Visvesvaraya Technological University,

Jnana Sangama, Belgaum - 590014

A Project Report on

"Patent Comparison Using Similarity Graph"

Submitted in partial fulfillment of the requirements for the award of degree of

Computer Science & Engineering

Submitted by

Anirudh Agarwal	1PI13CS199
Rohan Agarwal	1PI13CS124

Under the guidance of

Dr. R. Srinath

Professor, CS Dept.

Jan – May 2017

Department of Computer Science & Engineering

PES Institute of Technology,
100FT RING ROAD, BSK 3RD STAGE,
BENGALURU – 560085

PES INSTITUTE OF TECHNOLOGY

(An Autonomous Institute under VTU, Belgaum)

100 Feet Ring Road, BSK- III Stage, Bangalore – 560 085

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

Certified that the eighth semester project work titled **"Patent Comparison Using Similarity Graph"** is a bonafide work carried out by

Anirudh Agarwal	**1PI13CS199**
Rohan Agarwal	**1PI13CS124**

in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belgaum during the academic semester January 2017 – May 2017. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Bachelor of Engineering.

Signature of the Guide	Signature of the HOD	Signature of the Principal,
Dr. R. Srinath	**Prof. Nitin V. Pujari**	**Dr. K S Sridhar**

External Viva

Name of Examiners	Signature with Date

Acknowledgment

Firstly, we would like to express our gratitude to all the people who have been associated with this project and have helped us with working on it.

We would like to extend our sincere thanks to Dr. M. R. Doreswamy, founder of PESIT, Prof. D. Jawahar, CEO of PESIT and Dr. K. S. Sridhar, our Principal, for the encouragement, motivation and continuous support they have provided to us.

We would also like to thank Prof. Nitin V. Pujari, Head of the Department CSE and Prof. V. R. Badri Prasad, Project Coordinator for their constant support and encouragement.

We would like to thank our guide for the project, Dr. R. Srinath, professor at department of Computer Science and Engineering for providing us an opportunity to work on a novel concept, and also for his guidance throughout the duration of this project. We are grateful to him for his continuous inputs and ideas regarding this project.

We would like to extend gratitude towards the panel members of the various reviews/discussions who provided us very thoughtful and constructive feedback.

We would like to thank our parents and other family members for their continuous support to us, without which, we would not have been able to achieve whatever we have.

We would also acknowledge all our friends who have directly or indirectly helped us throughout the duration of the project.

Abstract

Patent novelty assessment via prior art search is keyword based which does not account for the use of different terminology for the same concepts. The need of the hour is to bring semantic matching into patent comparison.

We propose to create a system for semantic comparison of patents based on a similarity graph created from ***Wordnet***. The graph would connect semantically related words and weight on the edges would signify degree of semantic similarity. The system would thus traverse through the graph to give an aggregate semantic score between words being compared. This would be used to create semantic vectors for sentence comparison. Document comparison is an aggregate of sentence similarity across documents. Thus, the proposed solution would detect novelty better than existing keyword based solutions.

Table of Contents

List of figures

List of tables

CHAPTER 01

INTRODUCTION

Consider a patent being written about vitality of ascorbic acid for human body. Suppose there exists a scientific document talking about the same concept but instead of using ascorbic acid terminology, the document is about vitamin C. It would be beneficial if prior art system could somehow account for semantic closeness between these documents in its search and retrieval. As per researchers at IBM & MIT, search is divided into two categories. 'Navigational Search', which is to narrow down to a specific document using keywords or patent numbers. It is quick and effective. The other is 'Research search' where user finds all relevant documents pertaining to a particular idea, which is very effective while working with patents. Keyword match based algorithms are used to perform navigational search, for research search, there is a requirement for semantic search.

The problem of semantic search and retrieval is critical in patent search systems as it can improve the quality of prior art search being carried, compared to keyword matching algorithms. It is effective in comparing documents which talk about same concept using different terminologies, documents that are short in length and documents which have only functional words in common, all of which are quite common scenarios in patent filing.

There are various drawbacks associated with keyword based search systems for patent and scientific document retrieval. Being terminology driven, search for exact matches within these documents can be quite futile as scientific terminologies change/evolve over time. Also, different industries have different terms for same concepts. Patenting being a competitive field, often people use alternate terminologies to obfuscate existing technology and present their idea to be 'state of the art'. Thus, considering above scenarios, using only keyword match for patent retrieval is both ineffective and risky.

There is a need for a system that saves semantic knowledge and uses it to identify the concept behind terminologies for relevant document retrieval. There are multiple ways to bring in the concept of semantics in retrieval. One is to include all possible phrases describing a concept and then proceed with keyword match. Quite evidently, the aforementioned method is exhaustive, inefficient and almost impossible in many cases. Some systems create a synonym list mapping and use it to perform keyword search. Such systems are restricted in their scope and accuracy.

We propose the idea of creating a similarity graph to store semantic knowledge of related terminologies. We first model entire wordnet into our data structures to create our custom wordnet, for the purpose of scalability and flexibility. Information from this custom wordnet is used to create different types of edges around each word, with weight of edge being a function of type of edge. The semantic closeness between two words is assessed via crawling from one word to another and creating a feature vector in process. This feature vector is then further converted to a metric value that represents the semantic score between terminologies.

The problem of semantic similarity is challenging as it involves understanding the context in which the word is being used and consider one of its multiple interpretations. This is known as 'Word Sense Disambiguation' and is readily performed by human mind but not so accurately by machines. One of the approaches to tackle it is using most likely interpretation of a word using corpus statistics but it would lead to loss of crucial data. We address the problem using probabilistic approach where each interpretation of the word is assigned certain probability based on its likeness derived from corpus statistics.

Semantic search finds application in many areas. For instance, if some research personnels want to see patent citations similar to an idea they are evaluating, they can clearly use semantic search. A recommendation system for scientific documents can be created, based on the same idea, which would suggest 'more like this' documents to user's current document. It can be used in Prior art searches, Invalidation research before patent filing and possible infringement detection. Beyond patents, the technology can also be used in dialogue systems, text classification and movie searches with a part of the plot being used as query.

1.1. Problem Definition

To facilitate comparison of relevant patent documents on semantic level. The semantic knowledge is leveraged from Wordnet based custom Similarity graph. It gives a semantic metric between terminologies which is clubbed with order similarity in sentences to give document collation score.

1.2. Generic proposed solution

We propose the use of a comprehensive lexical database for the purpose of finding terms in the vocabulary that are related to other terms in a given query. The current system makes use of Wordnet which is a large database of words in the english language along with their meanings and example use cases which are linked to each other primarily via the relation of synonimity. When a more domain centric approach is required, certain frequently occurring words in the domain can be leveraged, which modifies the database accordingly. Consequently a directed graph with edge weights is constructed using this database. The paths between any two words are retrieved and their edge weights are calculated, which gives an estimate of the semantic similarity of the words.

For the task of sentence similarity comparisons, the ordering of the words in the sentences are also considered apart from the value of semantic similarity score between the words.

The user interface/user design is such that 2 pieces of text can be entered by the user in the area specified or any 2 documents be uploaded to the system. The user then presses a button to semantically compare the two documents. The results of the comparison are seen on the same page with some other metrics such as total run time, number of paths found in graph etc shown as interactive graphs on another page.

The project must be accepted as complete when the system is stable and is consistently able to determine the semantic similarity scores of any two pieces of text that are provided as inputs to the system. The system must produce satisfactory results on certain training data, as verified against standard benchmarks.

1.3. Acknowledgement

Department of Computer Science & Engineering

PES Institute of Technology

(An Autonomous Institute under VTU, Belgaum)

100 Feet Ring Road, BSK- III Stage, Bangalore – 560 085

Project ID: PW010

Project Title: Patent Comparison Using Similarity Graph

Project Team:	1PI13CS199	Anirudh Agarwal
	1PI13CS124	Rohan Agarwal

This project report was submitted for review on 19th April, 2017. I acknowledge that the project team has implemented all recommended changes in the project report.

Guide signature with date:

Guide Name: Dr. R. Srinath

CHAPTER 02

LITERATURE SURVEY

A profusion of literature has been published on using semantics to enhance information retrieval. They can be broadly classified in supervised and unsupervised methodologies. The supervised methods are in need of a training set to make the system learn semantic relations, which is quite different from our approach. Also, term semantics varies from one domain to other. A model trained for a specific domain would break for any other domain.

The most common approach to include semantics into retrieval is query expansion and bag of words[2],[3],[9]. Words are expanded using various relations like synonyms, hypernyms, hyponyms etc. from knowledge bases such as Wordnet[2]. A certain weight is attached to each expanded term which is proportional to its distance from the main word in the knowledge base. This is quite effective when query length is small and a keyword matching algorithm needs to be implemented for retrieval. It also does not need any training data or model creation. The solution is exhaustive, inefficient and non-probabilistic. As a result, precision and recall are relatively low.

Vector space model after document expansion is another path[14]. Substantial documents rely on co-occurring words for vectors to be nearly oriented. For shorter queries though, when represented in high dimensional space form sparse vectors which is both inaccurate and computationally expensive.Various Natural language processing techniques can be used to analyse the subject/concept of document to retrieve relevant information[11],[12]. This approach is non-probabilistic and cannot be used to define a document metric. Also, it has high computational requirements.

Latent Semantic Analysis is a well known approach that has the advantage of not relying on any external knowledge source[13]. It tackles the issue of chronic amnesia in search(each search has to begun from start without any past context) and reduces vector dimensionality by deriving context of words from huge corpuses[7]. As per this concept, proximity of words and phrases are a measure of their conceptual similarity. Although quite popular, the solution does not scale and the quality of data obtained is inferior.

Wordnet[15][16] is a paramount instrument in today's semantic world. It is the backbone behind various substantial and upcoming technologies like artificial intelligence and automatic text analysis. It is in consistence with human semantic memory for contextual recognition and is different from normal thesaurus as it groups words into lexical concepts called synsets. The knowledge graph itself is modelled as a hierarchy of synsets, linked to each other via various semantic relationships. It has on online interface[17] which gives a feel of its operative and usage. Although, wordnet is one of the best tools to tackle word sense disambiguation, it has its inconsistencies. It lack etymology i.e. pronunciation of words and hence cannot be used in speech recognition. Also, it covers day to day english words and does not contain domain specific terminologies.

Our approach is based on creation of a semantic network using a knowledge base. The network would not only connect direct words but also indirect relationships as modelled by information derived from Wordnet[5],[8]. Each edge will have a weightage depending on its type. Similarity is assessed by traversing through this network, from one term to another, via edges, aggregating weights as a metric[4]. This metric can be used to semantically cluster similar documents[1], instead of relying on keyword based cosine similarity. This network can be further augmented using more than external knowledge bases like wikipedia[6]. Certain domain specific terminologies can be infused into the network via domain specific thesaurus.

Finding Semantic similarity between two documents is not sufficient as two documents having same terms in different order convey variant meanings. Word order represents structural information which can be compared by forming an order similarity vector.[10]
discusses the importance of functional words as how they form an integral part of a documents structural frame and unlike popular belief, should not be discarded. This order similarity is combined with semantic similarity obtained from Wordnet. It is different from our approach as it does not take into account indirect relationships between words that do not exist in Wordnet.

Machine learning models such as Long Short Term Memory(LSTM) have been used successfully to determine sentence and/or word pair similarity as demonstrated in [20]. One such work is described in []. The key to this approach is the ability to create vectors that imbibe the meaning that is emphasized in a sentence. Such a vector captures the meaning of a sentence irrespective of the use of vocabulary to make that sentence or the word order of the sentence. These vectors are input to a sophisticated LSTM network which learns parameters that are used further to get similarity values.

[21] discusses two stages to efficiently retrieve patent documents against a user provided query. The first stage is focussed on getting a good recall value for the given query. This stage uses the techniques of stopword deletion, allomorph expansion, and related term expansion. Stopword deletion refers to removing of all stopwords from the user query. Synonyms of words in the query are appended to the query to increase its length. This directly results in improved recall and is called related term expansion. The second stage involves running the query against only the top N results of the first stage. The claim structure of the patent document is considered with meticulous detail to give a score to each patent document against the query in this stage.

[22] discusses about the creation of a database of patent applications that have been filed with the Indian Patents Office. It emphasizes the need of such a database of patent applications, its benefits and advantages it could provide to researchers. It also highlights the use of such a database to know about the research topics that are being pursued across the country and formulate a pattern based on this data over a given period.

