDDIQUI, T.J. An Unsupervised Hindi Stemmer with Heuristic Improvements. In: *Proceedings of the 2nd Workshop on Analytics for Noisy Unstructured Text Data*, Singapore, July 2008: ACM, pp. 99-105.

90. PARBERRY, I. (2001) *Lecture Notes on Algorithm Analysis and Computational Complexity* [WWW] University of North Texas. Available from: http://larc.unt.edu/ian/books/free/lnoa.pdf [Accessed 28/12/2012].

91. PETERS, I. (2009) *Folksonomies: Indexing and Retrieval in Web 2.0.* Berlin: Walter de Gruyter.

92. PETERS, I. and BECKER, P. (2009) *Folksonomies: Indexing and Retrieval in Web 2.0.* Germany: De Gruyter/Saur.

93. PIANTA, E., BENTIVOGLI, L., and GIRARDI, C. MultiWordNet Developing an aligned multilingual database.

94. PIRKOLA, A. (2001) Morphological Typology of Languages for IR. *Journal of Documentation*, 57 (3), pp. 330-348.

95. POPOVIC, M. and NEY, H. Towards the Use of Word Stems and Suffixes for Statistical Machine Translation. In: *Proceedings of the 4th Int. Conference on Languages Resources and Evalution*, Lisbon, May 2004.

96. POPOVIC, M. and WILLETT, P. (1992) The Effectiveness of Stemming for Natural-Language Access to Slovene Textual Data. *Journal of the American Society for Information Science*, 43 (5), pp. 384-390.

97. PORTER, M. (2001) *Snowball: A Language for Stemming Algorithms* [WWW]. Available from: http://snowball.tartarus.org/texts/introduction.html [Accessed 20/8/2013].

98. PORTER, M. (2006) *The Porter Stemming Algorithm* [WWW]. Available from: http://tartarus.org/martin/PorterStemmer/ [Accessed 25/9/2013].

99. SAHARIA, N., SHARMA, U., and KALITA, J. (2012) Analysis and Evaluation of Stemming Algorithms: A Case Study with Assamese. In: *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Chennai, August 2012: ACM, pp. 842-846.

100. SANCHEZ-ZAMORA, F. and LLAMAS-NISTAL, M. Visualizing tags as a network of relatedness. IEEE, pp. 1-6.

101. SAVOY, J. (1993) Stemming of French Words based on Grammatical Categories. *JASIS*, 44 (1), pp. 1-9.

102. SCOTT, S. and MATWIN, S. Feature Engineering for Text Classification. In: *Proceedings of the 16th Int. Conference on Machine Learning (ICML99)*, Bled, June 1999, pp. 379-388.

103. SEMANTIC WEB (n.d.) *Main Page* [WWW] Semantic Web,. Available from: http://semanticweb.org/wiki/Main_Page [Accessed 18/9/2014].

104. SEMBOK, T.M.T. and ATA, B.A. Arabic Word Stemming Algorithms and Retrieval Effectiveness. In: *Proceedings of the World Congress on Engineering (WCE 2013)*, London, July 2013.

105. SEMBOK, T.M.T. and BAKAR, Z.A. (2011) Effectiveness of Stemming and N-grams String Similarity Matching on Malay Documents. *International Journal of Applied Mathematics and Informatics*, 5 (3), pp. 208-215.

106. SHARMA, D. (2012) *Improved Stemming Approach used for Text Processing in Information Retrieval System.* MEng, Thapar University.

107. SMITH, P. (2012) *Professional Website Performance: Optimizing the Front-End and Back-End.* Indianapolis: Wiley.

108. STROPPA, N. (2006) *Algorithms & Complexity* [WWW] Dublin City University. Available from: http://www.computing.dcu.ie/~nstroppa/teaching/ca313_space.pdf [Accessed.

109. STUDY ADVICE AND MATHS SUPPORT (n.d.) *Developing your Literature Review* [WWW] University of Reading. Available from: http://www.reading.ac.uk/internal/studyadvice/StudyResources/Essays/sta-developinglitreview.aspx [Accessed 7/12/2014].

110. SUBA, K., BHATTACHARYYA, P., and JIANDANI, D. Hybrid Inflectional Stemmer and Rule-based Derivational Stemmer for Gujarati. In: *Proceedings of the 2nd Workshop on South and Southeast Asian Natural Language Processing (WSSANLP 2011)*, Chiang Mai, November 2011: Citeseer, pp. 1-8.

111. THE GLOBAL WORDNET ASSOCIATON (n.d.) *Wordnets in the World* [WWW]. Available from: http://globalwordnet.org/wordnets-in-the-world/ [Accessed 12/12/2014].

112. THE UNICODE CONSORTIUM (n.d.) *What is Unicode?* [WWW]. Available from: http://www.unicode.org/standard/WhatIsUnicode.html [Accessed 8/1/2013].

113. UDELL, J. (2004) *Collaborative Knowledge Gardening* [WWW] InfoWorld. Available from: http://www.infoworld.com/d/developer-world/collaborative-knowledge-gardening-020 [Accessed 30/12/2012].

114. VENNERS, B. (n.d.) *The Java Virtual Machine* [WWW] Artima. Available from: http://www.artima.com/insidejvm/ed2/jvm3.html [Accessed 10/5/2013].

115. W3C (n.d.) *Semantic Web* [WWW]. Available from: http://www.w3.org/standards/semanticweb/ [Accessed 8/7/2013].

116. WAL, T.V. (2005) *Explaining and Showing Broad and Narrow Folksonomies* [WWW] vanderwal.net. Available from: http://www.vanderwal.net/random/entrysel.php?blog=1635 [Accessed 20/12/2012].

117. WAL, T.V. (2007) *Folksonomy* [WWW] vanderwal.net. Available from: http://vanderwal.net/folksonomy.html [Accessed 20/12/2012].

118. WEISS, D. (2005) Stempelator: A Hybrid Stemmer for the Polish Language. *Institute of Computing Science: Poznan University of Technology Research Report*, RA-002/05.

119. WOOTTON, C. (2007) *Developing Quality Metadata: Building Innovative Tools and Workflow Solutions.* Oxford: Focal Press.

120. WORDNET (2006) *WordNet v3.0 Database Statistics* [WWW]. Available from: http://wordnet.princeton.edu/wordnet/man/wnstats.7WN.html [Accessed 19/9/2013].

121. YANG, M. and KIRCHHOFF, K. Phrase-Based Backoff Models for Machine Translation of Highly Inflected Languages. In: *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, Trento, April 2006, pp. 3-7.

122. ZAITSEV, P. and TKACHENKO, V. (2012) *High Performance MySQL: Optimization, Backups, and Replication.* Sebastopol: O'Reilly.

123. ZHANG, S., LEE, J.-H., and FANG, L. (2013) The Effect of Shopping Motivation of Social Commerce on Purchase Intention. *International Journal of Information Processing & Management*, 4 (6), pp. 137-149.