Knowledge Graph-Based Approach For Dynamic Ontology Generation

Hansika Gunasekara[1], Thushari Silva[2]
[1]*Department of Computational Mathematics, University of Moratuwa,* Moratuwa, Sri Lanka.;198077l@uom.lk
[2]*Department of Computational MathematicsUniversity of Moratuw*Moratuwa, Sri Lanka.thusharip@uom.lk

Abstract- **A knowledge graph is a data model representing real-world entities and relationships in a machine-readable format providing a comprehensive view of a specific domain or multiple domains. Dynamic ontology is a concept that refers to the idea that the fundamental nature of reality changes over time. Dynamic ontology generation using a knowledge graph automatically creates a new ontology or updates an existing ontology. This process keeps the ontology up-to-date with the changing real-world information. The most important part of dynamic ontology generation is integrating the domain ontology into the knowledge graph and updating the knowledge graph accordingly. The primary purpose of the research is to develop a dynamic environment that integrates the domain ontology with the knowledge graph. The main problem here is the dynamic integration of new data and concepts with changes in real-world information. To achieve this task graph-based ontology mapping framework is developed with matrix-based graph merging, graph clustering, cluster label propagation and matrix-based ontology mapping. The new mapping algorithm was tested with real-time dynamic data and compared with existing systems. The proposed approach outperforms the existing system in accuracy, relevancy and introducing new concepts to the ontology.**

1.Introduction

Ontologies are a crucial element of the semantic web. Ontologies are used in the Semantic Web, Artificial Intelligence, Systems Engineering, Biomedical Informatics, Software Engineering, Enterprise Bookmarking, Library Science, and Information Architecture to represent knowledge about or a part of the domain. They aim to capture basic knowledge by providing appropriate terms and formal relationships between them. Ontologies are an integral approach mainly used to represent acquired knowledge and ensure data and knowledge integration. However, most of them are generated manually by domain experts and ontology engineers familiar with the theory and practice of ontology construction [1]. Automatic ontology generation aims to convert new knowledge into ontological form by enabling related processing techniques, such as semantic search, recommendation and retrieval.

Moreover, automatic ontology generation will significantly reduce the expert involvement and time required to build ontologies [2]. There are two groups of ontologies: static and dynamic. A static ontology elaborates on items in existence, their attributes and their relationship with them. Dynamic ontology explains the domain as states and their transitions, including processes according to changes in information. Dynamic ontology generation has become an essential topic in semantic engineering in every field, with massive amounts of data emerging. Most dynamic ontologies are generated using structured data [3], such as relational schema and web-based XML data. Approaches to converting unstructured text into dynamic ontological format have yet to develop fully. Most approaches for ontology learning convert text to ontology format and need expert involvement [4] [5]. A few studies have been carried out in dynamic ontology generation using unstructured text corpus[6]

The knowledge graph is one of the emerging fields of knowledge representation. Many studies [5] have been done on knowledge representation using a knowledge graph because it is presented in graph format. Knowledge graphs aim to construct large structured knowledge repositories that machines can understand [7]. Knowledge graphs have gained much attention in natural language processing as a new type of knowledge representation. Knowledge graphs can efficiently organise and represent knowledge in advanced applications [8]. Knowledge Graphs (KGs) such as Freebase [9]and YAGO [10]have been widely adopted in a variety of natural language processing. The knowledge graph consists of a set of triplets {(h, r,t)} where h,t and r are the head and tail entities and the relationship, respectively. Ontologies are stored in various forms like RDF, RDFS, XML, OWL etc.

Building ontologies using learning techniques and knowledge graph representation of unstructured text are familiar problems. But, building a framework for lightweight log generation through the knowledge graph must overcome many issues. It should facilitate the system to integrate any knowledge graph anytime without problem for domain-independent ontology. The graph clustering-based approach to ontology introduced in [11]generates dynamic ontology. It ensures dynamicity even in the presence of the new device in the Internet of Things domain. This method is improved to generate dynamic ontology using a knowledge graph, preserving the dynamic nature of a new concept or source of text.

This paper aims to develop a domain-independent dynamic ontology generation framework that generates ontological forms from unstructured text data. The study is motivated by the need for more fully-automated domain-independent dynamic ontology generation systems. The framework utilises a knowledge graph to be mapped and tailored to suit target domain ontologies. This framework facilitates continuous integration of unstructured text data to maintain dynamicity. Further, this dynamic ontology ensures lightweight representation and speedy access to information through the index for future development in querying and recommendation.

The rest of the paper is organised as follows: the next section reviews the relevant literature. Then, the Proposed approach

section provides for knowledge graph integration of unstructured text data to the dynamic ontology process. Next, the implementation section describes all aspects of the techniques used in the dynamic ontology generation process. Then the result and discussion describe the implementation and evaluation of the proposed system. Finally, the conclusion of the proposed approach and future works are discussed.

2.literature review

The dynamic ontology generation approach is based on ontology generation, which enables it to update itself based on the changes in data. Studies regarding dynamic ontology generation can be divided into two types according to the input data source. They are Dynamic ontology generation using structured data and dynamic ontology generation using unstructured data. Although many studies were carried out on dynamic ontology generation using structured data sources, a few studies were conducted on the dynamicity of unstructured text data. But there are a massive amount of ontology learning approaches that generate ontologies from text data automatically. The dynamic ontology generation is an extension of ontology learning in a sustainable environment that enables the evolving and versioning.

2.1 Dynamic ontology generation using structured data

Most dynamic ontology generation studies are based on structured data transformation [12] using structured databases. Evolution changes are defined in [13]as a succession of operations the user wants to apply to the schema or the ontology data. This evolution aims at adapting the ontology to the changed domain. According to [14], these tasks usually occur during the use phase of the ontology. Ontology Dynamics clearly define the evolution criteria. [15]qualify the maintenance of ontology as the most crucial criterion. Evolution has to maintain whatever relies on ontology. [16] presents a new way to manage the lifecycle of an ontology incorporating both versioning tools and the evolution process.

2.2 Ontology Learning

The research [6]aims to develop a domain-independent automatic ontology generation framework that converts unstructured text into a domain-consistent ontological form. The framework generates knowledge graphs from an unstructured text corpus and refines and corrects them to be consistent with domain ontologies. Moreover, the knowledge graph can automatically update the knowledge without expert involvement, comparing the ontology learning algorithms discussed in [17]. The study in [18] has been carried out to automatically transfer the relationships to the ontology using semantic graphs and convolutional neural networks. They have used semantic and thematic graphs are used for knowledge extraction.

Further [18] have produced a hierarchical structure with in-depth integration of data for reducing the matching time and dynamic labelling to remove domain dependencies. According to the survey in[17], ontology learning from text methods has been presented to convert text data to ontology components in 22 studies. But no work was only found on dynamic ontology generation using text in the survey. Although the automatic generation of the ontology using text based on ontology learning is vital, rare work was found for dynamic ontology construction.

The summary of ontology learning approaches in Table I represents a mixture of natural langue processing, machine learning and network-based technologies used in each approach. Expert involvement is needed for all approaches and does not address dynamic evolution.

