Journal of Computer Science

Original Research Paper

OntoJob Query Processor: An Ontology Driven Query Processing Method in Jobology Information System

Ranjna Jain and Neelam Duhan

Department of Computer Engineering, J.C. Bose University of Science and Technology, YMCA, India

Article history
Received: 10-05-2019
Revised: 12-12-2019
Accepted: 03-06-2020

Corresponding Author:
Ranjna Jain
Department of Computer Engineering, J.C. Bose University of Science and Technology, YMCA, India
Email: ranjna.gupta@gmail.com

Abstract: There are a large numbers of jobsites/job portals that provide information about employment on the internet. These websites facilitate employers to post job lists. Job seekers go through those job posts and apply for the same. But, due to the availability of dozens of job portals, job seekers are unable to concentrate on the efforts to see the best outcomes. The overall objective of the paper is to develop a prototype system that provides a platform for the job seeker to access all the job lists from various job sites on a single click, at the same place with respect to the fired query. For this purpose, an ontology driven information system named as Jobology is discussed that integrates various Jobboards using the approach of ontology alignment. The system takes user's query in keyword format and in response generates a set of SPARQL queries. These SPARQL queries are then fired on their respective ontologies and in turn they yield the results. These results are merged and finally presented to user. As a contribution of this paper, we have proposed an "OntoJob" Query processor that takes job seeker query in keyword format and in turn generates a set of SPARQL queries with respect to every jobboard. The proposed approach is implemented in JAVA using OWLAPI on window platform. To evaluate the proposed work, comparison analysis between Jobboards, proposed ontologies and integrated system was performed. The results came out to be very satisfactory.

Keywords: Ontology, Semantic Web, Data Heterogeneity, Query Processing, SPARQL

Introduction

In today's time, internet (Brin and Page, 1998) is the biggest data source used by job seekers for job search. It provides a number of jobboard sites where job-seeker enrolls himself with his expertise and qualification and gets job related information according to his requirements. The internet is considered as an excellent tool for locating the job but job-seeker faces a number of challenges while handling various accounts on the respective jobboard sites such as visiting each site individually to look for opportunity and updating same data on each registered site. To deal with these issues, data integration seems to be a very convenient solution. Data integration provides the ability to manipulate data transparently across multiple data sources. But, the biggest challenge that comes across data integration is data heterogeneity. As data sources can be heterogeneous in syntax, schema, or semantics, thus making data interoperation a difficult task (Jain *et al*., 2018a). Syntactic heterogeneity is caused by the use of different models or languages. Schematic heterogeneity results from structural differences and Semantic heterogeneity is caused by different meanings or interpretations of data in various contexts. To achieve data integration, the issues posed by data heterogeneity need to be eliminated. To handle semantic heterogeneity, ontology plays a very important role in semantic web. Ontology (Jain *et al*., 2018b) is considered as a backbone of the semantic web (Berners-Lee *et al*., 2006). It is an explicit specification of a domain which gives a formal defined meaning to the terms used in annotation associated with data. Ontologies have been realized as the key knowledge representation technology for shaping and exploiting information for the effective management and evolution of semantic web.

In semantic web, it is the general perception that in order to define a domain, there exists a single ontology

which is used by different data sources. But, in reality, it is just opposite of what is assumed. For instance, today there are already more than hundred ontologies available for public access in only DAML (Mcguinness *et al*., 2002) ontology library and with the increasing number of user participation in semantic web, this number will increase significantly in such kinds of ontology libraries. This will ultimately create issue during query processing. The problems which will occur in this situation are discussed as below:

1. Since the developer is using its own ontology to represent its data, establishing link between two data sources to handle complex queries will become a challenge because the search system would not be able to communicate with each other due to different ontology as vocabulary they are using
2. Conventional query processing techniques cannot be applied on these structured data as they are unaware of the annotation used by the data
3. Users generally fire query in a Natural Language, handling of which on a structured data is a major challenge
4. Ontologies are defined at conceptual level and normally, they do not contain instances. Rather they are used as annotation. User gives a query at an instance level and not at conceptual level. Mapping instances with its respective concepts and then further plan a query in SPARQL (O'Connor and Das, 2007) format is a big challenge

Therefore, looking at the rapidly evolving technological environment, "OntoJob" a query processing method, is being proposed in this paper that describes the working of retrieving relevant information corresponding to the query from the N number of data sources at one place. The proposed method is divided into two phases: First phase covers the identifying concepts of the instances and the Second phase covers generating SPARQL query to be fired on ontologies to retrieve relevant results. The paper has been organized in six sections afterwards, wherein section 2 describes the preliminaries required before starting the work. Section 3 describes the related work that has been done in this domain. Section 4 describes the proposed work "ONTOJOB" query processor which explains the architecture and formation of SPARQL queries. Section 5 shows the implementation of the proposed work followed by performance evaluation and at last; section 6 concludes the paper with some light on future work.

Preliminaries

This section describes the indexes and various datasets maintained by the proposed system which are used during query processing.

(a) Indexes

The proposed system aligns selected ontologies and with respect to those maintains three global indexes named as Global Concept Index (GCI), Global Object Property Index (GOPI) and Global Data Property Index (GDPI) using following steps:

i. First it takes n number of ontologies as an input which are to be aligned concurrently
ii. then, it performs semantic matching on concepts, data properties and object properties
iii. And develops knowledge base of synonym of concepts and properties to fasten the matching process
iv. And at last it finally builds Global Concept Index, Global Data Property Index, Global Object Property Index which stores all information of the aligned ontologies and also maps them with their local ontologies to which they actually belong, thereby supporting backward engineering

(b) Datasets

A repository of different datasets is maintained which plays a very important role during query processing because ontologies are normally defined at conceptual level. User gives query in keyword or in natural language form. These dataset helps in finding to which concept a keyword may belong; which ultimately helps in planning query. For example; if a user is finding 'java' related jobs and he enters 'java jobs' then looking at the skill dataset, it can be identified that java is a skill and therefore this would help in understanding that user is looking for job on the basis of skill named as 'Java'. In the same way, if the user is looking for "Java, Delhi", then location dataset would identify that Delhi is a location and thus makes a high probability of belongingness to Indlocation concept as defined in Table 1, which would ultimately help in planning a query in SPARQL format. In this work, following datasets are maintained for finding the context of the keywords of the query given by user.