CHAPTER 03

SYSTEM REQUIREMENT SPECIFICATION

3.1. High level block diagram

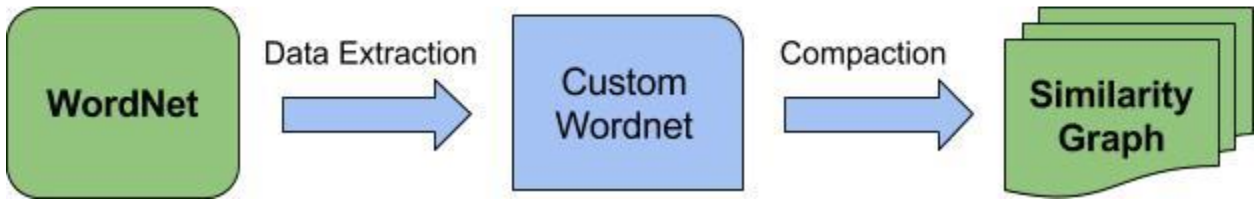


Fig. 1.1 Similarity Graph

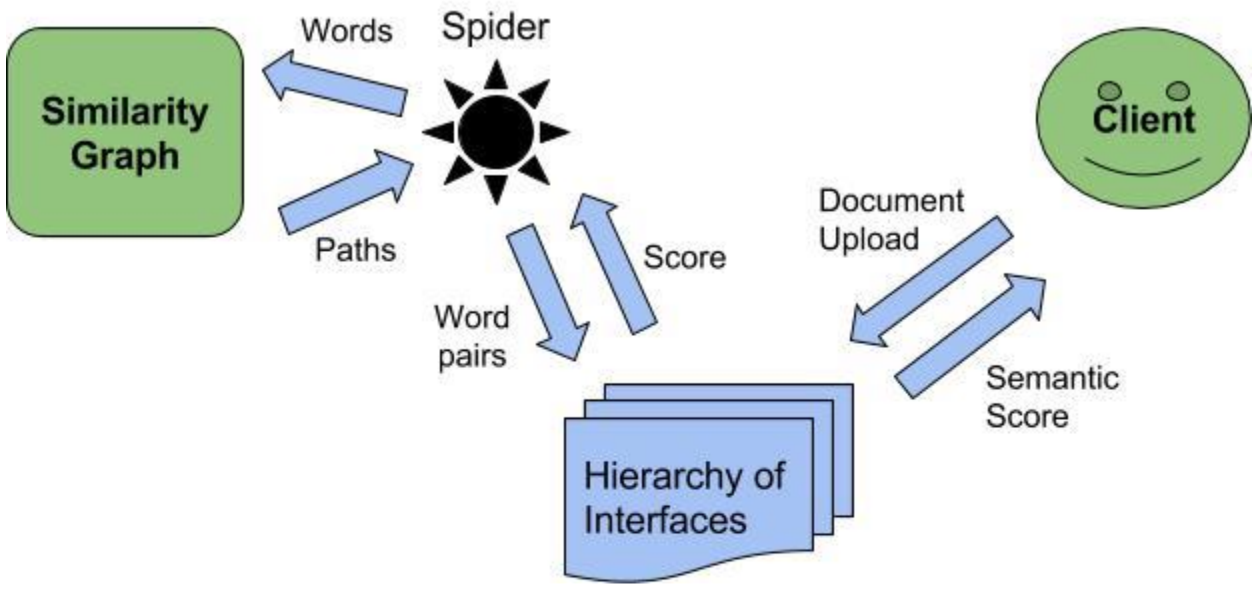


Fig. 1.2 Document Similarity Interface

1. The wordnet lexical database is modelled into a custom wordnet of our creation, for inclusion of domain specific terminologies later.
2. This custom wordnet is then used to initialize the nodes of the similarity graph and the relations between those nodes. The relations are represented by edges. Edge weights are assigned to the edges of the similarity graph using various heuristics.
3. A graphical user interface is required to specify the documents (that are to be compared) to the system for their processing.
4. These documents are compared to each other via hierarchy of intermediate interfaces.
5. document_client breaks the document into sentences and passes it to sentence_client.
6. sentence_client creates order vectors and semantic vectors for the pair of sentences it receives.
7. All the paths in the newly formed graph between the nodes that represent the documents to be compared are traversed and the edge weights are used to cumulatively determine the similarity score of the documents after all the relevant calculations are made.
8. Semantic score is obtained between pair of words through score aggregation during similarity graph traversal by spider as shown in Fig 1.2.
9. These scores obtained are sent back to sentence_client to form semantic vectors. Aggregate of semantic and order vectors serves as semantic similarity scores between pair of sentences, which is sent back to document_client.
10. The client interacts with the by uploading documents onto GUI and receives back an aggregate similarity score between documents along with paths amidst word pairs as identified by the system.

3.2. Environment used for the project

3.2.1. Hardware Required

The system essentially is based on processing and traversing a graph data structure with nodes and edges in the order of lacs. Thus sufficient computation speeds and random access memory are required as the entire graph structure is loaded onto RAM. The hardware for execution of the included source files must satisfy the following requirements -

1. 2.5 Ghz or more clock speed
2. Intel Core i5 chipset
3. 8gb RAM

3.2.2. Software Required

The software has been written in python programming language. The software for execution of the included source files must satisfy the following requirements -

1. Python version 2.7
2. NLTK Wordnet corpus
3. NLTK stopwords module
4. Apache lucene
5. Django version 1.7.11 or higher

3.2.3. Requirements for the project

3.2.3.1. Functional Requirements

F1 : Creation of a similarity graph. The system must be able to create a graph as described in the paragraphs above using the primary knowledge source of wordnet and certain rules defined for the edge weights.

F2 : Integration of documents into the similarity graph. The non noise words in the documents (to be compared) and the similarity graph must be consolidated into a larger graph structure without any loss of information in neither the similarity graph nor the documents.

F3 : Calculation of the similarity score of two or more documents. This requirement implies that the value of the similarity score of two documents must be determined via traversal of the impending edges in the graph.

F4 : Providing system capability to discern the novelty of provisional patent application input to the system under a particular class of patents that are awarded.

3.2.3.2. User Interface Requirements

U1 : The user interface is a web browser based and allows the user to input the documents that are to be compared. The input will be a text box where the user can type the text to be compared. Alternatively the user can load documents into the system by clicking on a button and specifying the path to the document. A button for initiating the process of comparison will be provided.

U2 : Another activity of the user interface would let users view the results of the comparison of the documents that were input to the system.

3.2.3.3. Non Functional Requirements

NF1 : Performance/Time complexity of the primary algorithm for calculating similarity values for documents must be under acceptable limits so as to be feasible.

NF2 : Scalability of the similarity graph. The similarity graph has to be designed in a way such that additional/secondary sources of knowledge(eg. Articles containing keywords that occur frequently in the subject matter etc) can be integrated into it with ease, without making any fundamental modifications to the original structure of the graph.

NF3 : Scalability of the system. The system has to be designed so that its multiple instances can run in order to accommodate more documents being input to the system for comparisons. Techniques such as multithreading have to be imbibed at a conceptual level.

NF4 : Reliability of the system. Every run of the algorithm must be in conformance to a minimum viable accuracy value as accepted beforehand.

3.2.4. Constraints

The biggest constraint for the project is its computational complexity. The solution would perform an exhaustive traversal of graph created from 118,000 word forms. There is no part of this chunk that can be discarded. As a result, the time complexity is going to be high. Also creation of such a convoluted graph would require high time and space. The algorithm would perform better perform on a machine with higher configuration. The Wordnet is an elaborate database, thus to leverage all possible relationships offered by it is not feasible given the limited timeline of the project. Thus, to put it into points :-

- ➔ Less computational power
- ➔ High Time and Space complexities of algorithm
- ➔ Insufficient time for In depth exploration of Wordnet knowledge base

3.2.5. Dependencies

Although the solution is proposed to be an unsupervised one, without the need of human intervention, the development has some major dependencies :-

- ➔ Wordnet - The entire project is centered around the information obtained from wordnet. We need to leverage advantageous relationships between word forms from this lexical database to build our semantic corpus.
- ➔ NLTK - It is a leading platform and is required for language processing of our patent corpus before they are semantically compared.
- ➔ Wikipedia - For ‘art’ centric terms that are not present in Wordnet, we would need external knowledge base that we will get by using wikipedia API.

3.2.6. Assumptions

Certain assumptions have been made regarding the requirements of the project and the expected use of those requirements towards the fulfillment of the project goals.

- ➔ The availability of the Wordnet database under the adequate licensing agreements for use by other third party software.
- ➔ No major loss in efficiency in results of the algorithm due to absence of any mechanism to process image data in patents.
- ➔ Availability of the required hardware, necessary for the resource intensive job of computation of semantic similarity scores of documents.
- ➔ A highly exhaustive wordnet library that satisfies the vocabulary needs of the documents input by the users, i.e. all non noise words, phrases, word forms in the document must be present in the wordnet database.

3.2.7. Use Case Diagram

The System namely has following use cases (i)Ranked Patent Retrieval (ii)Clustering Similar Patents (iii)Comparing congruent Patents. The mentioned had been represented via a use case diagram.

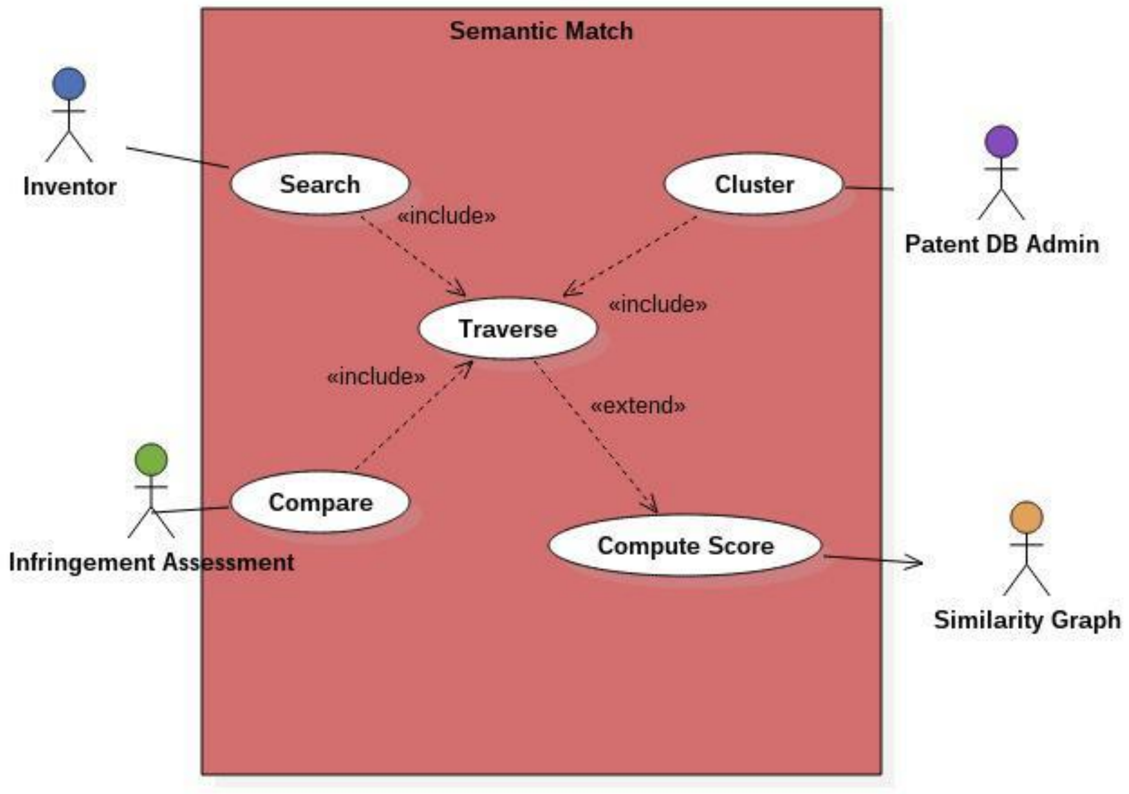
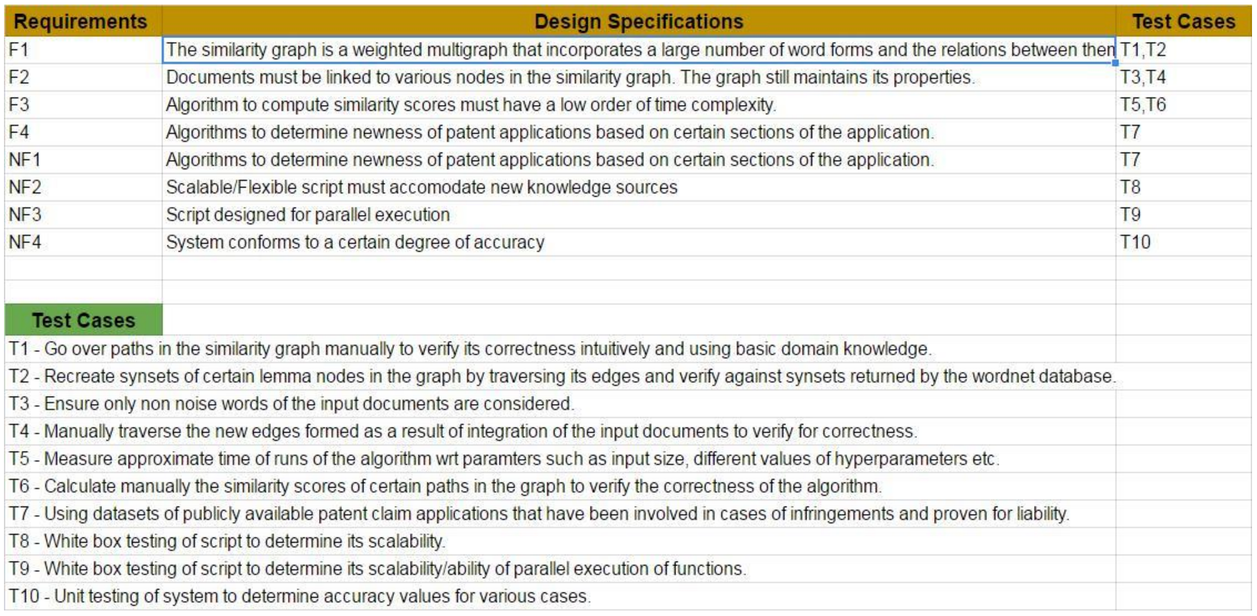


Fig.2.1 Use Case Diagram

3.2.8. Requirements Traceability Matrix

Updated RTM reflecting new functionality and corresponding test cases.



Requirements	Design Specifications	Test Cases
F1	The similarity graph is a weighted multigraph that incorporates a large number of word forms and the relations between then	T1,T2
F2	Documents must be linked to various nodes in the similarity graph. The graph still maintains its properties.	T3,T4
F3	Algorithm to compute similarity scores must have a low order of time complexity.	T5,T6
F4	Algorithms to determine newness of patent applications based on certain sections of the application.	T7
NF1	Algorithms to determine newness of patent applications based on certain sections of the application.	T7
NF2	Scalable/Flexible script must accomodate new knowledge sources	T8
NF3	Script designed for parallel execution	T9
NF4	System conforms to a certain degree of accuracy	T10

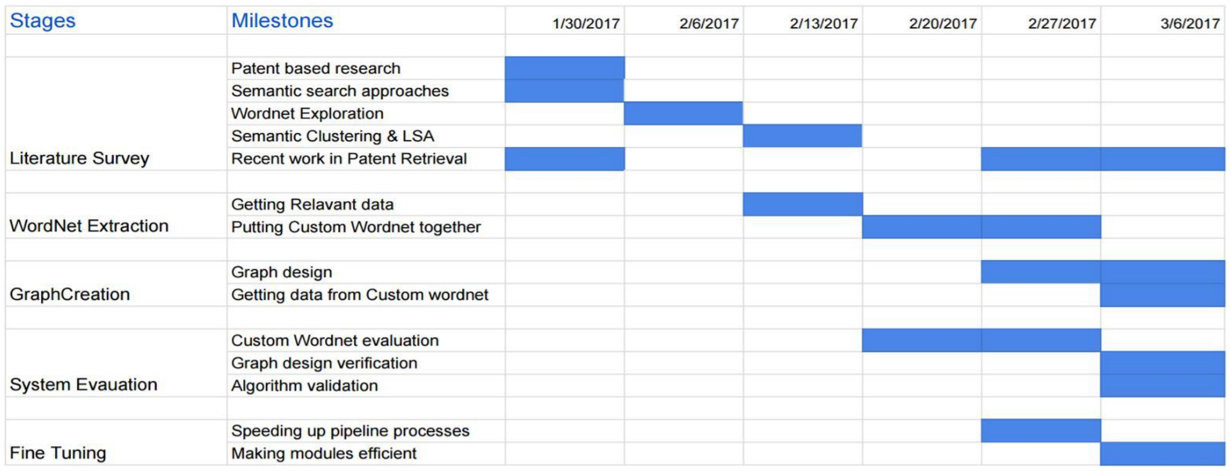
Test Cases
T1 - Go over paths in the similarity graph manually to verify its correctness intuitively and using basic domain knowledge.
T2 - Recreate synsets of certain lemma nodes in the graph by traversing its edges and verify against synsets returned by the wordnet database.
T3 - Ensure only non noise words of the input documents are considered.
T4 - Manually traverse the new edges formed as a result of integration of the input documents to verify for correctness.
T5 - Measure approximate time of runs of the algorithm wrt paramters such as input size, different values of hyperparameters etc.
T6 - Calculate manually the similarity scores of certain paths in the graph to verify the correctness of the algorithm.
T7 - Using datasets of publicly available patent claim applications that have been involved in cases of infringements and proven for liability.
T8 - White box testing of script to determine its scalability.
T9 - White box testing of script to determine its scalability/ability of parallel execution of functions.
T10 - Unit testing of system to determine accuracy values for various cases.

