University of Eastern Finland

School of Computing

Master Thesis

String Metrics and Word Similarity applied to Information Retrieval

Hao Chen

2012

Abstract

Over the past three decades, Information Retrieval (IR) has been studied extensively. The purpose of information retrieval is to assist users in locating information they are looking for. Information retrieval is currently being applied in a variety of application domains from database systems to web information search engines. The main idea of it is to locate documents that contain terms the users specify in their queries.

The thesis presents several string metrics, such as edit distance, Q-gram, cosine similarity and dice coefficient. All these string metrics are based on plain lexicographic term matching and could be applied to classical information retrieval models such as vector space, probabilistic, boolean and so on. Experiment results of string distance metrics on real data are provided and analyzed.

Word similarity or semantic similarity relates to computing the similarity between concepts or senses of words, which are not lexicographically similar. The popular methodologies in measuring word similarity with the help of a thesaurus, e.g. WordNet, can be classified into two categories: one uses solely semantic links, the other combines corpus statistics with taxonomic distance. Five similarity measures (Resnik, Lin, Jiang & Conrath, Path and Wu & Palmer) belonging to these two categories are selected to conduct the experiment on the purpose of comparison.

Hierarchical clustering algorithms including both single-linkage clustering and complete-linkage clustering are studied by employing word similarity measures as clustering criteria. Stopping criteria including Calinski & Harabasz, Hartigan and WB-index are used to find the proper hierarchical level in the clustering algorithms. Experiments on both synthetic datasets and real datasets are conducted and the results are analyzed.

Keywords: string metrics, semantic/word similarity, information retrieval, WordNet, hierarchical clustering, clustering stopping criteria.

Acknowledgements

I would like to thank Dr. Pasi Fränti for the advice, encouragement and support he provided to me in supervising this thesis effort. I would also like to thank Qinpei Zhao, for her critical analyses, technical advice and recommendations. Special thanks go to Dr. Olli Virmajoki for his reviews and recommendations.

Tables of Contents

1. Introduction

Information Retrieval (IR) has been a widespread topic for last three decades [1]. Its purpose is to assist users in locating information they are looking for by locating documents with the terms specified in their queries. Information retrieval is currently being applied in a variety of application domains in database systems [2] to web search engines [3]. The search engines like Google[1] and Yahoo[2] are the most well-known applications, which are used by many people on a daily basis.

According to classical information retrieval models (vector space[4], probabilistic[5], boolean[6]), retrieval is based on lexicographic matching between terms. For example, a query and a document term are considered similar if they are lexicographically the same. However, plain lexicographic analysis is not always sufficient to determine if two terms are similar and consequently whether two documents are similar. Two terms can be lexicographically different but have the same meaning (they are synonyms), they have approximately the same meaning (they are semantically similar) or they belong to the same class in some categorization. For example, word *orange* is more similar to word *range* than word *apple* by applying lexicographic matching because they only have one alphabet difference. On the other hand, word *orange* have higher similarity value with word *apple* than word *range* by using word similarity measures because both word *orange* and word *apple* belong to the category of fruit. The lack of common terms in two documents does not necessarily mean that the documents are irrelevant to each other. Relevant documents may contain semantically similar but not necessarily the same terms. Semantically similar terms or concepts may be expressed in a different way in the document, and direct comparison is

[1] http://www.google.com/

[2] http://www.yahoo.com/

therefore not effective. For example, Vector Space Model [7] (VSM) will not recognize synonyms or semantically similar terms.

String metric reflects the relation between the two strings lexicographically. It is used to find approximate matches to a query string. String metrics have been developed and applied in different scientific fields like the detection and correction of spelling errors [8], statistics for probabilistic record linkage [9], database for record matching [10], artificial intelligence for supervised learning [11] and biology [12]. String metrics such as Levenshtein distance, Hamming Distance, Damerau–Levenshtein distance, Q-gram similarity, cosine similarity and dice coefficient are studied and discussed in this thesis.

Word similarity is a widespread topic in natural language processing (NLP)[13]. It has been applied in a number of applications, including word sense disambiguation [14] and detection of malapropisms [15]. Word similarity is a concept where a set of documents, or terms within term lists, are assigned a metric based on the likeness of their meaning, semantic content. Approaches to compute word similarity by mapping terms to ontology and by examining their relationships in that ontology are investigated. Some of the most popular word similarity measures are implemented and evaluated using WordNet as the underlying reference ontology.

Popular methodologies in measuring word similarity with the help of a thesaurus can be classified into two categories [16]: thesaurus-based algorithm and distributional algorithms. A thesaurus contains all the information such as hyponymy, glosses, example sentences and derivational relations needed to compute a semantic distance metric. Distributional algorithms compare words based on their distributional context in corpora. These approaches represent words as points in an N-dimensional space and require an appropriate distance measures in this space. Algorithms including Path, Wu & Palmer, Lin and Jiang & Conrath are introduced and discussed in this thesis.

Clustering is used in information retrieval systems to enhance the efficiency and effectiveness of the retrieval process [17]. In conceptual clustering, the objective is to identify classes of objects in a collection such that objects with the same set of features are assigned to the same class. In addition, each class must be properly characterized in terms of the features used for classification [18]. Clustering is a common technique widely used in many fields, such as machine learning [19], pattern recognition [20], data mining [21] and image analysis [22]. Hierarchical clustering, as one of most significant and widely applied categories in clustering community offers a flexible and non-parametric approach to indicate the true cluster structure known to exist in the data. In this thesis, the hierarchical clustering is adopted for grouping amounts of words/texts, by which easier management and faster search are achieved. Meanwhile, stopping criteria like Calinski & Harabasz, Hartigan and WB-index are used to determine the proper level and the correct number of hierarchical clusters.

The outline of this thesis is the following:

In chapter II, we study the string metrics including variants of edit distance, Q-gram, cosine similarity and dice coefficient. In chapter III, we learn word similarity measures including Path, Wu & Palmer, Lin and Jiang & Conrath. In chapter IV, we have an insight into clustering, especially hierarchical clustering for grouping texts. To find the correct hierarchical level of the clustering result, stopping criteria such as Calinski & Harabasz, Hartigan and WB-index are introduced and studied. In chapter V, experimental results from string metrics, word similarity measures and hierarchical clustering are reported. In chapter VI, conclusions of this study and some directions of research are given.

2. String Metrics

String metrics (also known as similarity metrics) are the class of textual based metrics resulting in a similarity or dissimilarity (distance) score between two text strings. A string metric provides a floating point number indicating the level of similarity based on plain lexicographic match. For example, similarity between the strings *orange* and *range* can be considered to be much more than the string *apple* and *orange* by using string metrics.

String metrics have been widely applied in various areas such as approximate string matching [23], comparison [24] and in fuzzy string search [25]. Recently, Spell checking [26] has become one of the most common applications. Given an input document, a spellchecker needs to find possibly mistyped words by searching in its dictionary. A spellchecker recommends potential matching candidates similar to those words that are not in the dictionary based on string metrics. Data cleaning is another significant application. Information from different data sources often has various inconsistencies. The same real-world entity could be represented in slightly different formats. Data cleaning [27] keeps the consistencies of information from different data sources.

In this section, string metrics including three edit distance algorithms with different edit operations, Q-gram similarity, cosine similarity and dice coefficient are introduced and studied.

To express the string metric more formally, the following notations used in the thesis are defined:

- *S*: source string
- *T*: target string
- *M*: length of source string *S*
- *N*: length of target string *T*

2.1 Edit distance

Edit distance is an important class of string metrics, which determines the distance between two strings *S* and *T* by calculating the cost of best sequence of edit operations that convert *S* to *T*. Typical edit operations are character insertion, deletion, and substitution, and transposition. There are several variants to calculate the edit distance depending on which edit operations are allowed.

2.1.1 Levenshtein distance

Levenshtein distance [28] is defined as the minimum number of edit operations needed to transform *S* into *T* with the allowable edit operations being insertion, deletion, or substitution of a single character. For example, we can calculate the distance between two strings *apple* and *orange* by Levenshtein distance with a simple matrix in Figure 1. The distance value is 5, which is in the lower right hand corner of the matrix. This corresponds to our intuitive realization that *apple* can be transformed into *orange* by adding “O”, substituting “R” for “A”, “A” for “P”, “N” for “P” and “G” for “L”. One insertion and four substitutions equals to 5 changes.

		A	P	P	L	E
	0	1	2	3	4	5
O	1	1	2	3	4	5
R	2	2	2	3	4	5
A	3	2	3	3	4	5
N	4	3	3	4	4	5
G	5	4	4	4	5	5
E	6	5	5	5	5	5

Figure 1. Example of Levenshtein distance

The following steps show how to construct the matrix in Figure 1:

1. Construct a matrix *d* containing *M* rows and *N* columns.
2. Initialize the first row from 0 to *M* and first column from 0 to *N*.
3. Examine each character of *S* (*i* from 1 to *M*) and each character of *T* (*j* from 1 to *N*).
4. If *S*[*i*] equals *T*[*j*], the cost is 0. Otherwise, the cost is 1.
5. Set cell d[*i*, *j*] of the matrix equal to the minimum of:
    A. The cell immediately above plus 1: *d* [*i*-1, *j*] + 1.
    B. The cell immediately to the left plus 1: *d* [*i*, *j*-1] + 1.
    C. The cell diagonally above and to the left plus the cost: *d* [*i*-1, *j*-1] + cost.
6. After the iteration steps (3, 4 and 5) are complete, the distance is found in cell *d* [*N*,*M*].

The pseudocode for calculating Levenshtein distance is given in Figure 2.

```
int LevenshteinDistance(char S[1..M], char T[1..N]){
    declare int d[0..M, 0..N]
    for i from 0 to M
        d[i,0] := i //the distance of any first string to an empty second string
    for j from 0 to N
        d[0, j] := j //the distance of any second string to an empty first string
    for j from 1 to N
    {
        for i from 1 to M
        {
            if S[i] = T[j] then
                d[i, j] := d[i-1, j-1] //no operation required
            else d[i, j] := minimum(d[i-1, j] + 1, //a deletion
                                    d[i, j-1] + 1, //an insertion
                                    d[i-1, j-1] + 1) //a substitution
        }
    }
    return d[M, N]
 }
```

Figure 2. Pseudocode of Levenshtein distance

2.1.2 Hamming distance

Hamming distance [29] allows only edit operation of substitution to transform *S* into *T*. Therefore, the length between *S* and *T* is the same. This algorithm is often used for error detection and correction for two strings with the same lengths. A straightforward implementation of Hamming distance is given as pseudocode in Figure 3. It takes two strings *S* and *T* as inputs and returns their Hamming distance.

```
int HammingDistance(S[1..M], T[1..N])
 {   If M !=N then return -1
    declare int HammingDistance=0
    for i from 1 to M{
            if S[i] != T[i] then HammingDistance++
    }return HammingDistance
 }
```

Figure 3. Pseudocode of Hamming distance

2.1.3 Damerau–Levenshtein distance

Damerau–Levenshtein distance [30] is quite similar to Levenshtein distance. The only difference is that Damerau–Levenshtein distance allows one more edit operation: the transposition of two adjacent characters. The pseudocode for calculating Damerau-Levenshtein distance is in Figure 4.

```
int DamerauLevenshteinDistance(char S[1..M], char T[1..N])
 {
    declare int d[0..M, 0..N]
    declare int i, j, cost
    for i from 0 to M
       d[i,0] := i //the distance of any first string to an empty second string
    for j from 0 to N
       d[0, j] := j //the distance of any second string to an empty first string
    for i from 1 to M
    {
          for j from 1 to N{
          if S[i] = T[j] then cost := 0
                         else cost := 1
             d[i, j] := minimum(d[i-1, j] + 1, //a deletion
                                d[i, j-1] + 1, //an insertion
                                d[i-1, j-1] + 1) //a substitution

           if(i > 1 and j > 1 and S[i] = T[j-1] and S[i-1] = M[j]) then
                    d[i, j] := minimum(
                                          d[i, j],
                                          d[i-2, j-2] + cost) // transposition

       }
    }
    return d[M, N]
 }
```

Figure 4. Pseudocode of Damerau–Levenshtein distance

2.2 Q-gram

Q-gram is a consecutive substring of size that can be used as a signature of the entire string. Q-gram is typically used in approximate string matching by sliding a window of length q over the characters of a string to create a number of substrings. Since Q-gram can have fewer than q characters, characters “#” and “%” are used to extend the string by prefixing it with q-1 occurrences of “#” and suffixing it with q-1 occurrences of “%”. When q equals to 1, the Q-gram is the same as edit distance.