These datasets maintain a list of their respective data. They have built by extracting relevant data from various jobboard sites. With the recognition of new data, it gets updated in its corresponding dataset.

Table 1: List of dataset

S. No.	Dataset	Description
1.	Skill dataset	List of skillsets.
2.	Indlocation dataset	List of locations.
3.	Salary dataset	List of salary packages from minimum to maximum range.
4.	Experience dataset	List of experiences in terms of years a job can ask for.
5.	Designation dataset	List of job titles.

Literature Background

In the recent past, several ontology-based approaches have been proposed. This literature work illustrates some of the query processing methods proposed by researchers by using ontologies.

Sander *et al*. (2014) proposed Ontology based Translation of natural Queries to SPARQL where it interprets natural language input under consideration of domain specific concepts, individuals, relations and restrictions in the ontology, as well as additional knowledge e.g., keywords, synonyms etc. which are stored in a Simple Knowledge Organization System (SKOS) lexicon. After the interpretation or extraction of required information to parse a Natural Language query successfully, the input is mapped to the predefined SPARQL Inferencing Notation (SPIN) rules which provide the skeleton for the required SPARQL query. Delbru *et al*. (2011) has examines the shift from the traditional web document model to a web data object (entity) model and has studies the challenges faced in implementing a scalable and high performance system for searching semi-structured data objects over a large heterogeneous and decentralized infrastructure. Towards this goal, they have defined an entity retrieval model and developed methodologies for supporting this model and show how to achieve a high performance entity retrieval system introducing an indexing methodology for semi--structured data which offered a good compromise between query expressiveness, query processing and index maintenance compared to other approaches. They have address high performance by optimization of the index data structure using appropriate compression techniques. Finally, they have demonstrate that the resulting system could index billions of data objects and provide keyword based as well as more advanced search interfaces for retrieving relevant data objects in subsecond time. Weiand (2011) has developed KWQL; a keyword based querying for the social semantic search. Zhou *et al*. (2007) explored a novel approach of adapting keywords to querying the semantic web. This approach automatically translates keyword queries into formal logic queries so that end users can use familiar keywords to perform semantic search. A prototype system named 'SPARK' has been implemented in light of this approach. Given a keyword query, SPARK outputs a ranked list of SPARQL queries as the translation result. In the methods for (Shekarpour, 2011) automatically transforming keyword based queries into SPARQL has been suggested. Also work has been done in improving those methods in order to apply them on (a large subset of) the Linked Data Web. A heuristic method has been proposed for generating SPARQL queries out of arbitrary number of keywords. Yahya *et al*. (2012) has proposed Natural Language questions for the web of data. They identified that structured query language is a difficult task even for skill person and thus presented a methodology for translating natural language questions into structured SPARQL queries over linked data sources. They proposed a framework DEep Answers for maNy Naturally Asked question (DEANNA), that composes a full suite of components for question decomposition, mapping constituents into the semantic concept space, generating alternative candidate mappings and computing a coherent mapping of all constituents into a set of SPARQL triple patterns that can be directly executed on one or more linked data sources.

"ONTOJOB" Query Processor: Query Processing in Aligned Ontologies

This section describes ONTOJOB, architecture for query processing in global information systems as shown in Fig. 1 motivated by the problems discussed in the introduction section.

At an abstract level, the process starts with tokenizing the query given by user which resides at the token buffer. Query is entered by the user in keyword form and keywords are separated by the delimiter. Once the tokenization is done, tokenizer sends signal to token mapper to find if token belongs to any dataset and respective information gets stored in Token_Dataset Table.

At the back end, dataset_concept mapping table is maintained which contains a list designating which concept is mapped with which dataset. For instance, skill dataset contains a list of keyskills which can be the instance of skill concept. Once this is done, token mapper sends the signal to token_concept mapper to map the tokens with their respective concepts. For this, token_concept mapper refers token_dataset table and concept_dataset table and generates instance_concept table. Once, it is found to which concept a token belongs; next task is to find the relation between the classes and for this, Token_Concept mapper sends the signal to property finder to find the relation and the ontologies in which those concepts and property exist. The generated information gets stored in the property table. Once this is done, property finder sends signal to property transformer which creates an inverse property table by placing all the properties at one place corresponding to the each ontology which would be helpful in planning a query with respect to selected ontologies. After this, inverse query transformer sends signal to query generator process to take input from inverse property table and generate individual SPARQL queries for selected ontologies and then fires them to the respective ontologies which in turn generate results. At last, Result merger merges all the results and display it to the user at one place.