Fig.2.2 RTM

The Requirements Traceability Matrix has been maintained at this link. This matrix will be updated further in accordance with the progress of the project and the addition and/or modifications of the requirements of the project.

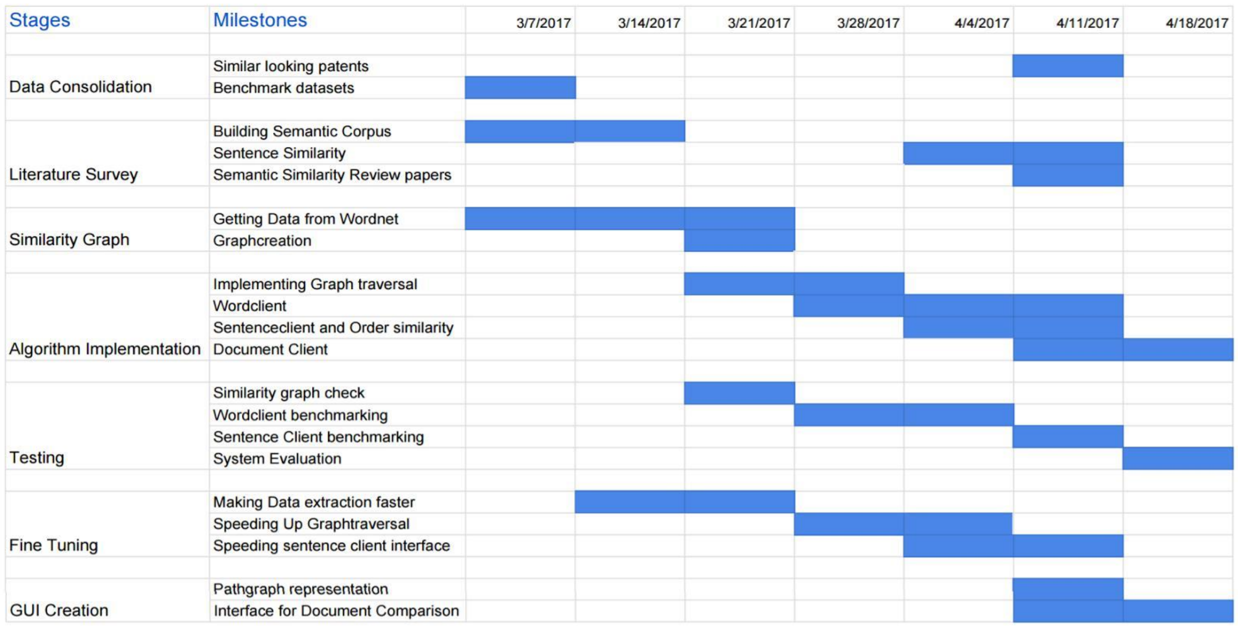
CHAPTER 04

SCHEDULE



Stages	Milestones	1/30/2017	2/6/2017	2/13/2017	2/20/2017	2/27/2017	3/6/2017
	Patent based research	🟦					
	Semantic search approaches	🟦					
	Wordnet Exploration		🟦				
	Semantic Clustering & LSA			🟦			
Literature Survey	Recent work in Patent Retrieval	🟦				🟦	🟦
	Getting Relavant data			🟦			
WordNet Extraction	Putting Custom Wordnet together				🟦	🟦	
	Graph design					🟦	🟦
GraphCreation	Getting data from Custom wordnet						🟦
	Custom Wordnet evaluation				🟦	🟦	
	Graph design verification						🟦
System Evauation	Algorithm validation						🟦
	Speeding up pipeline processes					🟦	
Fine Tuning	Making modules efficient						🟦

<u>Fig. 3.1 Schedule till march</u>



Stages	Milestones	3/7/2017	3/14/2017	3/21/2017	3/28/2017	4/4/2017	4/11/2017	4/18/2017
	Similar looking patents						🟦	
Data Consolidation	Benchmark datasets	🟦						
	Building Semantic Corpus	🟦	🟦					
	Sentence Similarity					🟦	🟦	
Literature Survey	Semantic Similarity Review papers						🟦	
	Getting Data from Wordnet	🟦	🟦	🟦				
Similarity Graph	Graphcreation			🟦				
	Implementing Graph traversal			🟦	🟦			
	Wordclient				🟦	🟦	🟦	
	Sentenceclient and Order similarity					🟦	🟦	
Algorithm Implementation	Document Client						🟦	🟦
	Similarity graph check			🟦				
	Wordclient benchmarking				🟦	🟦		
	Sentence Client benchmarking						🟦	
Testing	System Evaluation							🟦
	Making Data extraction faster		🟦	🟦				
	Speeding Up Graphtraversal				🟦	🟦		
Fine Tuning	Speeding sentence client interface					🟦	🟦	
	Pathgraph representation						🟦	
GUI Creation	Interface for Document Comparison						🟦	🟦

<u>Fig.3.2 Schedule till may</u>

CHAPTER 05

SYSTEM DESIGN

5.1. Architectural Diagram

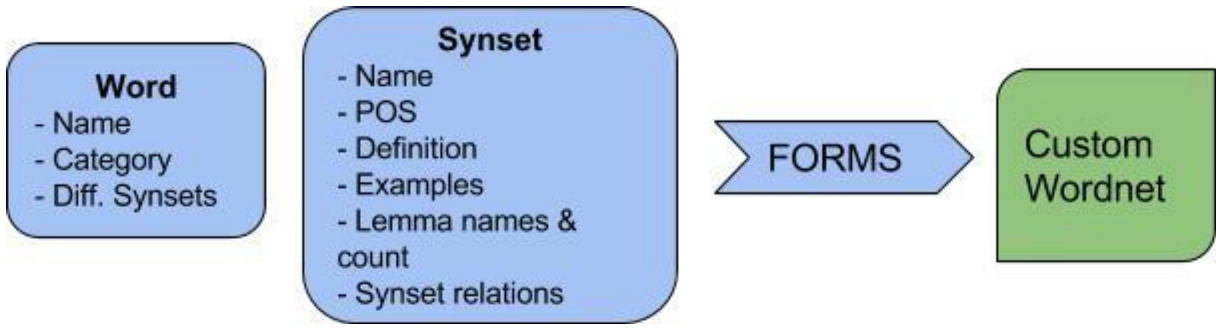


Fig. 4.1 Custom wordnet structure

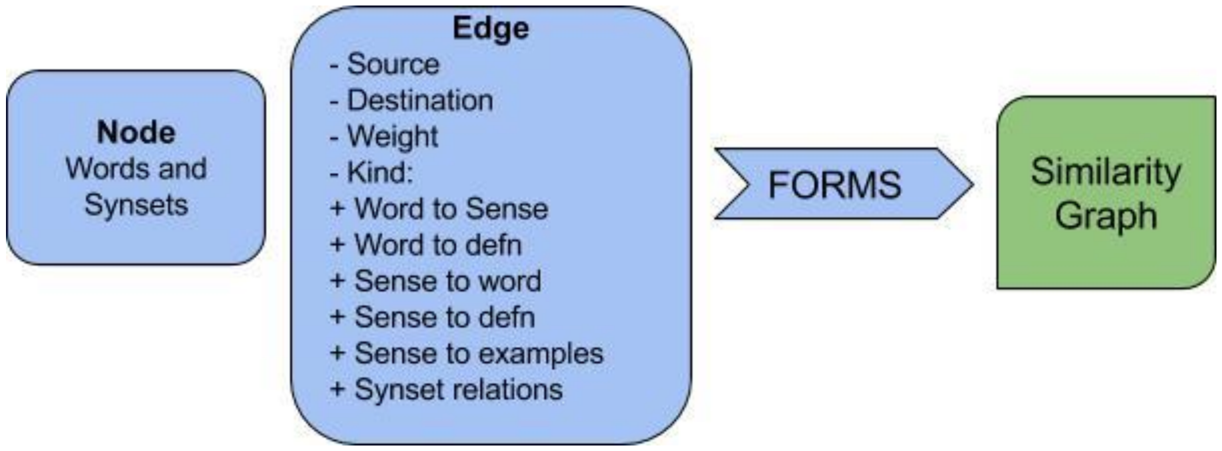


Fig. 4.2 Similarity graph structure

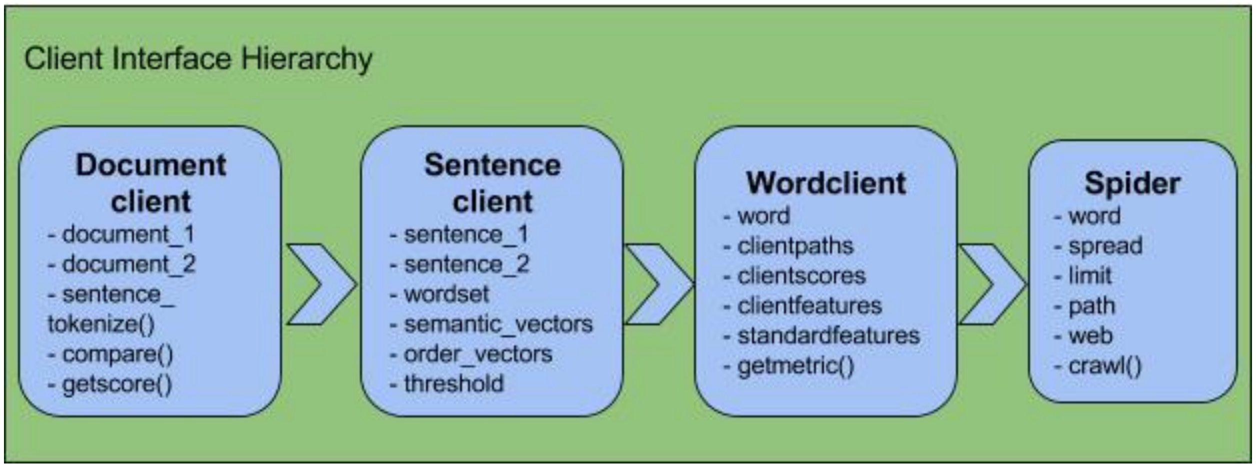


Fig. 4.3 Client Interface hierarchy

1. Word class has category attribute for easy addition and removal of different categories of words from system.
2. A word has four types of synsets, Noun, Verb, Adjective and Adverb, each having pos as ‘n’,’v’,’a’ & ‘r’ respectively.
3. Synsets name signifies unique key used to represent the semantic concept in wordnet.
4. Lemma names include all lemmas/words that point to the same lexical perception and lemma count represents their likelihood of occurrence when is a sentence when synset is being used in that particular context.
5. These classes together constitute the custom wordnet.
6. Node represents either word or synset. Edge class has a source and destination for easy traversal and delinking functionalities.
7. It also has weight which is calculated on the basis of kind of edge. Different kinds have different functions for calculation of weight using various hyperparameters. They are delegated when an edge instance is created.
8. Thus, similarity graph comprises of nodes and edges.
9. In client hierarchy, spider has word to search in graph, spread & limit to control its crawling and web to store various nodes and paths in the vicinity of word.

10. It does not perform normal dfs as even visited nodes can have more than one paths to them and we try to find all possible non-cyclic paths to a destination.
11. word_client interface has 'clientpaths' and 'clientscores' members to store paths obtained to reach from word to client word and scores of those paths.
12. Word member in word_client interface is the node around which the spider revolves.
13. 'clientfeatures' member is used to store feature vector of semantic similarity from word to client word, 'standardfeatures' member stores feature vector of a word to itself.
14. getmetric() is used to obtain score between standard feature vector and client feature vector. It gives measure of how semantically close words are as per our system.
15. In sentence_client, both sentences are stored after symbol and stopword removal. Morphological parsing and bigram assimilation.
16. Semantic_vectors and order_vectors store both types vectors representing the sentences. These vectors are aggregated to similarity scores.
17. Document_client compares sentences pairwise and obtains similarity score in the process.
18. GUI is made using django and thus follows MVC architectural pattern.
19. A commons file is maintained across the system containing commonly used functionalities across various modules as shown in Fig 3.4.

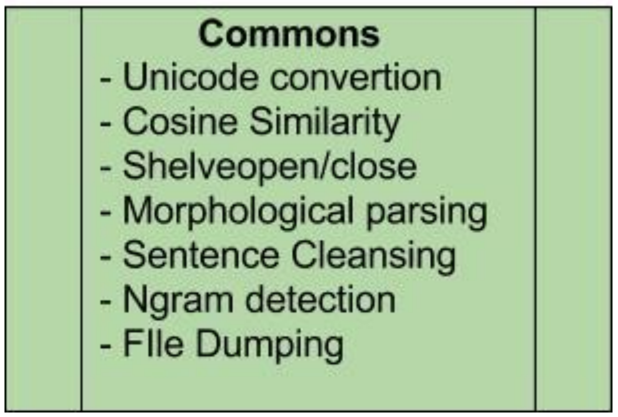


Fig. 4.4 System commonalities

5.2. User Interface Design Flow

The UI has been designed as a web app with a minimalistic approach in order to keep the overall usability intuitive to the end user. The home page of the app itself allows the user to upload files or write/copy-paste text into text boxes. A button at the bottom of the input boxes can be clicked to fire up the semantic comparison between the pieces of text.

The results of the computation are shown on the same page. The ‘results’ section displays the text entered by the user with words highlighted that have a semantic score value above a particular threshold between them. These words can be clicked which invokes a graph displaying the paths traversed from the said word to other matching words in the paired text.