The foundation of the use of Q-gram is that when S and T are within a small edit distance of each other, they share a large number of Q-gram in common. Getting the Q-gram for two query strings makes the count of identical Q-gram of these two strings and the total Q-gram available. The algorithm contains the following steps.

1. Extend the string by prefixing it with q-1 occurrences of “#” and suffixing it with q-1 occurrences of “%”.
2. Split the S and T into two sets of Q-gram *arrayS*, *arrayT*.
3. Get the total grams number L by adding number of grams in S and T.
4. Combine two sets of Q-gram *arrayS* and *arrayT* to a set of Q-gram *arrayTotal*.
5. Remove the duplicate Q-gram in *arrayTotal*.
6. Calculate the number m_1 of same Q-gram shared between *arrayS* and *arrayTotal*.
7. Calculate the number m_2 of same Q-gram shared between *arrayT* and *arrayTotal*.
8. Calculate the absolute value *difference* by $|m_1 - m_2|$.
9. The similarity of Q-gram is calculated as below:

$$similarity = \frac{L - difference}{L}$$

The pseudocode for a function of Q-gram is given in Figure 5.

```
int Q_grams(char S[1..M], char T[1..N], q)
 {
    declare arrayS, arrayT, arrayTotal
    declare L
    declare $m_1$, $m_2$
    declare difference
    declare Q_grams_similarity

    arrayS := stringToArray(S[1..M], q) //parse S into a set of Q-gram with
                                        //length q
    arrayT := stringToArray(T[1..N], q) //parse T into a set of Q-gram with
                                        //length q
    arrayTotal := arrayS.concat(arrayT) //combine arrayS and arrayT into
                                         //arrayTotal
    arrayTotal.unquie()     //remove duplicate terms in arrayTotal
     L := arrayS.length + arrayT.length //the total grams number L


    for i from 0 to arrayTotal.length
    {
       for j from 0 to arrayS.length
          if arrayS[j] = arrayTotal[i]
          then $m_1$++    //identical Q-gram number of S  over the total Q-gram
                          //available

       for k from 0 to arrayT.length
          if arrayT[k] = arrayTotal[i]
          then $m_2$++    //identical Q-gram number of S  over the total Q-gram
                          //available

          difference += $|m_1-m_2|$

    }

    Return Q_grams_similarity = (L- difference) / L
```

Figure 5. Pseudocode of Q-gram

2.3 Cosine similarity

Cosine similarity is a vector based similarity measure. Cosine of two vectors *a*, *b* can be derived by using the Euclidean dot product formula.

$$a \cdot b = |a||b| \cos\theta$$

Where, θ represents the angle between *a* and *b*.

The input string is transformed into vector space in order to apply the Euclidean cosine rule to determine similarity. The algorithm contains the following steps.

1. Split the *S* and *T* into two sets of 2-gram *arrayS*, *arrayT*.
2. Remove the duplicate 2-gram in *arrayS*, *arrayT and* get the number L_1, L_2 of 2-gram in *arrayS* and *arrayT*.
3. Combine two sets of 2-gram *arrayS* and *arrayT* to a new of 2-gram *arrayTotal*, remove the duplicate 2-gram in *arrayTotal*, and get the number *L* of 2-gram in *arrayTotal*.
4. The variable *C* is calculated as follows.

$$C = (L_1 + L_2) - L$$

5. The cosine similarity is calculated as follows.

$$Cosine\_Similarity = \frac{C}{\sqrt{L_1} \cdot \sqrt{L_2}}$$

The pseudocode for a function of cosine similarity is given in Figure 6.

```
int Cosine_similarity (char S[1..M], char T[1..N] 2)
 {
    declare arrayS, arrayT, arrayTotal
    declare $L_1$, $L_2$, L
    declare C
    declare Cosine_Similarity

    arrayS = stringToArray(S[1..M]) //parse S into a set of 2-gram
    arrayS.unquie()    //remove duplicate terms in the arrayS

    $L_1$ := arrayS.length //the number of different terms in arrayS
    arrayT = stringToArray(T[1..N], 2) //parse T into a set of 2-gram

    arrayT.unquie() //remove duplicate terms in the arrayT
    $L_2$ := arrayT.length //the number of different terms in arrayT

    arrayTotal = arrayS.concat(arrayT) //combine arrayS and arrayT into
                                       //arrayTotal
    arrayTotal.unquie() // remove duplicate terms in the arrayTotal

     L := arrayS.length + arrayT.length //the total different terms in arrayTotal

     $C = (L1 + L2) - L$

     $Cosine\_Similarity = \frac{C}{\sqrt{L_1} \cdot \sqrt{L_2}}$

      Return Cosine_Similarity

}
```

Figure 6. Pseudocode of cosine similarity

2.4 Dice coefficient

Dice coefficient [31], named after Lee Raymond Dice and also known as the Dice's coefficient, is a term based similarity measure. It is calculated as follows.

$$Dice\ coefficient = \frac{2 * C}{L_1 + L_2}$$

Where C is the number of character bigrams found in both strings S and T, L_1 is the number of unique bigrams in string S and L_2 is the number of unique bigrams in string T.

The algorithm contains the following steps.

1. Split the S and T into two sets of 2-gram *arrayS*, *arrayT*.
2. Remove the duplicate 2-gram in *arrayS*, *arrayT* and get the number L_1, L_2 of 2-gram in *arrayS* and *arrayT*.
3. Combine two sets of 2-gram *arrayS* and *arrayT* to a new set of *2-gram arrayTotal*, remove the duplicate 2-gram in *arrayTotal*, and get the number L of 2-gram in *arrayTotal*.
4. The variable C is calculated as follows.

$$C = (L_1 + L_2) - L$$

5. The dice coefficient is calculated as follows.

$$Dice\ coefficient = \frac{2 * C}{L_1 + L_2}$$

The pseudocode for a function of dice coefficient is given in Figure 7.

```
int Dice_coefficient (char S[1..M], char T[1..N] Q)
 {
    declare arrayS, arrayT, arrayTotal
    declare $L_1$, $L_2$, L
    declare C
    declare Dice_coefficient

    arrayS = stringToArray(S[1..M]) //parse S into a set of 2-gram

    arrayS.unquie()     // remove duplicate terms in the arrayS
    $L_1$ = arrayS.length   // the number of different terms in arrayS

    arrayT = stringToArray(T[1..N], 2) // parse T into a set of 2-gram
    arrayT.unquie()     // remove duplicate terms in the arrayT
    $L_2$ = arrayT.length   // the number of different terms in arrayT

    arrayTotal = arrayS.concat(arrayT) //combine arrayS and arrayT into
                                                    //arrayTotal
    arrayTotal.unquie()     // remove duplicate terms in the arrayTotal
     L = arrayS.length + arrayT.length // the total different terms in arrayTotal

     C = ($L_1$ + $L_2$) - L

     Dice_coefficient = $\frac{2 * C}{L_1 + L_2}$

    Return Dice_coefficient

 }
```

Figure 7. Pseudocode of dice coefficient

3. Word Similarity

Word similarity is to assess the degree of similarity between two concepts by the similarity in the meaning of their annotations have used. The application of word similarity has been applied in more and more fields. In Biomedical Informatics, word similarity is used to compare genes and proteins based on the similarity of their functions rather than on their sequence similarity. And in GeoInformatics, it is used to find similar geographic features or feature types. In the field of Natural language processing (NLP), word similarity measures are used to detect plagiarism. Recent researches on word similarity have developed several numbers of tools for measuring word similarity summarized in Table 1.

Table 1. Tools for measuring word similarity

MSR[3]	Finding semantically related words by using many external sources such as Google search engine, Yahoo search engine, Wikipedia and many others – or all of them at once.
UMLS::Similarity::path[4]	Perl module for computing word similarity of concepts in the UMLS by simple edge-counting.
SenseBot[5]	A semantic search engine that generates a text summary of multiple Web pages on the topic of search query.
SenseLearner[6]	A Tool for all-words word sense disambiguation.

[3] http://cwl-projects.cogsci.rpi.edu/msr/

[4] http://cpan.uwinnipeg.ca/htdocs/UMLS-Similarity/UMLS/Similarity/path.html

[5] http://www.sensebot.net/

[6] http://lit.csci.unt.edu/~senselearner/

Meanwhile, many competing approaches for measures of word similarity have been proposed. Budanitsk [32] presents an extensive survey and classification of measures of semantic similarity. One category of such measures has been spurred by the advent of networks such as MeSH[7] and WordNet[8]. These vary from simple edge-counting [33] to attempts to factor in peculiarities of the network structure by considering link direction relative depth [34] and density [35]. These analytic measures now face competition from statistical and machine learning techniques. A number of hybrid approaches have been proposed that combine a knowledge-rich source, such as a thesaurus, with a knowledge-poor source, such as corpus statistics.

In selecting measures to analyze and compare in this thesis, we focused on those that used WordNet as their knowledge source. We used Java language programming to call existing similarity functions. The selected measures are Path, Wup, Lin, Rensik, and Jiang.

3.1 WordNet

WordNet [36] is a large organized lexical database of English as a main language which can be downloaded and used freely. It was created and is being maintained at the Cognitive Science Laboratory of Princeton University. It is an online thesaurus and has all aspects of a dictionary. It can also be seen as ontology representing knowledge as a set of concepts within a domain, and the relationships between those concepts for natural language terms. Table 2 shows the number of words, synsets, and Word-Sense Pairs in WordNet. There is also a multilingual WordNet for European languages such as French, German, and Italian which is structured in the same way as the English language WordNet [37].

7 http://www.nlm.nih.gov/mesh/

8 http://WordNet.princeton.edu/

Table 2. Number of words, synsets, and senses in WordNet

	Unique Strings	Synsets	Total Word-Sense Pairs
Noun	117,798	82,115	146,312
Verb	11,529	13,767	25,047
Adjective	21,479	18,156	30,002
Adverb	4,481	3,621	5,580
Totals	155,287	117,659	206,941

WordNet divides the lexicon into four categories: noun, verb, adjective and adverb that are grouped into synonym sets (synsets). Single synset is the smallest unit in WordNet, which represents a specific meaning of a word. It contains the word, its explanation and its synonyms. The specific meaning of one word is called a sense. The synsets are also organized into senses corresponding to different meanings of the same term or concept. For example, the words "night", "nighttime", and "dark" constitute a single synset that has the following gloss: the time after sunset and before sunrise while it is dark outside. The synsets (or concepts) are related to other synsets higher or lower in the hierarchy through different types of relationships. The most common relationships are the hyponym/hypernym (Is-A relationships), and the meronym/holonym (Part-Of relationships). There are nine noun and several verb Is-A hierarchies (adjectives and adverbs are not organized into Is-A hierarchies) in WordNet totally. Figure 8 illustrates a fragment of the WordNet Is-A hierarchy.

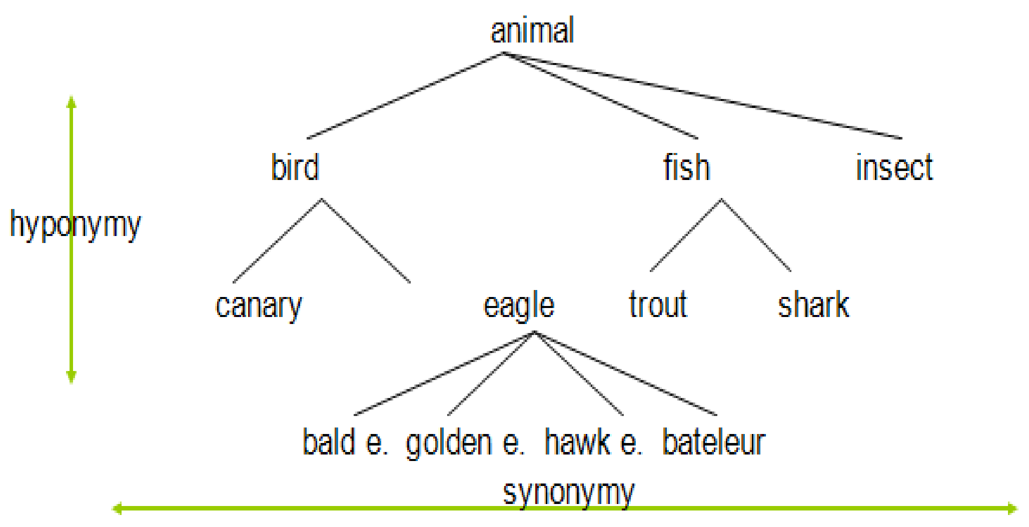


Figure 8. Example of WordNet hierarchical structure

WordNet is particularly suitable for similarity measures since nouns and verbs are organized into taxonomies of Is–A relations hierarchically where each node is a set of synonyms (synsets). A word will locate in multiple synsets at various locations in the taxonomy.