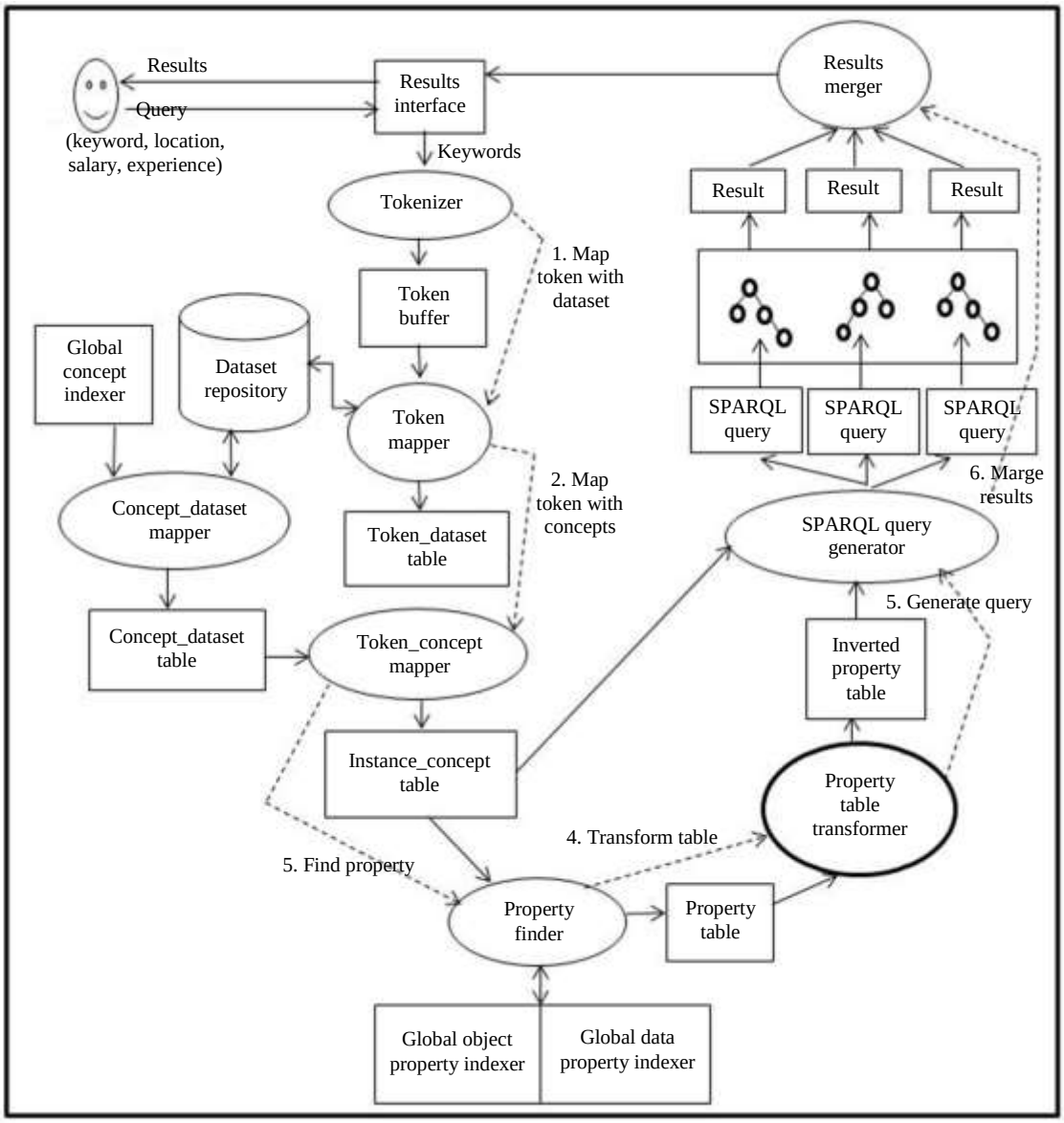


Fig. 1: Architecture of ONTOJOB query processor

The schema of various data structures used during query processing is shown in Table 2.

The schema of data structures used during query processing is defined as follows.

Token Buffer

This buffer contains the tokens generated by tokenizer. The descriptions of various fields maintained in this table are described in Table 3.

Token_Dataset Table

Token_dataset process finds whether a token belongs to any dataset maintained in dataset repository and its corresponding information gets stored in Token_Dataset table. The descriptions of various fields maintained in this table are described in Table 4.

Concept_Dataset Table

This table stores the mapping information between concept and dataset. For instance, skill dataset is mapped with skill concept. The descriptions of various fields maintained in this table are described in Table 5.

Instance_Concept Table

This table stores the mapping information between token and concept. For instance, if a token ‘PHP’ belongs to skill dataset and skill dataset is mapped with skill concept then PHP is considered as instance of skill

concept. The descriptions of various fields maintained in this table are described in Table 6.

Table 2: Schema of various data structures used in query processor

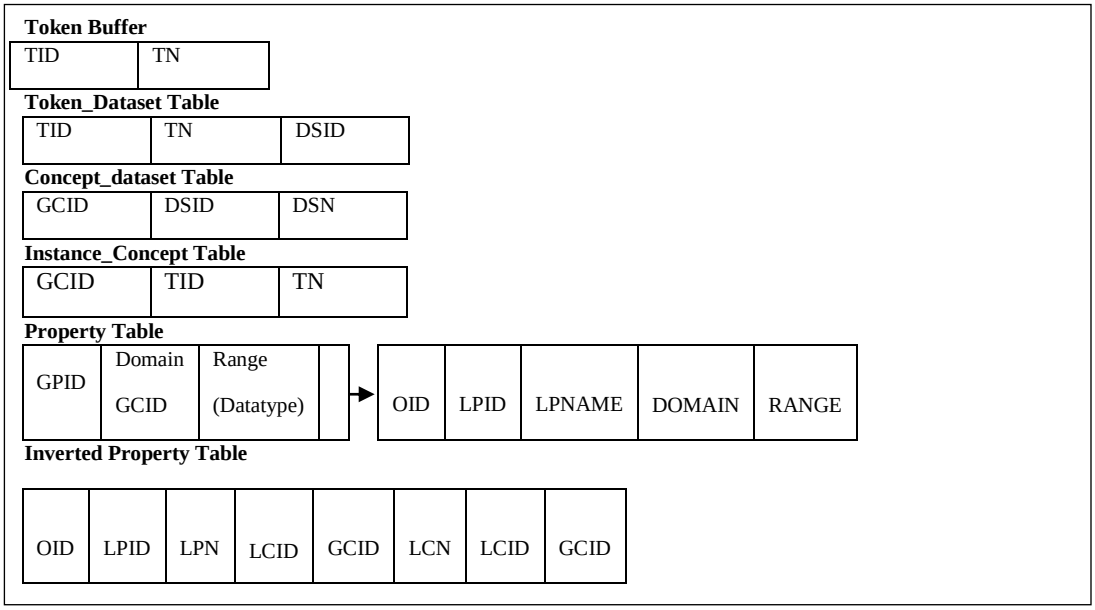


Table 3: Description of token buffer

Field	Description
TID	Unique id of token t in the query q.
TN	token name in the query q

Table 4: Description of Token_Dataset table

Field	Description
TID	Unique id of token t in the query q.
TN	Token name in the query q.
DSID	Unique dataset id collected in dataset repository.