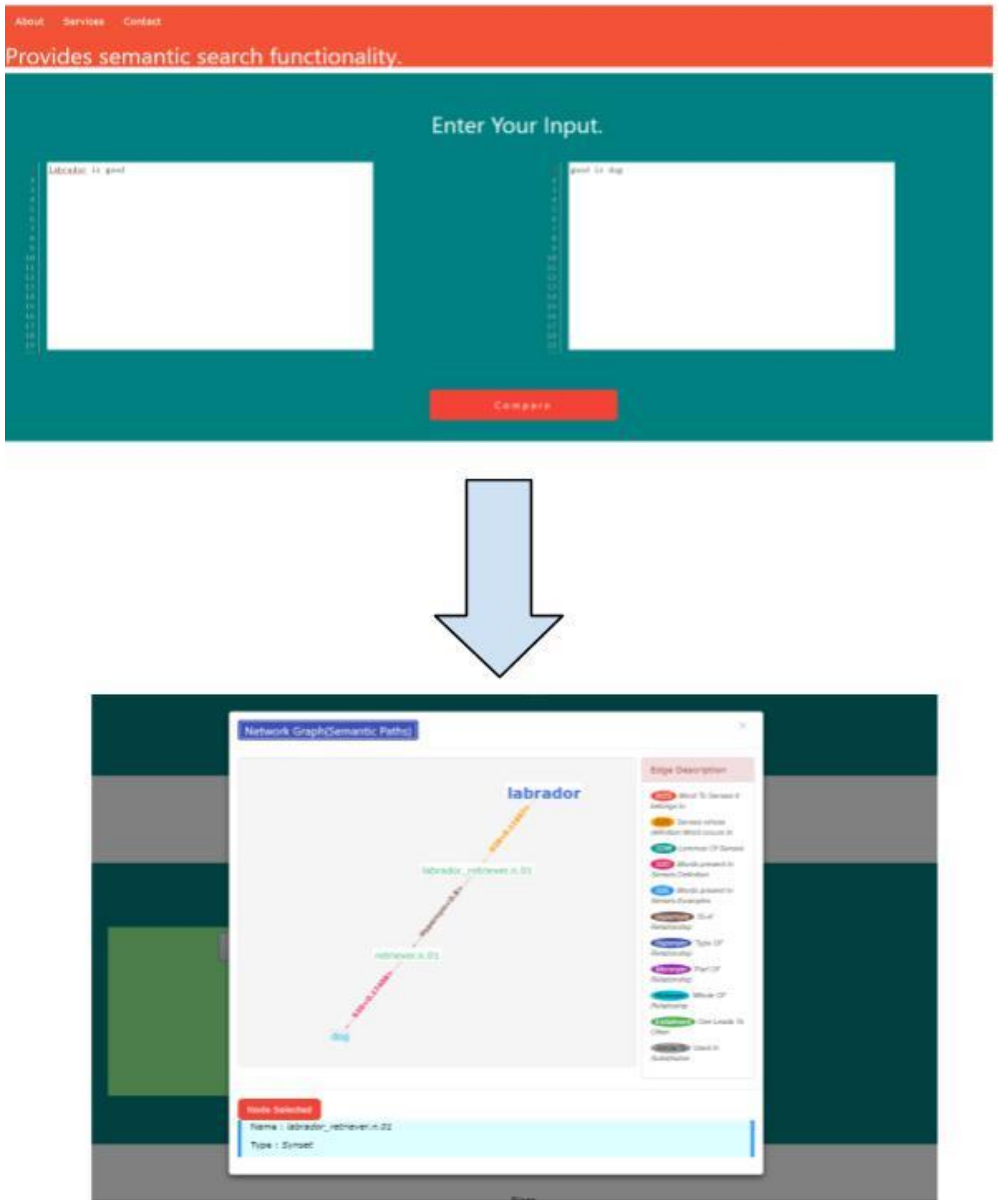
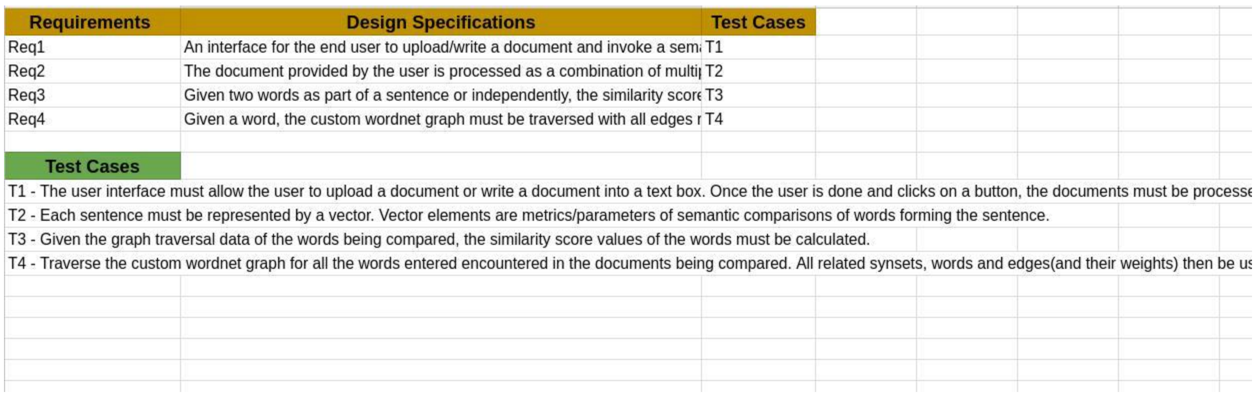


Fig.5.1 User Interface Design

5.3. Updated RTM

Updated RTM reflecting new functionality and corresponding test cases.



Requirements	Design Specifications	Test Cases
Req1	An interface for the end user to upload/write a document and invoke a sem	T1
Req2	The document provided by the user is processed as a combination of multi	T2
Req3	Given two words as part of a sentence or independently, the similarity score	T3
Req4	Given a word, the custom wordnet graph must be traversed with all edges r	T4

Test Cases
T1 - The user interface must allow the user to upload a document or write a document into a text box. Once the user is done and clicks on a button, the documents must be processe
T2 - Each sentence must be represented by a vector. Vector elements are metrics/parameters of semantic comparisons of words forming the sentence.
T3 - Given the graph traversal data of the words being compared, the similarity score values of the words must be calculated.
T4 - Traverse the custom wordnet graph for all the words entered encountered in the documents being compared. All related synsets, words and edges(and their weights) then be us

Fig.6.1 Updated RTM

CHAPTER 06

DESIGN

6.1. Data Flow Diagram

6.1.1. Similarity Graph Creation

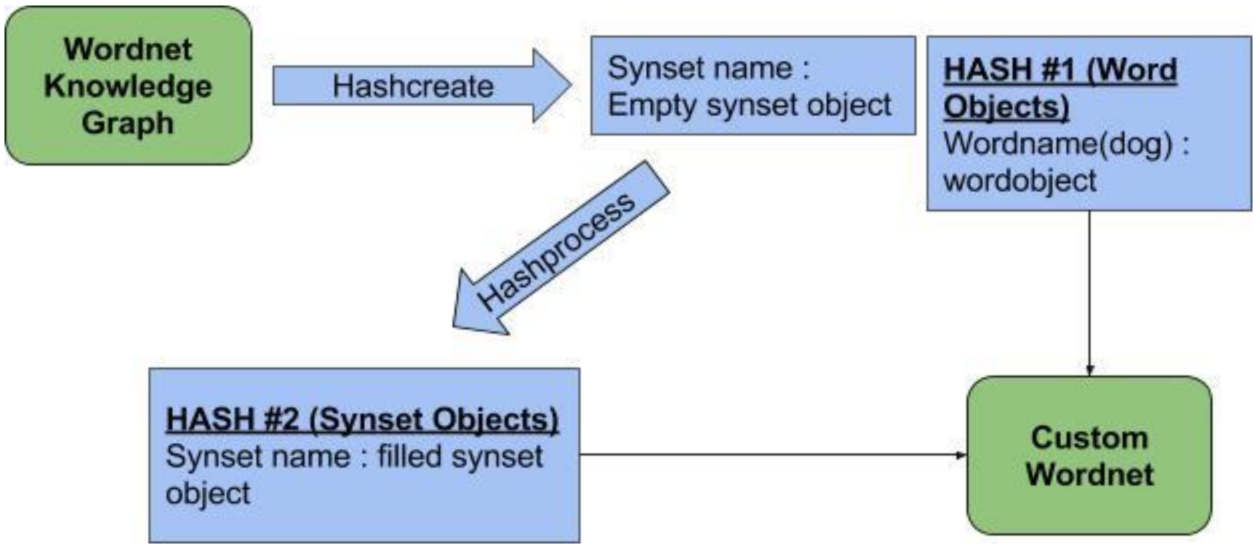


Fig. 7.1 Creating custom wordnet

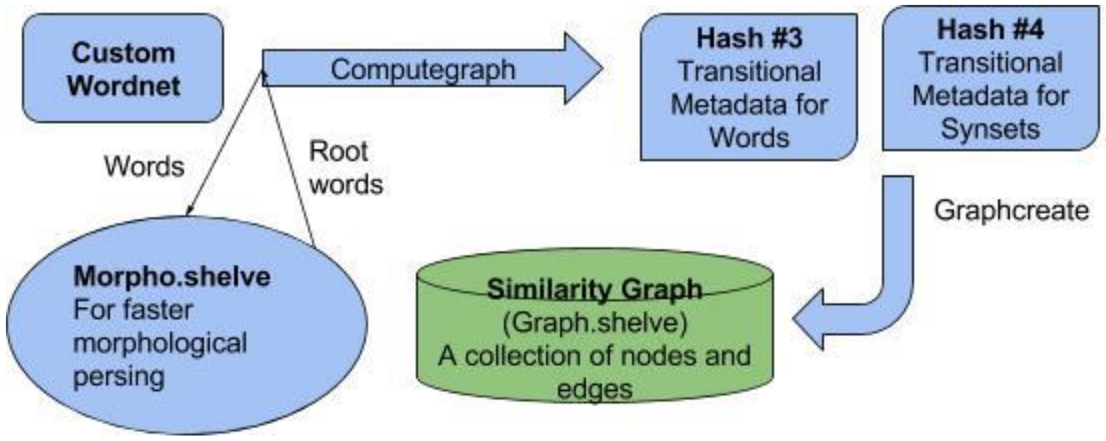

Custom Wordnet
Computegraph
Hash #3 Transitional Metadata for Words
Hash #4 Transitional Metadata for Synsets
Words
Root words
Graphcreate
Morpho.shelve For faster morphological persing
Similarity Graph (Graph.shelve) A collection of nodes and edges

Fig. 7.2 Similarity Graph

6.1.1.1. Modelling Custom Wordnet

Wordnet is a lexical database containing most of the commonly used english language terms and phrases.

It contains group of words having the same meaning clustered together into synsets, each expressing a unique concept. These synsets are linked to other synsets through semantics relations and lexical concepts. We model useful relations from wordnet into our custom wordnet, relations that help find semantic similarity between terms. The idea is to add words like scientific terminologies, thus scale system to include non-english domain specific terms which may not be present in wordnet.

Word class models words from Wordnet whereas Synset class models synsets. We take all data from wordnet that is required to assess semantic closeness between terms. The properties for each class are shown in Fig 3.1. 'Hashcreate' module creates two shelves, one filled with our word class instances, having data of word instances of wordnet, the other is for synset instances having wordnet synset data but without semantic relations. 'Hashprocess' module parses through this empty skeleton of synset instances in Hash#2 and fills it with appropriate semantic synset relations. The reason being our synset instances should have semantic relations with our synset instances itself, not with wordnet synset instances, and so all synset instances need to be first created and then populated with each other. These two hashes together, Hash#1 and Hash#2 forms out custom wordnet. We use nltk[23] interface in python to access wordnet.

6.1.1.2. Similarity Graph Creation

We then get various relationship data to create edges in similarity graph. From words we take, (i)Synsets they are part of (ii)Synsets whose definition and examples they are used in. From synsets (i)Lemmas (ii)Words used in their definition (iii)Words used in their examples (iv)Hypernyms [words with broader meanings] (v)Hyponyms [words whose meanings are a part of hypernyms] (vi)Meronyms [part of something bigger] (vii)Holonyms [bigger things meronyms constitute] (viii)Entailments [if doing X one must be doing Y, eg: sleep and snore] (ix)Similar_tos

Thus, ‘Computegraph’ module uses custom wordnet to create Hash#3 and Hash#4, the former storing transitional data for word edge creation and the latter storing data for synset edge creation. ‘Graphcreate’ module takes transitional metadata and creates Node and edges. Weight of edge is computed inside edge class by passing type being created. Graph.shelve is a dictionary with nodes as keys and edges as its values thus, serving as our Similarity Graph.