Measures of similarity between two synsets (senses) are based on information contained in an Is–A hierarchy. WordNet–based similarity measures are limited to making judgments between noun pairs (e.g., “cat” and “dog”) and verb pairs (e.g., “run” and “walk”) due to the limitation of Is-A hierarchies of WordNet. Although WordNet also contains adjectives and adverbs, semantic measures on adjectives and adverbs are not available as they are not organized into Is–A hierarchy.

A word may have several senses. Similarity between two words is the highest similarity value between two synsets (senses) coming from two compared words respectively.

The formula is as following:

$$sim\ (word1, word2)\ =\ \max sim\ (sense1, sense2)$$
$$sense1 \in \text{senses}\ (word1) \quad sense2 \in \text{senses}\ (word2)$$

For example, the noun “tree” has three senses in WordNet:

- A tall perennial woody plant having a main trunk and branches forming a distinct elevated crown; includes both gymnosperms and angiosperms
- A figure that branches from a single root; "genealogical tree"
- English actor and theatrical producer noted for his lavish productions of Shakespeare (1853-1917)

The noun “shrub” has one sense in WordNet:

- A low woody perennial plant usually having several major branches

The similarity value of each of the three senses of “tree” and “shrub” will be computed. The first sense of “tree” that gives the highest score will be assigned as similarity value between these two words “tree” and “shrub”.

There are two approaches that aim to calculate the semantic similarity from quite different angles. Path-length based measures are more intuitive, while information-content based measures approach is more theoretically sound.

3.2 Path-length based measures

Path-length based measures are to determine the similarity between two synsets (senses) as a function of the length of the path linking the synsets and on the position of the synsets in the WordNet. In this thesis, Path-length based measures including two similarity measures: Path and Wu & Palmer.

To express Path-length based measures more formally, the following notations are defined:

- *Synset*: the smallest unit in WordNet that represents a specific meaning of a word
- *Sense*: each sense (concept), which is represented by node in WordNet
- *Nodes*: a set of synonyms (synsets) represent one sense (concept)
- *Sub-sumer*: a shared parent of synsets
- *Least common sub-sumer (LCS)*: the lowest common ancestor node of two synsets (senses) in the hierarchy of WordNet.
- *Path-length*: the length of shortest path in the WordNet graph between two sense nodes

Figure 9 shows an example of the hyponym taxonomy in WordNet used for path-length similarity measure. The path-length between "car" and "auto" is 1. The path-length between "car" and "truck" is 3. The path-length between "car" and "object" is 8. The *LCS* of {car, auto} and {truck} is {automotive, motor vehicle}. The *LCS* of {automotive, motor} and {bike, bicycle} is {automotive, motor vehicle}. Please note that the path-length is measured in nodes/vertices rather than in links/edges. The length of the path between two members of the same synsets is 1 (synonym relations).

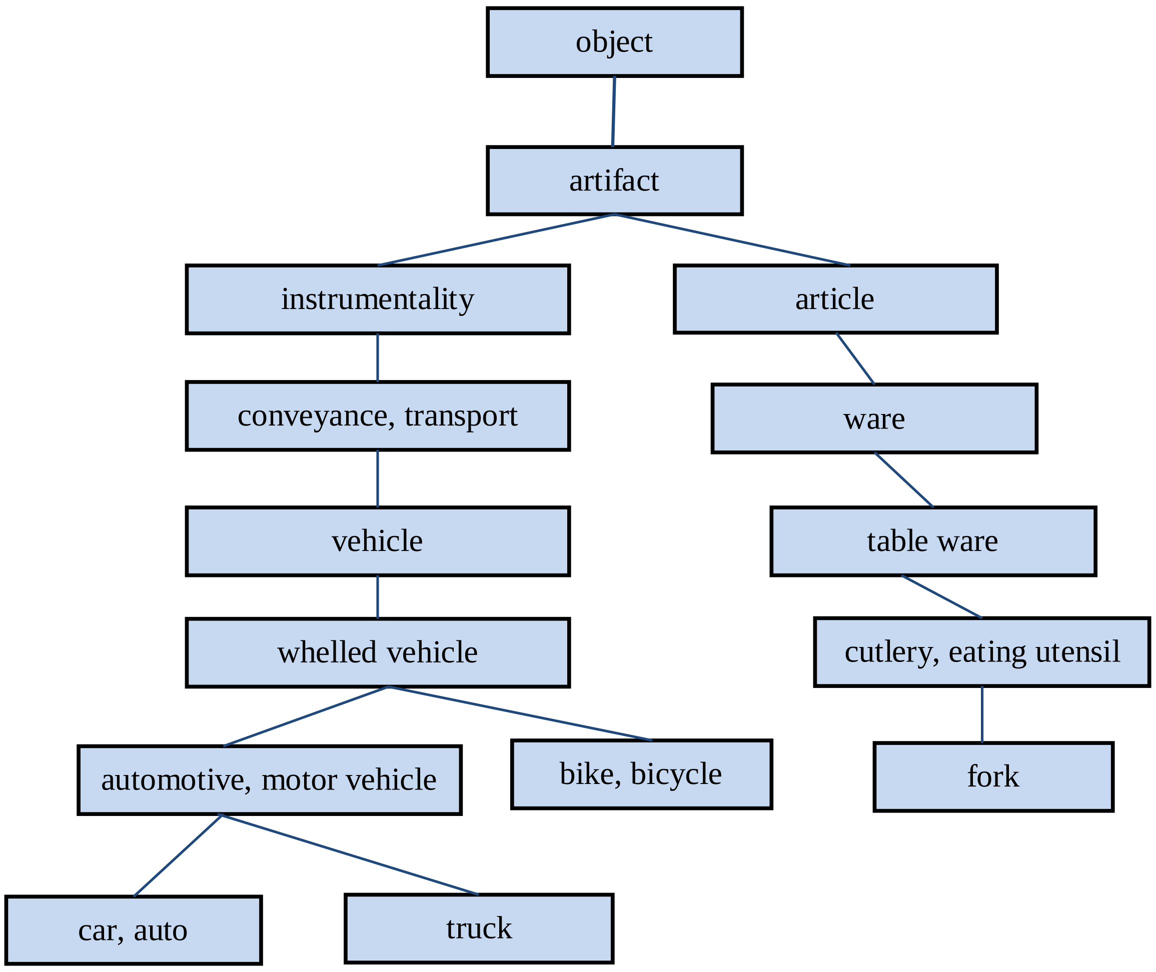


Figure 9. An example of the hyponym taxonomy in WordNet

3.2.1 Path

Path defines the similarity score as inversely proportional to the number of nodes along the shortest path between the synsets. The shortest possible path occurs when the two synsets are the same, in which case the length is 1. The formula can be calculated as follows:

$$Path = \frac{1}{shortest\ path\ length}$$

Thus, the maximum similarity value is 1. For example, the path-length between “car” and “truck” is 3 according to Figure 9. The similarity between “car” and “trunk” by path is calculated as follows:

$$Path\ (car, trunk) = \frac{1}{shortest\ path\ length} = \frac{1}{3} = 0.33$$

3.3.2 Wu & Palmer

Wu & Palmer [38] measure calculates similarity by considering the depths of the two synsets in the WordNet taxonomies, along with the depth of the *LCS*. The formula is as following:

$$Wup = \frac{2 * \text{depth}\ (LCS)}{\text{depth}\ (s1) + \text{depth}\ (s2)}$$

The *Wup* value is under the range of 0 < *Wup* <= 1. The score can never be zero because the depth of the *LCS* is never zero (the depth of the root of taxonomy is one). The score is one if the two input synsets are the same.

For example, according to the Figure 9, the depth of “car” is 8; the depth of “trunk” is 8. The *LCS* of “car” and “trunk” is {automotive, motor vehicle}. The depth of *LCS* is 7. The similarity between “car” and “trunk” by Wu & Palmer measure is as following:

$$Wup = \frac{2 * \text{depth}\ (LCS)}{\text{depth}\ (s1) + \text{depth}\ (s2)} = \frac{2 * 7}{8 + 8} = 0.875$$

3.3 Information-content Based Measures

Information-content based measure is a corpus–based measure of the specific application. Probabilistic information is added from a corpus to thesaurus hierarchy. The similarity between two synsets (senses) is defined as the measurement of the difference in information-content of the two terms as a function of their probability of occurrence in a corpus. WordNet is used as a statistical resource for computing the probabilities of occurrence of terms. In this thesis, Information-content Based Measure includes three similarity measures: Resnik, Lin and Jiang & Conrath. Figure 10 shows an example of WordNet hierarchy augmented with probabilities *P(s)*.

The following notations are defined:

- *s:* sense (concept) that is represented by node in the WordNet.
- *Words(s): a* set of words subsumed by *s*.
- *P(s):* the probability that a random word (*w*) in the corpus is an instance of *s*. *N* is the number of words in the corpus. The formula is as following:

$$P(s) = \frac{\sum_{w \epsilon Words(s)} count(w)}{N}$$

- *IC:* Information-content of *s*. The formula is as following:

$$IC(s) = -\log P(s)$$

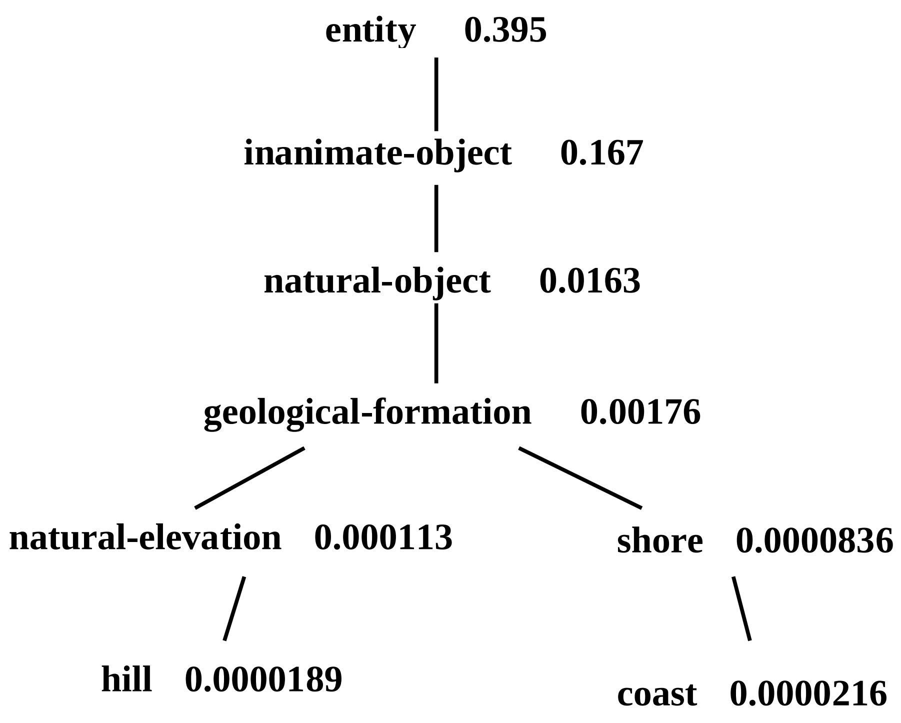


Figure 10. WordNet hierarchy augmented with probabilities *P(s)*

3.3.1 Resnik

The similarity between two words is related to their common information. The more two words have in common, the more similar they are. *Resnik* [39] measures the common information as the information-content (IC) of the Least Common Subsumer (LCS). The value will always be greater-than or equal-to zero. The formula is as following:

$$sim_{Resnik}(s,s') = -\log P(LCS(s,s'))$$

For example, according to Figure 10, the similarity between "hill" and "coast" by Resnik is calculated as following:

$$\mathrm{sim}_{\mathrm{Resnik}}(\mathrm{hill},\mathrm{coast}) = -\log \mathrm{P}\big(LCS(\mathrm{hill},\mathrm{coast})\big)$$

$$= -\log P(\text{geological} - \text{formation})$$

$$= -\log P(0.00176) = 2.75$$

3.3.2 Lin

Lin [40] modifies the Resnik. In this measure, similarity is measured not only by common information between two words but also by the difference between two words. The more information two words have in common, the more similar they are. The more differences between the information in two words, the less similar they are. Lin measures the similarity between two words by the ratio between the amounts of information needed to state the commonality of two words and the information needed to fully describe what two words are. The formula of Lin is as following:

$$sim_{lin}(s, s') = \frac{2 * \log P(LCS(s, s'))}{\log P(s) + \log P(s')}$$

For example, according to Figure 10, the similarity between "hill" and "coast" by Lin is calculated as following:

$$sim_{lin}(\text{hill}, \text{coast}) = \frac{2 * \log P(LCS(\text{hill}, \text{coast}))}{\log P(\text{hill}) + \log P(\text{coast})}$$

$$= \frac{2 * \log P(\text{geological} - \text{formation})}{\log P(\text{hill}) + \log P(\text{coast})}$$

$$= \frac{2 * \log P(0.00176)}{\log P(0.0000189) + \log P(0.0000216)}$$

$$= \frac{2 * (-2.75)}{(-4.72) + (-4.67)}$$

$$= 0.59$$

3.3.3 Jiang and Conrath

Jiang and Conrath [41] is another approach to measure the similarity between two words. It also takes into account the common information between two words and the difference between two words. The formula of Jiang and Conrath is as following:

$$sim_{jcn}(s,s') = \frac{1}{2 * \log P\big(LCS(s,s')\big) - (\log P(\text{s}) + \log P(s'))}$$

