Reducing the Search Space of Ontology Alignment Using Clustering Techniques

Zhiming Gao

Examiner: Patrick Lambrix
Supervisors: Kristian Sandahl, Yin Chen
Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finnslösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/.

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a period of 25 years starting from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© Zhiming Gao
Abstract

With the emerging amount of information available in the internet, how to make full use of this information becomes an urgent issue. One of the solutions is using ontology alignment to aggregate different sources of information in order to get comprehensive and complete information. Scalability is a problem regarding the ontology alignment and it can be settled down by reducing the search space of mapping suggestions. In this paper we propose an automated procedure mainly using different clustering techniques to prune the search space. The main focus of this paper is to evaluate different clustering related techniques to be applied in our system. K-means, Chameleon and Birch have been studied and evaluated, every parameter in these clustering algorithms is studied by doing experiments separately, in order to find the best clustering setting to the ontology clustering problem. Four different similarity assignment methods are researched and analyzed as well. Tfidf vectors and cosine similarity are used to identify the similar clusters in the two ontologies, experiments about threshold of cosine similarity are made to get the most suitable value.

Our system successfully builds an automated procedure to generate reduced search space for ontology alignment, on one hand, the result shows that it reduces twenty to ninety times of comparisons that the ontology alignment was supposed to make, the precision goes up as well. On the other hand, it only needs one to two minutes of execution time, meanwhile the recall and f-score only drop down a little bit. The trade-off is acceptable for the ontology alignment system which will take tens of minutes to generate the ontology alignment of the same ontology set. As a result, the large scale ontology alignment becomes more computable and feasible.

Keywords: Ontology alignment; Clustering; automated procedure; Large scale ontology; Evaluation
Acknowledgements

First of all, I want to thank my lab supervisor Patrick Lambrix to give me this interesting topic, and help me whenever I needed anything, thanks for his patient discussion with me. Thanks to Kristian Sandahl, the school supervisor, who talked us about how to write thesis and gave me feedback for my thesis. I want to thank Yin Chen, my supervisor in China, who helped me during whole my graduate student life. Thank you for the good lectures you brought to me when I was doing my bachelor. I appreciate every student in our double degree program, who have been accompanying me for this whole year, I feel thankful for the good experience you brought to me.
Contents

Uphovsrätt ............................................................................................................... II
Copyright.................................................................................................................. II
Abstract ................................................................................................................... III
Acknowledgements ................................................................................................. IV
1 Chapter 1 Introduction ......................................................................................... 1
   1.1 Background ..................................................................................................... 1
   1.2 The purpose of project .................................................................................. 3
   1.3 The status of related research ....................................................................... 3
       1.3.1 Clustering ............................................................................................... 3
       1.3.2 Similarity assignment ............................................................................. 9
       1.3.3 Similar clusters identification ............................................................... 12
       1.3.4 Related Study ....................................................................................... 14
   1.4 Main content and organization of the thesis ............................................... 15

2 Chapter 2 System Requirement Analysis ......................................................... 16
   2.1 The goal of the system .................................................................................. 16
   2.2 The functional requirements ....................................................................... 17
   2.3 The non-functional requirements .................................................................. 20
   2.4 Brief summary ............................................................................................... 22

3 Chapter 3 System Design .................................................................................... 23
   3.1 High level architecture ............................................................................... 23
   3.2 System overview ............................................................................................ 23
   3.3 Ontology Parsing and Similarity Assignment .............................................. 25
   3.4 Clustering ..................................................................................................... 28
       3.4.1 Chameleon .............................................................................................. 29
       3.4.2 K-Means ................................................................................................. 29
       3.4.3 Birch ...................................................................................................... 31
   3.5 Label Parsing .................................................................................................. 33
   3.6 Similar Clusters Identification ...................................................................... 34
Chapter 1 Introduction

1.1 Background

In computer science and information science, an ontology is a formal naming and definition of the types, properties, and interrelationships of the entities that fundamentally exist for a particular domain of discourse. It is thus a practical application of philosophical ontology, with a taxonomy.

Intuitively, ontologies define the basic terms and relations of a domain of interest, as well as the rules for combining these terms and relations [1]. In recent years many ontologies have been developed. Ontologies serve as a key technology for the Semantic Web [2]. The benefits of using ontologies include reuse, sharing and portability of knowledge across platforms, and improved documentation, maintenance, and reliability. Ontologies lead to a better understanding of a field and to more effective and efficient handling of information in that field. Involvement of ontologies are of great essential in some of the grand challenges, e.g. genomics research. There is much international research cooperation for the development of ontologies. The number of researchers working on methods and tools for supporting ontology engineering is constantly growing and more and more researchers and companies use ontologies in their daily work [3].

As different parties may adopt different standard for ontology, and the completeness of the data differs, combining heterogeneous ontologies in order to gain a more comprehensive and complete knowledge is a crucial question. Ontology alignment is the technique aiming to solve this problem.

Ontology alignment is the result of ontology matching [4]. Ontology matching aims at finding correspondences between semantically related entities of different ontologies. These correspondences may stand for equivalence as well as other relations, such as consequence, subsumption, or disjointness, between ontology entities. Ontology entities, in turn, usually denote the named entities of ontologies, such as classes, properties or individuals. However, these entities may also be more complex expressions, such as formulas, concept definitions, queries or term building expressions. Ontology alignments, result of ontology matching, can thus express with various degrees of precision the relations between the ontologies under consideration.
Alignments can be used for various tasks, such as ontology merging, query answering, data translation or for browsing the semantic web. Matching ontologies enables the knowledge and data expressed in the matched ontologies to interoperate. It is thus of utmost importance toward making use of heterogeneous ontologies.

Many different ontology matching solutions have been proposed so far from various viewpoints, e.g., databases, information systems, artificial intelligence. They take advantage of various properties of ontologies, e.g., structures, data instances, semantics, or labels, and use techniques from different fields, e.g., statistics and data analysis, machine learning, automated reasoning, and linguistics. These solutions share some techniques and tackle similar problems, but differ in the way they combine and exploit their results.

The ontology matching process usually takes a lot of time, so the efficiency of ontology matching is of great essential while dealing with large ontologies. As a result, in [5], the authors identify efficiency of matching techniques as a challenge for ontology matching. The efficiency of matchers is of prime importance in dynamic applications, especially, when a user cannot wait too long for the system to respond or when memory is limited. In the meantime, to obtain a complete and accurate aligned ontology, involvement of domain expert is necessary, the suggestions proposed by some existing alignment systems will be too large and time-consuming for domain expert. Reduce the workload of domain experts is an important focus of ontology study, different methods can be utilized in order to achieve this goal.

Many methods have been proposed to promote efficiency, a straightforward approach is to reduce the number of pairwise comparisons in favor of a top-down strategy as implemented in QOM [3], or to avoid using computationally expensive matching methods, such as in RiMOM [6]. RiMOM uses the techniques of suppressing the structure-based strategies and applying only a simple version of the linguistic-based strategies to reduce computation. Another direction is to use segment-based approach to avoid exhaustive pairwise comparisons, which seems to be very promising when handling large ontologies, e.g., COMA++ [7], Anchor-Flood [8] and SAMBO [28, 32]. These methods target at matching only the similar enough segments (clusters). But this type of study has to be further and more systematically developed. For example, some systems need to merge similar segments manually, so it is also worth investigating how to automatically partition large ontologies into proper segments [9]. The efficiency of the integration of various matchers can be improved by minimizing
(with the help of clustering, such as in PORSCHE [10], XClust [11] and LOAD [12]) the target search space for ontology alignment systems.

1.2 The purpose of project

The purpose of this project is to construct a certain series of automated processes leading to reduced search space for ontology alignment. The most important step is using clustering algorithm to cluster the two ontologies, so that by identifying similar clusters, only the concepts in those cluster pairs, with each concept in each cluster, will be considered to generate ontology matching suggestions. In the meantime, the similarity between concepts needs considerations and to be evaluated as well.

Following questions are answered in this thesis:

- Which clustering algorithm is best suited for the purpose of ontology clustering? How to evaluate that?
- Which similarity assignment method is suitable for the task of ontology clustering?
- What kind of procedures can generate best result for the purpose of reducing search space of ontology alignment?

1.3 The status of related research

To answer the first question in chapter 1.2, different clustering methods have to be studied and evaluated. Clustering can lead to the partition of ontologies, by aligning only the similar clusters, the search space for ontology alignment system can thus be reduced.

1.3.1 Clustering

Clustering [13] refers to the process of partitioning given data sets to a homogeneous data set group. This is based on features such that objects that tend to be similar are classified into one group and objects that tend to be dissimilar are classified into a different group. Clustering algorithms can be divided into four major kinds: hierarchical clustering, centroid-based clustering, distribution-based clustering and density-based clustering. Under the certain category, dozens of algorithms and variations exists with their different applicability, pros and cons. This paper presents the usage of hierarchical clustering and centroid-based clustering.
1.3.1.1 Hierarchical clustering

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. With different threshold, the cluster result is different with various precision. Strategies for hierarchical clustering generally fall into two types [14]:

- Agglomerative: A "bottom up" approach with each observation starts in its own cluster, and pairs of clusters are merged as one moves up the hierarchy.
- Divisive: A "top down" approach with all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

As more researches have been done on agglomerative way than the divisive one and the less complexity of implementing it, agglomerative algorithms are chosen. Among the many systems, Chameleon [15] and Birch [16] in many cases outperform the other algorithms and hence become our first choices.

1.3.1.1.1 Chameleon

Chameleon [15] is a hierarchical clustering method. The key feature of Chameleon algorithm is that it accounts for both interconnectivity and closeness of clusters in identifying the most similar pair of clusters. With these two measurements, it thus avoids the limitations of only focusing on one aspect, for example, another popular hierarchical clustering algorithm ROCK [17] only considers interconnectivity. Furthermore, Chameleon uses a novel approach to model the degree of interconnectivity and closeness between each pair of clusters. This approach considers the internal characteristics of the clusters themselves. As a result, it no longer depends on a static, user-supplied model and can automatically adapt to the internal characteristics of the merged clusters.

Chameleon operates on a sparse graph in which nodes represent data items, and weighted edges represent similarities among the data items. This sparse graph representation allows Chameleon to scale up to large data sets and to successfully use data sets that are available only in similarity space and not in metric spaces. Data sets in a metric space have a fixed number of attributes for each data item, whereas data sets in a similarity space only provide similarities between data items.

Chameleon finds the clusters in the data set by using a two-phase algorithm. In the first phase, Chameleon uses a graph-partitioning algorithm to cluster the data
items into several relatively small sub-clusters. Then in the second phase, it uses an algorithm to find the desired clusters by repeatedly combining these sub-clusters.

The first phase’s graph-partitioning algorithm can have many selections, in the original paper [15] it uses K-nearest-neighbor (KNN) to get the initial clusters. In second step, relative interconnectivity (RI) and relative closeness (RC) of clusters are used to judge whether the two clusters can be merged. The definitions of them are as follows.

Assume $C_i$ and $C_j$ are two clusters, RI is defined as the absolute interconnectivity between $C_i$ and $C_j$ normalized with respect to the internal interconnectivity of the two clusters $C_i$ and $C_j$. The absolute inter-connectivity between a pair of clusters is defined to be as the sum of the weight of the edges that connect vertices in $C_i$ to vertices in $C_j$, which is essentially the edge-cut of the cluster containing both $C_i$ and $C_j$ such that the cluster is broken into $C_i$ and $C_j$. We denote this by $EC(c_i,c_j)$. The internal inter-connectivity of a cluster $C_i$ is captured by the size of its min-cut bisector $EC_c_i$. See (1-1) for formula of the RI definition.

$$RI(C_i, C_j) = \frac{|EC(c_i,c_j)|}{|EC_c_i|+|EC_c_j|},$$  

(1-1)

In the meantime, RC represents the absolute closeness between $C_i$ and $C_j$ normalized with respect to the internal closeness of the two clusters. There are many ways to define it, in [15] the authors choose to compute the average similarity between the points in $C_i$ that are connected to points in $C_j$. Since these connections are determined using the k-nearest neighbor graph, their average strength provides a very good measure of the affinity between the data items along the interface layer of the two sub-clusters as well as being resistive to outliers and noise. Define $\bar{S}_{EC_c_i}$ and $\bar{S}_{EC_c_j}$ are the average weights of the edges that belong in the min-cut bisector of clusters $C_i$ and $C_j$, respectively, and $S_{EC(c_i,c_j)}$ is the average weight of the edges that connect vertices in $C_i$ to vertices in $C_j$. See (1-2) for the formula definition of RC.
Chameleon selects cluster pairs to merge for which both RI and RC are high. That is, it selects clusters that are well interconnected as well as close together. It usually yields reasonable clustering result with appropriate clustered data set.