6.1.2. Document Similarity Interface

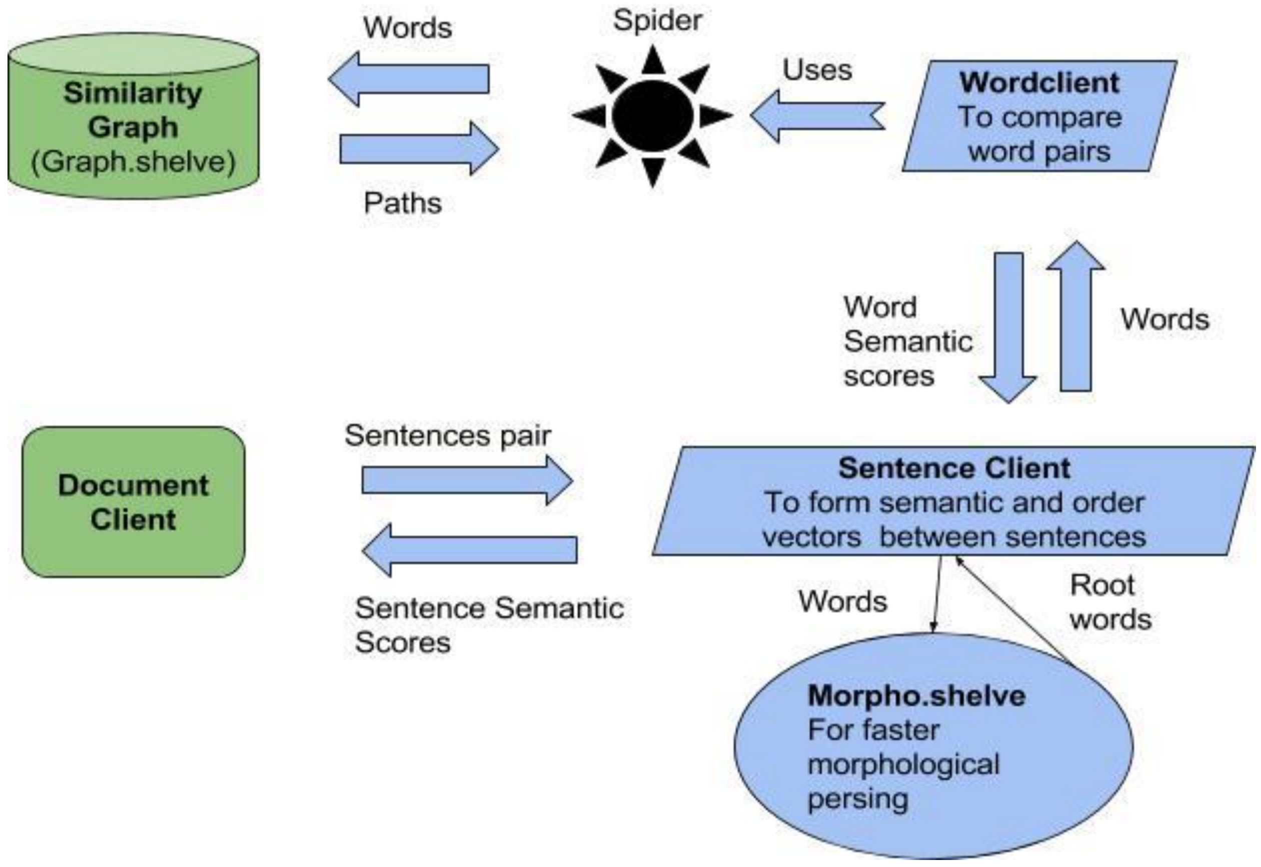

Similarity Graph (Graph.shelve)
Words
Paths
Spider
Uses
Wordclient To compare word pairs
Word Semantic scores
Words
Document Client
Sentences pair
Sentence Semantic Scores
Sentence Client To form semantic and order vectors between sentences
Words
Root words
Morpho.shelve For faster morphological persing

Fig. 7.3 Client interfaces hierarchy

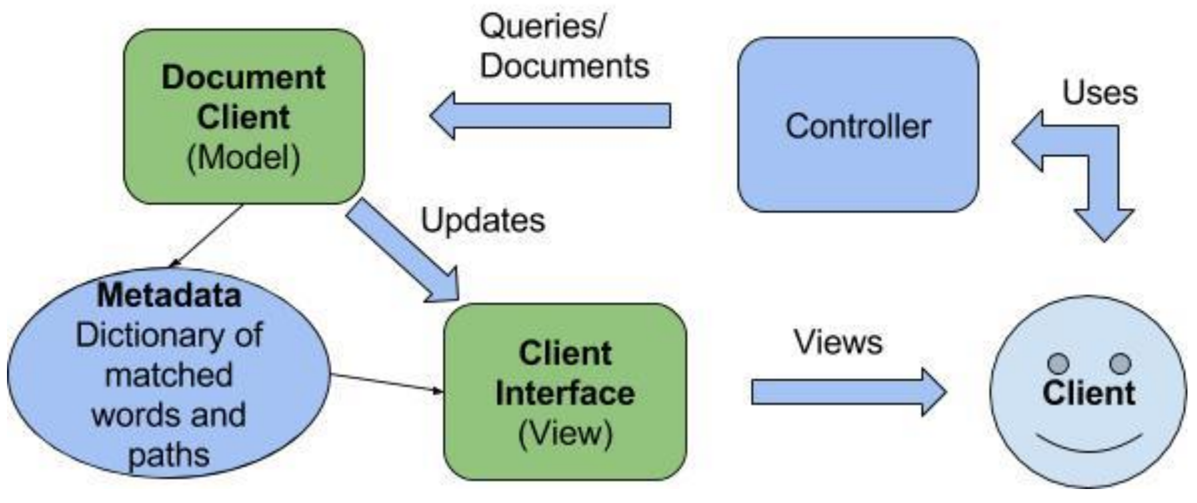


Fig. 7.4 Document Comparison Interface

6.1.2.1. Client Interface Hierarchy

6.1.2.1.1. Word_Client

Ideally it takes around 45 to 50 minutes to form complete graph. Word_client is used to compare pair of words using our similarity graph and give back semantic score obtained. It is the lower most interface in our client hierarchy as words are the smallest non-breakable units in a document. Word_client uses a Spider class to crawl through Similarity graph and obtain all possible paths between pair of words. The depth of crawling can be controlled by (i)Spider spread which signifies maximum path length (ii)Spider limit which signifies minimum path score. Thus, all paths obtained are bound by these two limits.

In Order to crawl faster from word1 to word2, the spider obtains a spread of nodes around word1. It then finds common nodes between web of word1 and outgoing edges of word2. Since all outgoing edges of word2 also have a back edge to word2, these common points are used to obtain complete paths from word1 to word2, by crawling one spread length lesser from word1. This decreases spider crawling time which is the prime computational bottleneck in the entire system.

These paths obtained from spider are used to create feature vectors which signify similarity obtained from similarity graph. The features are (i)Total no of paths (ii)Maximum path score (iii)Mean score (iv)Total score of all paths. The cosine similarity between feature vector from word1 to word2 and word1 to word1 gives semantic similarity score of word1 towards word2. It is because for our system, feature vector from a word to itself signifies similarity limit = 1 and so vectors obtained for other words are correlated to it. Since the graph is directional, semantic score from word1 to word2 is different from score from word2 to word1.

6.1.2.1.2. Sentence_Client

Sentences are short documents and hence their similarity metric is different from both big documents and words. We take two metrics for sentence similarity (i)Semantic meaning of sentence (ii)Order of sentence. Before performing any metric calculations, the sentences are checked for stop words, special words. Morphological parsing is performed on derived words to find more significant semantic paths from similarity graph. To decipher the semantic meaning of sentences(S1, S2) we first form a Wordset(W). This wordset contains all non-repeating words of S1+S2 eg: S1-"RAM is powerful" S2-"RAM are cheap". Then wordset(W) - "RAM is powerful are cheap".

This wordset is then used to obtain semantic vector for each sentence by using the following methodology. Each word in wordset is represented as one dimension, if it is present in S1, then that dimension is given value 1, if it is not present in S1, then word is found in S1 which is semantically closest to the dimension word. If the closest word's semantic score is below a threshold value, then that dimension is assigned value 0, else it is given the semantic score as its value. In Order to obtain order vector, similar wordset(W) is formed from sentences(S1,S2) and following methodology is followed. Each word in wordset again signifies a dimension. For Order vector of S1, If the dimension word is present in S1, then it is assigned the index of word in wordset, if not then closes word is found in S1. If semantic score of the closest word in S1 is beyond a threshold value, then the index of the closest word in S1 is found in 'W' and is assigned to the dimension else value assigned = 0.

Thus, after obtaining semantic vectors of Sentences(S1,S2) as s1,s2 we obtain semantic score between them using cosine similarity : $Ss = (s1.s2) / (\|s1\| * \|s2\|)$. For Order vectors o1, o2, we obtain an order similarity using normalization : $So = 1 - (\| o1 - o2\|) / (\| o1 + o2\|)$. These two scores are combined together to give a complete sentence semantic metric as :
$Score = \alpha*Ss + (1 - \alpha)*So$, where $\alpha \varepsilon (0.5,1]$.

6.1.2.1.3. Document_client

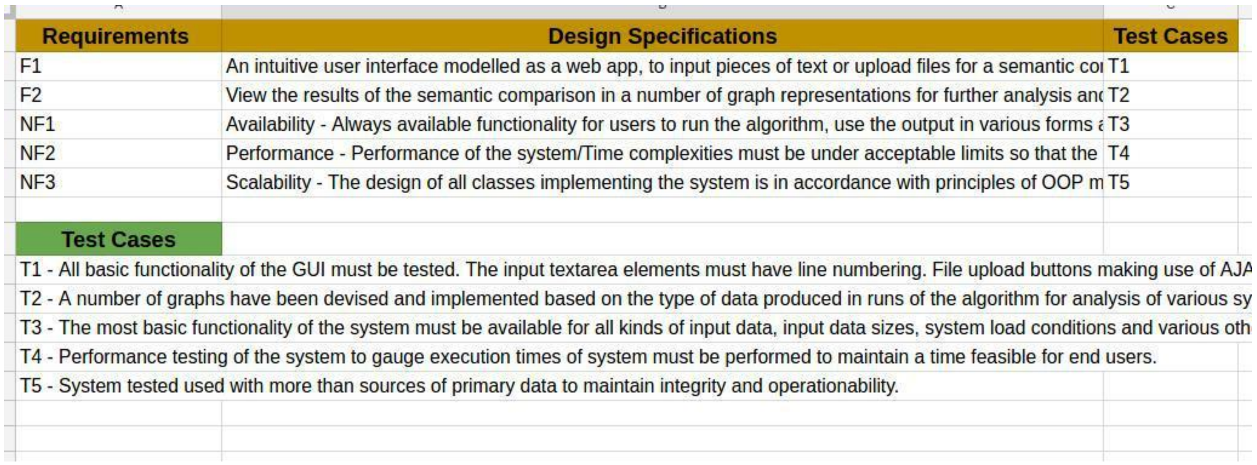
This is the interface exposed to client and is at the top of client hierarchy. It receives documents from users, fragments them into sentences and then delegates sentence client to find most semantically similar sentence in the other document. If an exactly similar sentence is found, then one is added to score and comparison for that sentence is stopped, else highest semantic score for that sentence is added. Total score is divided by no of sentences in the smaller document to get aggregate document score. The intent is to find if the smaller document is a constituent of the bigger document. A network graph shows semantically matched words from one document to another, if there are no semantic matches then this graph remains empty.

6.1.2.1.4. Graphic User Interface for Document

For User interface we use django, for the simple reason that it is a python based platform and our entire codebase is in python. Django follows mvc architectural pattern, so screen exposed to client are views. Client has the flexibility of either typing the documents being compared or uploading from local file system. When similarity assessment begins, client uses controllers to call document_client interface internally, passing client documents as input parameters. The document_client delegates work across the internal client hierarchy and gives back cumulative semantic scores. It also returns paths identified over similarity graph. Thus, views gets updated to show document score along with highlighted words that are semantically similar across documents. On clicking on these words, user again invokes another controller to show word trees representing paths obtained from our similarity graph. The system also has interface for user to inject contemporary words that are not present in wordnet. For this, user has to fill a form pertaining to various details associated with a word in order to find its contextual standing in our graph. After successful submission, the word becomes part of the graph and is now recognized by the system. Different categories of words can be dynamically infused and removed from the system graph.

6.2. Updated RTM

Updated RTM reflecting new functionality and corresponding test cases.



Requirements	Design Specifications	Test Cases
F1	An intuitive user interface modelled as a web app, to input pieces of text or upload files for a semantic coı	T1
F2	View the results of the semantic comparison in a number of graph representations for further analysis anc	T2
NF1	Availability - Always available functionality for users to run the algorithm, use the output in various forms a	T3
NF2	Performance - Performance of the system/Time complexities must be under acceptable limits so that the	T4
NF3	Scalability - The design of all classes implementing the system is in accordance with principles of OOP m	T5

Test Cases
T1 - All basic functionality of the GUI must be tested. The input textarea elements must have line numbering. File upload buttons making use of AJA
T2 - A number of graphs have been devised and implemented based on the type of data produced in runs of the algorithm for analysis of various sy
T3 - The most basic functionality of the system must be available for all kinds of input data, input data sizes, system load conditions and various oth
T4 - Performance testing of the system to gauge execution times of system must be performed to maintain a time feasible for end users.
T5 - System tested used with more than sources of primary data to maintain integrity and operationability.

Fig. 7.5 Updated RTM

CHAPTER 07

IMPLEMENTATION

7.1. Pseudo code/Algorithms

7.1.1. Custom Wordnet Creation

```
worditerator ← wordnet
for word in worditerator:
        custom_word = Wordfactory(word) //Extracts properties and gives back populated
instance
        custom_word -> Hash1
        store synsets of word
        for synset in synsets:
                custom_synset = Synsetfactory(synset)
                custom_synset -> Hash2 store
Factory(concept):
        // Fill various properties
        if concept.property:
                custom_instance.populate(concept.property)
        return custom_instance
populate(prop):
        // Different functions for diff
        property if prop = prop1:
                prop1(prop)
        if prop = prop2:
                prop2(prop)
        Else: unknown property
```

7.1.2. Graph Data Extraction

```
Synsetfactory(synset)
```

```
    data[prop1] ← synset.prop1
    data[examples] ← parsed, refined, bigramed list
    data[words] ← synset.lemmas


Wordfactory(data, value)
    for words in data[words] if
        word in worddata
            worddata[data] ← append value
        else
            worddata[data] entry created
```

7.1.3. Similarity Graph Traversal (Spider)

```
Crawl(word)
    graph ← OpenGraph
    edges ← graph[rootword]
    //Traverse through each edge one by
    one for edge in edges:
        if edge.weight != 0: # Prevents division by zero error
            current_score *= edge.weight
            current_depth += 1
            if current_depth <= pread and current_score >=
                limit: current_path ← append(edge)
                DFS(edge.dest)
                pop ← current_path.pop()
            current_depth -= 1
            current_score /= edge.weight
        Else: pass
DFS(node)
    if node in visited_nodes:
        # Check if path is a cycle of existing path
        paths ← Currentpaths(node)
        if path a subset of existing paths of node:
            # It is a cycle
            return

        # Check  if  word  entry  for  first  time  in
        web if node not in web:
            web[node] = []

        # To prevent from object getting copied
        ls ← deepcopy Currentpaths(node)
        word_web ← append(ls)
        if node in graph:
            edges ← graph[node]
            for edge in edges:
                if edge.weight != 0: # Prevent division by zero
                    error current_score *= edge.weight
                    current_depth += 1
                    if current_depth <= spread and scurrent_score >= limit:
// To Bound crawling
                        current_path ← append(edge)
                        if node not in visited:
                            # To prevent double entry in
                            visited visited ← append(node)
                        DFS(edge.dest)
                        current_.path.pop()
                    # After recursion, coming back
                    current_depth -= 1
                    current_score /= edge.weight
                Else: return
```

```
          Else: Node not in Graph only
```

7.1.4. Handling Inflectional Morphology

```
Morphoparse(word)
          root = Lemmatizer(word)
          if word != root:
                    return root
          else lemmatizer could not recognize the root of word
                    synsets ← wordnet(word)
                    lemmas ← synsets
                    root ← closest_word(lemmas, word)
                    return root
```