2.3 Dynamic ontology generation from text

Knowledge graph-based automatic ontology development for consistent dynamic features was ensured using knowledge graph embedding techniques [6]. The framework utilises refined KGs mapped and tailored to fit into target domain ontologies. However, generating ontologies from refined KGs will not only overcome the limitations of ontologies, such as data integration. Therefore, evaluation or performance analysis has yet to present in this research.

Recent research has addressed the problems of complex, heavyweight static ontology on the Internet of Things systems. To address this concern, [11] proposed a lightweight ontology using only the most essential concepts and clustering techniques. It provides dynamic semantics automatically to include additional concepts using machine learning techniques. The proposed model somewhat reduces the query response time and memory consumption compared to the existing ontology. The dynamicity of the ontology is addressed in the presence of new device or sensor data in the system using clustering. However, considering the literature, such a lightweight log for unstructured text data still needs to be addressed.

2.4. Cluster label propagation

Cluster label propagation is a method for cluster labelling, which assigns a label or description to a group of similar data points that have been grouped in a cluster. This technique can be applied to different data types and used for tasks such as image classification, text mining, and bioinformatics. Label propagation is an iterative technique for assigning labels to unlabeled data points based on the labels of their nearest neighbours. The idea is that data points close to each other are likely to have similar labels. The algorithm starts with a small set of labelled data points. Then, it propagates the labels to the unlabeled data points by iteratively updating them based on the labels of their nearest neighbours[20]

Several methods for graph cluster labelling, including Keyword-Based Labeling, identify the most frequent words or phrases in the data associated with each cluster and use them to generate a label. Centrality-Based Labeling labels clusters based on the centrality measures of the vertices in the cluster, such as degree centrality, betweenness centrality, or eigenvector centrality[21].

Table 1
Summaries of ontology learning approaches

Approach	Aim	Main technique	Source	Main goal	Assessment of acquired knowledge
Aussenac-Gilles and colleagues" approach [19]	To create a domain model by using NLP tools; To create a domain model by using linguistics technique	Knowledge extraction with syntactic patterns with a concordancer; Relation extraction based on linguistic patterns; Term extraction based on distributional analysis	Text; Existing ontologies; Terminological databases	To learn concepts, To learn new relations	User; Domain expert; Human intervention
Faatz and Steinmetz approach [5]	To enrich the concepts by extracting meaning from the WWW	Statistical approach; Semantic relativeness	Corpus from WWW	To enrich the ontology with new concepts	Domain expert
Khan andLuo'ss method [4]	To create a domain ontology using clustering techniques and WordNet	Clustering	Text	To learn new Concepts	Domain expert

Attribute-Based Labeling: labels clusters based on the attributes of the vertices in the cluster, such as labels or categorical variables [22]. Community Detection-Based Labeling labels clusters based on the community structure of the graph, where communities are defined as groups of vertices that are more densely connected than vertices in other groups[23]. Centroid-Based Labeling labels clusters based on their centroid or representative vertex properties.[24]

In network-based clustering, the most influenced node in the cluster is elected as the cluster label. According to[25], this cluster labelling algorithm can easily embed into the new class name suggestion in the ontology generation process. Therefore, this is one of the best-suiting algorithms for labelling knowledge graph-based clusters.

The proposed study aims to develop a fully automatic, domain-independent lightweight dynamic ontology generation framework using various types of unstructured text and enabling the integrating of one or many ontologies at any stage to fill this gap in the literature.

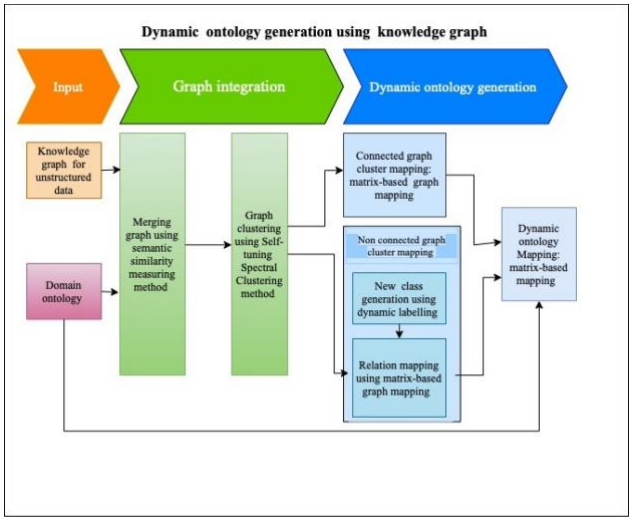
3. The Proposed Approach

The main focus of the proposed approach is to develop a platform to generate dynamic ontology using a knowledge graph, which enables to accommodate the ontology through knowledge behind unstructured data. The proposed approach for the dynamic ontology generator is described in Figure 1.

Initially, the dynamic ontology is generated by the steps introduced in the overview. Inputs of the dynamic ontology generator are static domain ontology and knowledge graph of unstructured data. The dynamic ontology generation process is briefly discussed in the following sub-sections. First, the dynamic ontology generation is based on the knowledge graph integration to the ontology using an RDF alignment method. The RDF alignment merges two graphs using semantic similarity and self-tunning spectral clustering using graph matrices. Then the connected graph matrix is transferred to the ontology iteratively with the presence of new text content. The framework updates the ontology every time new data is added to the knowledge graph.

Fig. 1 The proposed approach for the dynamic ontology generator

3.1 Identified Inputs



The Domain ontology and the knowledge graph for the unstructured data are the primary inputs of the proposed framework. The graph representation consists of two parts. Firstly the graphs are stored in Adjacency matrices, and then the hashing index is defined for the references of matrix elements and graph data. All vertices and edges of the graph are transferred to the hashing index while storing graphs in the adjacency matrix, as both graphs are large. The RDF data and Knowledge graph are stored in the Adjacency Matrix [26]. The Adjacency Matrix is an nxn matrix (where n is the number of nodes), with rows and columns labelled by graph vertices. Adjacency Matrix describes a graph by representing which vertices are adjacent to which other vertices; the cell of the Adjacency Matrix is filled with the label of the edge.

After working with the adjacency matrix, referring to the ontology again for querying or recommendation is necessary. If the graph is large, then this will take more time to search for the relevant node. For this task, hashing can be applied, giving an index of the desired node in constant time. The Minimal Perfect Hash Function is used for minimising collisions. Hence, this will save storage and decrease the Time Complexity[27].

3.2 Merging graph using the semantic similarity measuring method

After storing graphs in adjacency matrices, semantic similarities between each vertex and each edge were stored in a list of vectors. Then those similarities can be used when comparing matrix points for vertices and edges of the graph. The merging steps of the two adjacency matrices are described in Algorithm 1. Here Q is the adjacency matrix of the RDF graph, G is the adjacency matrix, and the final merged graph is defined as Z. Two matrices are merged by comparing the similar points between two matrices. Next, non-matching elements are appended to the matrix where there is no matching column, and a new column is added. Similarly, new rows are added when a non-matching row element arrives. Finally, the relations among nodes are defined by the matrix body referring to column names and row names when new relationships are introduced and updated in the matrix's j^{th} position.