For example, according to Figure 10, the similarity between "hill" and "coast" by Jiang and Conrath is calculated as following:

$$sim_{jcn}(\text{hill, coast}) = \frac{1}{2 * \log P(LCS(\text{hill, coast})) - (\log P(\text{hill}) + \log P(\text{coast}))}$$

$$= \frac{1}{2 * \log P(\text{geological} - \text{formation}) - \big(\log P(\text{hill}) + \log P(\text{coast})\big)}$$

$$= \frac{1}{2*\log(0.00176)-(\log(0.0000189)+\log(0.0000216))}$$

$$= \frac{1}{2*(-2.75)-[(-4.72)+(-4.67)]}$$

$$= 0.26$$

4. Clustering

Clustering is a division of data into groups of similar objects. Representing the data by fewer clusters necessarily loses certain fine details, but achieves simplification. It is a method of unsupervised learning and a common technique widely used in many applications such as scientific data exploration, information retrieval and text mining, spatial database applications, Web analysis, CRM, marketing, medical diagnostics, computational biology, and many others [42]. The goal of clustering is to separate a finite unlabeled data set into a finite and discrete set of natural hidden data structures, rather than provide an accurate characterization of unobserved samples generated from the same probability distribution [43]. The clustering problem is defined to partition a set of data objects into subsets (called clusters) so that objects in the same cluster have similar features. The general clustering problem includes three sub-problems [44]: (a) the choice of the clustering algorithms; (b) decision of the number of the clusters; (c) selection of the evaluation function. Many clustering methods have been proposed in the literature. These methods can be roughly classified into following five main categories [43]: partitioning-based, hierarchical, density-based, grid-based, and model-based methods. However, partitioning-based and hierarchical algorithms are the most significant and widely used in clustering communities. Figure 11 illustrates an example of hierarchical clustering of data objects.

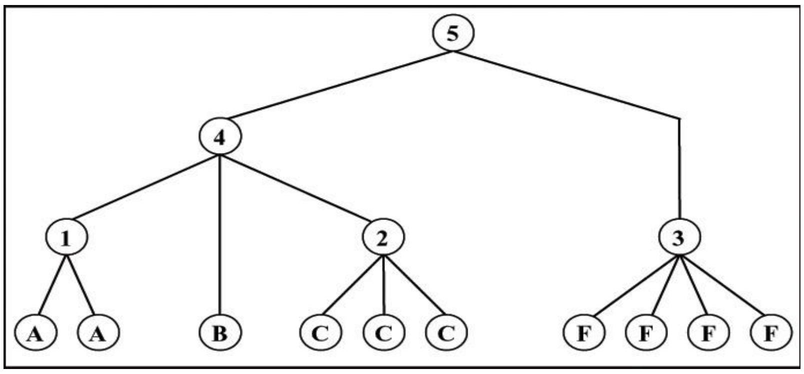


Figure 11. Example of hierarchical clustering of data objects

Hierarchical clustering provides an approach to increase the efficiency and effectiveness of the information retrieval process. For example, a dataset composed of words can be managed easier and searched faster by using fewer clusters representing the whole dataset grouped by hierarchical clustering.

In this section, we will first study the hierarchical clustering algorithms. Then, we discuss the stopping criteria for hierarchical clustering, which is an important sub problem in hierarchical clustering.

4.1 Hierarchical clustering algorithms

Hierarchical clustering methods are to build a hierarchy of clusters according to a given similarity function, which can be shown as a tree diagram called dendrogram. The top of dendrogram contains only one cluster including all the objects, whereas every cluster contains only one object at the bottom of dendrogram. Hierarchical clustering methods are generally classified into two types: agglomerative (bottom-up) methods and divisive (top-down) methods. Agglomerative algorithms treat each observation as a singleton cluster at the outset and then successively merge (or agglomerate) pairs of clusters until all clusters have been merged into a single cluster that contains all observations. Divisive clustering requires a method for splitting a cluster. It proceeds by splitting clusters recursively until individual observation is reached. In this thesis, hierarchical clustering is used with agglomerative strategy because it is more frequently used than top-down clustering.

The choice of which clusters to merge or split is determined by a linkage criterion, which is a function of the pairwise distances between objects. For agglomerative clustering, there are three main methods: single-link, complete-link and average-link, which differ in the similarity measures they employ. Single-linkage clustering, also called nearest neighbor technique, defines the distance between clusters as the distance between the closest pair of objects, where only pairs consisting of one object from each clusters are considered. Complete-linkage clustering, also called farthest neighbor, is opposite to single-linkage clustering. Distance between clusters is now defined as the distance between the most distant pair of objects, one from each cluster. In average link (or average linkage) clustering, two clusters whose average of all members have the smallest distance (two clusters with the smallest average pairwise distance) at each step. Figure 12 demonstrates a screenshot of a hierarchical clustering used in word clustering by semantic metrics. The input is a list of noun words, and the output is the hierarchical structure of the words by their semantic meaning.

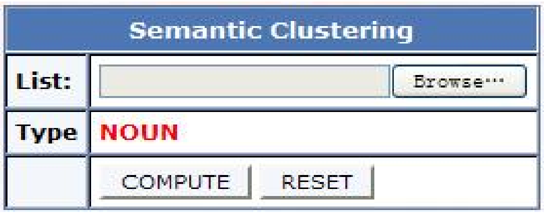


Figure 12. Hierarchical clustering demo

4.2 Stopping criteria

The number of clusters is an essential input parameter for clustering, which changes the behavior and the execution of clustering substantially. The output of clustering over the same dataset varies a lot according to the inputted parameter of the number of clusters. For a certain class of clustering algorithms such as k-means [45], the number of clusters *k* is specified already. Other algorithms, for example, DBSCAN [46] and OPTICS [47], do not require the specification of the number *k*.

Unlike nonhierarchical procedures that usually require user to predefine the number of the clusters, hierarchical methods routinely provide a series of results from *n* clusters to only one cluster (*n* is the number of objects in the data set). Therefore, a common question of automatically identifying an appropriate number of clusters to determine the optimal hierarchical level in a given dataset caught much attention from researchers.

Many statistical criteria or clustering validity indices have been investigated in the sense of determining the optimal number of clusters or discovering the natural structure of the datasets. When applied to the outcomes of hierarchical clustering methods, these techniques are sometimes referred as stopping criteria or stopping rules. Obviously, stopping criteria or clustering validity criteria must be carefully defined not only according to a presumably known data distribution of clusters but also to the specification of the input datasets. More importantly, those stopping criteria serve as a tool to measure the goodness of groups in clustering as well as a principle for selecting the “best” number of clusters meanwhile. A number of efforts have been made in the previous literatures, e.g., Milligan and Cooper [48] presented a comparison study over thirty validity indices for hierarchical clustering algorithms. Zhao, Xu, and Fränti [49] have proposed a new cluster validity index called WB-index. In this thesis, one common type of cluster validity index, called sum-of-squares based indexes is adopted to find the proper hierarchical level.

The sum-of-squares based indexes are usually used in hierarchical clustering as stopping criteria. In the analysis of sum-of-squares type indexes, within group variance and between group variance are calculated as sum-of-squares within cluster (*SSW*) and sum-of-squares between clusters (*SSB*) respectively:

$$SSW(k) = \sum_{i=1}^{m} \sum_{t}^{n_i} \sum_{l>t}^{ni} d(s_t, s_l)$$

$$SSB(k) = \frac{\sum_{t=1}^{m} \sum_{s>t}^{m} min_{i,j}\, d(s_i, s_j)_{s_i \in c_t,\, s_j \in c_s}}{m(m-1)/2}$$

where s_t, s_l, n_i correspond to t^{th} string in cluster k (c_k), l^{th} string in cluster k (c_k) and the size of the cluster c_k. $d(s_t, s_l a)$ equals to 1-$similarity\,(s_t, s_l)$. k is the number of clusters. c_t, c_s, s_i, s_j correspond to cluster t, cluster s, i^{th} string in cluster k (c_k) and j^{th} string in cluster k (c_k). In this thesis, we used three popular sum-of-squares based indexes shown in Table 3 as stopping criteria to detect correct level of hierarchical clustering.

Table 3. Three Sum-of-squares based indexes

Index Name	Formula
Calinski & Harabasz [50]	$CH = \frac{SSB/(\mathrm{m}-1)}{\mathrm{SSW}/(\mathrm{n}-\mathrm{m})}$
Hartigan [51]	$H = \log(SSB/SSW)$
WB-index [49]	$WB = m * SSW/SSB$

All these three sum-of-squares based indexes are based on *SSW* and *SSB*. In chapter 5, a simulation experiment based on these three indexes is done to find out a proper hierarchical level for hierarchical clustering. The result on both artificial generated and real data sets are presented in chapter 5. Clustering algorithms based on Lin, Jiang, Wup, Path similarity measures mentioned in chapter 3 are applied in the experiment. Analysis and conclusions are reported to determine the validity of these three selected indexes with word similarity measures.

5. Experiments and Result

In this chapter, we present the experimental results on the following areas:

- String metrics
- Word similarity
- Hierarchical clustering & stopping criteria

5.1 Experimental system

All the experiments were performed in the system as follows.

- Processor: Intel(R) Core(TM) 2 Duo CPU P7350 @ 2.00GHZ
- Memory: 3072 MB DDR2 800MHZ (PC6400)
- Hard disk: TOSHIBA mk3252GSX 320G
- Chipset: Intel Mobile 4 Series Chipset
- Operating system: Microsoft Windows Vista Home Premium (SP2)

All the real data comes from a project which is called MOPSI[9]. MOPSI is a Finnish abbreviation for **M**obiili **P**aikkatieto**S**ovellukset ja **I**nternet which can be translated in English as "Mobile Location-based Applications and Internet". The MOPSI project implements different location-based services and applications such as mobile search engines, data collection, user tracking and route recording.

[9] http://cs.joensuu.fi/mopsi/

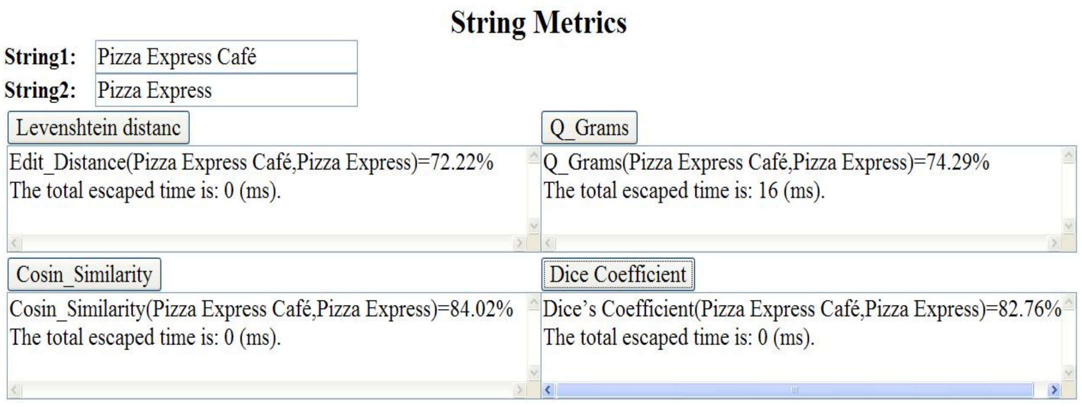
5.2 Experimental results and discussion

This section introduces the methodology of experiments and shows the experimental results.

5.2.1 String metrics

The experimental data contains 14 pairs of strings. All of them are the names of places. Some of them mean the same places with the different expressions.

String metrics have been programmed in JavaScript and user interface is normal web page. First, two experimental strings are needed to be inputted in string1 and string2 input box respectively. Secondly, choose the string metrics by pressing the button of the name of string metrics. Finally, the value of string similarity between two inputted strings appears in the blank area below the button. The Screenshot of string metrics user interface is given in Figure 13.