There is an equation to combine RC and RI as well, a parameter $\vartheta$ is used to trade-off RC and RI. The equation is shown as follows (1-3)

$$F = RC \times RI^{\vartheta}$$

If $\vartheta$ is larger than 1, it means the RI takes more part in the equation. If $\vartheta$ is less than 1, the RC is dominant in the equation. If $\vartheta$ is 1, it means RI and RC are of equal importance. This parameter can be tuned in order to control the priority of RI and RC.

1.3.1.1.2 BIRCH

BIRCH [16] (balanced iterative reducing and clustering using hierarchies) is an unsupervised data mining algorithm used to perform hierarchical clustering over particularly large data-sets. One of its advantages is its ability to incrementally and dynamically cluster incoming, multi-dimensional metric data points in an attempt to produce a good quality clustering for a given set of resources. It usually just needs a single scan of the data.

The algorithm takes N data points as input, represented as real-valued vectors, and probably a desired number of clusters K. Four phases are operated, the second and fourth of which are optional.

Firstly, build a CF tree using the N data points. A CF tree has following definitions:

- Given a set of N d-dimensional data points, the clustering feature CF of the set is defined as the triple $CF = (N, LS, SS)$, where LS is the linear sum and SS being the square sum of data points.
- Clustering features are organized in a CF tree, a height-balanced tree with two parameters: branching factor B and the threshold T. Each non-leaf node contains at most B entries of the form [CF, child], child of each entry...
represents the pointer to a child node of the cluster, and CF being the clustering features of that child node. A leaf node contains at most L entries each of the form [CF]. It also has two pointers prev and next which are used to chain all leaf nodes together, so that one can easily scan all leaves quickly. Parameter T determines the tree size. It is a very compact representation of the dataset because each entry in a leaf node is not a single data point but a subcluster.

In the second phase, the algorithm scans all the leaf entries in the initial CF tree to rebuild a smaller CF tree, while removing outliers and grouping crowded subclusters into larger ones.

In the third step an existing clustering algorithm is used to cluster leaf entries. It provides the flexibility of allowing the researcher to specify either the desired diameter threshold for clusters or the desired number of clusters. After this phase, a set of clusters is obtained that captures major distribution pattern in the data.

Fourth step is used to deal with possible existence of minor and localized inaccuracies. The centroids of the clusters produced out of previous phase are used as seeds and redistribute all data points to its closest seeds to obtain a new set of clusters. This step could also help researchers abandon some outliers. Outliers are points that are too far from its closest seed.

1.3.1.2 Centroid based clustering

In centroid-based clustering, clusters are represented by a central vector, which may not necessarily be a member of the data set. This type of clustering algorithm usually has a fixed number k, indicating the dataset is clustered into k clusters. A typical instance of this category is K-means clustering [18]. K-means clustering gives a formal definition as an optimization problem: find the k cluster centers and assign the objects to the nearest cluster center, such that the squared distances of each object to its cluster centroid are minimized.

The optimization problem itself is known to be NP-hard, and thus the common approach is to search for approximate solutions. In centroid-based clustering, a particularly well known approximatively method is k-means algorithm. It does however only find a local optimum, and is commonly run multiple times with different random initializations. Variations of k-means often include such optimizations as choosing the best of multiple runs, and there exists many kinds of restriction, such restricting the centroids to members of the data set (k-medoids),
choosing medians (k-medians clustering), choosing the initial centers less randomly (K-means++) or allowing a fuzzy cluster assignment (Fuzzy c-means).

1.3.1.2.1 K-Means

K-Means is a popular clustering technique attempting to find a user-specified number of clusters (K), represented by their centroids, which means the average of all points in the cluster.

Once the number of desired clusters has been set, the number which reflects the analyst’s understanding of the specific question, the algorithm starts to apply its iterative procedure. It firstly guesses the initial centroids, then populates the K clusters by delivering each item to its closest centroid. Finally, each centroid is updated based on the points assigned to the cluster. The assignment and update steps are repeated until no point changes the clusters configuration or until all the centroids remain the same. A detailed standard mathematical explanation is shown as follows:

Given an initial set of k means $m_1^{(1)}, \ldots, m_k^{(1)}$, the algorithm proceeds by alternating between following two steps:

Assignment step: Assign each observation to the cluster whose mean yields the least within-cluster sum of squares (WCSS). Since the sum of squares is the squared Euclidean distance, this is intuitively the "nearest" mean, see the formula (1-3)

$$S_i^{(t)} = \{ x_p : \| x_p - m_i^{(t)} \|^2 \leq \| x_p - m_j^{(t)} \|^2 \ \forall j, 1 \leq j \leq k \}, \quad (1-3)$$

In (1-3), each $x_p$ is assigned to exactly one $S_i^{(t)}$, even if it could be assigned to two or more of them.

Update step: Calculate the new means to be the centroids of the observations in the new clusters, see (1-4)

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j \quad (1-4)$$

Since the arithmetic mean is a least-squares estimator, this also minimizes the WCSS objective.
The algorithm has converged when the assignments no longer change. Since both steps optimize the WCSS objective, and there only exists a finite number of such partitions, the algorithm must converge to a local optimum. There is no guarantee that the global optimum is found using this algorithm.

The algorithm is often presented as assigning objects to the nearest cluster by distance. The standard algorithm aims at minimizing the WCSS objective, and thus assigns by “least sum of squares”, which is exactly equivalent to assigning by the smallest Euclidean distance. Using a different distance function other than squared Euclidean distance may stop the algorithm from converging. Various modifications of k-means such as spherical k-means and k-medoids have been proposed to allow using other distance measures.

1.3.2 Similarity assignment

To answer the second question in chapter 1.2, different similarity assignment methods have to be studied and evaluated.

The similarity assignment takes an important part in the quality of clustering result. Here, the definition of similarity assignment defines how similar every two data points are. Usually, the similarity value is between 0 and 1, values are nearer to 1 indicates the points are more similar, with the value of 0 indicating they are completely different. Depending on the specific properties of different dataset, different similarity assignment methods are preferred.

In this thesis, we try to explore the structure of ontology and select the best similarity assignment method for the purpose of clustering. As the ontology is a directed graph rooted in the most general concept “thing”, with the property that the farther the distance from a data point to the root, the more concrete that data point is. Thus, the similarity assignment methods which consider the depth of data point are researched, in the rest of this paper these methods are called hierarchical based similarity assignment methods. In the meantime, some similarity assignment methods which only focuses on the context of each concept also show some good ability, the context means the relationship around the concept, such as parents, children, grandparents and so on. In the rest of this paper they will be called context based similarity assignment methods.
1.3.2.1 Wu and Palmer similarity assignment method

The method is proposed in [19]. Concerning the ontology is a directed graph, we try to exploit the information of the distance of every point to the root and relationship with others. As a result, Wu and Palmer’s method is selected. This similarity method uses depth from root, least common superconcept and distance to least common superconcept information to get the similarity. A typical tree is shown as Fig.1-1, and the formula to calculate similarity is shown as (1-7).

![Fig.1-1 tree information to show the computation of Wu and Palmer](image)

\[
ConSim(C1, C2) = \frac{2+N3}{N1+N2+2+N3} \quad (1-7)
\]

C3 is the least common superconcept of C1 and C2. N1 and N2 are distance from C3 to C1 and C2 respectively. N3 is the distance from least common superconcept to the root. By using this similarity assignment method, the concepts which are far away from root and have close least common superconcept will score a higher similarity.

1.3.2.2 Jaccard Similarity

The Jaccard Similarity method is particularly suitable in case of binary data (or vectors). Because not only the value of the attributes is taken into account, but also
the relative positions of these values. Considering the initial data as finite sets and taking two sample items i and j:

- \( M_{11} \) includes the attributes whose i, j entry weighs (1,1) (i.e. elements present in both set i and set j, or intersection set between the two).
- \( M_{10} \) includes the attributes whose i, j entry weighs (1,0) (i.e. elements present in set i but not in set j).
- \( M_{01} \) includes the attributes whose i, j entry weighs (0,1) (i.e. elements present in set j but not in set i).
- \( M_{00} \) includes the attributes whose i, j entry weighs (0,0) (i.e. elements absent from both of the sets).

With the above definition, the Jaccard similarity is defined as (1-5):

\[
J(i,j) = \frac{M_{11}}{M_{10} + M_{11} + M_{01}}
\]  

(1-5)

A value close to 1 indicates a strong similarity between i and j, while close to 0 means they are of strong dissimilarity.

### 1.3.2.3 Dennai Similarity

This similarity assignment method is proposed in [20], it is an improvement of Wu and Palmer’s method illustrated in chapter 1.3.2.1. The expression is represented by the following formula (1-8), (1-9):

\[
ConSim(C1, C2) = \frac{2 \times N^3}{N1 + N2 + 2 \times N3 + FPD_{SG}(C1, C2)}
\]  

(1-8)

\[
FPD_{SG}(C1, C2) = \begin{cases} 
0 & \text{if Condition 1} \\
(N3 + N1) \times (N3 + N2) & \text{if Condition 2}
\end{cases}
\]  

(1-9)

Condition 1: \( C1 \) is ancestor of \( C2 \) or conversely

Condition 2: \( C1 \) and \( C2 \) are contained by a common superconcept (excluding root "thing")

The FPD (Function Produces Depths by Smaller Generalizing) is a function which penalizes the similarity of two neighboring concepts that are not in the same level of hierarchy. Because in Wu and Palmer’s formula, two concepts with subclass relationship have higher similarity value than the two located in the same level of hierarchy, which is not reasonable in this semantic information retrieval condition.
1.3.2.4 Alsayed similarity

This method is proposed in [21], which can be seen as an improvement of Jaccard similarity, the author compares clustering result using different combination of information of nodes. Information of children, parents, grand parents and siblings, known as the context of nodes, are used and evaluated separately and combined. It shows that when children and parents are considered the experiment can yield best result. So that the formula is illustrated as follows (1-10):

\[
\text{ConSim}(C_1, C_2) = \frac{\text{Context}(C_1) \cap \text{Context}(C_2)}{\sqrt{|\text{Context}(C_1)| \cdot |\text{Context}(C_2)|}} \tag{1-10}
\]

\(\text{Context}(C_1) \cap \text{Context}(C_2)\) represents the number of common nodes between their contexts and \(\sqrt{|\text{Context}(C_1)| \cdot |\text{Context}(C_2)|}\) represents the size of geometric mean of the two contexts used to normalize the value of the structure similarity. This formula guarantees that the more common nodes the two nodes share, the higher context similarity they have.

1.3.3 Similar clusters identification

This section will introduce the techniques to be used for the purpose of identification of similar clusters, clusters will be automatically identified similar using these techniques, tf-idf is used to identify the degree of importance for each word in a cluster, Vector Space Model holds the tf-df elements and represents each cluster, the cosine similarity is used to define the degree of similarity between clusters.

1.3.3.1 tf-idf

tf-idf, short for term frequency-inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus [22]. It is frequently used as a weighting factor in different areas, such as information retrieval, user modelling and text mining.

Typically, the tf-idf weight is composed by two terms: the first computes the normalized Term Frequency (TF), \(tf\) is the number of times a word appears in a document (in our case, cluster), divided by the total number of words in that document; the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears. Follows are formula (1-11), (1-12), (1-13) which defines two terms and the final tf-idf which are used in this thesis:
\[ tf(t, d) = \frac{\text{Number of times term } t \text{ appears in a document } d}{\text{Total number of terms in the document } d} \]  
(1-11)

\[ idf(t) = \ln \frac{\text{Total number documents}}{\text{Number of documents with term } t \text{ in it}} \]  
(1-12)

\[ tf_idf(t, d) = tf(t, d) \cdot idf(t) \]  
(1-13)

By using above formula, we can capture how important a word is to a cluster in the context of ontology. The importance increases proportionally to the number of times a word appears in the cluster but is offset by the frequency of the word in the ontology.

### 1.3.3.2 Vector Space Model

VSM (Vector Space Model) is an algebraic model for representing objects as vectors of identifiers, in ontology clustering case, each cluster in ontologies is represented by a vector. Each vector contains tfidf of every word occurred in the general context, general context can be context, ontology or ontologies depending on the different problems. \( n \) being the word count of the general context, vector of every cluster is represented as follows (1-13):

\[ d_j = ( tfidf_1, tfidf_2, ..., tfidf_n ) \]  
(1-13)

Each value of the vector represents a tf-idf for the represented word in that index, every cluster has the same size of vector because tf-idf of all words in the ontology is counted even some words do not exist in that cluster.