3.3 Graph clustering using the self-tunning Spectral Clustering method

Clustering is a widely used technique for unsupervised data analysis, with applications in statistics, computer science, biology, social sciences, and psychology. In clustering, the goal is to assign unlabelled data to groups; similar data points are assigned to the same group.

After merging two graphs using the matrix, it is necessary to integrate the graph elements to identify the semantic connectivity with existing graphs. Graph clustering was used to gather proper integrating elements. Clustering is one of the best methods for finding similar nodes. The spectral clustering algorithm was introduced [28]for clustering graphs with high noise. This method is reasonably fast, especially for sparse datasets.

Moreover, spectral clustering treats data clustering as a graph-partitioning problem without assuming the form of the data clusters. Spectral clustering is a technique in graph theory used to identify a group of nodes in a graph on the edges connecting them. This method is flexible and allows the clustering of non-graph data. It uses information from the eigenvalue spectrum of special matrices built from a graph or dataset. The following sections discuss the construction of matrices, the interpretation of their spectra, and the use of eigenvectors to assign data to clusters.

Then, clusters in this subspace are obtained using various clustering algorithms such as k-means. The adjacency matrix is an affinity matrix determining how close or similar any pair of points are in space. Spectral clustering performs well with standard clustering algorithms such as K-means [29]. However, since K-means clustering requires knowledge of the number of clusters, the expected number of clusters and a parameter for the similarity threshold need to be given. Therefore the optimal number of clusters for spectral clustering was derived in the self-tunning spectral clustering. Under spectral clustering, the elements should be gathered in the design space to estimate the number of clusters. Then an optimal number of clusters is determined based on the perturbation theory and the proposed spectral graph theory [30]. An improvement of the self-tunning clustering method that overcomes three problems of spectral clustering, i.e., cluster initialisation, cluster specification, and noise-robustness.[31] was used.

Algorithm 1. The matrix-based graph merging algorithm

Q= RDF graph matrix
G= knowledge graph matrix
Subjects of Q & G,
Q_s= { q_1,q_2……q_{ms})
G_s= {g_1,g_2,……..g_{ns}),
Objects of Q & G,
Q_o= { q_1,q_2……q_{mo}),
G_o={g_1,g_2,……..g_{no}),
Initialise
Z_s= Q_s, Z_o= Q_o, Z_r = Q_r, p= ms, s= mo
Check the cosine similarity index for G_s and Q_s, Z_o and G_o, Z_r and G_r
For all i
If G_{si} not equal to O_{si}
(where the similarity index< threshold)
Add a column to Z_s
p= p+1
Append the whole column of the matrix to the Z matrix
For all j
If Z_{oj} not equal to G_{oj}
(where cosine similarity index< threshold)
Add node to rows of Z
s=s+1
Append the whole row of the G matrix to the Z matrix
Else
For all j
If Z_{oj} not equal to G_{oj}
Add node to rows of Z
s=s+1
Append the whole row of the G matrix to the Z matrix
Else if Z_{rj} is equal to Grj
Increase the number of relations at Z_{rij} (update its positio

3.4 Dynamic ontology generation

Integrating relevant elements to a static ontology requires little effort as it is connected to existing matching classes. But dynamic ontology has to connect relevant elements to existing classes and generate new classes and relations. Therefore, all the clustered elements should be closely analysed, leading to dynamic ontology integration. Furthermore, elements should have a close relationship when a set of elements are gathered into a cluster.

After clustering all elements, there are three types of nodes in clusters according to their behaviour, closeness and alignment. There are two types of clusters: connected graph clusters and non-connected graph clusters. The connected graphs have two main parts: fully connected-graph clusters and partially-connected clusters. This cluster is defined as elements not connected to any class branch and integrated using the matrix-based graph mapping method.

Then non-connected graph cluster consists of unrelated elements that can be considered a new concept. In such cases, many nodes cannot connect as individuals of an existing class, mainly belonging to a cluster. Therefore, generating a new class of unconnected nodes is necessary,

considering the cluster closeness and node names. The output of the spectral clustering is a vector of elements. First, a set of adjacency matrices were derived from the set of vectors for clusters. Then these matrices were used to integrate the dynamic ontology.

Algorithm2 Dynamic ontology transformation using integrated graph cluster

Let us consider the dynamic ontology matrix as D and the integrated matrix as R: Here, R is partitioned into clusters,

then R is defined as R= (R_1,R_2,……..R_k) for k number of clusters

Columns of matrix D ,D_s= (d_1,d_2,..,d_i,…,d_n),

Columns of matrix $R_{k'}$,$R_{k's}$=(r_1,r_2,…,r_i,…,r_m),

Rows of matrix D D_O= (d_1,d_2,..,d_j,…,d_p),

Rows of matrix R $R_{k'O}$= (r_1,r_2,…,r_j,…,r_q),

For all clusters $R_{K'}$

If DS=Rk'S if

DO==Rk'O NO

ACTION

else

create individuals of r_j

if d_{ij} == r_{ij}

no action

else

introduce an object property (trem in ij^{th} position)

domain= r_i

range= r_j

else

create individuals of r_i

if d_{ij}== r_{ij} (relation mapping)

no action

else

introduce an object property (trem in ij^{th} position)

domain= r_i range= r_j

3.5 Connected graph cluster mapping using matrix-based graph mapping

In dynamic ontology, classes and properties are defined that can gather individuals and their individuals. Therefore, it is necessary to connect knowledge graph elements to dynamic ontology with the changes according to the data or concept changes. This section addresses data changes and updated individuals and new relations. Here an algorithm was built to compare the adjacency matrix of dynamic ontology with the adjacency matrix of each cluster of the above-integrated graph.

3.6 Non-connected graph cluster mapping

The elements of this cluster are not directly connected to the ontology classes. Instead, they are connected or show similar characteristics within the cluster and are integrated as a new concept. As per the example in Figure 2, if a new concept is introduced in the domain. In such cases, ontology does not know about this update or change. Therefore those concepts adding to the ontology should be done carefully.

The cluster analysis with the ontology and knowledge graph does not contain a cluster name that can be transferred as a class name. Therefore these elements belong to one class in ontology, and it is needed to define a new class. For this task, class names are derived using dynamic cluster labelling. Then dynamic ontology was generated automatically by integrating a newly generated graph into an appropriate class. Next, the integrated graph matrix was mapped with a domain-graph matrix representing the existing classes and their connections. Finally, existing individuals in the domain ontology were updated with their connected relations.