String Metrics

String1: Pizza Express Café

String2: Pizza Express

Edit_Distance(Pizza Express Café,Pizza Express)=72.22%
The total escaped time is: 0 (ms).

Q_Grams(Pizza Express Café,Pizza Express)=74.29%
The total escaped time is: 16 (ms).

Cosin_Similarity(Pizza Express Café,Pizza Express)=84.02%
The total escaped time is: 0 (ms).

Dice's Coefficient(Pizza Express Café,Pizza Express)=82.76%
The total escaped time is: 0 (ms).

Figure 13. Screenshot of string metrics user interface

When two strings with the similar meaning are inputted, cosine similarity has the highest value and edit distance has the lowest. As for running time, edit distance, cosine similarity and dice coefficient spent very short time, while Q-gram has spent the most time among 4 string metrics because Q-gram in this thesis has the most complicated steps to calculate the similarity value. Table 4 shows the experimental results on fourteen pairs of strings from MOPSI service list.

Table 4. Experimental results on our string metrics

Compared strings	Edit distance	Q-gram (q=2)	Consine similarity	Dice coefficient
Kioski Piirakkapaja Kioski Marttakahvio	47%	45%	50%	50%
Kauppa Kulta Keidas Kauppa Kulta Nalle	68%	67%	67%	67%
Pizza Express Café Pizza Express	72%	79%	82%	82%
Pizza Express Café Pizza Express Cafe	94%	85%	94%	94%
Pikko-Mikko baari baari Pikko-Mikko	29%	63%	85%	85%
Lounasravintola Pinja Ky - Ravintoloita Lounasravintola Pinja	54%	68%	63%	63%
Ravintola Beer Stop Pub Baari, Beer Stop R-kylä	39%	42%	50%	50%
Ravintola Foxie s Bar Siirry hakukenttään Baari Foxie Karsikko	31%	25%	24%	24%
Play baari Ravintola Bar Play - Ravintoloita	21%	31%	32%	32%
Average	51%	56%	63%	62%

The experimental results from string metrics lead to the following observations:

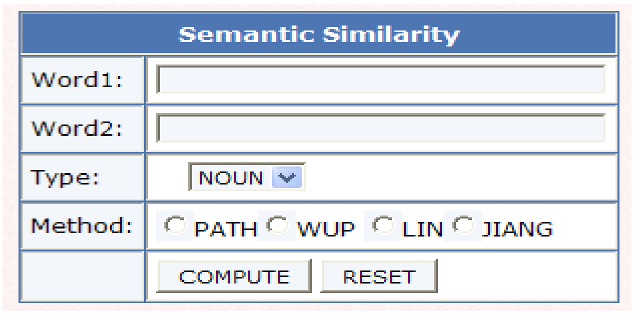
- In table 4, all pairs of strings means the same place except two pairs “Kioski Piirakkapaja”, “Kioski Marttakahvio” and “Kauppa Kulta Keidas”, “Kauppa Kulta Nalle”. They mean different places. But the value of string metrics for these two pairs of strings is not extremely low. String metrics are based only on lexicographic matching between terms without considering meaning.

- All four string metrics give excessively high scores to pairs of strings including terms that only differ by one or two characters. For example, two strings “Pizza Express Café” and “Pizza Express Cafe” achieve the highest similarity value 94% with Edit distance. The rest three string metrics Q-gram, Cosine Similarity and dice coefficient also give high similarity value of 85%, 94%, and 94% respectively.
- Edit distance gives inherent preference to the order of substrings in a string. For example, although two strings “Pikko-Mikko baari” and “baari Pikko-Mikko” contain the same two words, the edit distance metric assigns a low similarity score of 29.41% compared with other three string metrics Q-gram, Cosine Similarity and dice coefficient giving a relative high similarity value of 63%, 84.62% and 84% respectively.

5.2.2 Word similarity

The experimental datasets come from an experiment done by Miller and Charles [52]. In their study, 38 undergraduate subjects were given 30 pairs of nouns extracted from the previous study by Rubenstein and Goodenough [53], and were asked to rate the similarity of each pair on a scale from 0 (not similar) through 4 (perfect synonymy). In my thesis, the experimental results are converted into a new scale from 0 (not similar) to 1 (perfect synonymy), which are easy to be compared with the results of other similarity measures.

Similarity measures are coded in Java. The user interface is programmed in JSP (Java Server Pages). First, input experimental words in the Word1 and Word2 input box respectively. Secondly, choose noun or verb as word type for two inputted words. Then select one word similarity measure mentioned in this thesis. Finally, press the compute button to get the word similarity value of two inputted words. The Screenshot of word similarity user interface is given in Figure 14.



Semantic Similarity	
Word1:	
Word2:	
Type:	NOUN
Method:	○ PATH ○ WUP ○ LIN ○ JIANG
	COMPUTE RESET

Figure 14. Screenshot of word similarity user interface

Table 5 shows the performance of similarity measures on synthetic datasets.

Table 5. Performance of similarity measures on synthetic datasets

Word pair		Human judgment	Edge-counting based measures		Information-content based measures	
			Path	Wu & Palmer	Lin	Jiang& Conrath
car	automobile	0.78	1	1	1	1
gem	jewel	0.77	1	1	1	1
journey	voyage	0.77	0.97	0.92	0.84	0.88
boy	lad	0.75	0.97	0.93	0.86	0.88
coast	shore	0.74	0.97	0.91	0.98	0.99
asylum	madhouse	0.72	0.97	0.94	0.97	0.97
magician	wizard	0.70	1	1	1	1
midday	noon	0.68	1	1	1	1
furnace	stove	0.62	0.81	0.46	0.23	0.39
food	fruit	0.62	0.81	0.22	0.13	0.63
bird	cock	0.61	0.97	0.94	0.60	0.73
bird	crane	0.59	0.92	0.84	0.60	0.73
tool	implement	0.59	0.97	0.91	0.93	0.97
brother	monk	0.56	0.97	0.94	0.91	0.91
crane	implement	0.34	0.89	0.67	0.37	0.59
lad	brother	0.33	0.89	0.71	0.20	0.28
journey	car	0.23	0	0	0	0.33
monk	oracle	0.22	0.81	0.59	0.22	0.34
food	rooster	0.18	0.64	0.13	0.08	0.40
coast	hill	0.17	0.89	0.67	0.63	0.71
forest	graveyard	0.17	0.75	0.18	0.06	0.19
monk	slave	0.11	0.89	0.71	0.23	0.39
coast	forest	0.08	0.83	0.40	0.10	0.29
lad	wizard	0.08	0.89	0.71	0.21	0.32
chord	smile	0.03	0.72	0.44	0.28	0.35
glass	magician	0.02	0.75	0.31	0.21	0.68
noon	string	0.02	0	0	0	0.18
rooster	voyage	0.02	0	0	0	0.08
Average		0.41	0.80	0.63	0.49	0.61

The experimental results from word similarity lead to the following observations:

- In Edge-counting measures, Wu & Palmer performs better than plain path-length measure Path because its average value is closer to average human judgments value. Wu & Palmer considers the positions of the terms in the hierarchy.
- Information-content measures Lin performs well as its average similarity value is very close to average human judgment value.
- The information-content approach provides an improvement over the traditional edge-counting measure.

Edge-counting based measures are less accurate than Information-content based measures when they are applied to WordNet. One reason is that irregular densities of links between concepts result in unexpected conceptual distance outcomes. Without causing serious side effects elsewhere, the depth scaling factor does not adjust the overall measure well due to the general structure of WordNet (e.g. higher hierarchical sections tend to be too similar to each other). Furthermore, the Edge-counting based measures depend upon the subjectively pre-defined WordNet hierarchical structure. Since the original purpose of the design of the WordNet is for electronic lexical database instead of similarity computation purpose, some local network layer constructions may not be suitable for the direct distance manipulation.

5.2.3 Hierarchical clustering & stopping criteria

Both synthetic datasets and real datasets are used to implement a hierarchical clustering based on semantic similarity measures.

The synthetic datasets (car, pencil, pen, orange, cat, bus, bike, apple and dog) includes 9 nouns. According to the semantic meanings of each noun, the datasets are divided into 4 clusters (car bus bike, pencil pen, dog cat, apple orange). This is the proper hierarchical level. The ideal cluster number of the synthetic datasets is 4.

The real datasets contains 36 nouns, which are translated from the service list in MOPSI (http://cs.joensuu.fi/mopsi/tahti.php) by Google Translate. To make the cluster number of the real datasets more accurate, a survey of how to cluster the real datasets was done. 20 persons were asked to group these nouns into clusters based on semantic meaning. 2 persons divided the datasets into 10 clusters. 5 persons divided the datasets into 9 clusters. The rest of 13 persons divided the datasets into 8 clusters. Therefore, it is common to divide the real datasets into 8 clusters (hostel hostelry lodge auberge, film cinema movie, luncheon lunch snack collation meal, cafeteria coffeehouse cafe eatery restaurant, arena stadium, gymnasium gym, pharmacy shop market outlet stall store drugstore cubicle kiosk booth, barroom saloon taproom bar ginmill). This is the proper hierarchical level. The ideal cluster number of real dataset is 8.

Hierarchical clustering including single-linkage clustering and complete-linkage clustering is coded in Java. The user interface is programmed in JSP (Java Server Pages). First, choose the correct word type noun or verb for the dataset to be clustered. Then, press the browse button to upload a dataset in the TXT file format. Finally, press the compute button to get the result of both single-linkage clustering and complete-linkage clustering. The screenshot of hierarchical clustering user interface is given in Figure 15.

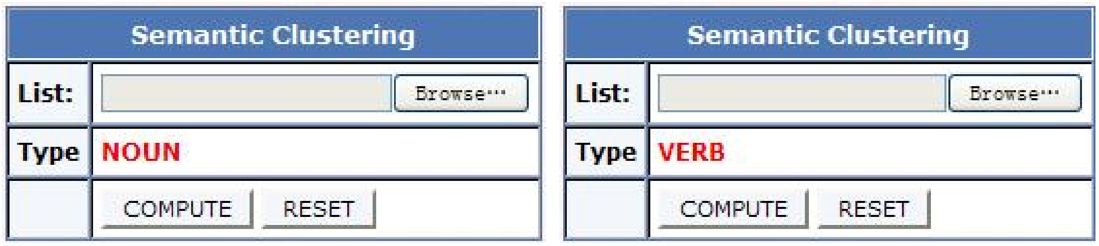
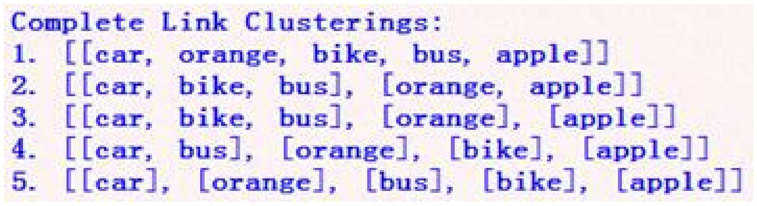


Figure 15. Screenshot of hierarchical clustering user interface

The screenshot of hierarchical clustering result is given in Figure 16.



```
Complete Link Clusterings:
1. [[car, orange, bike, bus, apple]]
2. [[car, bike, bus], [orange, apple]]
3. [[car, bike, bus], [orange], [apple]]
4. [[car, bus], [orange], [bike], [apple]]
5. [[car], [orange], [bus], [bike], [apple]]
```

Figure 16. Screenshot of hierarchical clustering result

Based on the results of hierarchical clustering, three validity indexes Calinski & Harabasz[50], Hartigan[51] and WB-index[49] are calculated by every level of hierarchical clustering and are plotted in clustering validity curve where x-axis indicates the number of clusters and y-axis presents the corresponding value of each clustering validity index.

Figure 17 shows Calinski & Harabasz validity index for synthetic datasets.