7.1.5. Sentence Pair Semantic Feature Extraction

```
Vectorcalc(sentence1, sentence2)
          for word in wordset of pair of
                    sentence: if word in sentence:
                              semantic vector ← 1
                              order vector ← index
                    else:
                              instance ← Wordclient(word)
                              for token in sentence:
                                        score ← similarity between token and word
                                        allscores ← append(score)
                                        allpaths ← append(paths between word and token)
                              maxscore, maxindex ← allscores
                              if macscore > threshold:
                                        proper_index ← Index of token with highest score in
                                        Wordset semantic vector ← maxscore
                                        order vector ← proper_index
```

```
        else:
            semantic vector ← 0
            order vector ← 0
```

7.1.6. Word pair Semantic Feature Extraction

```
Calcmetric(client_word)
    edges ← graph[client_word]
    for edge in edges:
        dests ← append(edge.dest)
    for node in web of word:
        if node in dests:
            common_points ← append(node)
    # Completing half path to client_word
    extrapath = {}
    for node in common_points:
        edges ← graph[node]
        for edge in edges:
            backedge of common node to client_word
            extraedge[common_point] ← map(backedge)
    for node in common_points:
        paths ← paths to common
        points for path in paths:
            path ← append(extraedge[node])
```

7.2. Codebase Structure

The project directory is named “SS_Graph”. It contains the following files and folders -

Files -

1. Commons.py - contains a number of hyperparameters used in calculation for edge weights. Also has definitions of various commonly used functions.
2. Computegraph.py - module implementing code to create associations such as ‘Sense to Definition(S2D)’, ‘Definition word to sense(D2S)’, ‘Sense to Example words(S2E)’ etc. and maintained in various hash data structures on the disk.
3. Documentclient.py - module implementing the process of comparing 2 documents, decomposing them into sentences and collating the overall score.
4. Edge.py - module implementing the class structure for the edges in the graph.
5. Graphcreate.py - module implementing code for the calculation of edges for a particular node in the graph.
6. Hashcreate.py - module implementing code that populates the synset classes with relevant data.
7. Hashprocess.py - module implementing code to generate word and synset classes and populate the word classes
8. Hypers.py - contains data for a number of hyperparameters used for calculations.
9. Parser.py - module implementing code to parse patent documents into various sections/subsections as required by the user.
10. README.md - readme file for the software.
11. Sentenceclient.py - module implementing code that calculates similarity scores at a sentence level; provides an interface for calculation of semantic similarity of sentences.
12. Spider.py - module implementing code that crawls the generated graph based on the source and destination words.
13. Synset.py - module implementing the synset class.
14. Word.py - module implementing the word class.

15. Wordclient.py - module implementing code that calculates semantic similarity scores at a word level.
16. Wordnet.py - module implementing code to retrieve data from the wordnet database.

Folders -

1. ssgraph - contains module for the implementation of the front end of the software system.
2. Shelves - contains intermediate files generated by the various system modules at run time, stored as persistent shelve data structures.

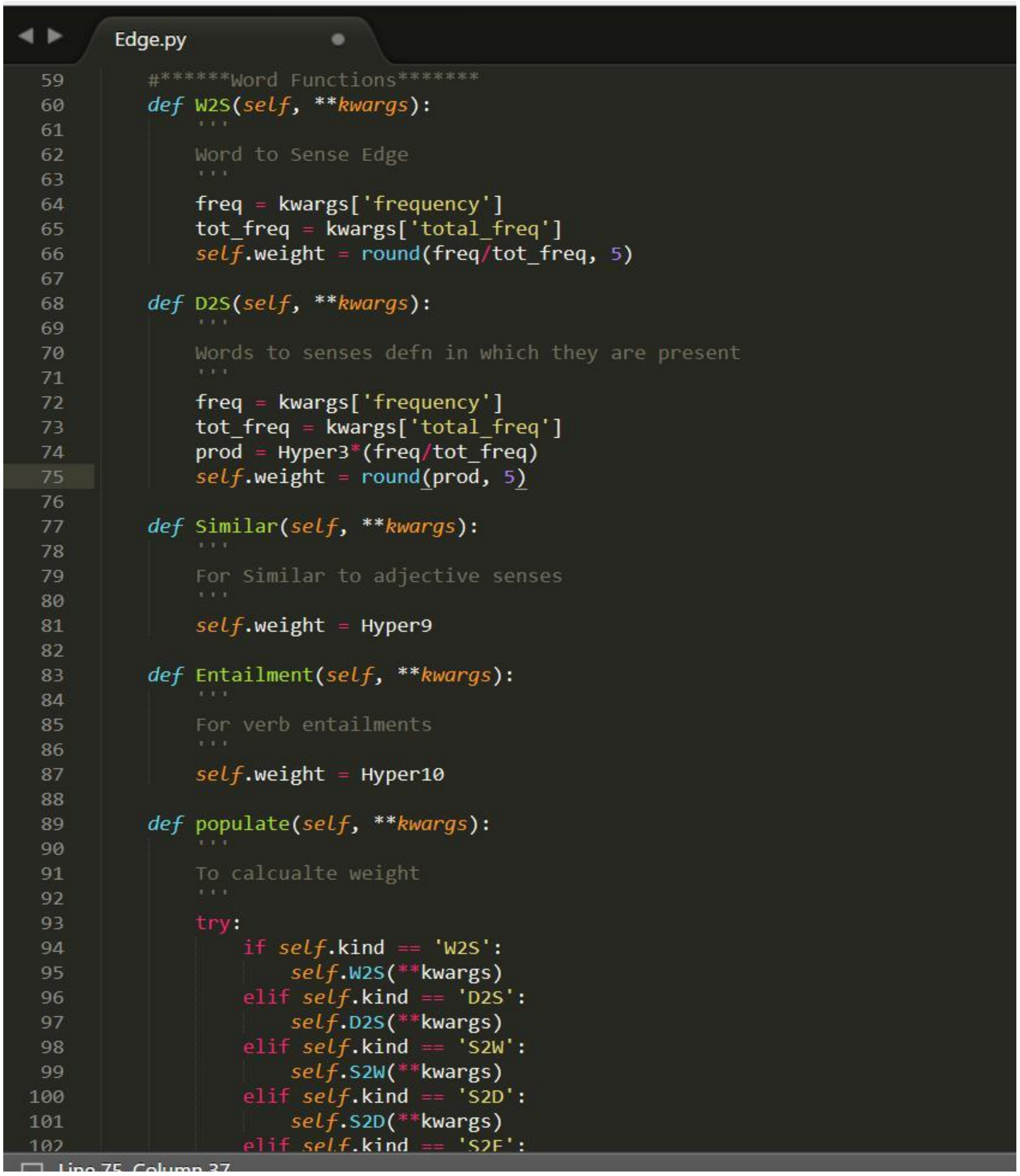
7.3. Coding Guidelines

The entire codebase adheres to a well established set of guidelines which are as follows :

1. Camelcase naming for classes and functions, lowercase for member variables.
2. Multi Line comments inside functions stating purpose of each, single line comments beside operations and anomalous conditions, separated by one tab space.
3. Most of the modules are modelled as classes with each having unique attributes and properties, encapsulated in a single unit.
4. Interfaces are exposed to client whereas implementation is hidden throughout the solution pipeline.
5. Some classes have functions with variable parameters. The goal is to make adding and removal of values feasible from interface without changing signature of functions on implementation end.
6. Class members have fixed convention with '_' preceding their names.
7. Interfaces delegate work to other classes internally, thus maintaining system abstraction.
8. Crucial class properties are made accessible through getters and setters, name of properties cannot be derived from getter/setter names, thus maintaining abstraction at class level.
9. Factory creation pattern is used for creating objects with fixed set of properties
10. Default parameters are used in reusable functions for polymorphism via method overloading, as functions can be called with their default behaviour or by passing particular values to them.
11. Try catches around code in main and separate handling of different types of exceptions like KeyboardInterrupt, StopIteration etc.
12. Various hyperparameters and other commonalities are stored in separate files with global access so as to promote code re-usability throughout the system.
13. Modularity maintained throughout code as each dissociable functionality/design is modelled as a separate function/class.

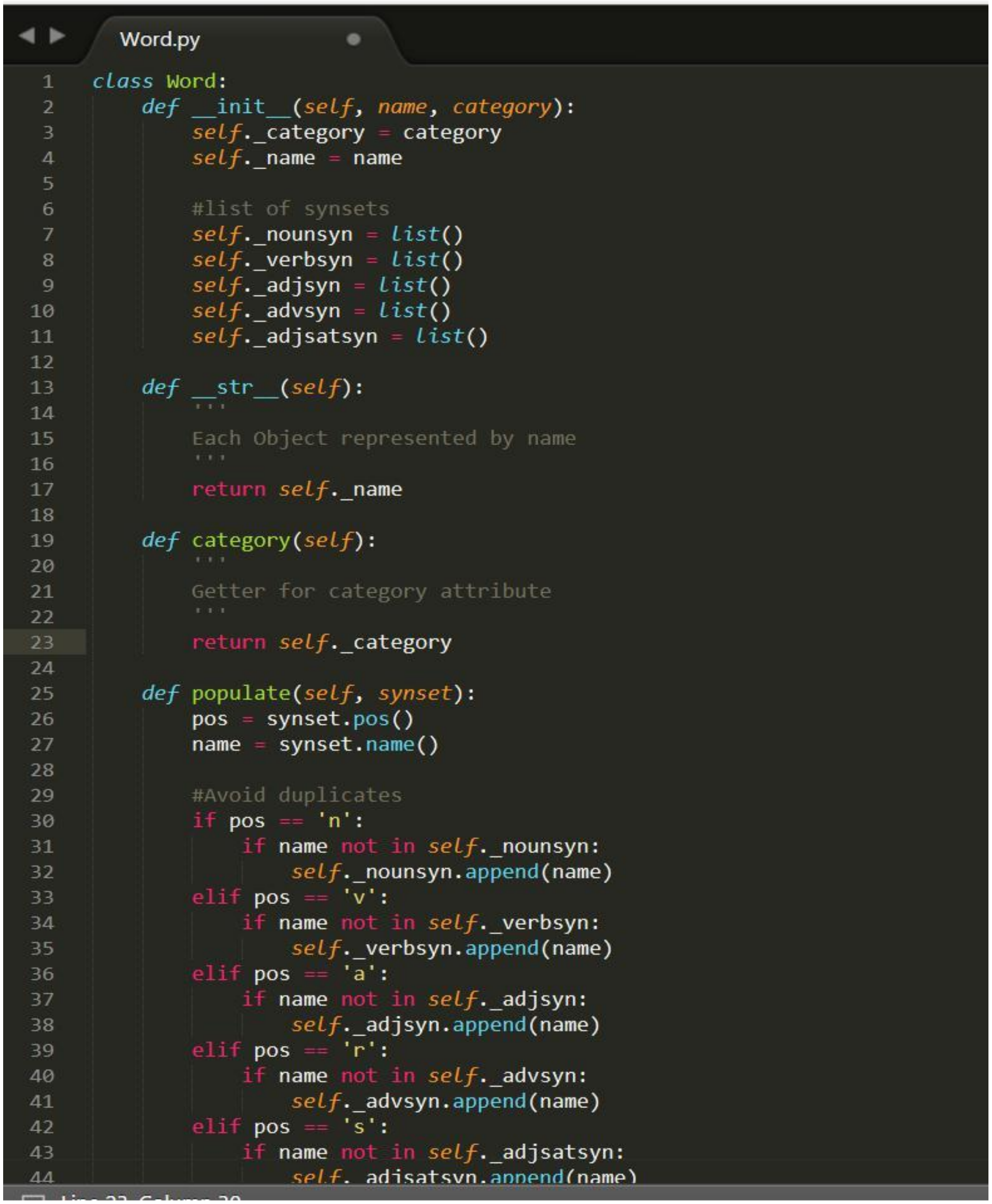
14. Generic functions to populate properties are present inside the class itself(encapsulation). These functions delegate/fill various properties on the basis of parameter values received.
15. Most of the classes have 'main' functions to either generate intermediate data or create instances to test class functionalities.
16. Use ideally 4 spaces per level of indentation.
17. Disable ‘hard tabs’ in the text editor being used.
18. Do not mix spaces and tabs in source code files to maintain interoperability.
19. Include a space after the comma(“,”) in dictionaries, lists, inside function arguments etc before writing the next element.
20. Include a space character around variable assignments, equality/inequality comparisons.
21. However do not use spaces for assignments when inside parentheses or inside function argument lists.

7.4. Sample Code



```
    #******Word Functions*******
    def W2S(self, **kwargs):
        '''
        Word to Sense Edge
        '''
        freq = kwargs['frequency']
        tot_freq = kwargs['total_freq']
        self.weight = round(freq/tot_freq, 5)

    def D2S(self, **kwargs):
        '''
        Words to senses defn in which they are present
        '''
        freq = kwargs['frequency']
        tot_freq = kwargs['total_freq']
        prod = Hyper3*(freq/tot_freq)
        self.weight = round(prod, 5)

    def Similar(self, **kwargs):
        '''
        For Similar to adjective senses
        '''
        self.weight = Hyper9

    def Entailment(self, **kwargs):
        '''
        For verb entailments
        '''
        self.weight = Hyper10

    def populate(self, **kwargs):
        '''
        To calcualte weight
        '''
        try:
            if self.kind == 'W2S':
                self.W2S(**kwargs)
            elif self.kind == 'D2S':
                self.D2S(**kwargs)
            elif self.kind == 'S2W':
                self.S2W(**kwargs)
            elif self.kind == 'S2D':
                self.S2D(**kwargs)
            elif self.kind == 'S2E':
```

Fig 8.1. Edge class, follows various guidelines like delegation, variable parameter, naming.



```python
class Word:
    def __init__(self, name, category):
        self._category = category
        self._name = name

        #list of synsets
        self._nounsyn = list()
        self._verbsyn = list()
        self._adjsyn = list()
        self._advsyn = list()
        self._adjsatsyn = list()

    def __str__(self):
        '''
        Each Object represented by name
        '''
        return self._name

    def category(self):
        '''
        Getter for category attribute
        '''
        return self._category

    def populate(self, synset):
        pos = synset.pos()
        name = synset.name()

        #Avoid duplicates
        if pos == 'n':
            if name not in self._nounsyn:
                self._nounsyn.append(name)
        elif pos == 'v':
            if name not in self._verbsyn:
                self._verbsyn.append(name)
        elif pos == 'a':
            if name not in self._adjsyn:
                self._adjsyn.append(name)
        elif pos == 'r':
            if name not in self._advsyn:
                self._advsyn.append(name)
        elif pos == 's':
            if name not in self._adjsatsyn:
                self._adjsatsyn.append(name)
```

Fig 8.2. Word class, follows various guidelines like abstraction. getters/setters.