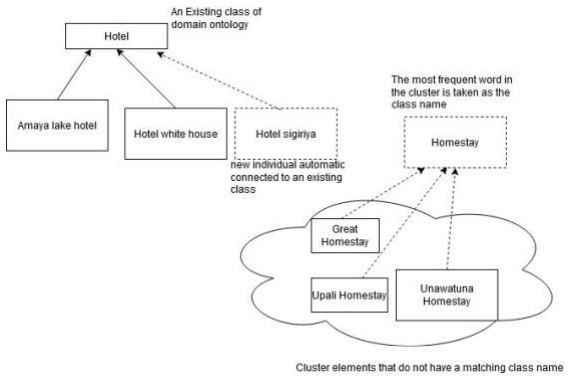


Fig. 2. Example of non-connected graph clusters and generation of new classes

3.7 New class generation using dynamic labelling

For the knowledge graph, clusters do not connect to the ontology classes, and subclasses tend to generate new classes. A Lable propagation Algorithm (LPA) is used to derive the new class labels for the unconnected graph clusters. According to the study in [32], they influence the structure and attributed dimensions to identify clusters in the attributed graph. Influencing the structure and attributed dimensions to identify clusters in the attributed graph. Here two new Concepts have been introduced and used by t the high-precision Algorithm LPA to solve the problem of attribute clustering. First, the proposed algorithm in [32], a weighted graph, is developed, every single edge of which combines the similarity of structures and attributes of two nodes with an edge. In the weighted graph, the influence of nodes will be calculated using Laplacian centrality. Afterwards, the updating stage is performed, in which the node of a two-member set will have a label and label influence. Each node that is supposed to be updated will select a label based on the more decisive influence of the label among the adjacent nodes. Also, it will cause nodes with more decisive influence in structure and attributes to update many tags. After a few steps, it is expected that the homogeneous nodes in terms of structure density will have the same tags, similar to tags of the same cluster of the graph. The algorithm is named in [32] the SAS-LP algorithm.

The SAS-LP algorithm is defined to find the most influencing node of the graph's adjacency matrix. Then the most influencing node is elected as the label of the cluster. Then this label is used to build the class in ontology, and all other elements in the cluster become the individuals of the cluster. Finally, the ontology is updated using the new class and its connected classes. Here it needs to check whether the cluster elements are mapped with ontology elements. If elements do not match, it introduces a new class in ontology with the name of cluster label. Finally, it calls algorithm 2 to establish relations among nodes and create object properties in the ontology. Algorithm 3 is called whenever a new text is integrated.

Algorithm 3:- New class introduction to the ontology using cluster labelling

---

Generate Adjacency matrix A for RDF graph ontology
Compare the clusters
For all clusters in P=(p_1,p_2……p_j…….N)
J=(1,2……j,…N)

For cluster j
Generate Adjacency matrix P_j of cluster elements
Compare Pj and A
If at least one element of P_j is mapped with A
Algorithm 2
Else
Call SAS-LP cluster labelling algorithm

New Class name = label of the cluster (output of SAS-LP algorithm)
New individuals=, all nodes carrying the class name
Update A with new nodes
Update relations using algorithm 2

4. Implementation

This research is focused on generating a dynamic ontology using a knowledge graph. First, a travel domain ontology and a knowledge graph were used to implement the dynamic ontology generation. Here the knowledge graph was built using some articles, comments and reviews. Then the graph-based dynamic ontology generator was implemented by dynamically integrating the knowledge graph into a static domain ontology.

The implementation is focused on Graph-based dynamic ontology generation. The Graph-based dynamic ontology generator consists of three main modules 1) Matrix representation and merging, 2) Graph clustering, and 3) Dynamic ontology mapping.

4.1 The Graph-based dynamic ontology generator

The domain ontology and the knowledge graph are the input of the graph-based dynamic ontology generator. It consists of three modules which fulfil three dedicated tasks for the dynamic ontology generation process. Each task and its implementation of 1) Matrix representation and merging, 2) Graph clustering and 3) Dynamic ontology mapping are summarised in Fig. 3

4.2. Matrix representation and merging

After building the knowledge graph, it was integrated into the domain ontology in the RDF format. Then, both graphs were represented in separate adjacency matrices, and a Laplacian matrix was constructed to find the eigenvector. Next, the NumPy Python package calculated the clustering and graph projection eigenvalues. Finally, an eigenvector was found using the Laplacian equation in Python.

In this module, both the RDF graph and the knowledge graphs were converted to adjacency matrices to easier to implement merging algorithms. Then each graph node and edge reference was transferred to an index for further usage. An example is to show the functioning of the technique that transfers the RDF graph to the matrix using a small RDF graph, as illustrated in Fig.4.

This small RDF Graph is stored in Adjacency Matrix in the exact figure. As there are nine nodes in the graph, hence Adjacency Matrix creates against it is a 9×9 square matrix. The name of nodes was considered as names of rows and columns. If nodes were connected, the label of the edge was added in the corresponding matrix's sell. Then the corresponding adjacency matrix rows, column labels, and matrix cell addresses were listed as an index for representing nodes and edges with their types (class names) and superclass hierarchy. Finally, the referencing index was introduced to make connecting with the original RDF graph and the ontology easier, as shown in Fig 5.

4.3 Merging graph using semantic similarity methods

After the graphs are represented in adjacency matrices and the referencing index, algorithm1 was applied for merging. Here both adjacency matrices of the RDF graph and knowledge graph are integrated. Algorithm 1 was implemented in a Python environment. First, semantic similarities were calculated for each pair of nodes and edges of comparing graphs. Next, cosine similarity, Jaccard similarity, and Euclidian distances were calculated to find the best matching techniques for semantic similarity calculation. The mean of all three similarity values was used for the similarity checking. Then, the integration algorithm was implemented, considering calculated semantic similarities. Finally, the merged matrix was transferred to module 2 as an adjacency matrix of the merged graph.