Table 5: Description of Concept_Dataset Table

Field	Description
GCID	Global concept id of the concept maintained in global concept indexer (Jain *et al*., 2017).
GCN	Global concept name of the concept maintained in global concept indexer.
DSID	Unique dataset id collected in dataset repository.
DSN	Name of the dataset stored in dataset repository.

Table 6: Description of Instance_Concept Table

Field	Description
GCID	Global concept id of the concept maintained in global concept indexer.
GCN	Global concept name of the concept maintained in global concept indexer.
TID	Unique id of token t in the query q.
TN	Token name in the query q.

Table 7: Description of Data Property Table

Field	Description
GPID	Global concept id of the concept maintained in global concept indexer.
Domain	Local concept name of the concept maintained in global data property table.
Range	Local concept name of the concept (if object property) or data type (if data property) maintained in global object/data property table.
OID	Ontology ID.
LPID	Local property id of the property.
Domain	Local concept name of the concept maintained in its respective local property table.
Range	Local concept name (if object property) or data type (if data property) maintained in its respective local property table.

Table 8: Description of property table

Field	Description
OID	ontology ID.
LPID	local property id of the property in its respective ontology.
LPN	local property name of the property in its respective ontology.
Domain	Local concept id, global concept id and local concept name of domain concept.
Range	Local concept id, global concept id and local concept name of range concept (if object property); data type (if data property)

Property Table

Once the tokens get mapped with their respective concepts, next step is to find the relationship that exists between the concepts. For this, all the relationships which are maintained between job concept and identified concept c are collected from Global Object Property Indexer and Global Data Property Indexer which in turn are maintained in Data Property table and Object Property Table. The descriptions of various fields maintained in this table are described in Table 7.

Inverted Property Table

This table is an inverted version of Property Table. It contains the list of properties with respect to individual ontology which will help in constructing SPARQL query. The descriptions of various fields maintained in this table are described in Table 8.

The details of the various modules along with their working are outlined as below:

Tokenizer

It takes user query keywords as an input and split it into tokens. The generated tokens are stored in token buffer. Once the tokens have been generated, it then sends signal to token mapper for further process.

Token Mapper

Upon getting the signals from tokenizer, it finds to which dataset the token may belong to. For instance, if a java as token is received from token buffer and if that is found in the skill dataset, then at the generalized level it would be considered as skill.

Dataset_Concept Mapper

This process maps datasets present in dataset repository with the concept indexed in GCI. This is a one- time process in which concepts are already mapped with the datasets. For instance, skill concept is mapped with skill dataset; location concept is mapped with location dataset and so on.

Token_Concept Mapper

This process maps tokens with their respective concepts by referring token_dataset table which contains a list of tokens along with the dataset (to which they belong) and concept_dataset table which holds a list of concepts mapped with datasets. By joining these two tables, the resultant table instance_concept table is generated which contains a list of instances with their respective concepts.

Property Finder

Once the concept has been identified with respect to query keywords, token_concept mapper sends a signal to property finder to find the relation that exists between job and the concept. Property finder refers to GOBJPI and GDPI and retrieves the property that exists between the two concepts (Job and other concept identified from token_concept mapper) and stores it into the Dataproperty table (if retrieved from GDPI) and Objectproperty table (if retrieved from GOBJPI). Along with this, it retrieves other information such as ontologies in which this property exists; and domain and range concepts which would be required during query planning. Once it is done, it sends transform property signal to property transformer process.

Property Table Transformer

The data generated from Property Finder process gets collected in Property Table. In this table, the head of every row is the property followed by the nodes containing information about the ontology and local property. This defines the ontologies in which the property exists. Property table transformer represents the same information but in inverted form. This process upon getting the signals from property finder process creates an inverted property table in which each row is headed with ontology name followed by the nodes containing the properties that are identified from GOBJPI and GDPI with respect to the query. After this, a signal is sent to a SPARQL_query_generator process. With this step, writing SPARQL query becomes a simple process. Once the instances get mapped with their respective concepts and properties have been identified, next step is to generate SPARQL query.

In the next section, the process of generation of SPARQL queries with respect to ontologies is presented.

Generation of SPARQL Queries

This phase generates SPARQL queries with respect to selected ontologies using Instance_Concept Table. While generating the SPARQL queries; a number of things have been taken into consideration for retrieving better and relevant results for user:

1. Since all Job board results have to be displayed at the same place to its intended user, the topmost results should contain all the options given by user at the user interface. For instance, if a user has given java, nodeJS, AngularJS as keyword at the user interface, then system should display all the job posts from different ontologies which contain all these skills
2. It should also retrieve those job posts which contains 2 keywords followed by job posts with 1 keyword to provide maximum opportunities

The process of SPARQL Query Generator is shown in Fig. 2.

It starts working once it gets signal from the inverse property transformer. It plans to build separate queries corresponding to the each ontology listed in Inverse

Property Table. It collects ontology name, property name, domain and range and constructs separate SPARQL query with respect to the ontologies. It performs two tasks: First, builds SPARQL Query using SPARQL Querybuilder process and second, generates filters that will be appended to SPARQL query using Filter combination generator. The process of generating filters using Filter Combination generator is shown in Fig. 3.

At one side filter combination generator generates all the combinations of the instances and store them in filter combination table and on the other side SPARQL querybuilder builds SPARQL query using inverse property table.

Once the SPARQL queries have been generated, these queries are applied on their respective ontologies to retrieve results from ontologies one by one and stores the results in their respective result tables. Along with this, each row of the result table contains the count of filters taken from filter combination table. For instance, if the filter_count is of the retrieved jobpost is 4 then this designates that this job post contains 4 keywords given by user. This will be required during merging operation. Once this is done, it sends signal to result merger to merge the results and present it to the user.