### 1.3.3.3 Cosine Similarity

Given the vectors of every two clusters, the similarity between them can be calculated using cosine similarity, which is defined as follows (1-14), (1-15), (1-16):

\[ CosineSimilarity(d1, d2) = \frac{d1 \cdot d2}{|d1||d2|} \]  
(1-14)

\[ d1 \cdot d2 = d1[0] \ast d2[0] + d1[1] \ast d2[1] + \cdots + d1[n] \ast d2[n] \]  
(1-15)

\[ |d1| = \sqrt{d1[0]^2 + d1[1]^2 + \cdots + d1[n]^2} \]  
(1-16)

As it is shown, the numerator represents the dot product of the two vectors, while the denominator is the product of their Euclidean lengths. If every tf-idf of two clusters is similar, the angle between the two clusters is small, which leads to a value of cosine similarity closer to 1. In the meantime, if the two vectors have nothing in common, the angle is around 180, resulting the cosine similarity of -1 which
indicates dissimilarity. Thus this metric can be used to measure the similarity between clusters.

1.3.4 Related Study

In order to cope with matching two large ontologies, several techniques can be used, such as reduction of search space, parallel matching, and self-tuning [23]. Among them, reduction of search space is the focus of this paper. One way to reduce the search space is to partition the two ontologies [12,21,24,25,26,28,32]. This approach aims at partitioning input ontologies in such a way that each partition of the first ontology has to be matched only with a subset of the second ontology. It is similar to our system, the flow chart of this approach is shown as Fig.1-2 which is drawn in [21].

Fig.1-2 flow chart of partitioning the two ontologies [21]

As shown in Fig.1-2, the approach consists of four steps. The first step, partition identification, partitions the input ontologies into a set of disjoint clusters. The second step, determination of similar partitions, is devoted to identifying similar partitions. Once similar partitions are identified, in Step 3, matching algorithms can be used to determine local correspondences between similar partitions. Finally, in step 4, the final match result is constructed from these local correspondences.

Among them, [12] is the predecessor of this system. It proposed a LOAD framework with a set of procedures in order to reduce search space using partitioning the ontologies technique. The K-means and DBSCAN it deploys have increased precision, however drops the recall rate tremendously. The best solution illustrated in the paper presents 0.931 precision rate and 0.322 recall rate in all alignment and 0.929 precision rate and 0.897 recall rate for trivial alignment, compared with initially 0.89 of precision and 0.861 of recall presented by SAMBO [27]. Purpose of
reducing search space is achieved, before using the LOAD framework 9026626 comparisons is done and after that only 264536 times comparisons are needed.

1.4 Main content and organization of the thesis

Main purpose of this topic is to improve the solution proposed in [12], using clustering techniques to reduce search space of ontology alignment. We design the procedures based on the work from [12], but in an automated way. We do not need to identify each topic of cluster and merge similar clusters manually, instead we introduce VSM, tf-idf and cosine similarity as a whole part of procedure to identify similar clusters. Different similarity assignment methods and clustering algorithms are implemented and evaluated. Four similarity assignment methods are implemented and evaluated, which are proposed from different paper. Three clustering algorithms, Birch, K-means and Chameleon are implemented and evaluated in order to find the most suitable one for the ontology clustering problem. Finally, each changeable parameter is experimented and evaluated respectively.

Following chapters are introduced as follows. In chapter 2, we illustrate the system requirement, the functional and unfunctional requirements are extracted from holistic and vague problem, the reduction of search space for ontology alignment. In chapter 3, the system design is shown, we show the high level architecture design, as well as the design of each module, some class diagrams and flow charts are drawn in order to explain the design. In chapter 4, we illustrate the system implementation, the flow of each procedure is described, some key language functions, data structure and implementation methods are elaborated. In chapter 5, experiments regarding to each parameter are made and evaluated. Different comparisons are made to find out conclusions. In chapter 6, conclusions are made.
2 Chapter 2 System Requirement Analysis

2.1 The goal of the system

The system should finally be integrated into existing ontology alignment system, for example SAMBO. This part of system is in charge of reducing search space of ontologies, so that domain experts can use less time to integrate two ontologies. In [28], a typical alignment process is shown as Fig. 2-1. Firstly, an alignment algorithm receives two source ontologies as input. The algorithm can include several matchers. Linguistic matching, structure-based strategies, constraint-based approaches, instance-based strategies, using auxiliary information or a combination of above can all be implemented as a matcher. Each matcher utilizes knowledge from one or multiple sources. The matchers calculate similarities between the terms from the different source ontologies. In the end some alignment suggestions are given by one or combination of matchers. Users (domain experts) then can accept and reject those suggestions, the move which may further influence subsequent suggestions. Besides, a conflict checker is used to avoid conflicts introduced by the alignment relationships. The final output is a set of alignment relationships between terms from the two source ontologies.

This paper implements a set of procedures, which can finally be seen as the preprocessing part in Fig.2-1. Our system is to be integrated into the ontology alignment system. However, in the testing phase, instead of source ontologies shown in Fig. 2-1, the concept pairs generated by our system are replaced as input. The goal is to make the number of concept pairs as little as possible while still containing most correct alignment pairs. When we deal with large ontologies, reduction of input pairs can help save a lot of time for both the ontology alignment system and domain expert. Ontology alignment system needs less time to process the reduced input data, and with the precision going up, more incorrect alignment suggestions than the correct one are ruled out by our system, domain experts need to identify less ontology alignment than before. For instance, for the ontologies in the anatomy track of the Ontology Alignment Evaluation Initiative [33], before using our system, the product of the number of concepts in two ontologies is needed, i.e. 9026626 pairs.
need to be validated, while after using our system, around only 300000 pairs are to be validated.

In order to be compatible to the ontology alignment system, output produced by this program shall be the same format as the source ontologies in Fig. 2-1, so that the original ontology alignment system does not need to know the implementation detail of this program, our system is served as a black box which will make the integration much easier.

![Fig. 2-1 Alignment process shown in [28]](image)

2.2 The functional requirements

The final goal of this paper is to reduce the search space using designed sequential procedures, the most important part is clustering algorithm, thus evaluate different clustering algorithms are of great essence. In this section a virtual role named Researcher is created, the operations needed to be done by this Researcher are
shown in Fig. 2.2. In order to evaluate different algorithms, methods and procedures, a researcher shall be able to do following operations:

- Normalize the input: original data is constructed as format of RDF/OWL, the system is able to translate it into more applicable format, like matrix, for the sake of manipulating concepts’ relationship easier. In the meantime, assign similarity weights to relationship needs to be considered, the similarity can be defined in several ways. For example, in [29], one way to assign the similarity is shown as formula (2-1). It calculates the weights between two concepts based on the depth of the ontology and the distance between them. Closer they are, higher similarity (approaching 1.0) they get. Besides formula (2-1), [29] also proposes other similarity assignment methods which could be further analyzed and applied to the case in this study.

\[
    \text{Tradition}_\text{Sim}(C_i, C_j) = \frac{2 \times (\text{Depth} - 1) - \text{Dis}(C_i, C_j)}{2 \times (\text{Depth} - 1)}
\]

(2-1)

- Set parameters: first of all, researcher should choose the clustering algorithm to be run, and for each clustering algorithm it has some parameters need to be set, for instance, K-means [30] needs to specify the K, indicating how many clusters the algorithm needs to form in the end, the initialization method also needs to be specified; Hierarchical algorithm like chameleon [15] needs to specify the threshold of RI and RC to get different size of clusters, the first phase algorithm, which to get initial clusters, also needs to be chosen and parameterized; Birch[16] needs to specify the branching parameter B, cluster diameter T and leaf size L to control the size of each cluster and the size of the tree. Then, in order to identify similar clusters, the threshold for cosine similarity needs to be specified as well.

- Run clustering algorithm: given settings and input, the program is able to run the algorithms and get clustered concepts as output. The output is also going to be in the format of matrix, which is easier and more convenient for storage and further alignment purpose. Clustering algorithms being used cover the K-means algorithm, Birch algorithm and Chameleon algorithm.

Although K-means have been evaluated in [12, 25], still there are space for improvement and exploration. For example, firstly, in [12], only one kind of similarity assignment method is implemented and evaluated, more ways of similarity assignment can be implemented to make comparisons. Secondly, the procedures proposed in [12] is semi automated, with the K-means as the irrelevant variable, we can test if the automated procedures proposed in our paper can outrun than the semi-
automated one. [31] only considers the clustering phase without further alignment phase, and uses density and coherent as measurements, which shows that it is unclear about its practicality when applied to the measurement of precision, recall and f-score.

Chameleon and Birch algorithms are chosen because we hope to get more automation in the progress of clustering, with these two algorithms, the repeated process of discovering the number of clusters could be omitted, leading to a higher time efficiency. Besides, the promising good clustering result shown in many cases is also an important factor.

- Set cosine similarity threshold: tfidf vector and cosine similarity technique are used in order to find similar clusters, the threshold of cosine similarity determines the quantity of matched clusters. A higher threshold will filter out the less similar cluster pair while a lower one will generate more pairs.
- Make alignment: in this paper the real ontology alignment system is not used, in contrast, we use the data generated from the ontology alignment system as a template to evaluate our system. It is feasible because the ontology alignment system will only generate the alignment suggestion pairs that are from the input data, which can be seen as a subset of input data, as our system prunes the input data, the removed pairs can not be generated as ontology alignment from the system such as SAMBO. By doing so we can skip the long waiting time of running SAMBO system, meanwhile the alignment result is still the same.
- Evaluate system: the program uses three measurements to evaluate different clustering algorithms, precision, recall and f-score respectively, which will be illustrated later. The evaluation part will evaluate the procedures and each parameter respectively, in order to find the best setting for the program.
2.3 The non-functional requirements

(1) Language and platform
Java is chosen to be the language, with Eclipse being the IDE. Jena Api is used to assist the parse of ontology file and rdf file, the rdf file is used to make evaluation of final result.

(2) Response time requirement
The system shall be completed in limited time, i.e. 3 minute. As a time-consuming system, some intermediate status should be printed out so that the user would not think the program breaks down. Completion log shall be displayed after one module has done. The interval of two logs should be less than 1 minute.

(3) Interoperability requirement
The system excepts the evaluation part shall be able to be packed into a package and invoked by other program, the api format shall be interoperable to ontology
alignment systems. The mainstream ontology alignment systems are almost all implemented in java, so our system should try the best to accommodate to those systems. the utmost goal of the system is to be integrated into the ontology system.

(4) Robust requirement

The system shall not be affected by other factors. Computations shall always give correct results. The result or important cache data shall be stored in disk in case the computer powers off and the data is lost.

(5) Precision requirement

Precision (also called positive predictive value) is the fraction of retrieved instances that are correct.

The precision of the result shall be higher than the original ontology alignment system, since one of the principle of our system is to improve the efficiency for domain expert. The total number of suggestion in no doubt decreased and good clustered ontologies will rule out mainly outliers of concepts that do not have much common with others, in some way the clustered ontologies can be seen as have a filter that leaves out less probable ontology alignment suggestions. The result shown in [12] also verifies this assumption.

(6) Recall requirement

Recall (also known as sensitivity) is the fraction of relevant instances that are retrieved.

The goal of our system is to retain recall as high as possible. Due to the incompleteness of suggestions yielded by our system, the number of relevant instances is doomed to be equal or lower, recall can be no more than the one without using our system. However, with the accepted proposals made by the alignments of clustered ontologies, using a consistent group, similar to a session-based ontology alignment [32], can further separate the two whole ontologies, resulting in the complete alignments, some slightly recall decrease in the early phase is acceptable in some way.

(7) f-score requirement

F-score is the harmonic mean of precision and recall, which the formula is shown as follows (2-1).

\[
f = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}
\] (2-1)
This metric can be used in order to see how good the ontology alignment result is. It considers both precision and recall simultaneously, thus it provides a holistic view of the evaluation result. Either precision or recall is dramatically low will lead to a low value of f-score, which could indicate there is some problem needs fixing.