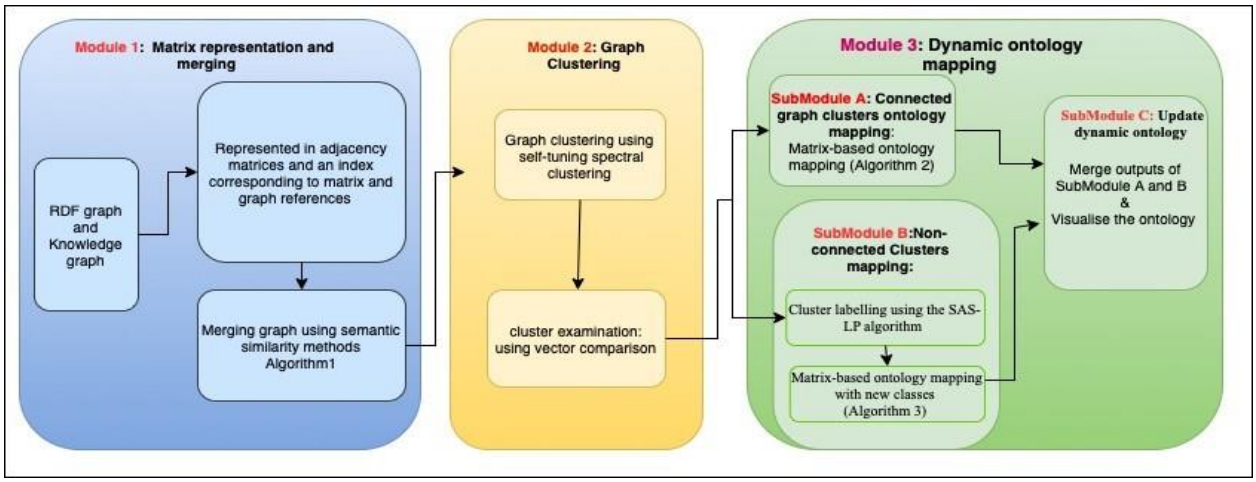


Fig. 3. Overview of Implementation. Modules

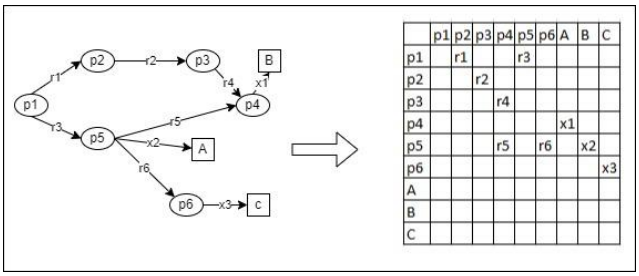


Fig. 4. RDF graph to the Adjacency matrix

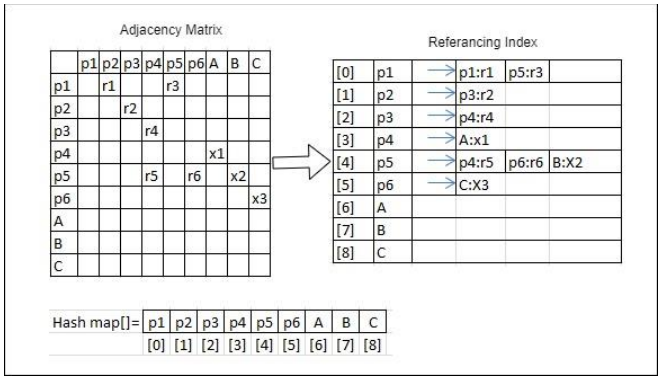


Fig.5. Referencing index storage from the adjacency matrix

4.3 Merging graph using semantic similarity methods

After the graphs are represented in adjacency matrices and the referencing index, algorithm1 was applied for merging. Here both adjacency matrices of the RDF graph and knowledge graph are integrated. Algorithm 1 was implemented in a Python environment. First, semantic similarities were calculated for each pair of nodes and edges of comparing graphs. Next, cosine similarity, Jaccard similarity, and Euclidian distances were calculated to find the best matching techniques for semantic similarity calculation. The mean of all three similarity values was used for the similarity checking. Then, the integration algorithm was implemented, considering calculated semantic similarities. Finally, the merged matrix was transferred to module 2 as an adjacency matrix of the merged graph.

4.4 Cluster the graph using self-tunning spectral clustering

The self-tuning spectral clustering algorithm was applied to the merged graph to split the graph into clusters. Here, the algorithm decides the number of clusters, and similar or nearby elements are gathered in clusters. Clustering best matching elements from the knowledge graph and the ontology are gathered. All similar elements are connected in each cluster. Those clusters should be sent to the ontology mapping for dynamic ontology generation. The behaviour of the cluster should be identified for this task, and then the corresponding mapping algorithm should be applied.

4.5 Cluster examination

The elements inside the cluster should be examined before applying an ontology mapping algorithm. For the dynamic ontology generation, It should be identified how the nodes are connected within the cluster. The implantation was extended to find connected and not connected graphs with the Python environment. Then the clusters can be categorised into two main categories, connected graph clusters and non-connected graph clusters sent to module 3 as inputs.

4.6 *Dynamic ontology mapping*

There are two types of clusters. All nodes are connected within the cluster, and clusters are not connected to ontology nodes. In module 3, suitable ontology mapping algorithms are applied to each category of clusters, as illustrated in Fig 6. Nodes of all connected clusters have connectivity to the ontology elements. Therefore, they can be mapped with existing components and relations in the cluster. But only connected nodes cluster with a connection to the ontology can be considered new concepts. Then they are introduced to the existing ontology with new classes. The following sub-sections will discuss both types of clusters treated with separate ontology mapping algorithms.

4.7 Connected graph clusters ontology mapping

The outcomes of the cluster examination in module 2, the connected graph clusters, are filtered. Then they are mapped with

the ontology using algorithm 2. Here the adjacency matrix of the current ontology is compared with the integrated matrix, and the graph is mapped with the ontology.

4.8 Not-connected graph cluster ontology mapping

This sub-module is designed to incorporate new concepts or topics introduced to the domain. For example, a not-connected graph cluster is considered a new topic with no class related to ontology. Therefore, it needs to introduce a new class and build connections.

Firstly, a label propagation algorithm was implemented for the cluster then a class in ontology was defined using the new label as the class name. Here the cluster with the most influencing node is found using the following equations discussed in (Berahmand et al., 2022). Then the most influencing node is elected as the label of the cluster. Then this label is used to build the class in ontology, and all other elements in the cluster become the individuals of the cluster. Finally, the ontology is updated using the new class and its connected classes.

4.9 Matrix-based ontology mapping

Here, the nodes were mapped with the domain ontology nodes, so the object nodes were checked where the subject nodes were mapped. When both subjects and objects were mapped, the matching relations remained or were recreated. New relations were introduced after matching existing relations in the domain ontology was verified. New nodes were added as class individuals, considering existing relations, and other connected nodes connected to a domain ontology were exited. Algorithm 2 was introduced to expand the ontology system.

The merged matrix of the RDF and knowledge graph was converted into a new ontology. Additionally, all the triple patterns and properties were transferred to the ontology. Finally, the final graph pattern was converted into an ontology with the initial class structure of the domain ontology. All the relationships and properties were converted accordingly, and all the individuals were stored in a new ontology. The new relations and nodes connected in the merged graph were converted to the ontology, and the equivalent, similarity, and subclasses were derived in the new ontology. For example, in graph merging, some relations coincided with existing relations; they were considered equivalent relations with reasoning. Finally, the output-connected graph ontology mapping and the non-connected graph ontology mapping were merged.

5. Results

This experiment focuses on dynamic ontology generation using a knowledge graph constructed using unstructured text data. Onto-Travel is a domain ontology initially defined as a travel ontology consisting of 27 classes, 19 subclasses, 81 datatype properties, 48 object properties, and 365 individuals in the first part of the research. All data and concepts related to structural data, such as destinations, hotel details, tour details, weather, and user details, are updated. This paper focuses on generating a dynamic ontology generator integrating a knowledge graph of unstructured text data into the Onto-Travel with dynamic updates.

Therefore, this experiment was designed to check whether the dynamic ontology generation and maintenance are done correctly in the presence of unstructured text such as social media comments, articles and descriptions.

In this experiment, the framework, from building the knowledge graph to integrating the ontology, was built with the domain ontology, social media comments extracted from Tripadvisor, and descriptions extracted from www.srilankatravel.com. Fifty sentences in ten social media comments about travelling and 100 sentences from ten descriptions were selected from the travel site www.srilanktravel.com. The framework comprises several components at different stages. Each key component, such as knowledge graph building and graph integration for dynamic ontology generation of the framework, was tested using real-time data; the results are recorded in the following subsections. First, the experiment was done on knowledge graph generation using real-time unstructured text related to tourism. The knowledge graph integration process is decomposed into three main modules in the implementation. The outputs of each module were reported in this section.