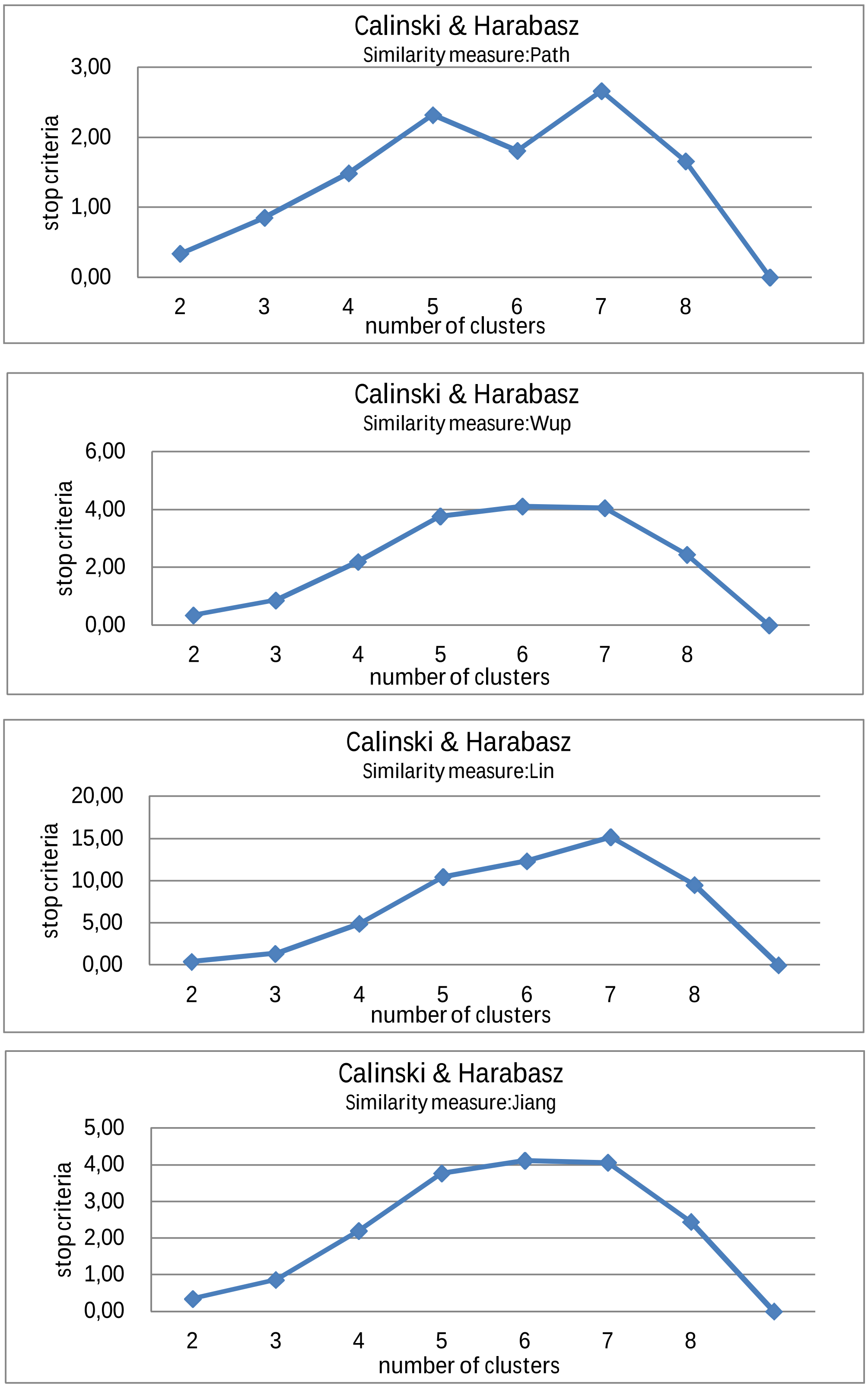


Figure 17. Calinski & Harabasz Cluster validity index for synthetic datasets

Figure 18 shows Hartigan validity index for synthetic datasets.

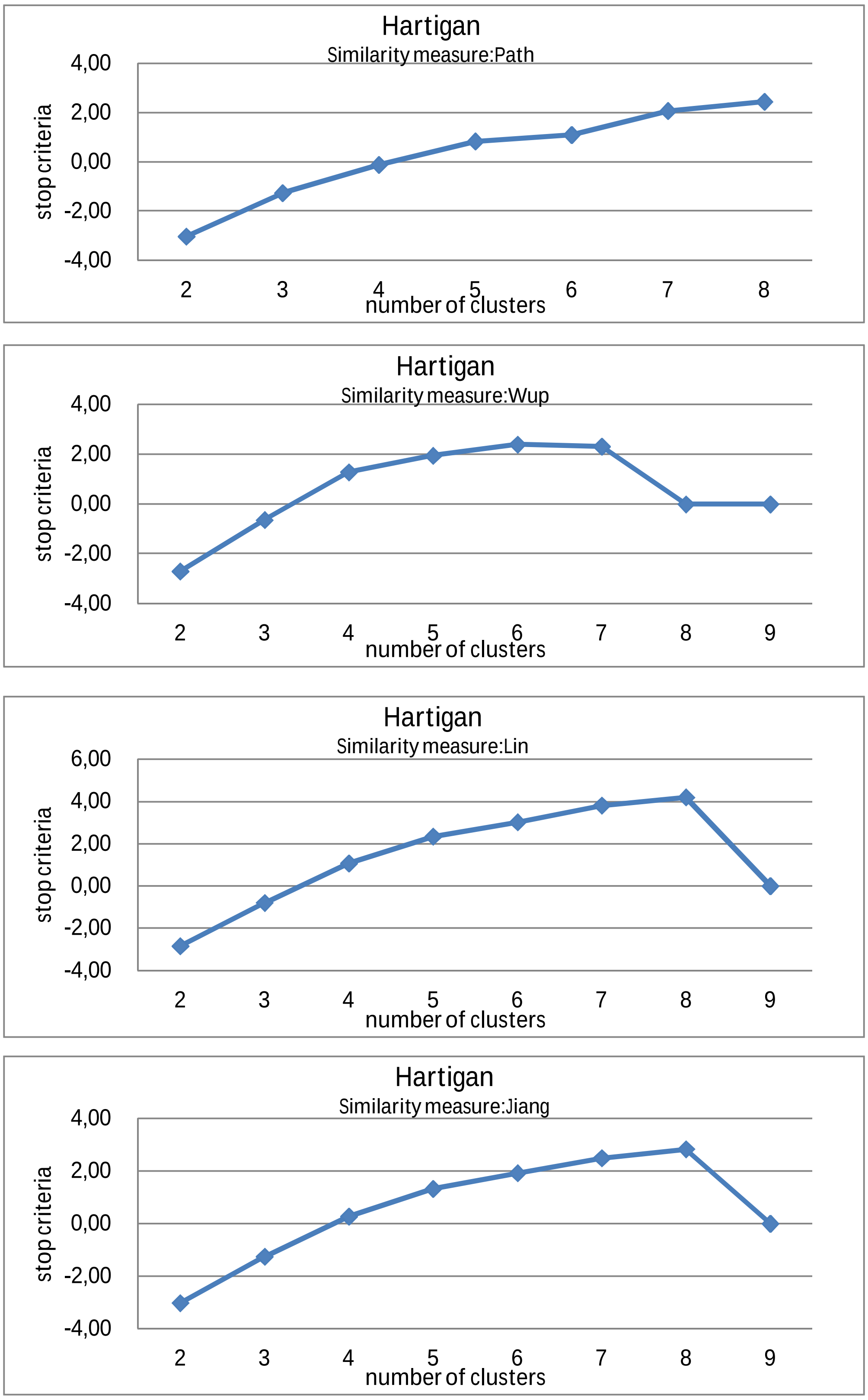
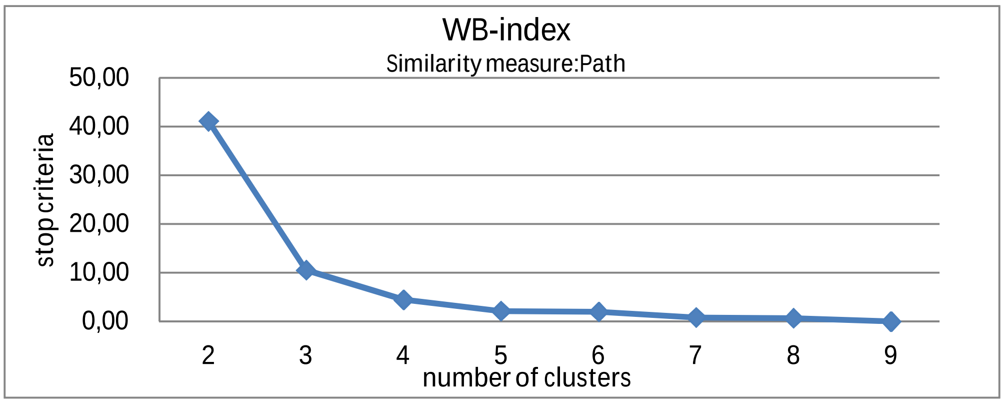
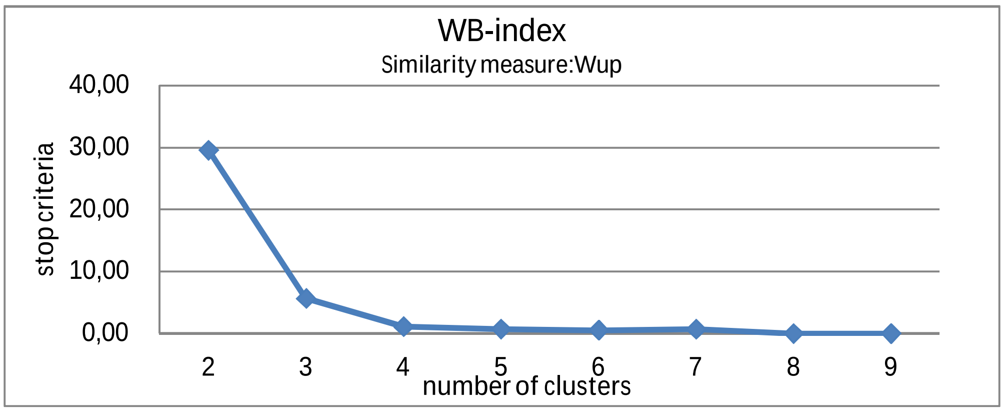
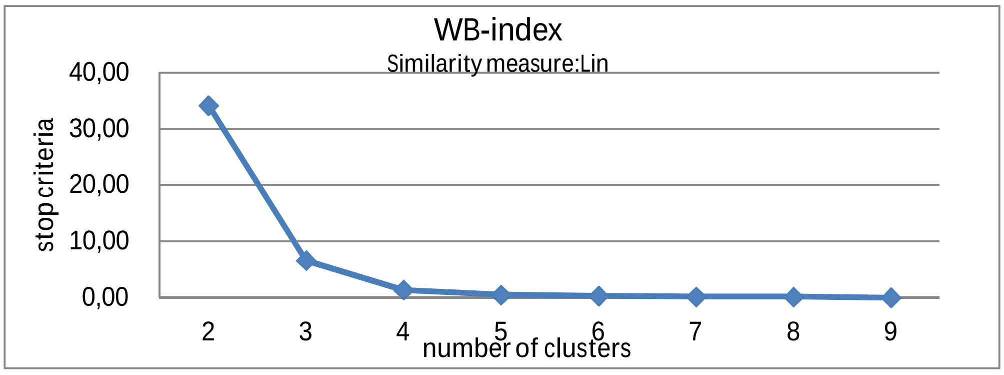


Figure 18. Hartigan Cluster validity index for synthetic datasets

Figure 19 shows WB-index validity index for synthetic datasets.







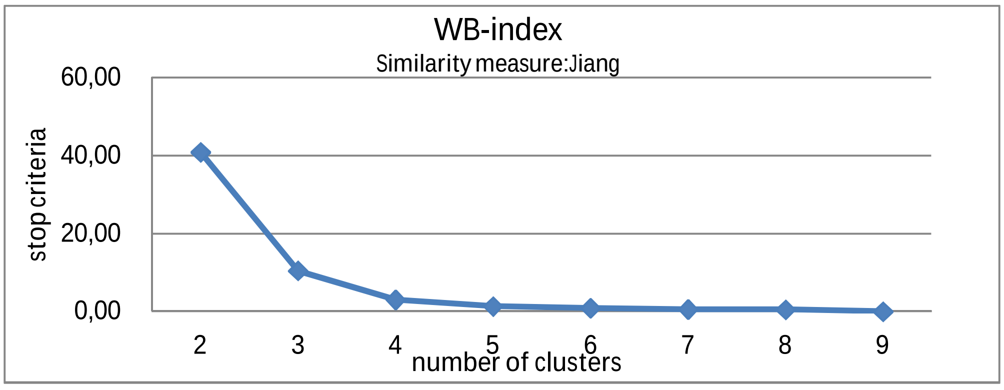


Figure 19. WB-index Cluster validity index for synthetic datasets

Figure 20 shows Calinski & Harabasz validity index for real datasets.