The f-score should be maintained as high as possible,

2.4 Brief summary

This chapter describes the requirement of this system, and analyze both functional and non-functional requirement, shed light on the focus of this study and the metrics goal we would like to gain. The study is mainly about using different kinds of clustering techniques applying to ontology and designing an automated procedure to prune the search space of ontology alignment systems. The non-functional requirements are illustrated, the major measurement for the success of this system is precision, recall and f-score.
3 Chapter 3 System Design

3.1 High level architecture

As illustrated in the system requirement analysis, the system this paper designs will be a separated part outside of SAMBO, the ontology alignment system. The processed data generated from our system will be the input of SAMBO. The system of this paper and SAMBO can see each other as black box, both of them do not need to know the implementation inside of each other.

Without our program, SAMBO system will take every pair from two ontologies, with one of pair in one ontology and one in the other. For example, in the anatomy ontology alignment problem, there are 2743 concepts in mouse ontology, and 3304 concepts in human ontology, so that without our system, SAMBO will examine 3304*2743, that is, 9062872 pairs; after using our system, in one case, only concept pairs from similar clusters are taken, so in the last our system would yield only 228412 pairs, which is nearly four times less than the original SAMBO input. As a result, the system running time of SAMBO will largely decrease.

The high level architecture is shown as Fig.3-1. All components of our system is invisible to the SAMBO system, vice versa.

3.2 System overview

This paper aims at dealing with problem of large scale ontology alignment by using a series of procedures that can lead to reduction of ontology mapping suggestions. The main technology it used is clustering, which clusters the similar
concepts in two ontologies respectively and then match only those clusters with cosine similarity higher than the threshold in order to dramatically reduce search space. The ontology alignment system combined with our system can significantly reduce time and effort to align ontologies compared with the original ontology alignment system.

The system interacts with the original ontology alignment system to some degree, the result this system yields will replace the input of the original ontology alignment system. This in some way indicates the consistency of the input and output of our system, which has to be the same format as the input of selected ontology alignment system. In the case of this paper, the ontology alignment system chosen is SAMBO system, and the input of it is OWL format, so both the parse of input and final result generation will be designed as the OWL format.

The flow chart of whole system is shown as Fig.3-2. The system will receive the ontology files as input, parse the ontologies and assign similarity to each pair of concepts. Then, cluster the ontologies respectively, parse the labels of each cluster, identify similar clusters between two ontologies and output every pair in identified similar clusters. Finally, evaluation to the system is made. Each technique and procedure the paper designs will be shown afterwards sequentially.
3.3 Ontology Parsing and Similarity Assignment

The system firstly imports two ontologies as input, using depth first search algorithm to extract concepts from file of .owl format. As the ontology files have some concepts irrelevant to our study, the input must be parsed first. Those concepts with the value of null or empty are excluded from the input. Then, as iterating through the ontology, depth and relationship between concepts are collected for the further use of similarity assignment. Depth is the steps of a concept need to take from the root concept. Relationship between concepts is the subclass and superclass.
attribute of each concept, considering the research result shown in [21], only relationship of children, parents and itself is considered.

After iterating through the ontology, the similarity is distributed to each pair of concepts. With the help of depth and relationship information acquired during iteration process, different kinds of similarity assignment methods are applied. Methods are selected from various source of literature after consideration and comparison. The reason why we choose different kinds of similarity assignment methods is that we want to evaluate which combination of techniques will yield best result. Methods shown in chapter 1.3.2, Wu and Palmer similarity, Dennai similarity, Jaccard similarity and Alsayed similarity are calculated and evaluated respectively.

In the end, a two dimensional array of similarity value between every two concepts are generated as the input of next phase. The flow chart of ontology parsing design is shown as Fig.3-3.

The four similarity assignment methods have similar generation behaviors, all of them use the information stored during the ontology parsing phase and have the need to use the information to compute similarity for every pair. So that we design to create a base class for these similarity methods. In order to hide the implementation detail of the similarity assignment methods and to achieve the design principle of low coupling and high cohesion, a factory method pattern is used to generate similarity. Meanwhile, the four similarity method name is stored in an enumeration type, and the factory class only accepts the enum type in order to avoid mistakes and make the program user-friendly. The class diagram of the similarity assignment part is illustrated as Fig.3-4.
Fig.3-3 flow chart of ontology parsing
3.4 Clustering

In this system clustering algorithm is used to cluster similar concepts in an ontology into the same cluster. Different clustering techniques are to be used in this phase, similarity generated in the previous phase is used as measurement to gather similar concepts. Clustering analysis is used in this phase as well in order to find best parameters for different clustering techniques.

Three clustering algorithms to be studied follow the same procedure in our system, construct itself with parameters, build clusters, then output clusters. Thus a super class is created in our system, so that the main program is able to pass the parameters to the algorithm without knowing the implementation detail of each algorithm. Cluster class is built in order to store the detail of clusters, such as the cluster number, and an array to store the indexes of concepts that are in the cluster. The class diagram of this part can be drawn as follows Fig.3-5.
3.4.1 Chameleon

Chameleon algorithm has been illustrated in previous chapter. In our system, the Chameleon class inherits from the Clustering class. For this ontology clustering case, firstly we use the similarity between concepts and implement k-nearest-neighbors algorithm to get the initial clusters. Then calculate the relative closeness (RC) and relative interconnectivity (RI) in order to find similar enough clusters, if similar enough then two clusters are merged. Repeat the previous phases until no two clusters have enough RI and RC value to merge. The procedure is shown as Fig.3-6.

3.4.2 K-Means

K-Means is a centroid based clustering method illustrated in previous chapter. The class inherits from the Clustering class. Considering the initialization of initial centroids of the algorithm, two kinds of methods are implemented, which are explained as follows:

- Random initialization: each centroid is a randomized vector, with each element value ranged from 0 to 1.
- Random data points initialization: choose k concepts out of all concepts.

The two methods will influence the convergence of the final centroids in different ways, so both of them are implemented and evaluated. Secondly, assign every concept to its nearest centroid by calculating Euclidean distance from each centroid and choosing the minimum one. Thirdly, recalculate centroids by sum up all concepts within each cluster and get the average. Repeat second and third phase until no
centroid has changed after the recalculation. The flow chart of procedure is shown as Fig.3-7.

Fig.3-6 Chameleon clustering for ontology flow chart
3.4.3 Birch

Birch is a hierarchical based clustering method illustrated in previous chapter. The Birch class inherits from Clustering class in our case. The most important part of Birch is the formation of CF tree, a CF tree is restricted by two parameters, a branching factor $B$ which is used to decide how many children a non-leaf node can have, and a threshold of cluster diameter $T$ which is to control the size of every cluster. Every concept in the ontology is iterated one by one. The first concept being iterated is considered to be the root of the CF tree. Later, whenever a new concept is added, distance of the concept to every leaf is calculated, the leaf which has the shortest distance with the concept will accept the new concept. Then, the diameter of the leaf, which has just accepted a concept, is calculated, if the value is smaller than the $T$, the leaf node absorbs the concept, otherwise if the node has less than $B$ clusters, add the concept in that leaf as well but as a new subcluster, if the number of
clusters in the leaf exceeds the $B$, split the leaf node. Every insertion of a concept will lead to the update of CF information of the node that absorbs it. The flow chart of the Birch is shown as Fig.3-8.

![Birch flow chart](image)

**Fig.3-8 Birch flow chart**

The design of class diagram of Birch is shown as Fig.3-9. CF is the most basic structure in Birch, a MinCluster has dependency to the CF class because the MinCluster contains CF, and TreeNode inherits from CF for the reason that TreeNode is made of many CF. We divide the TreeNode into NonLeafNode and LeafNode, since they have different operations. In order to iterate the clusters conveniently, every leaf has two pointers to the sibling leaves.
3.5 Label Parsing

This part will be developed as a separate module class, it receives the clusters from clustering phase and information of concepts from ontology parsing phase as input, output the parsed label of every concept and the tfidf vector of every cluster.

Label of concepts should be normalized into tokens in order to be used in latter phase to generate tfidf vectors of clusters. The procedures in this phase would be sequential, every operation has to wait until last procedure is over. The proposed procedures include tokenize words, normalize words, delete useless words, and stem words. The Flow chart of this part is shown as Fig.3-10.

Fig.3-9 Class diagram of Birch part
Most of concepts in ontologies have a label describing what the concept is, in this phase labels are parsed in order to get the words of it, so that in the next phase the tfidf vector can be generated for the clusters. The procedures are described as follows. Firstly, labels that have null value are excluded. Secondly, labels are tokenized by separating words with space. Thirdly, stopwords are excluded, stopwords mean the words that are too commonly used that they have no useful information. “a”, “an”, “of” are examples of the stopwords. Fourthly, stem the words so that all different forms of the same root word can be regarded as the same. For example, “feet” becomes “foot” after stemming.

### 3.6 Similar Clusters Identification

The functions of calculating tfidf vectors and computing cosine similarity shall be written in the same module of Label Parsing, because they are both trying to extract the meaning of each cluster.
Tf-idf vector shall be used in order to extract the most representative words for each cluster. By calculating the cosine similarity value of every two clusters, with the threshold preset by researcher, the two clusters with cosine similarity between their corresponding tfidf vectors higher than the threshold are considered to be similar and used for later ontology alignment suggestion phase.

### 3.7 Alignment Evaluation

This phase is isolated from our system to some degree, our final system does not have this part, this part is only used to evaluate our system. This part will use a reference file which is the standard ontology matching brought up by the domain experts, meanwhile, an ontology alignment file from ontology alignment system is used to mimic the result of the ontology alignment system. Precision, recall and f-score are computed in order to make evaluation of our system. Execution time and the number of matching concept pairs generated by our system shall be also recorded, one is to see how much time our system can save for the ontology alignment system, the other one is used to see the performance of our system in reducing the search space of ontology alignment system.

The execution time should be recorded as the time elapsed from the beginning to the end of our program except this alignment evaluation part. A separate module is used to initiate the timer and calculate the time elapsed.

The outcome of each round of evaluation is recorded in a log file, so that we can check and compare the evaluation results easily. Each record is along with the parameters from other procedures, like the clustering method used and its parameters, the similarity assignment method used.

### 3.8 Brief summary

In this chapter we design the overall structure of the system, decide the way our system used to cooperate with the SAMBO system, functions and flows of every module has been described. The procedures of our system are designed and explained, some key parts’ design is illustrated with class diagram and flow chart.
4 Chapter 4 System Implementation and Testing

In this chapter the environment of system implementation is shown, the implementation details of the system design are illuminated, and then we will show how testing and evaluation process are operated.

4.1 The environment of system implementation

The experiment is conducted in macOS Sierra system, with 2.5 GHz Intel Core i7 processor, 16 GB 1600 MHz DDR3 memory.


4.2 System Implementation

This section will show how the system is implemented. This system is entirely built by the author himself without using other applications and open source codes or software. Everything is written in java. For the ontology and rdf parse part, Jena Api is used. WordNet api is used to deal with label parsing. The delivered product only has a completed java program that can run, given the owl format input.

4.2.1 Ontology Parsing

A general flow chart of this phase is shown as Fig.3-2.

Ontology has a form of directed graph which everything rooted down the “thing”. The typical example of a concept in ontology is shown as Fig.4-1.

From Fig.4-1, we can see a concept has a Class name, which can be seen as a unique identifier, then there is a label describing what it is, the label of this concept it is “body cavity/lining”, finally a relationship of “subClassOf” is defined in last line which indicates this concept is the children of the Class http://mouse.owl#MA_0002433.

To exploit this structure, Jena Api is used in order to iterate through the owl file. Firstly, use ModelFactory.createOntologyModel() method to create a ontology reader, given the file path, the ontology of owl format is read from file to the memory, held by the handle. Then, use listHierarchyRootClasses() method to get all
the root concepts and use an iterator to wrap them. For each root classes it iterates, use depth first search algorithm to iterate the whole branch. During the iteration, depth information, label and class name of concepts are stored in three separate array list. Relationship between every two concepts are stored as a two dimensional array list. Besides, validity check is conducted during the iteration, for concepts that do not have class name is excluded, and since it is a directed graph, some concepts may occur more than once, only those concepts being iterated for the first time would be stored.

![Example of owl file](http://mouse.owl#MA_0000005)

**Fig.4-1 example of owl file**

### 4.2.2 Similarity Assignment

After recording the necessary information of the ontology, similarity assignment algorithm is implemented. As mentioned in previous chapter, different similarity assignment techniques are implemented in order to compare and evaluate.

Four similarity assignment methods can be divided into two groups, each with a certain type. *Wu and Palmer similarity* [19] and *Dennai similarity* [20] are similar, while *Jaccard similarity* and *Alaysed similarity* [21] are similar.