5.1 Knowledge graph generation

For the experiment task, a knowledge graph was built for the 50 sentences from comments, and 100 sentences from descriptions were selected. At the preprocessing stage, 13 incomplete sentences were removed, and others were transferred to the knowledge graph generation process. Then, the remaining sentences were transferred to the triplets. Finally, there were the remaining 137 sentences after preprocessing, which were decomposed into one predicate sentence. One hundred seventy-eight sentences were identified as one predicate sentence and transferred to the triplets. Here 167 sentences were transferred to the triple. A part of the knowledge graph is visualised in Fig 6.

5.2 RDF graph representation

The domain ontology is initially defined as a travel ontology consisting of 27 classes, 19 subclasses, 81 datatype properties, 48 object properties, and 365 individuals. The ontology is represented in RDF format, and the graph was drawn using resources and literals. Figure 7 shows a part of the graph of domain ontology.

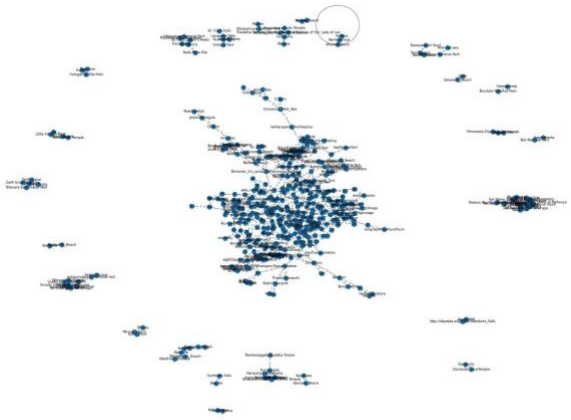


Fig 6: A part of the knowledge graph

Fig. 7 RDF graph of Onto-Travel

5.3. Calculation of semantic similarity between text components

The adjacency matrices for the knowledge and the RDF graphs are in the graph merging process. Initially, the adjacency matrix of the knowledge graph is a 321X321 matrix, and the RDF graph matrix is a 365X365 matrix. Calculating the semantic similarity between elements of each matrix was stored in a corresponding matrix. The semantic similarity matrix between two adjacency matrices generates a 321X365 matrix, and each matrix point is filed with a decimal value between -1 and 1. The semantic similarity between each component is used in the matching process. In this experiment, semantic similarities were calculated using three techniques. First, cosine similarity, Euclidean distance, and Jaccard similarities were calculated within the nodes of each graph. All three gave same-size matrices; the mean value matrix was stored as the semantic similarity matrix.

Two adjacency matrices were merged using semantic similarities and algorithm 1. The output was a 384 X384 matrix that connected all relevant nodes and expanded the matrix.

5.4Graph Clustering and ontology expansion

The merged matrix was used as the adjacency matrix for Spectral clustering in module 3, and after applying self-tuning spectral clustering generated 17 clusters. But six clusters had only one element. So these six clusters were removed, considering them outliers. Then all other 11 clusters were compared further with the dynamic ontology.

The merged matrix of the RDF and knowledge graph was converted into an ontology. Additionally, all the triple patterns and properties were transferred to the ontology. Finally, the final graph pattern was converted into an ontology with the initial class structure of the domain ontology. All the relationships and properties were converted accordingly, and the individuals were stored in a new ontology.

The conversion rate was calculated to check whether all nodes were converted to ontology. Here, the number of converted nodes over all available nodes was taken as the conversion rate. Then it showed that 95% of all the nodes and relations were converted into a new ontology. Further, the correctness of data conversion was tested, showing 89% correctness of the nodes. In the selected example, 65% of the relations were directly transferred.

5.5Overall system testing with real-time data example

In real situations, new comments and reviews are added rapidly. When a new comment or a review is added to the system, it must be dynamically embedded. The system dynamically identifies the upcoming knowledge and is automatically added to the system. For the collection of comments, the system is identified into two types. They are comments with existing contents in the ontology and the new contents to the ontology. These two types are examined by the system automatically.

Such a situation is tested with real-time data. A set of comments were added to the ontology at a given time. The first set of comments and articles about places considered in the ontology was added to the system for testing purposes. The second test data was selected from articles that are unrelated to ontology. Both datasets were added to the dynamic ontology generator and tested all components in the dynamic ontology.

6. Evaluation

The evaluation of dynamic ontology generation using knowledge for creating and updating consists of four main parts. It typically involves measuring its effectiveness in accuracy, completeness, efficiency, and scalability. The accuracy of dynamic ontology generation was evaluated by comparing the generated ontology with a manually curated ontology. The completeness of the generated ontology was evaluated by comparing it with existing ontologies in the travel domain. The number of concepts, relations, and axioms in the ontology was used to measure completeness.

The efficiency of dynamic ontology generation was evaluated by measuring the time and computational resources required to generate and update the ontology. In addition, the scalability of the approach was evaluated by measuring its performance on large-scale knowledge graphs and ontologies.

Here the system-generated dynamic travel ontology and a manually curated ontology were used to evaluate the dynamic ontology generation process. Initially, the domain ontology is developed to implement the ontology generation process. Then the dynamicity is generated and maintained using a knowledge graph based on social media comments, newspaper articles and descriptions on the web. Finally, the manually curated ontology is updated using expert knowledge by reading unstructured texts and integrating the relevant data.

6.1 Accuracy of the generated ontology

The accuracy of the generated ontology was tested with gold standards with a set of manually curated concepts, relations and axioms. First, Precision and Recall were used to evaluate the accuracy of the generated ontology. As equation (5) mentioned, Precision is the fraction of retrieved concepts, relations, and relevant axioms. At the same time, Recall is the fraction of relevant concepts, relations, and axioms retrieved, as mentioned in

equation 6. Then the F-measure was calculated as a weighted harmonic mean of Precision and Recall, as shown in equation 7.

The F-measure is a valuable measure for evaluating the overall accuracy of the generated ontology.

$$Precision = \frac{number\ of\ components\ that\ are\ relevant}{number\ of\ all\ integrated\ components} \quad (5)$$

$$Recall = \frac{number\ of\ \ components\ that\ are\ relevant}{number\ of\ all\ possible\ relevant\ components} \quad (6)$$

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (7)$$

Table 2
Accuracy evaluation measures

Ontology component	Precision	Recall	F-measure
Entities	0.87	0.83	0.85
Datatype Properties	0.82	0.78	0.80
Relationships	0.67	0.56	0.61
Axioms	0.78	0.65	0.71

The evaluation results show that Precision and Recall give more than 85% level in developed entities. Automatically, the f-measure shows an 85% accuracy level. Data type properties show 80% of the average accuracy level. But the relationship mapping shows 60% of accurate mapping levels because some relations needed to be included in the integration.

6.2 Completeness of the generated dynamic ontology

For completeness checking, a well-curated benchmark domain ontology was developed. The knowledge graph-based ontology (onto-travel) was tested with travel ontology. Therefore, the manually curated benchmark ontology(onto-travel-manual) was developed in the same domain. The system-generated dynamic ontology was compared with the benchmark ontology by comparing the number of concepts, relations, and axioms in both ontologies, as shown in TABLE 3. In addition, the coverage of the ontology was evaluated by measuring the number of instances that the ontology could classify.