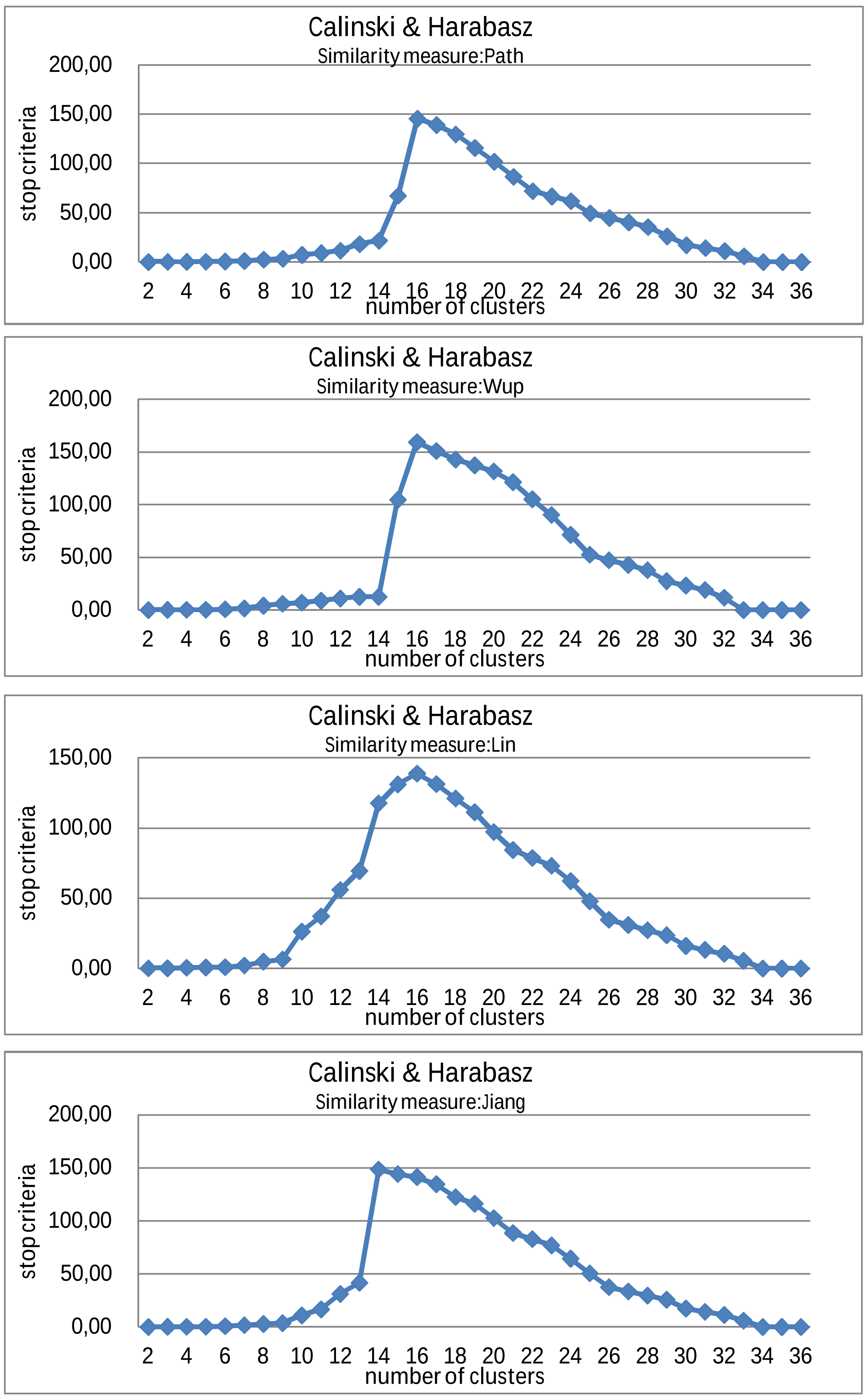
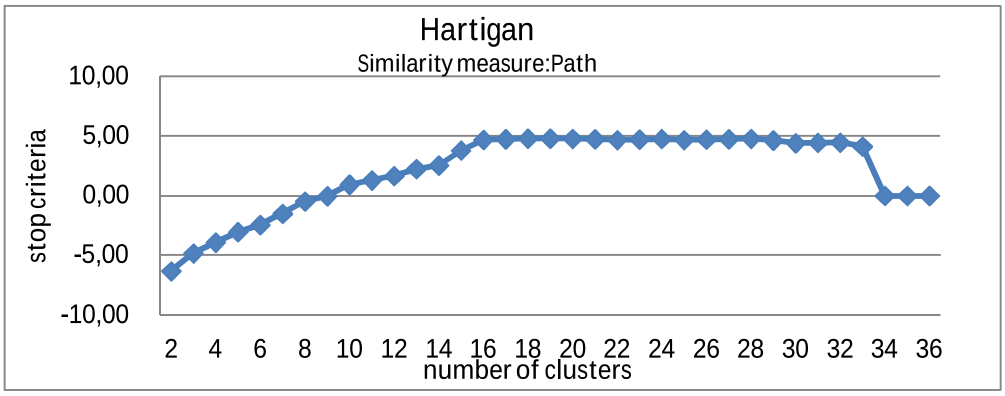
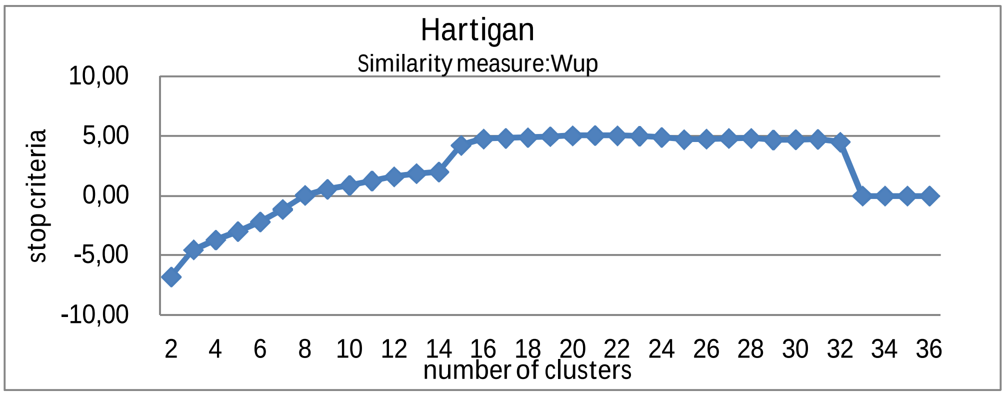
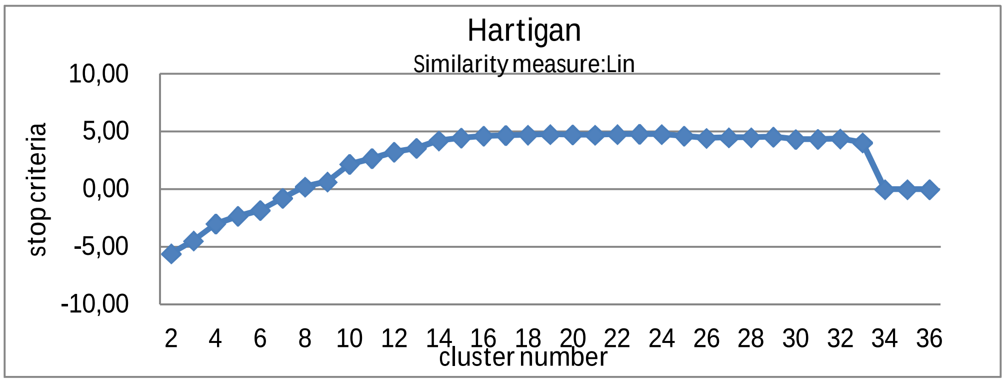


Figure 20. Calinski & Harabasz Cluster validity index for real datasets

Figure 21 shows Hartigan validity index for real datasets.







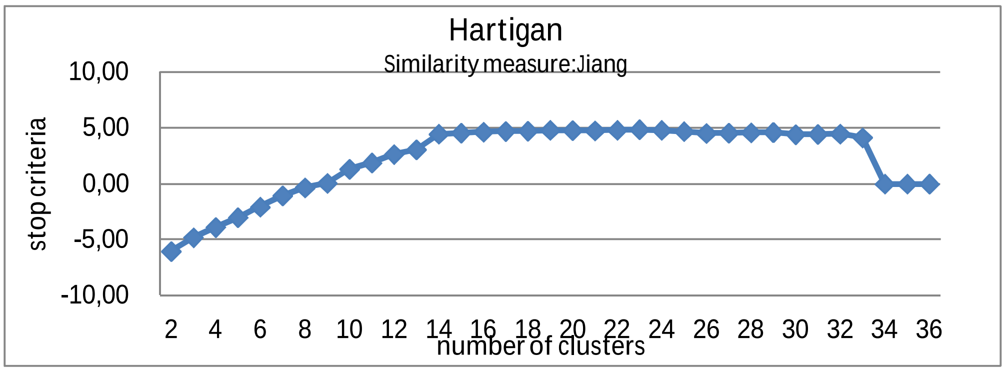


Figure 21. Hartigan Cluster validity index for real datasets

Figure 22 shows WB-index validity index for real datasets.

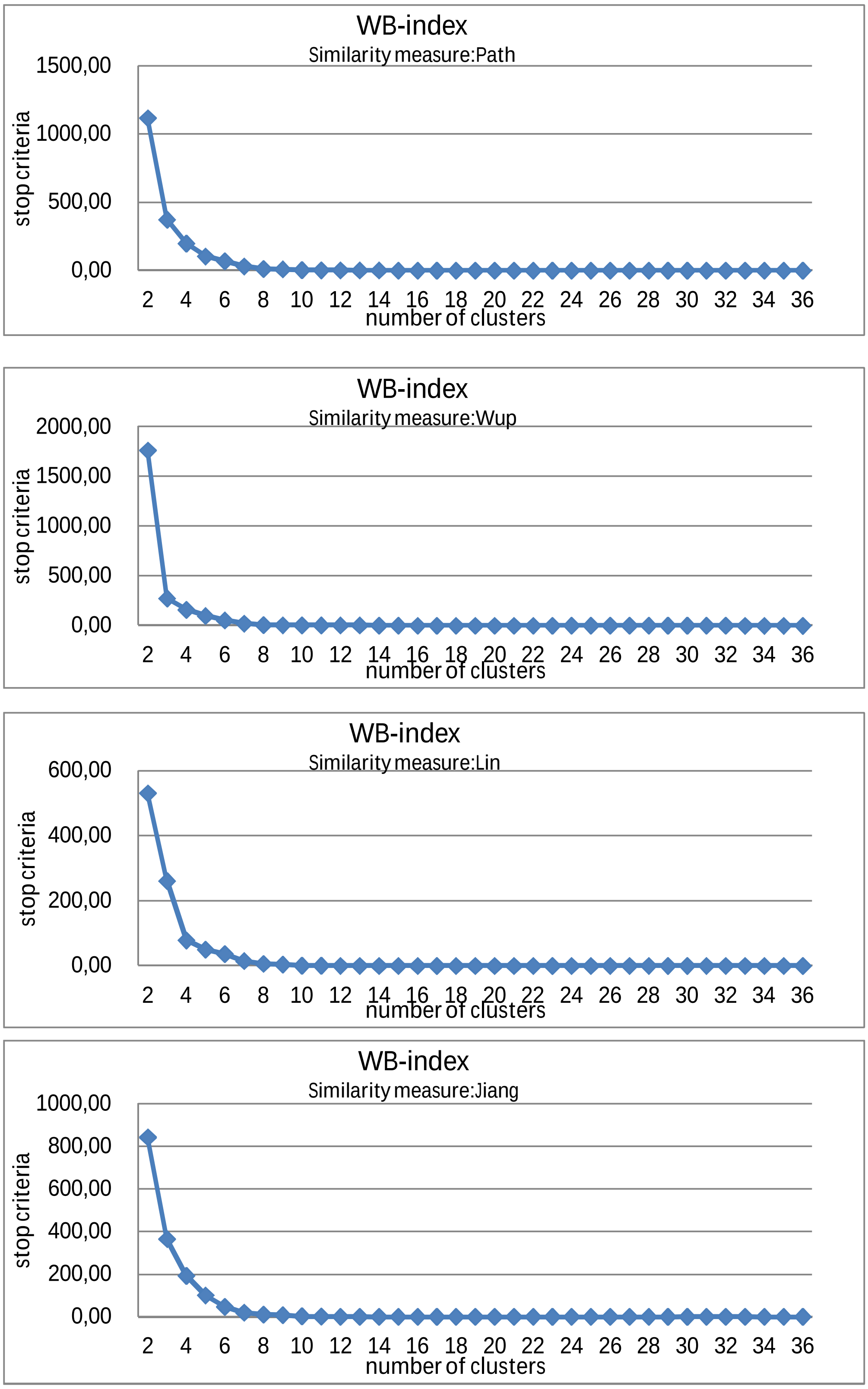


Figure 22. WB-index Cluster validity index for real datasets

The experimental results lead to the following observation:

- Hierarchical clustering can be used to cluster the words based on semantic meaning. Both single-linkage clustering and complete-linkage clustering with four different similarity measures find the proper hierarchical level for both synthetic datasets and real datasets.