7.5. Unit Test Cases

The system is extensively modular and source pipeline is segmented into dissociable sections, each independent of each other. Since we followed bottom-up design approach there was a need of testing each component separately before integration. Thus, the following unit cases were implemented.

1. *test_words* : The first section of solution pipeline extracts words from wordnet lexical graph, models it as an instance of our Word class and stores into onto a data structure. To make sure if outcome is persistent, an automated script extracts random Word instances from this data structure and compares all its properties with its equivalent in wordnet. Since the script is randomised, it ensures the consistency of the entire data structure.
2. *test_synsets* : Similar to its prior module, the second component extracts synset specifications from wordnet and stores it onto data structure. An automated script anon extracts random synsets instances and compares it with its wordnet equivalent. Any inconsistencies are reported into log files.
3. *test_compute* : After creating custom wordnet, relevant information is extracted from it to generate metadata for similarity graph creation. Since regularity and consistency of the similarity graph is dependent on this metadata, testing this module thoroughly is critical. Data is present in structured files from which random entries are extorted and checked for persistence against wordnet. This time instead of probing all node properties, selective properties to be casted in similarity graph are validated.
4. *test_graph* : This checks for pliability of graph created with respect to transitional metadata generated prior to this module. The number of nodes created, number of edges from each node are verified with the data. After this step, the underlying technology behind the system is good to go.
5. *spider_test* : Once graph created, it needs to be traversed efficiently and exhaustively at the same time. This is where spider comes into picture. The program performs a special case of dfs traversal, considering multiple paths to vicinity nodes, avoiding path cycles at

the same time. Since the output of this module is unpredictable and extensive, its testing cannot be automated and is done against theoretical concepts.

6. *word_benchmark* : The wordclient component of system is supposed to take word pairs and give back semantic score between them. Since there are various benchmarks available to evaluate system's semantic awareness against human annotated data, an automatic script picks one of these benchmarks[18][19] and compares system's outcome against them.
7. *manual_sentence_test* : Sentence client gives semantic scores between pair of sentences taking into account both word order and semantics. Manual test cases were written and system's output was checked against expected values.

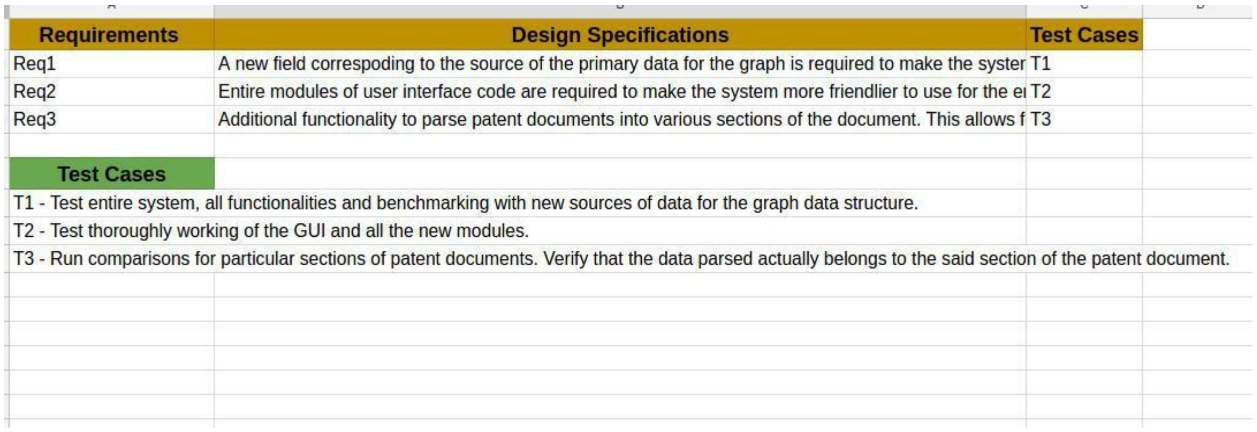
7.6. Metrics for Unit Test Cases

In order to satisfy aforementioned unit tests, certain metrics were established which are as follows :

1. *test_words* : To fulfill this test scenario, the automated script should complete its execution without any abnormal termination triggered by any inconsistency in properties.
2. test_synsets : Similar to its predecessor, this unit has uninterrupted execution of automated testing script as its pass criteria.
3. *test_compute* : The script is run on a hundred random data units in order to validate data persistence. If there are no cases of discrepancy during script run, the test is considered to be a success.
4. *test_graph* : The number of nodes and edges in the graph should be at par with transitional metadata generated beforehand for the test to succeed.
5. *spider_test* : Since crawling of graph is checked manually, the resulting vicinity web of theorized test cases should match with the expected value. Any mismatch will cause the test to fail.
6. *word_benchmark* : We primarily use Miller and Charles benchmark[18] which has human annotated semantics scores between certain pair of words. If the score obtained from our wordclient is within 0.05 range of the expected value as per the benchmark, the test is accepted.
7. *manual_sentence_test* : Due to manual unit testing, certain test cases are fixed, semantic and order vectors are computed for each and compared against the vectors generated from system. Any inconsistency with respect to expected values will cause the test to fail.

7.7. Updated RTM

Updated RTM reflecting new functionality and corresponding test cases.



Requirements	Design Specifications	Test Cases
Req1	A new field correspoding to the source of the primary data for the graph is required to make the syster	T1
Req2	Entire modules of user interface code are required to make the system more friendlier to use for the er	T2
Req3	Additional functionality to parse patent documents into various sections of the document. This allows f	T3

Test Cases

T1 - Test entire system, all functionalities and benchmarking with new sources of data for the graph data structure.

T2 - Test thoroughly working of the GUI and all the new modules.

T3 - Run comparisons for particular sections of patent documents. Verify that the data parsed actually belongs to the said section of the patent document.

Fig. 8.3 Updated RTM

CHAPTER 08

ILLUSTRATIVE EXAMPLE

Suppose comparing two documents. D1 - ‘’I love my dog. I adore him” and D2 - “I love my puppies. I love my dog too. His name is Rocket”. Document client is appointed to fragment these documents into sentences and find if shorter document (D1) is present in bigger document (D2). After fragmentation, morphological parsing and stop word removal, D1 - [‘love dog’, ‘adore’] and D2 - [‘love puppy’, ‘love dog’, ’name rocket’]. Now each sentence from D1 is compared with each sentence from D2 via sentence client.

Comparing ‘love dog’ and love puppy‘ using sentence client, a word set is formed W - [‘love dog puppy’]. Getting semantic vector for sentence ‘love dog’, each word from word set is compared with words in sentence. Since ‘love’ is present in sentence, 1 is added in semantic vector, same for word ‘dog’. Comparing word ‘puppy’, since it is not found in sentence, most semantically similar word to ‘puppy’ in sentence is ‘dog’. So its semantic similarity score - 0.8 is added and so semantic vector for sentence is - [1,1,0.8]. Similarly, order vector for sentence is calculated using similar methodology but instead of using semantic scores, we use indexes of words in word set. Thus, order vector for sentence ‘love dog’ is - [1,2,2]. Similarly order and semantic vectors for other sentence is obtained. Semantic score between sentences is computed via cosine similarity of their semantic vectors. Order score is obtained via normalizing order deviation using - $1 - \|v1-v2\| / \|v1+v2\|$. These two scores are merged into one score to give sentence similarity.

For words similarity, semantic graph is leveraged to get word vector. So from ‘puppy’ to ‘dog’ word vector - [3,0.6,0.36,1.72]. Meaning of each dimension is discussed in previous sections. To get a standard score, vector is obtained from ‘puppy’ to ‘puppy’ as - [4,0.75,0.33, 1.32]. Each dimension is pushed using certain hyperparameters to tune in accordance with [18] benchmark. Word similarity is thus cosine similarity between these two vectors.

Thus, In document client score obtained between ‘love dog’ and ‘love puppy’ - 0.94. But since ‘love dog’ is then compared with other sentence of document ‘love dog’, sentence score - 1, so comparison of ‘love dog’ is stopped. Similarly, all sentences are compared and then total score is divided by no of sentences in shorter documents. So, final score - (1 + 0.6875) / 2 = 0.844.

CHAPTER 09

TESTING

9.1. System/Function test specifications for the project

The system testing specification used for the project is derived from the software requirements specification and the functional requirements specification of the software system. The system testing specification includes test cases to maintain the overall acceptability/viability of the software for the end user apart from testing the most basic features and functionality of the software. It includes performance testing, checks for basic security practices, usability test scenarios and other general test cases.

Some of the test cases that were executed :

General test cases

1. Application crash messages, database errors and other implementation specific error details must not be displayed in the production version of the software. All such instances of error must be redirected to a particular error page.
2. Numeric values must be formatted in a proper way.
3. Timeout values wherever used must be configured adequately keeping the system states and expected behaviour for the user in mind.
4. Validation and error messages must be shown at correct locations in the user interface.
5. All error messages must follow a standard style procedure.
6. All exceptions arising due to the execution of the software must be handled by the software without affecting any system wide applicable variables or any other software/services that may be running on the system during the time of the crash.
7. Make sure text on all pages does not include any spelling or grammatical errors.
8. All system resources requested by the software must be freed as the software completes execution.

GUI Test scenarios

1. Formatting parameters such as font sizes, style, font color, background color for various elements on the user interface etc must have values in adherence to the ones agreed upon in software requirements specifications and functional requirements specifications.
2. Check all pages in the interface for any broken links.
3. User must not be able to type in text boxes used to display the results of any process. Such text boxes must be made read-only.
4. Tab key must work as expected throughout the user interface.
5. As the page loads, default values of radio buttons, placeholder values in input fields etc must be made.
6. Loader icons must be displayed when making a request, waiting for a response etc.
7. Make sure the user interface loads within an expectable time frame.
8. Check if the web application is able to load under conditions such as slow connection. Make sure all elements of the web app are loaded as expected under such conditions.
9. Response times of any user action such as clicking a button, making ajax requests, loading another web page etc must be under acceptable limits.

Security

1. Sanitize input fields for SQL injection attacks.
2. Page crashes must not reveal any data regarding the databases used, server information, geographical locations or any application related information in the production environment.

9.2. Test Environment Used

The codebase of the software being written in python language, the ‘unittest’ framework was used for testing.

Unittest provides support for test automation, writing of setup code that can be run before the execution of a test case, writing teardown/shutdown code that can be run after a test case has finished execution etc.

Unittest has a number of components that are used for test automation, namely

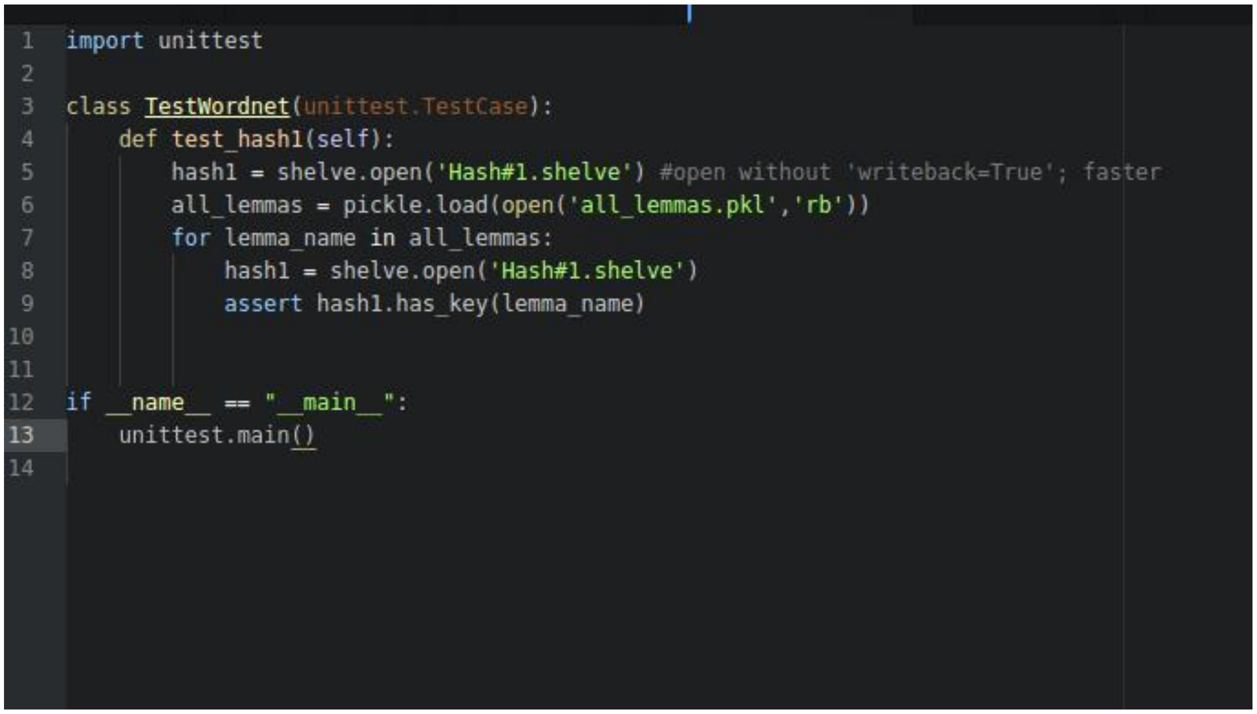
Test fixture - used for implementing any activities that must be done before the execution of a test case, such as creating any files, directories, starting up a server etc.

Test case - it is a function in the unittest class, that checks for a program output against an expected output for a particular input or set of inputs.

Test suite - is a number of test cases grouped together, that must all be run for testing.

9.3. Test procedure

Test cases under unittest are defined as functions inside a class which inherits from the unittest.TestCase class. The test cases essentially implement one of assertEqual(), assertTrue(), assetFalse() or assetRaises() to check the system produced output with the expected output values.



```python
import unittest

class TestWordnet(unittest.TestCase):
    def test_hash1(self):
        hash1 = shelve.open('Hash#1.shelve') #open without 'writeback=True'; faster
        all_lemmas = pickle.load(open('all_lemmas.pkl','rb'))
        for lemma_name in all_lemmas:
            hash1 = shelve.open('Hash#1.shelve')
            assert hash1.has_key(lemma_name)


if __name__ == "__main__":
    unittest.main()
```

Fig 9.1 An example of a testcase run.