Result Merger

It merges the results upon getting the signal from query generator. Now the motive is to provide those jobpost on the top from various job boards which contains maximum keywords given by user at user interface. Therefore, looking at the filter_count in each result table, merging operation is performed. For instance jobpost containing all the keywords will be displayed on the top. Now, on the basis of date and time of job post uploaded, these posts will be displayed at the interface and presented to user.

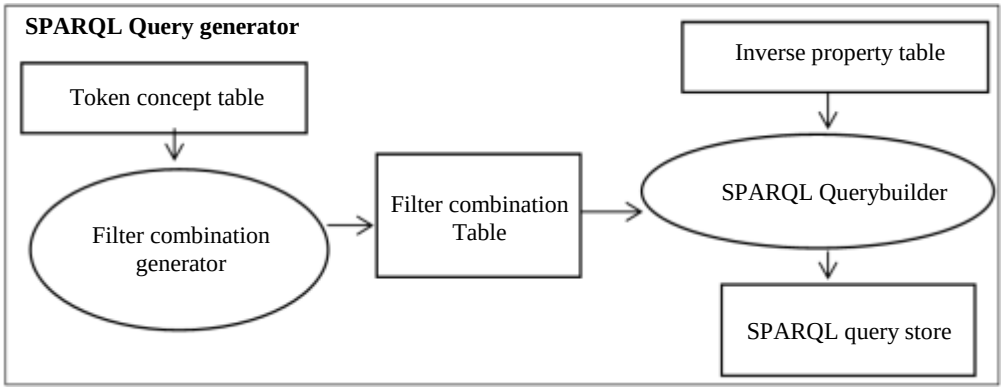


Fig. 2: SPARQL query generator

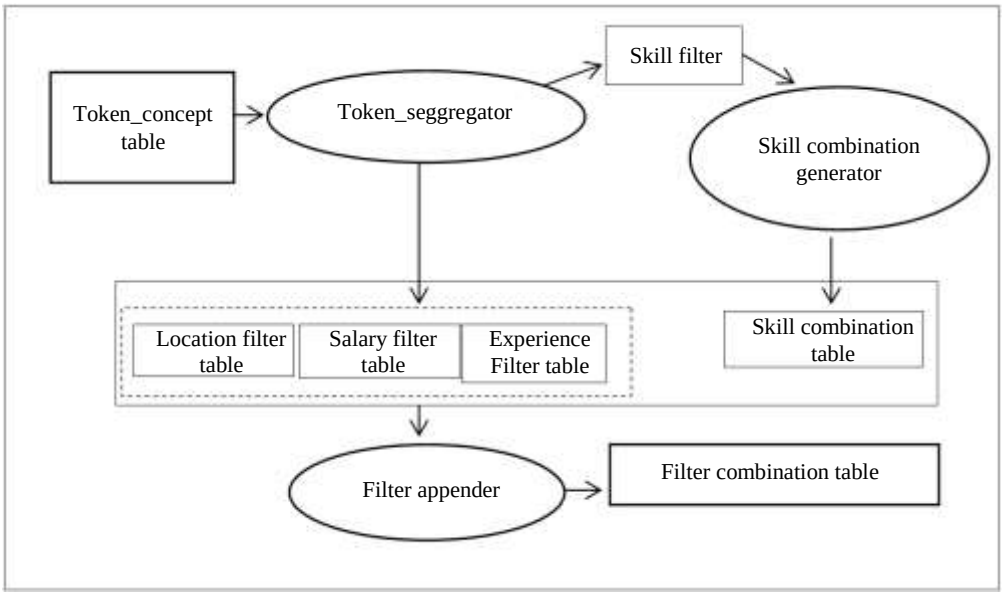
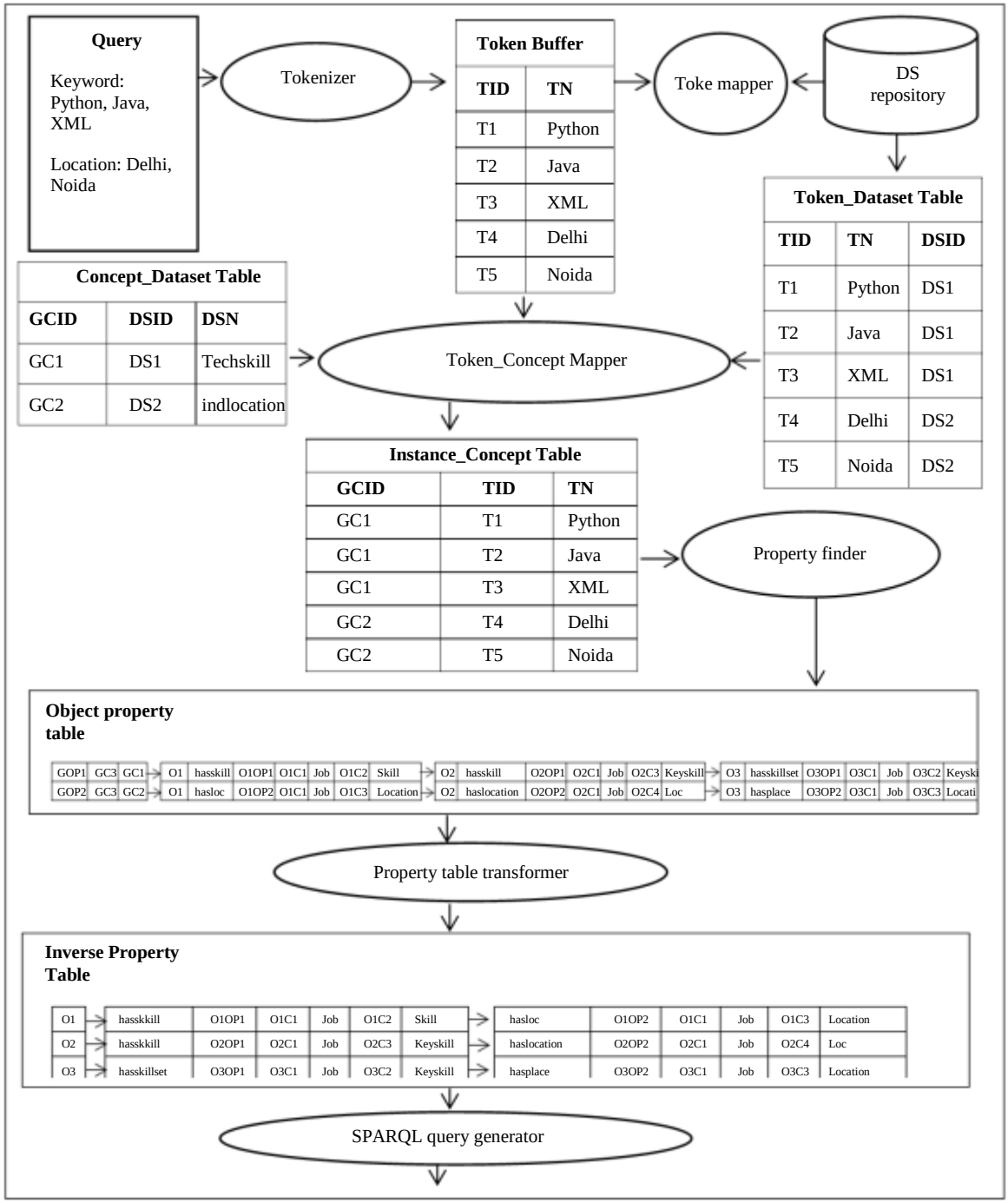