- Calinski & Harabasz criterion usually works well in numeric datasets. The maximum value of stop criteria indicates the correct number of clusters. However, it does not work in text datasets. We noticed that the maximum value indicates the wrong number of clusters for both synthetic data and real data.

- WB-index has the same problem as Calinski & Harabasz cluster validity index. Although there is a significant local change (knee point) in the WB-index clustering validity curve, but it is not the suitable point for the stopping criteria to indicate the correct hierarchical level.
- Hartigan cluster validity index provides a reliable solution for determining the number of hierarchical clustering. The minimum positive point in Hartigan cluster validity index curve indicates the proper number of clusters, which can be seen as a reasonable stopping criterion.

6. Conclusions

At first, we have investigated string metrics and compared the performance of edit distance, Q-gram, cosine similarity and dice coefficient by conducting an experiment on real data from MOPSI project. According to the experimental results, Levenshtein distance performs better to detect the order of substrings in strings than other three string metrics. Cosine similarity and dice coefficient are suitable to find the pairs of strings including several same terms with the same order that only differ by one additional term.

Then, we have studied the approaches of measuring the word similarity between two words. WordNet, which is commonly used in this field, is employed in our study. By analyzing the experiment results from two categories of word similarity measures based on WordNet, Information-content based measures (Lin and Jiang & Conrath) perform better than traditional edge-counting measures (Wu & Palmer and Path).

Finally, we have studied hierarchical clustering algorithms and three sum-of-squares based indexes. We have performed the experiments on both synthetic datasets and real datasets for detecting the proper stopping criterion in hierarchical clustering. We found that Hartigan cluster validity index achieves the best result, which works well not only in numeric datasets, but also in text datasets compared with other two cluster validity index Calinski & Harabasz and WB-index. The minimum positive point in Hartigan cluster validity index curve indicates the proper level of hierarchical clusters, which can be seen as a reasonable stopping criterion.

As for the future work, more ontologies such as MeSH and Dublin are possible to be tested in order to see how these measures perform. A bigger variety of results is useful and helpful for understanding which specific word similarity measures performs better in different situation. This kind of information is critical and can be used in order to find the difficulties and the limitations of each word similarity measure and design new measures that would overcome those problems and perform better.

A new information retrieval model based on the word similarity is possible to be developed and applied to a web information retrieval system (search engine) used to retrieve both images and documents. Instead of using vector space model that cannot recognize synonyms or semantically similar terms, the new information retrieval model with the use of WordNet and word similarity will achieve more accurate and reliable results related to query terms entered by users.

Reference

[1] J.M. Ponte and W.B. Croft, “A language modeling approach to information retrieval,” *Proceedings of 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 275-281, 1998.

[2] N. Fuhr and T. Rölleke, “A probabilistic relational algebra for the integration of information retrieval and database systems,” *ACM Transactions on Information Systems (TOIS),* Volume 15, Issue 1, pp. 32-66, 1997.

[3] M. Gordon and P. Pathak, ”Finding information on the World Wide Web: the retrieval effectiveness of search engines,” *Information Processing & Management,* Volume 35, Issue 2, pp. 141-180, 1999.

[4] V.V. Raghavan and S.K.M. Wong, “A critical analysis of vector space model for information retrieval,” *Journal of the American Society for Information Science* Volume 3, Issue 5, pp. 279–287, 1986.

[5] K.L. Kwok, “A network approach to probabilistic information retrieval,” *ACM Transactions on Information Systems (TOIS),* Volume 13, Issue 3, pp. 324-353, 1995.

[6] D.H. Kraft, G. Bordogna and G. Pasi, “An extended fuzzy linguistic approach to generalize boolean information retrieval,” *Information Sciences-Applications*, Volume 2, Issue 3, pp.119-134, 1994.

[7] G. Salton, A. Wong and C.S. Yang, “A Vector Space Model for Automatic Indexing,” *Communications of the ACM*, Volume 18, nr.11, pp. 613–620, 1975.

[8] G.V. Bard, “Spelling-error tolerant, order-independent pass-phrases via the damerau-levenshtein string-edit distance metric,” *ACSW '07 Proceedings of the fifth Australasian symposium on ACSW frontiers,* Volume 68, pp. 117-124, 2007.

[9] W. E. Winkler, “The state record linkage and current research problems,” *Technical report, Statistics of Income Division*, Internal Revenue Service Publication, Wachington, DC, 1999.

[10] A. Monge and C. Elkan, “The field-matching problem: algorithm and applications,” *Proceedings of the second international Conference on Knowledge Discovery and Data Mining*, pp. 267-270, 1996.

[11] S. Tejada, C.A. Knoblock and S. Minton, “Learning object identification rules for information integration,” *Information Systems 26*, pp. 607–633, 2001.

[12] B. Wellner, J. Castaño and A. J. Pustejovsky, “Adaptive string similarity metrics for biomedical reference resolution,” *ISMB '05: Proceedings of the ACL-ISMB Workshop on Linking Biological Literature, Ontologies and Databases: Mining Biological Semantics*, pp. 9-16, 2005.

[13] M. Bates, “Models of natural language understanding,” *The National Academy of Sciences of the United States of America*, Volume 92, No. 22, pp. 9977–9982, 1995.

[14] R. Sinha and R. Mihalcea, “Unsupervised Graph-basedWord Sense Disambiguation Using Measures of Word Semantic Similarity,” *Proceedings of Semantic Computing, 2007. ICSC 2007*, pp. 363-369, 2007.

[15] G. Hirst and D. St-Onge, “Lexical chains as representations of context for the detection and correction of malapropisms,” *Fellbaum*, pp. 305–332, 1998.

[16] B. Alexander and H. Graeme, “Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures,” *Workshop on WordNet and Other Lexical Resources, Second meeting of the North American Chapter of the Association for Computational Linguistics,* Pittsburgh, June 2001

[17] J.S. Deogun and V.V. Raghavan, “User-Oriented Do cument Clustering: A Framework for Learning in Information Retrieval,” *Proceedings of the ACM SIGIR Conference,* pp. 157-163, Pisa, Italy, 1986.

[18] B. Nordhausen, “Conceptual Clustering Using Relational Information,” *AAAI- 86: Proceedings of the Fifth National Conference on Articial Intelligence*, Volume 1, pp. 508-512, Philadelphia, PA, 1986.

[19] A. McGregor, M. Hall, P. Lorier and J. Brunskill, “Flow Clustering Using Machine Learning Techniques,” *Proceedings of PAM*, pp. 205-214, 2004,

[20] A. Baraldi and P. Blonda, “A survey of fuzzy clustering algorithms for pattern recognition,” *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions*, Volume 29, Issue 6, pp. 778-785, 1999.

[21] R.T. Ng and J. Han, “Efficient and Effective Clustering Methods for Spatial Data Mining,” *Proceedings of the 20th Conference on VLDB*, Santiago, Chile, pp. 144-155, 1994.

[22] J. Matas and J. Kittler, “Spatial and feature space clustering: Applications in image analysis,” *Computer Analysis of Images and Pattern*, pp. 162-173, 1995.

[23] G. Navarro, “A guided tour to approximate string matching,” *ACM Computing Surveys (CSUR)*, Volume 33, Issue 1, pp. 31-88, 2001.

[24] G. Stoilos, G. Stamou and S. Kollias, “A String Metric for Ontology Alignment,” *Proceedings of International Semantic Web Conference' 2005*, Volume 3729, pp. 624-637, 2005.

[25] L. Jin and C. Li, “Selectivity estimation for fuzzy string predicates in large data sets,” *VLDB 05: Proceedings of the 31st international conference on Very* large data bases, pp. 397-408, 2005.

[26] C. Li, J. Lu and Y. Lu, “Efficient Merging and Filtering Algorithms for Approximate String Searches,” *Proceedings of IEEE 24th International Conference*, pp. 257-266, 2008.

[27] W. Cohen, P. Ravikumar and S. Fienberg, “A comparison of string metrics for matching names and records,” *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web*, pp. 73-78, 2003.

[28] V.I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” *Soviet Physics Doklady*, Volume 10, pp. 707–710, 1965.

[29] W. R, Hamming, “Error detecting and error correcting codes,” *Bell System Technical Journal 29*, pp. 147–160, 1950.

[30] F.J. Damerau, “A technique for computer detection and correction of spelling errors,” *Communications of the ACM*, 1964, pp. 171-176.

[31] L.R. Dice, “Measures of the Amount of Ecologic Association Between Species,” *Ecology 26*, pp. 297–302, 1945.

[32] A. Budanitsky, “Lexical Semantic Relatedness and its Application in Natural Language Processing,” *Technical report CSRG-390*, Department of Computer Science, University of Toronto, 1999.

[33] R. Rada, H. Mili, E. Bicknell, and M. Blettner, “Development and application of a metric on semantic nets,” *IEEE Transactions on Systems, Man, and Cybernetics*, 19 (1): pp. 17–30, 1989.

[34] C. Leacock and M. Chodorow, “Combining local context and WordNet similarity for word sense identification,” *Fellbaum*, pp. 265–283, 1998.

[35] E. Agirre and G. Rigau, “Word sense disambiguation using conceptual density”, *Proceedings of the 16th International Conference on Computational* Linguistics, pp. 16–22, 1996.

[36] M.M. Stark, R.F. Riesenfeld, “WordNet: An Electronic Lexical Database,” *Proceedings of 11th Eurographics Workshop on Rendering*, 1998.

[37] P. Vossen, “EuroWordNet: building a multilingual database with wordnets for European languages,” *The ELRA Newsletter*, Volume 3, Issue 1, pp. 7-10, 1998.

[38] Z. Wu and M. Palmer, “Verb semantics and lexical selection,” *Proceedings of 32nd Annual Meeting of the Association for Computational Linguistics*, pp. 133–138, 1994.

[39] P. Resnik, “Using information-content to evaluate semantic similarity,” *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 448–453, Montreal, 1995.

[40] D. Lin, “An information-theoretic definition of similarity,” *Proceedings of the 15th International Conferenc on Machine Learning, Madison*, pp. 448-453 1998.

[41] J.J. Jiang and D.W. Conrath, “Semantic similarity based on corpus statistics and lexical taxonomy,” *Proceedings of International Conference on Research in Computational Linguistics*, Taiwan, 1997.

[42] R. Xu and D. Wunsch, “Survey of clustering algorithms,” *IEEE Trans on Neural Networks*, Volume 16, Issue 3, pp. 645-678, 2005.

[43] A. Baraldi and E. Alpaydin, “Constructive feedforward ART clustering networks—Part I and II,” *IEEE Trans on Neural Networks*, Volume 13, Issue 3, pp. 645–677, 2002.

[44] P. Fränti and J. Kivijärvi, “Randomised local search algorithm for the clustering problem,” *Pattern Analysis and Applications*, Volume 3, pp. 358-369, 2000.

[45] J. MacQueen, “Some methods of classification and analysis of multivariate observations,” *Proceedings of* 5*th Berkeley Symposium Matchmatical Statistical Probability*, Volume l, Issue 1, pp. 281-297, 1967.

[46] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” *Proceedings of Int. Conf. on Knowledge Discovery and Data Mining*, pp. 226-231, 1996.

[47] M. Ankerst, M.M. Breunig, H.-P. Kriegel and J. Sander, “OPTICS: Ordering Points To Identify the Clustering Structure,” *ACM SIGMOD international conference on Management of data*, pp. 49-60, 1999.

[48] R.A. Hansen and G.W. Milligan, “Objective assessment of cluster analysis output: Theoretical considerations and empirical findings,” *Proceedings of the American Institute for Decision Sciences*, pp. 314-316, 1981.

[49] Q. Zhao, M. Xu and P. Fränti, “Sum-of-Square Based Cluster Validity Index and Significance Analysis,” *ICANNGA'09, LNCS 5495*, pp.313–322, 2009.

[50] T. Calinski and J. Harabasz, “A dendrite method for cluster analysis,” *Communication in statistics 3*, pp. 1–27, 1974.

[51] J.A. Hartigan, *Clustering algorithms*, Wiley, New York and London, 1975.

[52] G.A. Miller and W.G. Charles, “Contextual correlates of semantic similarity,” *Language and Cognitive Processes*, Volume 6, Issue 1, pp. 1–28, 1991.

[53] H. Rubenstein and J. Goodenough, “Contextual correlates of synonymy,” *Communications of the ACM* , Volume 8, Issue 10, pp. 627–633, 1965.