9.4. Example test outcome

For the test case specified above, when executed using the unittest python testing framework, gives an outcome as follows,

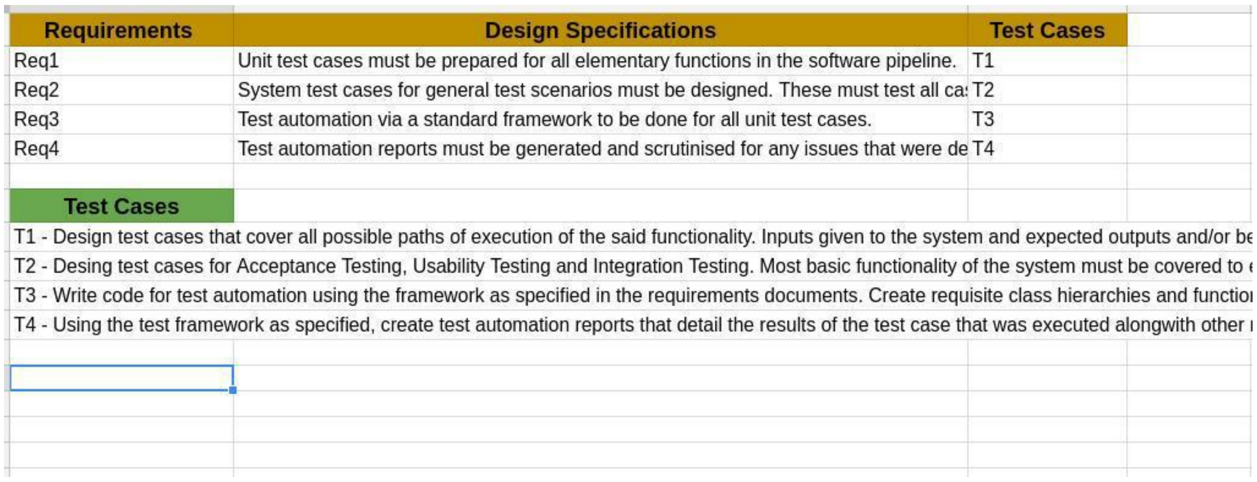
```
...
----------------------------------------------------------------------
Ran 1 test in 0.003s

OK
```

9.5. Test metrics

Out of a total of 10 test unit test and system test cases that were run, 8 of the cases passed. Because of the processing of a large number of elements and variations in the english language that are inherent due to differences in accent etc some words/ synsets and example use cases contradict the data received from wordnet. Due to this some of the test cases failed. We are trying morphological parsing, stemming and lemmatization techniques among other things to bring down the number of such occurrences significantly.

9.6. Updated Requirements Traceability Matrix

Updated RTM reflecting new functionality and corresponding test cases.



Requirements	Design Specifications	Test Cases
Req1	Unit test cases must be prepared for all elementary functions in the software pipeline.	T1
Req2	System test cases for general test scenarios must be designed. These must test all ca	T2
Req3	Test automation via a standard framework to be done for all unit test cases.	T3
Req4	Test automation reports must be generated and scrutinised for any issues that were de	T4

Test Cases
T1 - Design test cases that cover all possible paths of execution of the said functionality. Inputs given to the system and expected outputs and/or be
T2 - Desing test cases for Acceptance Testing, Usability Testing and Integration Testing. Most basic functionality of the system must be covered to e
T3 - Write code for test automation using the framework as specified in the requirements documents. Create requisite class hierarchies and functio
T4 - Using the test framework as specified, create test automation reports that detail the results of the test case that was executed alongwith other r

Fig 10.1 Updated RTM

CHAPTER 10

RESULTS AND DISCUSSION

The result obtained in a comparison of two words is a numeric value of semantic score that represents how similar the two words are. The graph made from the custom wordnet being quite exhaustive we are able to find a good number of similar words and the precision metric for the overall resulting score value is quite high. The system also performs well against human annotated benchmarks of word pairs and the semantic similarity values of those words. Thus the semantic similarity values being obtained in the case of word comparison is quite satisfactory.

Document semantic similarity and sentence semantic similarity on the other hand, required more deeply learned systems, that can give due weightage to parameters such as word order apart from the semantic closeness of words. It has to associate sentences in one document to sentences in another document for all the sentences in the document. This requires considerable computing power, with the final values being skewed in certain input cases.

In order to benchmark our system performance against existing patent norms, we used google patents. We queried for heart disease device patents, the first result obtained was ‘Cardiac Disease Treatment and Device’ (US656409B2). We performed a prior art search for this patent over google. We took top 10 results, in order of preference as arranged by google patent search, and compared summary of all these patents with our original patent i.e. (US656409B2). We compare same parts of patents as it would not make sense to compare abstract of one patent to background of other patent. User can select which section of patents needs to be compared via a dropdown in the interface. The following were the results obtained on comparing summaries:

Patent No. (In order of relevance) Comparison with - US656409B2	Semantic Similarity Score	Computation time (sec) on 8gb machine
US6702732B1	0.3838	4798.992
US7163507B2	0.4159	3091.64
US6432039B1	0.3557	1973.48
US6402781B1	0.4085	3981.013

US6685627B2	0.3964	8163.18
US6076013A	0.3992	1758.345
US20050234436A1	0.4689	6392.668
US6887192B1	0.365	1386.421

Table 1.1 Semantic scores of related patents from google patent search

We then compared same patent with itself, the score obtained was 1. When compared after introducing some noise i.e. changing active voice to passive, restructuring some sentences and using alternate terminologies, the score decreased to 8.701. When compared with totally unrelated patents, the following were the results:

Patent	Semantic Score	Time (sec)
Cooking (WO1995026636A1)	0.2738	1561.896
Berthing (US7287484B2)	0.3352	2161.272
Seating (US20030122662A1)	0.3508	1714.899
Ecommerce (US20140244451A1)	0.3325	146.377

Table 1.2 Semantic scores of unrelated patents from google patent search

After running several other tests we came to the conclusion that in case of patent comparison in our system, score between [0.65,1] would suggest patents are similar and might infringe, score between [0.35,0.65) suggest patents are about same topic but do not infringe, and score [0,0.35) suggest that patents are unrelated.

We also compared our system against SEMILAR[24] a framework/library to facilitate semantic search. Some corner cases were manually compared and the results were as follows :

Pair of Sentences	SemDoc	SemiLar
S1 : "She should be ashamed of herself for playing politics with this important issue," said state budget division spokesman Andrew Rush. S2 : "Senator Clinton should be ashamed of herself for playing politics with the important issue of homeland security funding," he said.	0.5278	0.543
S1 : The window was shut. S2 : The heat in the room was intense.	0.0838	0.00
S1 : My car skid over the wet road. S2 : I slipped over the slippery floor.	0.4032	0.10240
S1 : I was having fun near the waves. S2 : I was enjoying at the beach.	0.0903	0.00
S1: The cat jumped over the dog. S2: The dog jumped over the cat.	0.9592	1.00

Table 1.3 Semantic scores sentences in SemDoc vs. SemiLar

Thus, from above results it is evident that a well established semantic search platform like SEMILAR does not relate unrelated concepts like beach and waves. It also does not consider the case where same set of words when arranged differently mean different things as evident from the last test case.

A number of improvements can be made to the current system. From rudimentary improvements such as decreasing the time of execution of modules in the pipeline by increasing efficiency to incorporating deep learning neural networks, there are a host of features that can be added to the software. The ability to provide a user interface for the addition of a new word by the user that does not already exist in the database used by the software and more interactive and intuitive user interface for the use of the software are high priority and important future considerations for this project.

The requirements traceability matrices formulated during the entire duration of the project were duly followed with the final software honoring the requirements as stated in the Requirements Specification documents and the requirements traceability matrix. All the test cases and tasks aligned against the requirements and tracked as in the Architectural Design RTM, the system design RTM, the functional requirements RTM, non functional requirements RTM, the test cases RTM among others were implemented.

CHAPTER 11

RETROSPECTIVE

Semantic incorporation in information retrieval is a critical and appealing problem in modern day computer science. It is a delegation of how human brain processes speech and thus would make artificial intelligence more ‘human like’. Since it's an open research field, there is no correct answer, only a few standard approaches. It was a great opportunity to explore recent works in the field, weigh our options and cherry pick which combination would make more logical sense and hasn't been tried before, along with implementing our own learnings in the process. Word sense disambiguation and polysemy are most relevant and challenging problems in semantic perception which haven't been solved and are usually ignored by most semantic solutions. Our approach handles them very elegantly, using probability, like a human mind would. Rather than taking hierarchical model of entity relationships, we model entities in a graph, which is more realistic as it associated entities which might not be semantically related but are lexically linked by their usage in linguistics.

Patent prior art search is a significant problem in dire need of semantic search. Approaching the problem with our semantic solution lead to expectedly better results. Although, WordNet is a powerful tool for semantics, it has its own shortcomings. It is a compaction of commonly used English words but lacks domain centric terminologies. The solution to this is to incorporate external domain knowledge into our similarity graph information would, which we could not crack due to time and scope boundations. But we made sure that our solution is scalable to external knowledge admittance of such kind. We were deeply humbled to work with our guide whose experience, knowledge and passion for the field inculcated enthusiasm within us and propelled us in the right direction. We even got an opportunity to share some of our work with our juniors through a hands on session about relevance of semantics and wordnet exploration. Overall, treading on a path less explored presented a new dimension of education to us, one in which there are no precise answers. Semantics is a field of endless possibilities, one that we would like to pursue in our future.

CHAPTER 12

REFERENCES

[1] - L. Stanchev. Semantic Document Clustering Using a Similarity Graph. Tenth IEEE International Conference on Semantic Computing, 2016.

[2] - Pawan Sharma , Rashmi Tripathi, Vivek K. Singh, R.C.Tripathi, IIIT Allahabad, Automated Patent Search through Semantic Similarity. International Conference on Computer, Communication and Control, 2015.

[3] - Pawan Sharma , Rashmi Tripathi, Vivek K. Singh, R.C.Tripathi, IIIT Allahabad, Finding Similar Patents Through Semantic Query Expansion. Eleventh International Multi-Conference on Information Processing-2015.

[4] - L. Stanchev. Fine Tuning an Algorithm for Semantic Search Using a Similarity Graph. International Journal of Semantic Computing, 2015.

[5] - L. Stanchev. Creating a Similarity Graph from WordNet. Fourth International Conference on Web Intelligence, Mining and Semantics, 2014.

[6] - L. Stanchev. Creating a Phrase Similarity Graph from Wikipedia. Eighth IEEE International Conference on Semantic Computing, 2014.

[7] - Dr. Stuart McLean, LexisNexis, Patent Prior-Art Searching with Latent Semantic Analysis, Aug 2014.

[8] - L. Stanchev. Building Semantic Corpus from WordNet. First International Workshop on the Role of Semantic Web, 2012.

[9] - Vitaly Klyuev and Yannis Haralambous, Query Expansion: Term Selection using the EWC Semantic Relatedness Measure, 2011.

[10] - Yuhua Li, David McLean, Zuhair A. Bandar, James D. O'Shea, and Keeley Crockett, Sentence Similarity based on semantic nets and corpus statistics, IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 18, NO. 8, AUGUST 2006.

[11] - E. H. Hovy, L. Gerber, U. Hermjakob, M. Junk, and C. Y. Lin. Question Answering in Webclopedia. TREC-9 Conference, 2000.

[12] - D. Moldovan, S. Harabagiu, M. Pasca, R. Mihalcea, R. Goodrum, and R. Girju. LASSO: A Tool for Surfing the Answer Net. Text Retrieval Conference (TREC-8), 1999.

[13] - T. K. Landauer, P. Foltz, and D. Laham. Introduction to Latent Semantic Analysis. Discourse Processes, pages 259–284, 1998.

[14] - S.K.M. Wong, Vijay V. Raghavan, VECTOR SPACE MODEL OF INFORMATION RETRIEVAL, Department of Computer Science, University of Regina, Canada, 1984.

[15] - Christiane Fellbaum (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press.

[16] - George A. Miller (1995). WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: 39-41

[17] - Princeton University "About WordNet." WordNet. Princeton University. 2010. <http://wordnet.princeton.edu>

[18] - G. Miller and W. Charles. Contextual Correlates of Semantic Similarity. Language and Cognitive Processing, 6(1):1–28, 1991.

[19] - L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolf-man, and E. Ruppin. Placing Search in Context: The Concept Revisited. ACM Transactions on Information Systems, 20(1):116–131, January 2002.

[20] - Jonas Mueller and Aditya Thyagarajan. Siamese Recurrent Architectures for Learning Sentence Similarity. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16).

[21] - Hisao Mase, Tadataka Matsubayashi, Yuichi Ogawa, Makoto Iwayama, Tadaaki Oshio. Two-Stage Patent Retrieval Method Considering Claim Structure. Working Notes of NTCIR-4, Tokyo, 2-4 June 2004

[22] - Pawan Sharma, R.C. Tripathi. Patent Database: A methodology of information retrieval from PDF. International Journal of Database Management Systems (IJDMS) Vol.5, No.5, October 2013 DOI : 10.5121/ijdms.2013.5502 9

[23] - ETMTNLP '02 Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics - Volume 1.

[24] - Rus, V., Lintean, M., Banjade, R., Niraula, N., and Stefanescu, D. (2013). SEMILAR: The Semantic Similarity Toolkit. Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics, August 4-9, 2013, Sofia, Bulgaria.

CHAPTER 13

USER MANUAL

# SS(Semantic Similarity)-Graph [Git Repo Link - https://github.com/anirudhagar13/SS_Graph]
Custom Semantic Graph based on Wordnet

-The System leverages this graph to semantically compare two documents and give back a similarity score.
-User can view paths over UI as to how words are related to each other in our custom semantic graph

#Steps to Create:

----

1. Install python2.7.
2. Download nltk package in python using pip-install/easy-install.
3. Get wordnet, penntreebank from nltk corpus in python, incase any dependency missed get it using nltk.download() > dependency.
4. Run Creator.sh to create graphdata, it is a shell file.
5. If using Powershell run 'Measure-Command {start-process sh Creator.sh -Wait}', if bash run 'time Creator.sh'
6. Incase of any failures, clear Shelves folder and rerun Creator.sh
7. The System is now good to go, Open UI.
8. Set direct paths to respective folders in views.py inside ssgraph/graph.
9. Run server using 'python manage.py runserver'.
10. Open link displayed on console, have fun with the system.

#How to Use:

----

1. After the screen renders, either type two documents.
2. Compare them by clicking on Compare button.
3. To see semantic matches of word found in the other document, select wordtree radio button and click on word.

4. To see how in graph the words are related, select network graph radio button and click on on word.

5. To know details about each node just click on node, and see details appear below.

6. To know edge meanings refer to the color coded legends on the side.

7. To see how document gets broken down into sentences and then into words, go to logs and read each log.

8. Can be used to compare patents by uploading files, .txt or .docx only.

9. Select the section of patent you want to compare and click on compare.

10. Read logs to know the intricacies of patent comparison.