To generate Wu and Palmer and Dennai similarity, the program makes use of depth and distance array lists which are stored during the iteration of ontology. The common least parents are discovered by iterating the distance vector. The program looks into the distance vectors of two concepts and try to find the element of the same index in two vectors with smallest but non-zero number, the index of that element is the index of least parent of the two concepts. Such that the similarity value of every two concepts are generated one by one. The only difference between two similarity assignment methods is that for Dennai similarity assignment method we need to add a FPD in the denominator of the formula, which just adds a little more
computation of this phase. After this phase, the program outputs a two dimensional matrix of similarity. Given the two dimension as i and j, the element in the place of (i,j) is the similarity between i\textsuperscript{th} concept and j\textsuperscript{th} concept.

When it comes to Alsayed similarity and Jaccard similarity, given the specialty of the distance between every two concepts either be 1 or 0. The program utilizes BitSet to store the distances. To calculate Jaccard similarity, as the numerator being the entries which two elements are both 1, and() function is used upon two vectors in order to get the intersection of them, the return is the numerator of the formula. The denominator is composed of entries which not both of element being 0, so the or() function is used in order to get the union of the two vectors, the return is the denominator of the formula. In order to calculate the Alsayed similarity, the program uses BitSet to generate the intersection and union as well, only difference is the denominator of Alsayed similarity has the operation of square root. After theses two operations, Jaccard similarity and Alsayed similarity are generated and stored in a two dimensional array respectively.

### 4.2.3 Clustering

Given the similarity calculated, different clustering algorithms are implemented.

#### 4.2.3.1 Chameleon

A k-nearest-neighbor (knn) algorithm runs first. Initialize a two dimensional array, with initial value of 0, each element represents if there is connection between every two concepts. It receives k as a parameter to determine how many concepts a concept shall link to, k concepts with highest similarity will be connected to the concept, if there is less than k concepts have similarity of more than 0 to the concept, all those concepts with higher than 0 similarity value are connected. The connected concepts will have their corresponding element in connection array set as 1.

A depth first search algorithm is implemented to traverse whole connection array. The ontology is divided into multiple graph, and each graph is seen as a cluster. Then calculate relative inter-connectivity and relative closeness of each cluster, if the final score exceeds the threshold (taken from parameter), the two clusters will merge into one. After no more clusters can be merged, exit the program and output the clusters.
4.2.3.2 K-means

Given the parameters of number of desired centroids \( k \) and the similarity matrix of concepts, the k-means module is executed. Two overloaded functions GenerateKMeans() are implemented in order to initialize the centroids. One of them only has a parameter \( k \), which indicates the desired number \( k \) of centroids. This method will generate centroids among the whole space. The other one has parameters of \( k \) and similarity matrix, each row of the similarity matrix is a possible centroid in this case.

In the second phase, buildClusters() function is called, in this function, all concepts are assigned to the nearest centroid first, and then we repeat the procedure of reassignment of every concept, the function ReassignConcepts(), and recalculation of every centroid, the function RecalcMeans(). Reassignment of concept is based on the selection of shortest distance between the concept and all centroids.

4.2.3.3 Birch

Given the similarity matrix from the last phase, the buildTree() function is called to generate the CF tree. A leaf node is firstly instantiated, meanwhile the doubly linked list for iteration of leaves are instantiated and take the leaf node into the list. Then, scan all concepts and add them to the tree one by one.

Firstly, each record of concept is transformed into the structure of CF and denoted as a sub cluster. Then the sub cluster is absorbed to the tree, depending on the class of the root being leaf node or non-leaf node, polymorphic function of absorbSubCluster() is called. If the checked node is a non-leaf node, it calculates all distances between each child node and the to be absorbed cluster, then the selected child will call the absorbSubCluster() function again until getting to one of the leaf node. If the checked node is a leaf node, the distances between every sub cluster and the to be added cluster are calculated, and the one with shortest distance will be chosen. Then, the diameter of the merged cluster is calculated in order to decide if we need to split the cluster. Finally, the all the CF in the path to this leaf are updated.

4.2.4 Label Parsing

Given the cluster sets resulting from the previous phase, label parsing is provided to extract the useful information from the labels of each cluster. In order to do that, some common text mining techniques are used. Given the stored array of
concept labels from ontology parsing phase, concept labels in every cluster can be extracted and processed.

Firstly, tokenize every label into an array of words. In our program, we use split() function to split words by separators such as space, underline and vertical line. Secondly, transform all word into lower case using toLowerCase() function. Thirdly, delete those words which are too commonly used, because they do not have useful meaning we need, include them will lead to misinterpretation of the cluster. As a result, “a”, “an”, “the”, “for”, “of” and so on are removed from the list. Finally, WordNet api is used in order to stem the word, normalize the same rooted words as the same. For example, “feet” is transformed to “foot”. An array contains every possible part of speech that has variations, e.g. noun, verb, adverb and adjective. The program searches the word in each data file of part of speech in WordNet, if there is a match, stem the word to the corresponding word that in the WordNet file. For example, the word “feet” will be searched in verb WordNet file first, it returns null, and then it is searched in noun WordNet file, we get the return “foot” so the word is stemmed to “foot”.

4.2.5 Cluster Subject Identification

Given the appeared words of count $n$ of all concepts and the word vectors generated from the last phase, cluster subject identification is feasible. Create tf-idf vectors with size of $n$ for each cluster. For each cluster, iterate every concept in the cluster, parse the word vectors of those concepts. As a result, given count of every word and sum of all count of words, tf vectors is generated by calculating the number of times a word appears in the cluster divided by the total number of words in that cluster. During the iteration, frequency of each word is also stored for the use of calculating idf vectors. After iteration, the idf vectors are calculated as the logarithm of the number of the clusters in the ontology divided by the number of clusters where the specific term appears.

Given the tfidf vectors, two alternative methods could be used. One is to identify corresponding words of highest $k$ tfidf as the subject of the cluster, $k$ is defined and adjusted according to the size of the ontology.

Another option is to use cosine similarity to identify similar clusters. In this case, we do not need to identify the subject of cluster, just store the tf-idf and use it in the latter phase to match similar clusters.
4.2.6 Similar Clusters Identification

Given the tfidf vectors, cosine similarity can be calculated. Two nested loops are used in order to calculate the dot product of every two vectors and square root of every vector. Denominator is checked, if it is 0, the cosine similarity of that two clusters is set as -1, which indicates the most dissimilarity.

Threshold of similarity is set different in order to get different precision of cluster results and make evaluation of them. A Map structure is used to store the suggested matching pairs. The Map structure will store the concept indexes of the first ontology as entries, and the value being matched concept indexes in the second ontology.

4.2.7 Alignment Evaluation

In this phase, a reference file validated by domain expert is used, the file contains the standard correct ontology alignment of mouse.owl and human.owl, in the following part it is named standard reference. An ontology alignment suggestion file generated by SAMBO system is used as well. It contains the ontology alignment suggestion generated by inputting mouse.owl and human.owl to SAMBO system, in the following it is named SAMBO file. Both of them are in rdf file format. A typical alignment suggestion in standard reference file and SAMBO file is shown as follows Fig.4-2:

```
<map>
  <Cell>
    <entity1 rdf:resource="http://mouse.owl#MA_0001588"/>
    <entity2 rdf:resource="http://human.owl#NCI_C12234"/>
    <measure rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.99</measure>
    <relation>=</relation>
  </Cell>
</map>
```

Fig.4-2 a typical alignment suggestion in SAMBO file

As shown in Fig.4-2, the rdf:resource attributes of entity1 and entity2 are url of suggested alignment from mouse.owl ontology file and human.owl ontology file respectively. The value inside of pair measure is the confidence of two concepts being matched, for the SAMBO system, confidence threshold is 0.6, so the value could vary from 0.6 to 1.0, higher value means there is higher possibility the two
concepts can be matched. In our case, we do not use this measure, what is only concerned is the mapping pairs in these files.

The program iterates the rdf files using Jena Api, split the strings, which are extracted from two entities, with “#” and store the latter parts in an array of size of two. Taken the Fig as example, the stored array would look like following:

{MA_0001588, NCI_C12234}

In the end, all array is stored inside of a two dimensional array, which holds all ontology alignment suggestions in the file.

Then the intersection operation is made upon the alignment suggestion generated from the SAMBO file and our system, the result alignment suggestion is named preprocessed alignment. The reason for why the name is preprocessed alignment is that our system can be seen as a text mining procedure aiming to reduce the concept pairs to be validated. This phase is a simulation of SAMBO system, if we directly input result of our system into SAMBO, preprocessed alignment will be generated as output as well.

Given the preprocessed alignment, the evaluation program iterates every alignment pair extracted from the standard reference file, if the preprocessed alignment has the match, the matched count plus 1. After the iteration, precision is calculated by dividing the matched count with total amount of preprocessed alignment; recall is calculated by dividing the matched count with total number of alignment in standard reference; f-score is calculated by the formula composed of precision and recall.

4.3 System Testing Overview

System testing is conducted by writing separate test program. The test program has validated input and output, the tested functions are called by the test program with the designed parameters, the output or essential intermediate variables are printed and checked with the standard result which is calculated beforehand.

For example, a quick sort is implemented in our program during the phase of chameleon clustering. The function of this part of program is dedicated to sort the similarity vectors of a concept and return the index of top k largest elements. Given the requirement and function, we design the test case data as disordered float data, and create another array containing the indexes of top k largest elements. After preparation, the test program calls the function with data array as parameter, and
compare the function return with prevalidated result array to see if the function is implemented correctly.

A log file is created in order to automatically record the parameters and evaluation result of every system run. Similarity assignment method, clustering algorithm and its parameters, cosine similarity threshold are recorded, then the evaluation result of our system is recorded as well.

The execution time is also a focus of our study. Usually when the ontology alignment system deal with large ontology, it needs a lot of time like minutes, hours even days. Our system can help reduce the search space, thus reduce the time of the ontology alignment system needed to process the ontologies. `System.nanoTime()` is used to calculate elapsed time in our debug version of system. In the beginning of the program, we record the time using `System.nanoTime()` function, after everything finishes, call the function again and calculate the difference between two times call. The time difference is output in the format of milliseconds. However, due to the status of the machine, the execution time will be slightly different even do the same experiment twice, through the experiment we find out that the time difference could be up to 10 seconds.

### 4.4 Brief summary

In this chapter we have illustrated the environment the system deployed in. Then, the detail of implementation is explained, how does each module cooperate with others is illustrated. Key functions are described in the corresponding module part. Finally, we show the methods used to system testing, and elaborate how we calculate each evaluation metrics.
5 Chapter 5 experiment result

In the previous chapters, we have presented our clustering based pre-processing system, in this chapter, many experiments are conducted in order to show the results of our system. As a multi-procedures system, parameters of many procedures can be varied, and we use the varied parameters manually to show how those variables can affect the final performance results.

We use mouse.owl and human.owl as input of our experiments, which are the source ontologies of anatomy ontology alignment problem [33]. These ontologies have been widely used to test different ontology alignment systems [12,20].

5.1 SAMBO Result

SAMBO result is the alignment suggestion based on mouse.owl and human.owl, given out by SAMBO system. As the standard reference file available, the number of comparisons, precision, recall and f-score of SAMBO result are calculated and shown below as Table 5-1.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
<th>comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMBO</td>
<td>89%</td>
<td>86.1%</td>
<td>87.5%</td>
<td>9062872</td>
</tr>
</tbody>
</table>

Table 5-1 human and mouse anatomy ontology alignment result by SAMBO

As our system uses SAMBO system to generate ontology result, and our system could be thought of as a pre-processing of the ontology alignment system, the expectation of these metrics is to some degree predictable and meanwhile the metrics have important value in evaluating our system.

By using our system, the precision will be higher or equal than the SAMBO result. The reason is the pre-processed data has removed some matches, those alignment suggestions could be correct or incorrect. As the formula of precision is to divide the number of correct alignment with total alignment suggestions, the removal of an incorrect alignment suggestion will reduce the denominator by one, so that the precision goes higher. However, if our system removes a correct pair, with numerator and denominator both decrease 1, and denominator being always higher than numerator, the precision will become smaller. As a result, the precision is a good metric to evaluate the performance of system. If the precision goes lower, it means
our system removes some correct ontology alignment suggestions, and a higher precision means we delete much more incorrect ontology alignment suggestions.

The recall rate will be lower or equal to the result of SAMBO system by using our system. The formula of the recall is the total number of correct alignment suggestions divided by the total number of alignment in standard reference. As the denominator being unchanged, the numerator could decrease by accidentally removing correct alignment suggestions. We should also watch this metric carefully in order to maintain as much correct alignment suggestion as possible.