Fig. 3: Filter combination generator

Illustration

Let's consider an example whose illustration is shown in Fig. 4 which will help in understanding the construction of SPARQL query from keyword based query corresponding to ontologies. Consider a query: Python, Java, XML and Location: Delhi, Noida

The output of Filter Combination Generator i.e.; Filter Combination Table and Inverse property table is given to SPARQL Query builder which yields various SPARQL queries. The sample of newly generated SPARQL queries from the above process is shown in Appendix I. The illustration as explained in Fig. 4 shows the step by step process of converting keyword based queries into SPARQL queries. These queries are then fired over respective ontology independently. The generated results are then merged by result merger and finally get available to the user.



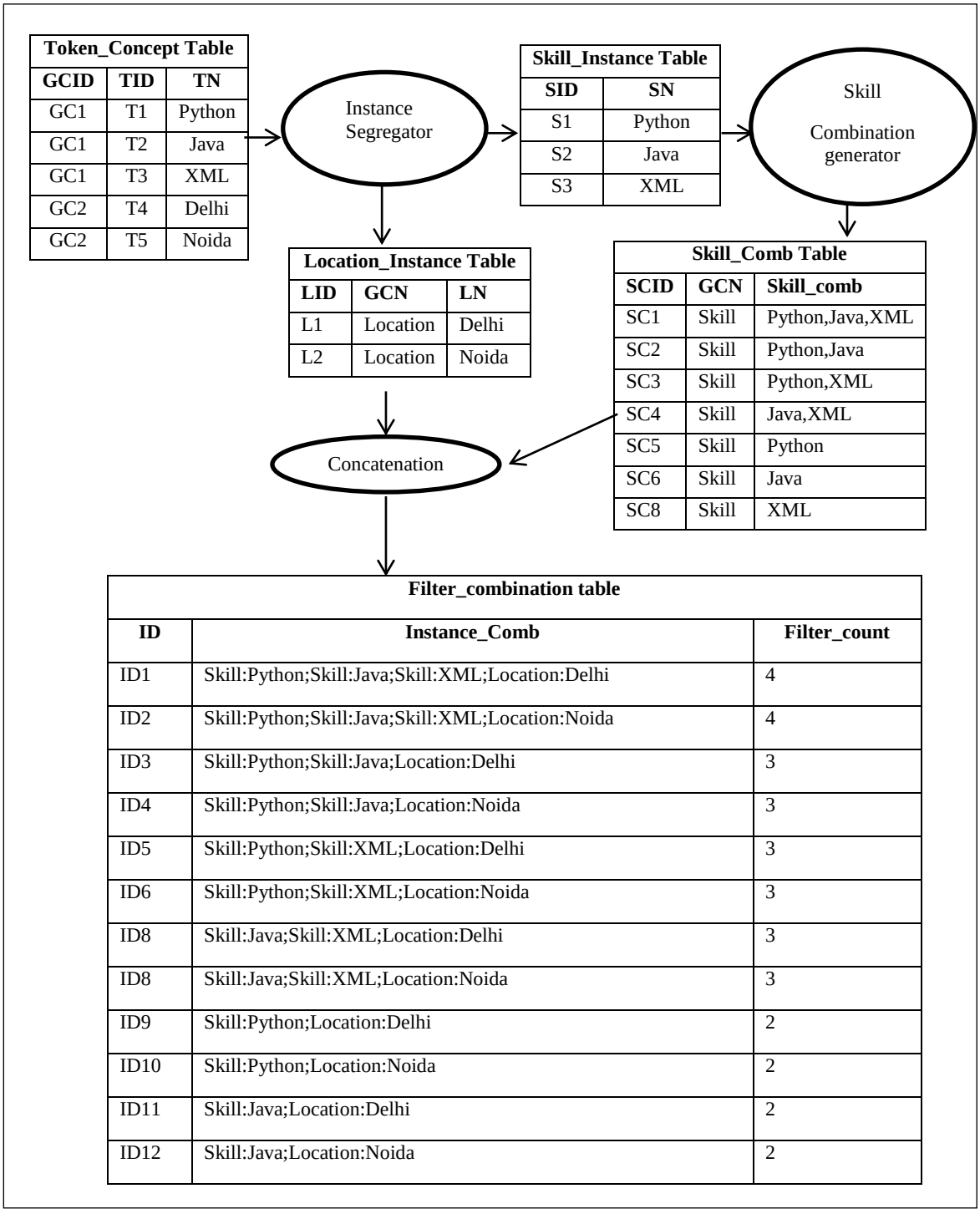


Fig. 4: Illustration of user query

Implementation of the Proposed Work

To analyze the proposed work, various experiments have been conducted. The proposed approach has been implemented in Java Eclipse. For the implementation of the proposed system, keyword based query is taken from user interface which retrieves results from ‘n’ number of ontologies and displays the results at the

same platform. The snapshot shown in Fig. 5 illustrates the results related to the user query "Java" as keyword and "Noida" as location.