The differences between the generated and benchmark ontologies were analysed to identify the missing concepts (entities), relations, and axioms. Furthermore, this analysis can identify areas where the dynamic ontology generation approach needs improvement. Here the system-generated automatic ontology is called Dynamic Onto- travel, and manually curated ontology is called onto Travel-manual. Comparing the completeness of the generated dynamic ontology with the manually curated ontology, some entities, properties and axioms were missed because of the need for proper relationship definitions. But all individuals were integrated into the dynamic ontology than the manual ontology.

Table 3
Comparison of completeness of generated dynamic ontology and manual ontology

Comparing component	Dynamic Onto-travel	Onto-Travel-manual	Percentage of completeness
Number of entities	32	35	0.91
Number of Data Properties	28	32	0.87
Number of object properties	54	58	0.93
Number of Axioms	32	33	0.97
Number of individuals	398	368	1.08

Table 4
Comparison of the efficiency for generation and updation

	Knowled ɔh-based generation	Manual logy ɾation	Knowl ɔh-based logy ɹtion	Manual tology updation
Time is generate the tology	Less than hour	More week	Less minutes	Impossible
Measure and CPU	High	Low	Low	Low

6.3 The efficiency of the ontology generation process

The efficiency of the generated ontology is measured in terms of time required to generate the ontology, memory usage, and CPU usage. First, the efficiency was checked for ontology generation and updation, as shown in TABLE 4. Then each test was done on three types of ontologies manual ontology and knowledge graph-based ontology. *Scalability of the dynamic ontology*

Dynamic ontologies are designed to be flexible and adaptable, allowing them to evolve and expand over time to meet changing requirements. However, as the size and complexity of the ontology increase, issues related to scalability can arise. Scalability is the ability of a system to handle increasing amounts of data or users without experiencing a significant decrease in performance.

Table 5
scalability test for three knowledge graphs with different sizes

Knowledge graph number	Number of nodes	Number of relations	Time is taken to	Time is taken to update

			generate ontology (seconds)	the ontology
1 (for newspaper articles)	132	38	125	48
2 (for description)	97	43	68	46
3 (social media comments)	243	74	180	58

In the case of dynamic ontologies, scalability is an essential consideration because as the ontology grows, the computational resources required to manage the ontology query can increase significantly. Here the scalability of the dynamic ontology generation process was tested for three knowledge graphs with different sizes. The first knowledge graph was made of newspaper articles. The second one is from descriptions of tourist websites. Finally, the third knowledge graph was created using social media comments. The number of nodes and the number of relations were used to measure the size of the knowledge graph. First, the time taken to generate the dynamic ontology and the time taken to update the ontology was checked for the scalability of the ontology. Then the size of the ontologies was tested for each knowledge graph by considering entities, datatype properties, object properties and the number of axioms. Considering the time taken to generate the ontology and the update, the ontology is shown in seconds. The ontology generation process is scalable and develops all components in dynamic ontology correctly and in a shorter period.

6.5 The evaluation of the approach

The fundamental techniques used in developing the dynamic ontology using knowledge were evaluated to measure whether the selected technique suits the situation. In addition, the graph merging and ontology mapping techniques were evaluated.

6.6 Ontology mapping evaluation

The final ontology was evaluated using Precision, Recall, and the F-measure [33] for relevancy checking for ontology generation. All the steps of the integration process were evaluated for relevance and accuracy. The final resultant ontology was also verified based on the Precision, Recall, and F-measure values. Precision is defined as the relevant number of integrated components, as shown in Eq. (5). Recall is the number of integrated relevant components of all the possible relevant components, as shown in Eq. (6). Finally, the F-measure is derived from the Recall and Precision, as shown in Eq. (7). The Precision of the developed final ontology was 0.787, meaning it had a 79% relevancy level for all the integrated components. The Recall measured the number of relevant integrated components over the possible relevant components, which was 0.92. During the evaluation, the F-measure of the nodes and related integration and conversion to the final ontology was 0.84. The evaluation of the final ontology revealed that after connecting the knowledge graph, the final ontology was 84% relevant to the expected results.

Furthermore, the novel method was compared with the keyword (string)-based matching based on their performance with ontology instances or the same set of comments with the same ontology. Finally, the relevance of the results was measured using Precision, Recall, and F-measure. Then, both methods were compared, as presented in Table 7.

Table 7
. Comparison of the proposed graph-based method with the keyword-based matching method.

	Introduced Graph-based method	Keyword-based matching method
Rate of integration of components	75%	60%
Precision	0.787	0.659
Recall	0.902	0.731
F-measure	0.84	0.693

Table 6
Scalability test for three knowledge graphs to ontology components

	Knowledge graph		Dynamic ontology			
	Number of nodes	Number of relations	Number of entities	Number of datatype properties	Number of object properties	Number of axioms
1 (for newspaper articles)	132	38	22	69	18	21
2 (for descriptions)	97	43	13	34	12	17
3(for social media comments)	243	74	34	56	24	28

The rate of integration using the keyword-based matching method was lower than that of the novel integration method. In addition, no other relations were found since keyword-based matching could not address unmatched elements matching methods based on relevancy. However, the novel approach addressed graph and keyword matching and graph clustering, leading to a better alignment of unmatched elements.

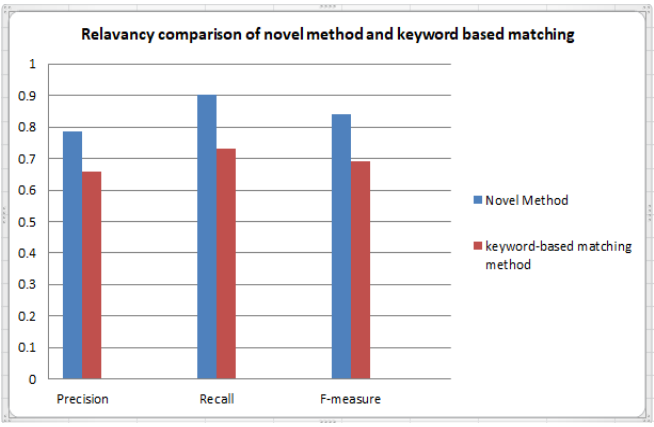


Fig 8: Comparison of the novel and keyword-based

According to the accuracy results, the Precision, recall, and F-measure values were 78, 90, and 84%, respectively. Furthermore, Fig. 8 illustrates that the novel method provides higher relevancy than the keyword-based matching method. In the case of dynamic ontologies, scalability is an essential consideration because as the ontology grows, the computational resources required to manage the ontology query can increase significantly.

7. Conclusions

This study is focused on dynamic ontology generation using a knowledge graph in the presence of continuously updating new data in different connected sources. This task was achieved using knowledge graph-based semantic integration for updating the unstructured text data for dynamic ontology. First, the novel graph-based integration and ontology transformation method was introduced and evaluated for real-time data. In addition, graph-based clustering was enriched with self-tuning spectral clustering, reducing the computational cost of semantic mapping using graph clustering. Finally, in the implementation, an index was maintained for the easy reference of the graph after merging the matrices. The continuous updating of the ontology in the presence of new text data self-tuning spectral clustering and adjacency matrix mapping enhanced the performances of the integration process.