F-score would usually go down because the recall rate is doomed to be lower and precision is unstable depending on the performance of our system.

The number of comparisons, also known as search space, are going to be dramatically lowered down. As we have introduced in previous chapters, by conducting our procedures, only mapping between similar clusters can be chosen to be the input of ontology alignment system, so that the workload of ontology alignment system would largely decrease.

In conclusion, the SAMBO result provide itself to be good criteria to be compared with, the result of comparison can illustrate instinct view of the performance of our system, as well as how much does our system affect the ontology alignment system such as SAMBO.

5.2 Experiments

Our system is composed of a set of sequential procedures that have many variables. The first step, ontology parsing, has no variables. In the second step, similarity assignment has four choices implemented in our system, Wu and Palmer similarity, Dennai similarity, Jaccard similarity and Alsayed similarity, different similarity assignment method can lead to various performance of our system. In third phase, K-means, Birch and Chameleon are to be chosen in our system, and all of them have various of combination of parameters to be chosen from. Finally, in the phase of similar clusters identification, threshold of cosine similarity also needs to be evaluated.

In order to evaluate the parameters, we have to set up the basic procedure so that the program can complete the run and give out the final ontology alignment result. Because all the metrics proposed in this study focus on the final result of the system instead of intermediate result, the system must have a complete run. Thus, some
basic setting is made in order to let the system have a complete run. In the similarity assignment phase the system chooses the Alsayed similarity assignment method as basic setting, in the similar cluster identification phase threshold of cosine similarity is set as 0.05, clustering is considered to be the trickiest part so no setting is made beforehand, we will start from analyzing different clustering methods. In the sections below, different parameters are evaluated both separately or combined.

5.2.1 K-means

In K-means, two parameters shall be made experiments upon. Firstly, there are two kinds of initialization method for this clustering algorithm as mentioned in the previous chapters. The random initialization among the whole space or random initialization among the data points. The former one is good at finding possible centroids in the space but also have the possibility of converging in remote or isolated points. The second parameter is the number of k we should choose for the algorithm. As the sensitivity experiment already been done in [12], we will choose the k both from that study or randomly in order to see the trend and make comparisons. In [12], depends on the ontology size and sensitivity analysis, a k of 65 for mouse ontology and 93 for human ontology is selected.

5.2.1.1 Random initialization among space

In this experiment, initial centroids are chosen among the corresponding dimensional space, each dimension element of every centroid is random ranged from 0 to 1, which is the possible range of similarity. Five system run, based on the proposed basic procedures setting, is recorded as Table 5-2.

<table>
<thead>
<tr>
<th>K(mouse)</th>
<th>K(human)</th>
<th>Precision</th>
<th>Recall</th>
<th>f-score</th>
<th>Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>50</td>
<td>89.02%</td>
<td>86.08%</td>
<td>87.53%</td>
<td>9062872</td>
</tr>
<tr>
<td>65</td>
<td>93</td>
<td>89.02%</td>
<td>86.08%</td>
<td>87.53%</td>
<td>9057496</td>
</tr>
<tr>
<td>300</td>
<td>300</td>
<td>89.08%</td>
<td>86.08%</td>
<td>87.55%</td>
<td>9028184</td>
</tr>
<tr>
<td>700</td>
<td>700</td>
<td>89.02%</td>
<td>86.08%</td>
<td>87.53%</td>
<td>9036916</td>
</tr>
<tr>
<td>2000</td>
<td>2000</td>
<td>89.02%</td>
<td>86.08%</td>
<td>87.53%</td>
<td>9062872</td>
</tr>
</tbody>
</table>

Table 5-2 K-means using random initialization among space

The experiment stops at both k with value of 2000, because it already takes around 7 minutes to run the program. The program could be optimized more in the future work, but compared with other algorithms right now it takes too much time already. In conclusion, use the initial centroid randomized among space is not a very
efficient way. Analysis as follows, for example, there is 2743 concepts in mouse ontology which means the similarity vector is of 2743 dimension. In our largest case, there are 2000 centroids randomized among whole 2743 dimensions, which will highly possibly converge to a few number of centroids easily. To make this initialization method actually work, we need far more initial centroids, so that every possibly cluster can be captured by these centroids.

5.2.1.2 Random initialization among data points

In this experiment, initial centroids are chosen among the concepts, five system run, based on the proposed basic procedures setting, is recorded in Table 5-3.

<table>
<thead>
<tr>
<th>$K(\text{mouse})$</th>
<th>$K(\text{human})$</th>
<th>Precision</th>
<th>Recall</th>
<th>f-score</th>
<th>Comparisons</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>50</td>
<td>89.17%</td>
<td>83.11%</td>
<td>86.04%</td>
<td>7800870</td>
</tr>
<tr>
<td>65</td>
<td>93</td>
<td>89.98%</td>
<td>83.51%</td>
<td>86.62%</td>
<td>7052996</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>89.80%</td>
<td>82.45%</td>
<td>85.97%</td>
<td>6914220</td>
</tr>
<tr>
<td>300</td>
<td>300</td>
<td>90.44%</td>
<td>81.79%</td>
<td>85.90%</td>
<td>4768262</td>
</tr>
<tr>
<td>500</td>
<td>500</td>
<td>90.55%</td>
<td>79.68%</td>
<td>84.77%</td>
<td>3705804</td>
</tr>
</tbody>
</table>

Table 5-3 K-means using random initialization among data points

As shown in the table, the sensitivity analysis proposed in [12] do have the best performance over all five experiments, which again validates the feasibility of the sensitivity analysis in K-means clustering for the ontology clustering purpose. The lower and higher number of $k$ both decrease the f-score, although the larger $k$ somehow would lead to worse performance in recall and f-score, we can see the precision is somehow increased. As discussed in section 5.1, the higher precision means that among the removed alignment pair, there is more incorrect one than the correct one, so that it means a better set of alignment suggestions to some degree. Meanwhile, the number of comparisons lowers down a lot. For the $k$ number of 65 and 93 case, the number of comparisons made in SAMBO is 7052996, while the original number of comparisons is 9026626. In this case our system does trim some comparisons away, but not very largely. In the case of both $k$ equals 500, the number of comparisons is lowered down more than two times, which could be a good sign when we are dealing with large ontology alignment problem.

5.2.2 Chameleon

Chameleon has three possible changeable parameters in our case. As illustrated in previous chapter, the first phase of chameleon uses k-nearest-neighbors to generate
initial clusters. For the formula composed of RI and RC in reconnecting clusters phase, a threshold of the formula is set to control if the two clusters should be merged. The formula is calculated by $RI \times RC^\alpha$, $\alpha$ is also a parameter that can be changed in order to reposition the relative interconnectivity and relative closeness. The three parameters of Chameleon are evaluated and recorded in the Table 5-4.

<table>
<thead>
<tr>
<th>K</th>
<th>Threshold</th>
<th>$\alpha$</th>
<th>Precision</th>
<th>Recall</th>
<th>f-score</th>
<th>Comparisons</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.05</td>
<td>1</td>
<td>91.36%</td>
<td>82.26%</td>
<td>86.57%</td>
<td>481028</td>
<td>61</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>91.37%</td>
<td>82.39%</td>
<td>86.65%</td>
<td>482543</td>
<td>55</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>0.5</td>
<td>91.37%</td>
<td>82.45%</td>
<td>86.69%</td>
<td>484082</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>0.1</td>
<td>2</td>
<td>91.39%</td>
<td>82.59%</td>
<td>86.76%</td>
<td>480643</td>
<td>60</td>
</tr>
<tr>
<td>2</td>
<td>0.1</td>
<td>1</td>
<td>91.81%</td>
<td>76.12%</td>
<td>83.23%</td>
<td>1198115</td>
<td>53</td>
</tr>
<tr>
<td>1</td>
<td>0.2</td>
<td>1</td>
<td>91.43%</td>
<td>82.39%</td>
<td>86.68%</td>
<td>478996</td>
<td>59</td>
</tr>
<tr>
<td>1</td>
<td>0.3</td>
<td>1</td>
<td>91.45%</td>
<td>82.52%</td>
<td>86.75%</td>
<td>481299</td>
<td>59</td>
</tr>
<tr>
<td>2</td>
<td>0.3</td>
<td>1</td>
<td>91.82%</td>
<td>77.77%</td>
<td>84.21%</td>
<td>1182646</td>
<td>58</td>
</tr>
<tr>
<td>1</td>
<td>0.4</td>
<td>1</td>
<td>91.44%</td>
<td>82.45%</td>
<td>86.72%</td>
<td>481610</td>
<td>59</td>
</tr>
<tr>
<td>1</td>
<td>0.5</td>
<td>1</td>
<td>91.44%</td>
<td>82.45%</td>
<td>86.72%</td>
<td>481679</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>1</td>
<td>91.51%</td>
<td>82.52%</td>
<td>86.78%</td>
<td>479783</td>
<td>59</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>0.5</td>
<td>91.51%</td>
<td>82.45%</td>
<td>86.75%</td>
<td>478213</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>1.0</td>
<td>2</td>
<td>91.45%</td>
<td>82.52%</td>
<td>86.75%</td>
<td>478449</td>
<td>60</td>
</tr>
<tr>
<td>1</td>
<td>1.5</td>
<td>1</td>
<td>91.38%</td>
<td>82.52%</td>
<td>86.72%</td>
<td>479495</td>
<td>58</td>
</tr>
<tr>
<td>1</td>
<td>2.0</td>
<td>1</td>
<td>91.45%</td>
<td>82.52%</td>
<td>86.75%</td>
<td>480158</td>
<td>95</td>
</tr>
<tr>
<td>1</td>
<td>MAX_VALUE</td>
<td>1</td>
<td>91.44%</td>
<td>82.45%</td>
<td>86.72%</td>
<td>479559</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 5-4 Chameleon evaluation

From the experiment result, first conclusion we can draw is the k of knn should just be 1, a bigger k will make more unnecessary connections thus lead to the decrease of all metrics except precision. Although it has the highest precision overall, the other metrics are far worse than the others. No more experiment of other value of k is done due to the above analysis.

Other experiments have shown that higher the threshold is, the result seems to become slightly better, the precision, recall and f-score all slightly go up, while the number of comparisons and execution time being about the same. As the evaluation results being better when the threshold goes up, we want to evaluate the situation when the threshold is the maximum, thus the threshold is set as MAX_VALUE, in this case, the Chameleon degrades into KNN, because the infinite positive threshold
prevents any two clusters from merging together. After KNN has generated initial clusters, in the second phase, as the threshold is unachievable, no cluster can be merged so that the final clusters are exactly the same as the clusters generated from KNN. It shows that there is only a little decline in the result compared with the best case of Chameleon. It illustrates that actually KNN is a good clustering algorithm to be used in ontology clustering case. With KNN, we do not need to tune the parameters, it can directly be applied to the ontology alignment problem.

To evaluate $\partial$, we set it with three value, 0.5, 1 and 2 in two cases. As we discussed in previous chapter, this parameter is used to balance the focus of RI and RC. If the parameter is less than 1, RI takes more part in the result of the metric. If the parameter is more than 1, RC takes more proportion than RI in the result of the metric. If the parameter equals to 1, RI and RC are equal in the composition of the metric. These experiments show that the variation of $\partial$ does not affect the result a lot. In K=1, threshold=0.1 case, $\partial$ value of 0.5 and 1 both increase the evaluation result a little bit, however, in K=1, threshold=1 case, $\partial$ value of 0.5 and 1 both decrease the evaluation result somehow. Taken the formula of threshold into consideration, $F = RC \ast RI^\partial$, the F will change according to the increase or decrease of $\partial$. And F is used to be compared with threshold, so if $\partial$ goes down or up, the relative difference between F and threshold changes. It actually the change of $\partial$ can be seen as another way to tune the parameter of threshold.

In conclusion, Chameleon help reduce the search space, its number of comparisons is around one twentieth of the original SAMBO system, the precision gets higher a little bit, although recall and f-score go down to some degree, the changes are acceptable. Thus Chameleon can serve well to the purpose of reducing search space for the ontology alignment.

5.2.3 Birch

The essential part of Birch is the formation of the CF tree, a CF tree has two parameters to be tuned. The first one is the branching factor B, which means a non-leaf node can only contain at most B entries. If the node has more than B entries, the node should split and repetitively check the parent of the node to ensure the number of branch still smaller than B. A large B indicates each cluster might be able to contain more concepts, from the instinctive view the CF tree becomes fatter. The second one T is the threshold of diameter of a subcluster, when a new concept is
about to be added to an entry, T is checked, if the new diameter exceeds the threshold T, the entry should split to two, with the two concepts have furthest distance being the origin of two new clusters respectively. T is vital in this case, on one hand, if T is too small the final clusters are all composed of only one concept. On the other hand, if T is too big all concepts will be in one cluster, the final clusters collection will be of only one large cluster. The following Table 5-5 is the evaluation of Birch regarding its different value of parameters.