Performance Evaluation of the Proposed System

To evaluate the Proposed System, the architecture has been implemented in JDK 1.8 Eclipse framework.

For analysis, three sets of queries as shown in Table 9 were given to the 20 users. Top 50 posts were considered as retrieved post and out of those, top 10 posts were used for making decision. The comparative analysis of Precision P, P', P'' for the queries belonging to corresponding query sets is shown in Fig. 6 where p depicts the average precision with respect to query q1 from all the jobboard, p' depicts the average precision with respect to q1 from all the individual proposed ontologies and p'' depicts the average precision from the proposed integrated system i.e., Jobology search system.

The average precision graph at system level is shown in Fig. 7.

It can be observed from Fig. 6a that the proposed system gives more relevant results as it exhibits high precision in comparison with Jobboards and individual Jobboard's ontologies. For QS3, the plotted values are comparatively low.

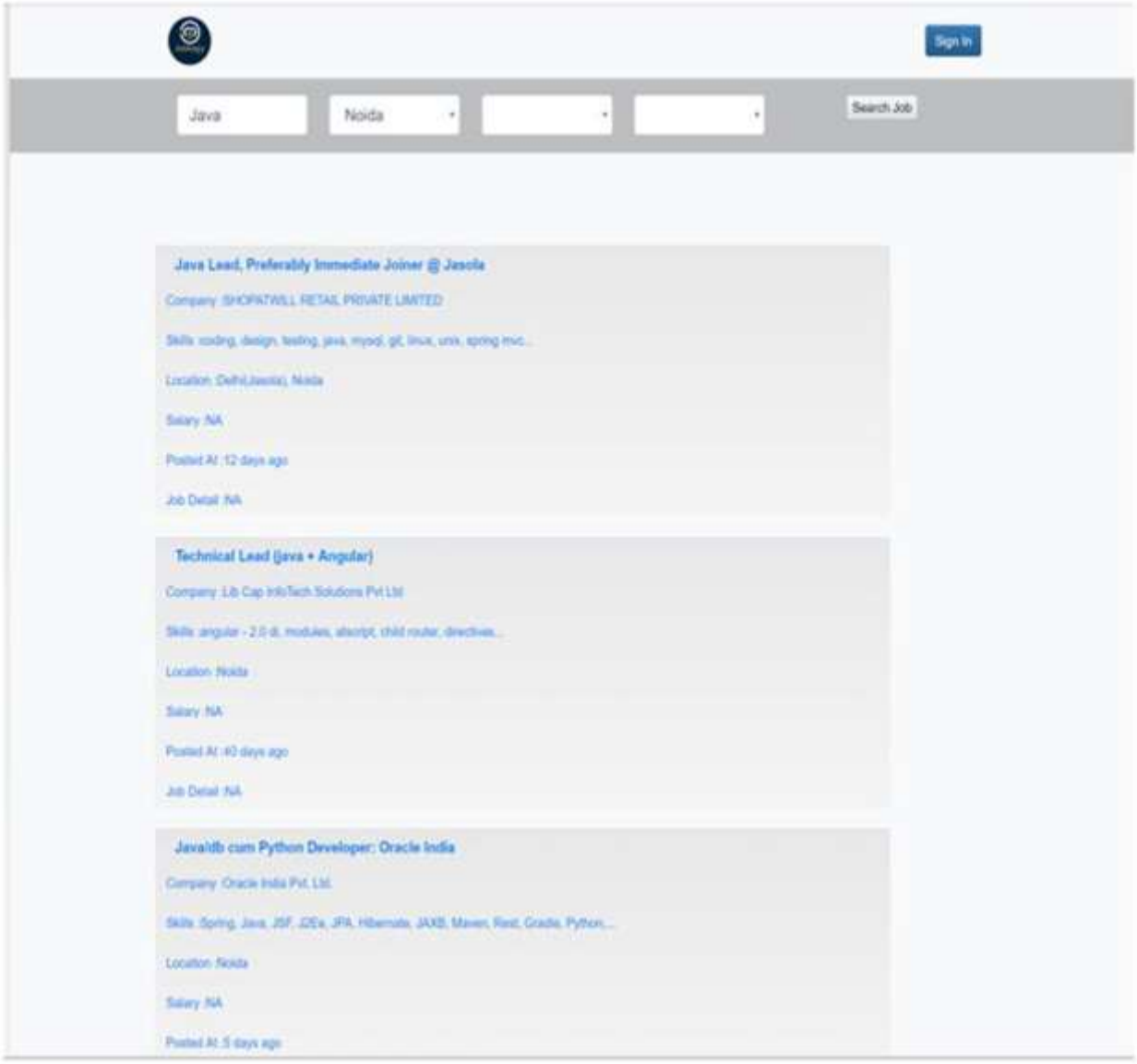


Fig. 5: Output

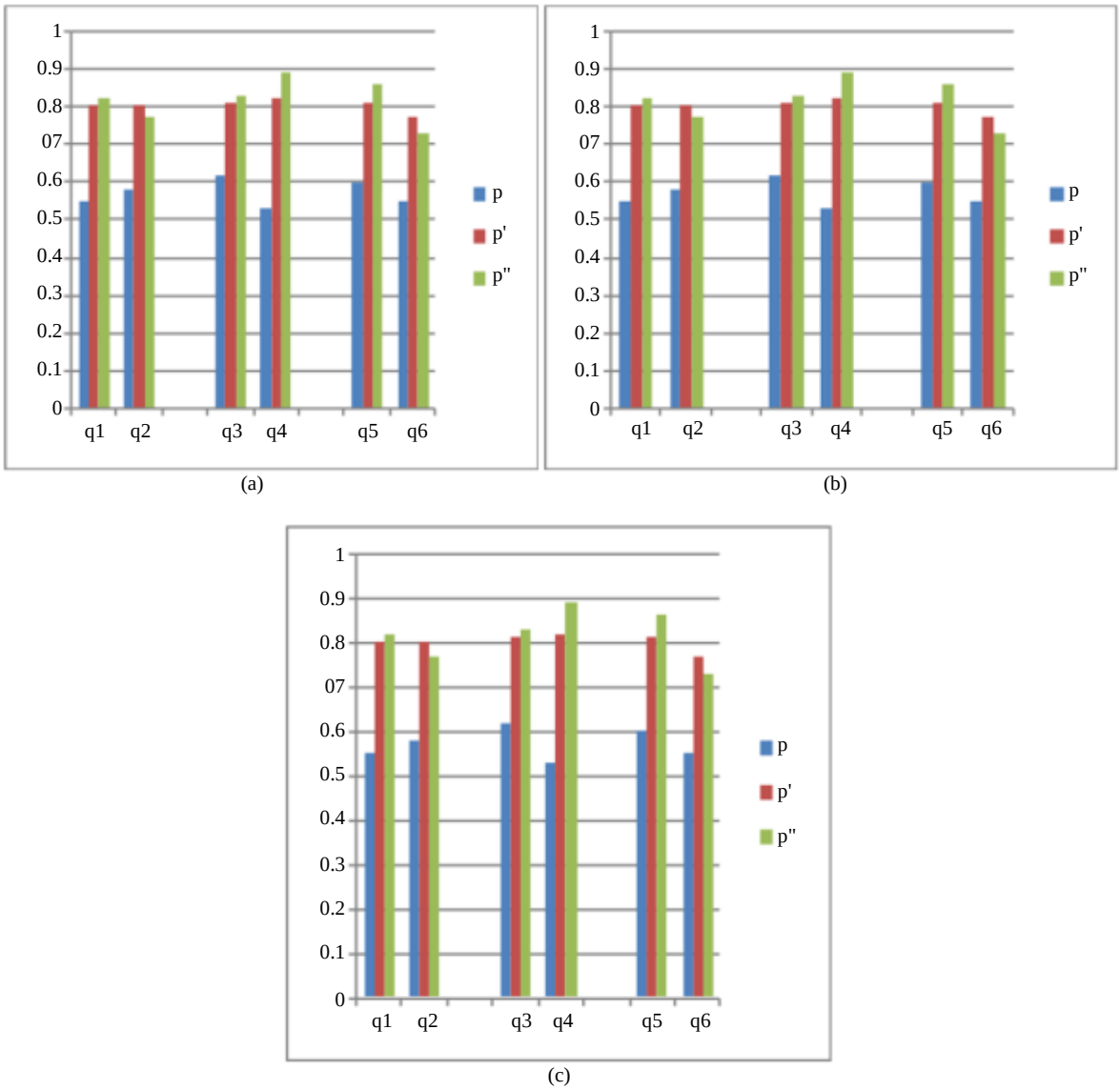


Fig. 6: (a) Precision analysis of queries for Query Set1 (b) Precision analysis of queries for Query Set2 (c) Precision analysis of queries for Query Set3

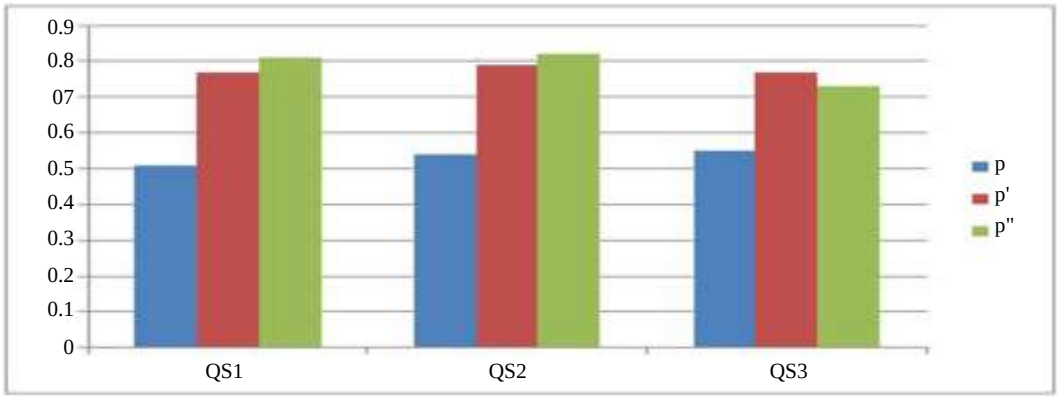