The proposed approach was applied to the travel domain to demonstrate its applicability. Dynamically updated data extracted from sources in the travel domain were converted into knowledge graphs after resolving semantic heterogeneity and were integrated into the ontology later. The performance of the proposed method was compared to that of the keyword-based approach. According to the results, the proposed method outperformed the keyword-based approach, with an accuracy level of 84%.

References

[1] K. I. Kotis, G. A. Vouros, and D. Spiliotopoulos, “Ontology engineering methodologies for the evolution of living and reused ontologies: Status, trends, findings and recommendations,” *Knowledge Engineering Review*, vol. 35, 2020, doi: 10.1017/S0269888920000065.

[2] Z. Ma, H. Cheng, and L. Yan, “Automatic construction of OWL ontologies from petri nets,” *Int J Semant Web Inf Syst*, vol. 15, no. 1, 2019, doi: 10.4018/IJSWIS.2019010102.

[3] J. An and Y. B. Park, “Methodology for Automatic Ontology Generation Using Database Schema Information,” *Mobile Information Systems*, vol. 2018, 2018, doi: 10.1155/2018/1359174.

[4] L. Khan and F. Luo, “Ontology construction for information selection,” in *Proceedings of the International Conference on Tools with Artificial Intelligence*, 2002. doi: 10.1109/tai.2002.1180796.

[5] A. Faatz and R. Steinmetz, “Ontology Enrichment with Texts from the WWW,” Aug. 2002.

[6] S. Elnagar, V. Yoon, and M. A. Thomas, “An automatic ontology generation framework with an organisational perspective,” in *Proceedings of the Annual Hawaii International Conference on System Sciences*, 2020. doi: 10.24251/hicss.2020.597.

[7] K. Kumar and A. Mukherjee, “Constructing knowledge graph from unstructured text,” *Siddhant Manocha*, no. 12375, 1237, [Online]. Available: http://home.iitk.ac.in/~kundan/report_365.pdf

[8] X. Chen, S. Jia, and Y. Xiang, “A review: Knowledge reasoning over knowledge graph,” *Expert Systems with Applications*, vol. 141. 2020. doi: 10.1016/j.eswa.2019.112948.

[9] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, “Freebase: A collaboratively created graph database for structuring human knowledge,” in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008. doi: 10.1145/1376616.1376746.

[10] M. Fabian, K. Gjergji, and W. Gerhard, “YAGO: A core of semantic knowledge unifying wordnet and wikipedia,” *16th International World Wide Web Conference, …*, 2007, doi: 10.1145/1242572.1242667.

[11] H. Rahman and M. I. Hussain, “A lightweight dynamic ontology for Internet of Things using machine learning technique,” *ICT Express*, vol. 7, no. 3, 2021, doi: 10.1016/j.icte.2020.12.002.

[12] B. Bachimont, A. Isaac, and R. Troncy, “Semantic commitment for designing ontologies: A proposal,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Bioinformatics)*, 2002. doi: 10.1007/3-540-45810-7_14.

[13] N. F. Noy and M. Klein, “Ontology Evolution: Not the Same as Schema Evolution,” *Knowl Inf Syst*, vol. 6, no. 4, 2004, doi: 10.1007/s10115-003-0137-2.

[14] E. Tovar and M.-E. Vidal, “REACTIVE: A rule-based framework to process reactivity,” *CEUR Workshop Proc*, vol. 519, Jan. 2009.

[15] R. Dividino, M. Romanelli, and D. Sonntag, “Semiotic-based ontology evaluation tool S-OntoEval,” in *Proceedings of the 6th International Conference on Language Resources and Evaluation, LREC 2008*, 2008.

[16] P. Pittet, C. Cruz, and C. Nicolle, “Guidelines for a dynamic ontology: Integrating tools of evolution and versioning in ontology,” in *KMIS 2011 - Proceedings of the International Conference on Knowledge Management and Information Sharing*, 2011. doi: 10.5220/0003653201730179.

[17] J. Watróbski, “Ontology learning methods from the text -an extensive knowledge-based approach,” in *Procedia Computer Science*, 2020. doi: 10.1016/j.procs.2020.09.061.

[18] S. R. Guruvayur and R. Suchithra, “Automatic relationship construction in domain ontology engineering using semantic and thematic graph generation process and convolution neural network,” *International Journal of Recent Technology and Engineering*, vol. 8, no. 3, 2019, doi: 10.35940/ijrte.C6832.098319.

[19] N. Aussenac-Gilles, B. Biebow, and S. Szulman, “Corpus analysis for conceptual modelling,” *Citeseer*, 2000.

[20] A. Turel and F. Can, “A new approach to search result clustering and labelling,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Bioinformatics)*, 2011. doi: 10.1007/978-3-642-25631-8_26.

[21] M. Alam and K. Sadaf, “Labeling of web search result clusters using heuristic search and frequent itemset,” in *Procedia Computer Science*, 2015. doi: 10.1016/j.procs.2015.02.014.

[22] H. Poostchi and M. Piccardi, “Cluster Labeling by Word Embeddings and WordNet’s Hypernymy,” *Proceedings of the Australasian Language Technology Association Workshop 2018*, 2018.

[23] B. Laassem, A. Idarrou, L. Boujlaleb, and M. Iggane, “Label propagation algorithm for community detection based on Coulomb’s law,” *Physica A: Statistical Mechanics and its Applications*, vol. 593, 2022, doi: 10.1016/j.physa.2022.126881.

[24] C. Jebari, “An improved centroid-based approach for multi-label classification of web pages by genre,” in *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, 2011. doi: 10.1109/ICTAI.2011.142.

[25] H. Sun *et al.*, “CenLP: A centrality-based label propagation algorithm for community detection in networks,” *Physica A: Statistical Mechanics and its Applications*, vol. 436, 2015, doi: 10.1016/j.physa.2015.05.080.

[26] Pat Morin, *Open Data Structures: An Introduction*. 2013.

[27] Atta-Ur-Rahman and F. A. Alhaidari, “Querying RDF data,” *J Theor Appl Inf Technol*, vol. 96, no. 22, 2018, doi: 10.4018/978-1-61350-053-8.ch015.

[28] R. Tous and J. Delgado, “A vector space model for semantic similarity calculation and OWL Ontology Alignment,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006. doi: 10.1007/11827405_30.

[29] S. Kosinov and T. Caelli, “Inexact multi subgraph matching using graph eigenspace and clustering models,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2002. doi: 10.1007/3-540-70659-3_13.

[30] L. Zelnik-Manor and P. Perona, “Self-tuning spectral clustering,” in *Advances in Neural Information Processing Systems*, 2005.

[31] G. Wen, “Robust self-tuning spectral clustering,” *Neurocomputing*, vol. 391, 2020, doi: 10.1016/j.neucom.2018.11.105.

[32] K. Berahmand, S. Haghani, M. Rostami, and Y. Li, “A new attributed graph clustering by using label propagation in complex networks,” *Journal of King Saud University - Computer and Information Sciences*, vol. 34, no. 5, 2022, doi: 10.1016/j.jksuci.2020.08.013.