<table>
<thead>
<tr>
<th>B</th>
<th>T</th>
<th>Precision</th>
<th>Recall</th>
<th>f-score</th>
<th>Comparisons</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.0</td>
<td>90.89%</td>
<td>82.26%</td>
<td>86.36%</td>
<td>347021</td>
<td>156</td>
</tr>
<tr>
<td>3</td>
<td>1.0</td>
<td>90.98%</td>
<td>83.18%</td>
<td>86.91%</td>
<td>374989</td>
<td>133</td>
</tr>
<tr>
<td>5</td>
<td>1.0</td>
<td>90.97%</td>
<td>83.05%</td>
<td>86.83%</td>
<td>368720</td>
<td>167</td>
</tr>
<tr>
<td>7</td>
<td>1.0</td>
<td>91.03%</td>
<td>83.05%</td>
<td>86.86%</td>
<td>351602</td>
<td>152</td>
</tr>
<tr>
<td>10</td>
<td>1.0</td>
<td>90.96%</td>
<td>82.92%</td>
<td>86.75%</td>
<td>371109</td>
<td>162</td>
</tr>
<tr>
<td>3</td>
<td>1.15</td>
<td>90.80%</td>
<td>83.31%</td>
<td>86.89%</td>
<td>388023</td>
<td>144</td>
</tr>
<tr>
<td>5</td>
<td>1.15</td>
<td>90.99%</td>
<td>83.31%</td>
<td>86.98%</td>
<td>382513</td>
<td>145</td>
</tr>
<tr>
<td>10</td>
<td>1.15</td>
<td>90.97%</td>
<td>83.05%</td>
<td>86.83%</td>
<td>389340</td>
<td>143</td>
</tr>
<tr>
<td>5</td>
<td>1.25</td>
<td>90.98%</td>
<td>83.18%</td>
<td>86.91%</td>
<td>414609</td>
<td>124</td>
</tr>
<tr>
<td>10</td>
<td>1.25</td>
<td>90.98%</td>
<td>83.18%</td>
<td>86.91%</td>
<td>412378</td>
<td>111</td>
</tr>
<tr>
<td>10</td>
<td>1.5</td>
<td>91.32%</td>
<td>82.59%</td>
<td>86.73%</td>
<td>698470</td>
<td>60</td>
</tr>
<tr>
<td>10</td>
<td>1.75</td>
<td>90.98%</td>
<td>76.52%</td>
<td>83.12%</td>
<td>1874179</td>
<td>31</td>
</tr>
<tr>
<td>10</td>
<td>2.0</td>
<td>90.50%</td>
<td>78.56%</td>
<td>84.11%</td>
<td>4244905</td>
<td>23</td>
</tr>
<tr>
<td>Any</td>
<td>Max_value</td>
<td>89.02%</td>
<td>86.08%</td>
<td>87.53%</td>
<td>9062872</td>
<td>11</td>
</tr>
</tbody>
</table>

Table 5-5 Birch evaluation

The experiment of variable B stops at 10, because this parameter cannot exceed the number of CF a leaf can have, in our case, the number of CF a leaf is no more than 10. From the experiments we can see the change of B does not affect the final result a lot, only minor difference is made and there is not much trend to determine which value of B could be the best setting. However, the T, as we discussed before, has affected the result prominently. A higher T shortens the execution time, increases the number of comparisons and lowers three evaluation metrics dramatically after exceeding a certain point. The phenomenon can be explained by the scale of clusters, because if the value of T is too big, a subcluster can hold too many concepts, thus there would be some big clusters in the ontology and the time to calculate the tfidf vectors and cosine similarity will dramatically decrease.
5.2.4 No clustering

In this section, we delete the similarity assignment and ontology clustering part of the system, the ontology parsing, label parsing, similar cluster identification is used. In this case, similar cluster identification becomes similar concept identification, because we consider every concept as a cluster. The evaluation result is shown as follows Table 5-6

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
<th>Comparisons</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>No clustering</td>
<td>90.96%</td>
<td>81.66%</td>
<td>86.06%</td>
<td>208970</td>
<td>413</td>
</tr>
<tr>
<td>SAMBO</td>
<td>89%</td>
<td>86.1%</td>
<td>87.5%</td>
<td>9026626</td>
<td></td>
</tr>
</tbody>
</table>

Table 5-6 Evaluation of our system without clustering algorithm

It gains extremely good result. The number of comparisons reduce around 43 times and precision goes little higher. The drawback is the downwards of recall and f-score but they are in acceptable degree. The running time is a little bit longer than the other system version with clustering algorithm. It is because by considering every concept as a cluster, we have to create an array of size of total number of different words in two ontologies for each cluster, and then compute cosine similarity for every two clusters. The less number of clusters, the smaller computation we have. However, this result also sheds light on the good performance of our universal pre-processing for the ontology alignment system. Unlike clustering algorithms mentioned above, we do not need to tune every parameter in this case, and still get a fairly good result.

This result shows that the good performance of our system, these automated procedures are of great value. They can benefit the system even without the involvement of clustering techniques.

5.2.5 Similarity Assignment

Four similarity assignment methods are implemented and evaluated in our system, they are Wu and Palmer similarity, Dennai similarity, Jaccard similarity and a Alsayed similarity respectively.

To evaluate similarity assignment, previous experiments of clustering algorithms are taken into consideration. In order to evaluate similarity assignment methods properly and efficiently, we choose two clustering algorithm setting with acceptable evaluation results, other parts of procedures remain as the basic setting.
First one we choose is Chameleon with $k$ of 1, threshold of 0.1 and $\vartheta$ of 1. For the Alsayed clustering we already got the result from section 5.2.2. The other three similarity assignment methods are evaluated and recorded in the Table 5-7 below.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
<th>Comparisons</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wu and Palmer</td>
<td>91.34%</td>
<td>82.12%</td>
<td>86.49%</td>
<td>597348</td>
<td>282</td>
</tr>
<tr>
<td>Dennai</td>
<td>91.42%</td>
<td>82.26%</td>
<td>86.60%</td>
<td>510788</td>
<td>290</td>
</tr>
<tr>
<td>Jaccard</td>
<td>91.24%</td>
<td>82.45%</td>
<td>86.63%</td>
<td>487024</td>
<td>62</td>
</tr>
<tr>
<td>Alsayed</td>
<td>91.37%</td>
<td>82.39%</td>
<td>86.65%</td>
<td>482543</td>
<td>55</td>
</tr>
</tbody>
</table>

Table 5-7 Evaluation of similarity assignment methods with Chameleon

Second one we choose the Birch with $B$ of 5 and $T$ of 1.15. The evaluation results of four similarity assignment methods are shown as Table 5-8 below.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
<th>Comparisons</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wu and Palmer</td>
<td>90.94%</td>
<td>82.78%</td>
<td>86.67%</td>
<td>429475</td>
<td>339</td>
</tr>
<tr>
<td>Dennai</td>
<td>90.86%</td>
<td>83.25%</td>
<td>86.88%</td>
<td>381725</td>
<td>389</td>
</tr>
<tr>
<td>Jaccard</td>
<td>90.84%</td>
<td>83.11%</td>
<td>86.81%</td>
<td>383330</td>
<td>136</td>
</tr>
<tr>
<td>Alsayed</td>
<td>90.99%</td>
<td>83.31%</td>
<td>86.98%</td>
<td>382513</td>
<td>145</td>
</tr>
</tbody>
</table>

Table 5-8 Evaluation of similarity assignment methods with Birch

As we have illustrated in previous chapter, the first two similarity assignment methods can be categorized into the hierarchical similarity assignment while the last two are categorized as context based methods. From these two groups of experiments we can see, Dennai, as a hierarchical based similarity assignment method, is better than Wu and Palmer’s almost in every aspect besides the execution time. The uprising of precision, recall and f-score can give the credit to the good performance of the introduction of FPD, while the slower execution time is for the reason that the Dennai similarity assignment method need more computation to generate similarity. The Alsayed similarity assignment method is slightly better than Jaccard similarity assignment method in the evaluation result.

By comparing two groups of similarity assignment methods we can see the context based group have better performance than the hierarchical based one. A presumption is the depth information cannot correctly reflect the degree of details, a concept that is far from the root does not guarantee it to be very detailed. The level of
details of a concept depends on how much information does the organizer of that ontology have for that part of knowledge. Meanwhile, the hierarchical similarity assignment methods have much more execution time than the other group, because it needs more information about the concept, like depth of a concept, distance between every two concepts, the computation is more complex and our chosen structure to store the similarity might not be efficient enough, which could be further studied. The context based group in turn has a slightly better evaluation result and help speed up more than 200 seconds.

5.2.6 Cosine Similarity Threshold

For the previous experiments, the cosine similarity has been set as 0.05 all the time, the reason is that we want to find as much similar clusters as possible. The possible value of cosine similarity ranged from -1 to 1, a value close to 1 means strong similarity, a value close to -1 means strong dissimilarity. In addition, a cosine similarity value of -1 is set when we have value of 0 in the denominator of the formula of cosine similarity. Generally speaking, a higher threshold will lead to smaller number of similar clusters that can be matched, many incorrect matching are ruled out, only the highly similar clusters will be selected as alignment suggestions, so that the precision will go up. The recall is going to be lower as the threshold getting higher, the numerator of recall is getting smaller because some possibly correct alignment with a low similarity value will be ruled out by our system.

The setting is applied the same as the experiments for similarity assignment methods, we choose two clustering algorithm setting with acceptable evaluation results, other parts of procedures remain as the basic setting.

First one we choose is Chameleon with \( k = 1 \), threshold of 0.1 and \( \delta \) of 1. For the Alsayed clustering we already got the result from section 5.2.2. The other three similarity assignment methods are evaluated and recorded in the Table 5-9 below.

<table>
<thead>
<tr>
<th>CS threshold</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90.11%</td>
<td>84.70%</td>
<td>87.32%</td>
<td>1709836</td>
<td>59</td>
</tr>
<tr>
<td>0.05</td>
<td>91.36%</td>
<td>82.32%</td>
<td>86.61%</td>
<td>481948</td>
<td>59</td>
</tr>
<tr>
<td>0.1</td>
<td>92.06%</td>
<td>77.97%</td>
<td>84.43%</td>
<td>294424</td>
<td>59</td>
</tr>
<tr>
<td>0.3</td>
<td>93.18%</td>
<td>65.83%</td>
<td>77.16%</td>
<td>125823</td>
<td>58</td>
</tr>
<tr>
<td>0.5</td>
<td>95.23%</td>
<td>56.66%</td>
<td>71.05%</td>
<td>69965</td>
<td>58</td>
</tr>
</tbody>
</table>
Table 5-9 Evaluation of cosine similarity threshold with Chameleon

Second one we choose the Birch with B of 5 and T of 1.15. The evaluation results of cosine similarity threshold are shown as Table 5-10 below.

<table>
<thead>
<tr>
<th>CS threshold</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
<th>Comparisons</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>90.46%</td>
<td>83.84%</td>
<td>87.02%</td>
<td>737305</td>
<td>138</td>
</tr>
<tr>
<td>0.05</td>
<td>90.87%</td>
<td>83.38%</td>
<td>86.96%</td>
<td>383803</td>
<td>144</td>
</tr>
<tr>
<td>0.1</td>
<td>91.06%</td>
<td>81.33%</td>
<td>85.92%</td>
<td>253463</td>
<td>145</td>
</tr>
<tr>
<td>0.3</td>
<td>92.80%</td>
<td>66.29%</td>
<td>77.34%</td>
<td>90681</td>
<td>136</td>
</tr>
<tr>
<td>0.5</td>
<td>94.38%</td>
<td>53.17%</td>
<td>68.02%</td>
<td>29247</td>
<td>149</td>
</tr>
<tr>
<td>0.7</td>
<td>96.89%</td>
<td>34.96%</td>
<td>51.38%</td>
<td>7531</td>
<td>136</td>
</tr>
<tr>
<td>0.85</td>
<td>99.15%</td>
<td>22.96%</td>
<td>37.28%</td>
<td>1844</td>
<td>136</td>
</tr>
<tr>
<td>1</td>
<td>100.00%</td>
<td>2.64%</td>
<td>5.14%</td>
<td>66</td>
<td>137</td>
</tr>
</tbody>
</table>

Table 5-10 Evaluation of cosine similarity threshold with Birch

As these experiments have shown, with the cosine similarity threshold going higher, the precision goes up, while recall, f-score and number of comparisons go down rapidly. These results have proved our hypothesis and the metrics change as expected. The execution time is around the same all the time because the computation amount remains the same. We can deduct that if the more precise suggestions are wanted, we can increase the cosine similarity threshold in order to get a set where majority of the suggestions are correct, and later reset the threshold to a low level in order to get more possibly correct suggestions. Generally speaking, a value of 0 for cosine similarity grants the highest f-score, however, considering the threshold being 0, almost every pair between every two clusters are to be processed by the ontology alignment system. It enlarges the number of comparisons a lot, for example in the first experiment quadruple number of comparisons have to be done after the threshold lowers down from 0.05 to 0, however in the case of Birch, only twice number of comparisons have to be done.

In conclusion, the experiments show that 0.05 is a reasonable value for the cosine similarity threshold, it has good trade-off between evaluation result and the purpose of reducing search space.
5.3 Alternative ontology alignment system

This paper only conducts experiment regarding to the SAMBO ontology alignment system, in order to test the applicability of our system, more experiments in other ontology alignment platforms should be conducted. Meanwhile, we also calculate the precision, recall and f-score metrics before using the SAMBO system. That is, the evaluation towards the output of our system (input of SAMBO system). Through this kind of evaluation, we can see the potential of our system, because SAMBO rules out some correct alignment suggestions during the processes.

Take one set of procedure as example, Chameleon with 0.1 of threshold, 1 of k, use Alsayed similarity assignment method, cosine similarity threshold is set as 0.05. We conduct the experiments for this procedures to see the difference after using SAMBO system. The result is shown as Table 5-11 below:

<table>
<thead>
<tr>
<th>Situation</th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>After SAMBO</td>
<td>91.36%</td>
<td>82.32%</td>
<td>86.61%</td>
</tr>
<tr>
<td>Before SAMBO</td>
<td>0.30%</td>
<td>94.00%</td>
<td>0.59%</td>
</tr>
</tbody>
</table>

Table 5-11 Before and After SAMBO evaluation

We can see from the table that the SAMBO system improve the precision and f-score dramatically while recall drops around 12%, it shows that SAMBO still rules out some correct alignment. Improve the ability of ontology alignment system to identify more number of correct alignment suggestions are not in the scope of the study of this paper, but we can clearly see that with the development of ontology alignment system, or using other ontology alignment system which is better matched with our system, recall and f-score could have some better performance. We now find out the context based similarity assignment methods can generate better results compared with hierarchical based similarity assignment methods, but maybe the matchers used during the procedure of SAMBO are better matched with context based similarity assignment. So, we can further evaluate other ontology alignment systems with different matchers, in order to see if they fit some different types of similarity assignment methods.

5.4 Comparisons with LOAD system

In this section we evaluate our system compared with the previous system LOAD. Three solutions generated from our system are compared with the best
solution in LOAD. The setting of parameters of our system are shown as follows. The similarity assignment method used here is Alsayed similarity assignment method. Firstly, two Chameleon set both uses k of 1, threshold of 0.1 and of 1, the difference is that the first set uses cosine similarity threshold as 0.05, and the second set uses cosine similarity threshold as 0.4. Then, the Birch uses B of 5 and T of 1.15 with cosine similarity threshold of 0.05. The experiment results are shown as Table 5-12.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F-score</th>
<th>Comparisons</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOAD</td>
<td>93.1%</td>
<td>32.2%</td>
<td>47.8%</td>
<td>264536</td>
<td></td>
</tr>
<tr>
<td>Chameleon 1</td>
<td>91.36%</td>
<td>82.98%</td>
<td>86.97%</td>
<td>459561</td>
<td>73</td>
</tr>
<tr>
<td>Chameleon 2</td>
<td>93.96%</td>
<td>61.61%</td>
<td>74.42%</td>
<td>96495</td>
<td>63</td>
</tr>
<tr>
<td>Birch</td>
<td>90.99%</td>
<td>83.31%</td>
<td>86.98%</td>
<td>382513</td>
<td>145</td>
</tr>
<tr>
<td>SAMBO</td>
<td>89%</td>
<td>86.1%</td>
<td>87.5%</td>
<td>9062872</td>
<td></td>
</tr>
</tbody>
</table>

Table 5-12 performance compared with LOAD

Firstly, as we compare the LOAD with Chameleon 1 and Birch set, although the precision is down around 2%, the number of comparisons increases doubly, the f-score and recall dramatically increase. It is due to we try to maintain as much number of correct suggestions as possible. Meanwhile, as the LOAD system is semi-automatic, the procedure involves the domain experts in order to identify the subject of clusters and merge similar ones manually, which is very time consuming. On the contrary, our system owns an automated procedure and the execution time is usually within 3 minutes, with best performance of only one minute.

Secondly, the Chameleon 2 set uses cosine similarity threshold as 0.4 in order to extract the alignment suggestions with higher possibility of correctness. All the statistics outperform the LOAD. With higher precision, recall and f-score, the number of comparisons drop extremely as well. In conclusion, our system shows that it can replace the LOAD system properly.

5.5 Brief summary

In this chapter we conduct experiments regarding to every possibly changeable parameter in our system. The results have shown that our system has ability to reduce a large amount of the search space while affecting little to the recall and f-score.

The comprehensive experiments show that Chameleon and Birch are better than K-means algorithm in dealing with ontology clustering problems. It is discovered
that KNN, which is the first phase of Chameleon, also has the ability to do a good job in ontology clustering, meanwhile the characteristic of no need to tune parameters is a great attribute of KNN, KNN combines with the default procedures of our system can generate a good result. Best parameters setting for Chameleon during the experiments is k equals 1, threshold equals 1.0 and $\partial$ equals 1, this setting of parameters gains a f-score of 86.78% and reduce the search space to 479783 with 59 seconds execution time. In the meantime, the best parameters setting for Birch is B of 5 and T of 1.15, it achieves 86.98% f-score and reduce the search space to 382513 in 145 seconds. Compared with the original SAMBO system, it reduces more than 20 times of search space while only at the cost of less than one percent of f-score. It shows our system is of good performance.

Similarity assignment methods, cosine similarity threshold are evaluated as well, and through the analysis we get the most suitable similarity assignment methods for our case, the Alsayed similarity assignment method. Another conclusion is that the cosine similarity threshold should be used differently in different situation, but a value of 0.05 is reasonable for most cases.

The experiment result shows that our proposed procedures have good performance even without using clustering algorithms, it mainly thanks to the good labeling of concepts, the similar concepts can be paired easily one by one, it also shows the extraordinary ability of text mining step. Although it takes much longer time than the other methods as well as a slightly smaller f-score, it reduces the most search space for the ontology alignment system.

By comparing the experiment results with previous system LOAD, we can see in every aspect our system outperforms the system LOAD.

In conclusion, the procedures proposed in this paper have managed to reduce the search space, improve the precision and control the downfall of the recall and f-score in a manageable scope, as well as taking small time for running these pre-processing procedures. The system is suitable for use and have a good performance.
6 Future work

The research work presented in this paper provides a first step towards the alignment of large ontologies by properly exploiting the data mining methods. Therefore, the proposed framework can be seen as the starting point for further improvements and refinements.

The paper focuses on the alignment of the two ontologies which is called Anatomy track for the OAEI initiative. Because the experiments are only restricted to these two certain ontologies, we do not know if the procedures and algorithms proposed in this thesis would have the same performance to the other ontology alignment problems. Further experiments towards the other ontology alignment problems shall be made to test if the word of this paper can be applied universally. The program illustrated in this paper also only can deal with the owl format file of ontology due to the same reason. If we want to use this program to make research on other format of ontology file, specific parse module needs to be implemented.

Structure to store similarity between concepts can be studied and get some changes. In this paper, we use a two dimensional array to store the similarity, but in some similarity assignment method, we observe that there are many concepts do not have any connection with each other, thus the similarity values are 0 in the corresponding positions. We presume that if we use linked lists or other structures to track the similarity value, it would be more time and space efficient. It needs experiments to prove if this presumption will work.

Clustering algorithms can be further studied and analyzed, now we have figured out the best parameters by making experiments and compare the result with the standard reference. In order to make it useful for the ontology alignment problems that without the standard reference, metrics like intra-connectivity can be utilized to make sure every cluster is in the desired scale. Study can be made to get the desired number of clusters for every specific ontology, for example, the size of the number of concepts and the structure of the ontology can be taken into consideration.

Birch and Chameleon can be further exploited after we get knowledge of the determination of proper number of clusters. Because these two are hierarchical clustering algorithms, we can simply iterate different levels of hierarchy to get different number of clusters. Take Birch as an example, a higher level non-leaf consists of all the lower non-leaves and leaves that connects to it. So the higher level
“supercluster” will have many subclusters inside and the overall number of clusters decrease. When we have the number of clusters conform to our preset threshold, then output it. Thus we can get the wanted size of the clusters.

Because our system uses cosine similarity of tfidf vectors to automatically identify similar clusters between two ontologies, the label of concepts become vital in this case. Presume a case that labels of similar clusters are vague or of great different, since the identification of similar clusters are based on text mining, a great amount of alignment suggestions will be lost. So that either in that case we identify similar clusters manually or some new methods should be brought up in order to improve performance.
7 Conclusion

A system is implemented in this paper for the purpose of reducing search space for large ontology alignment problem, it achieves the goal by partitioning two large ontologies and then identifying similar clusters, only pairs of concepts in similar clusters will be the input of ontology alignment system. The paper uses SAMBO system as the ontology alignment system, human and mouse anatomy ontologies as the source ontologies.

Our system follows a certain series of automated procedures. Firstly, the ontology file is parsed to get the information about concepts and their relationship. Secondly, similarity matrix is generated using the information extracted from the ontology. Thirdly, clustering algorithm uses the similarity information to cluster the concepts. Fourthly, calculate tfidf vector of each cluster and use cosine similarity to filter out the similar clusters. Finally, the pairs, which each element from each ontology, in the similar clusters are stored and used as the input of ontology alignment system.

Evaluation of precision, recall, f-score, number of ontology alignment suggestions and execution time is done regarding to the different parameters in our system. We evaluate four similarity assignment methods and draw the conclusion that a context based similarity assignment method, which each concept uses relationship with its children and parents, has best performance. The hierarchical type of similarity assignment method has poorer performance than the context based one, this conclusion reflects on the precision, recall, f-score and execution time.

In this paper three clustering algorithms are evaluated, each parameter of every clustering algorithm are evaluated separately. It turns out that Chameleon and Birch have better performance than K-Means. The best solution reduces 20 times number of comparisons for ontology alignment system while only slightly affecting the f-score and recall, meanwhile the execution time only takes one or two minutes. Best parameters setting for Chameleon during the experiments is k equals 1, threshold equals 1.0 and \( \delta \) equals 1, this setting of parameters gains a f-score of 86.78% and reduce the search space to 479783 with 59 seconds execution time. In the meantime, the best parameters setting for Birch is B of 5 and T of 1.15, it achieves 86.98% f-score and reduce the search space to 382513 in 145 seconds. Compared with the original SAMBO system, it reduces more than 20 times of search space while only at
the cost of less than one percent of f-score. It shows our system is of good performance.

There are two additional discoveries, the first one is that when the Chameleon only deploys the first phase, the evaluation result is satisfying. Only first phase of Chameleon means that it degrades to knn, so that we can use KNN to cluster the ontology, the most advantage is that with KNN we do not need to tune parameters. The second discovery is that our system has good performance even without clustering algorithm involved, the tfidf vector and cosine similarity can capture the most of information about the ontology.

Compared with the previous system LOAD, through the different setting of parameters, we can outperform LOAD system in every metric aspect.

In conclusion, the methods and procedures proposed in this paper has solved the proposed problem of reducing search space for ontology alignment.
8 Bibliography


[12] Chiatti, A., Dragisic, Z., Cerquitelli, T. and Lambrix, P., 2015, October. Reducing the search space in ontology alignment using clustering techniques and
topic identification. In Proceedings of the 8th International Conference on Knowledge Capture (p. 21). ACM.


[33] Zlatan Dragisic, Valentina Ivanova, Huanyu Li, Patrick Lambrix, Experiences from the Anatomy track in the Ontology Alignment Evaluation Initiative, 2017