Fig. 7: Plotted values of average precision of query sets

Table 9: Sample query set

Query Set1		Query Set2		Query Set3	
S. no	**Query**	**S. no**	**Query**	**S. no**	**Query**
1	Java, Delhi	1	Oracle, Delhi	1	Advanced Java, Mumbai
2	PHP, Bangalore	2	SAP, Gurugram	2	.net, Pune
3	Python, Chennai	3	ADO, Dehradun	3	HTML, Javascript, Delhi
4	CSS, Delhi	4	Core Java, Ahmednagar	4	NodeJS, Javs, Ahemdabad
5	.net, Indore	5	Java, Struts, Bhopal	5	Abndroid, Bangalore
6	AngularJS, Ahemdabad	6	Java, Hibernate, Bhubeneshwar	6	Java, Spring, Kolkata

Conclusion

In this paper, a novel approach has been proposed for that allows querying of aligned ontologies. The basic idea of proposed approach is to make use of ontology alignment systems for querying. The system supports writing queries using global indexers built during alignment process and dataset. By this, more relevant results from n number of websites are presented to the user at one place.

While the contributions in this paper provide a novel and in our opinion, a scalable querying approach for querying, there is a scope of extension also. Given the fact that, currently proposed system is considering only jobboard websites as data for building ontology driven knowledge base, coverage area of knowledge base can be enhanced by considering direct jobs posted at the company's websites. The proposed system is focusing on converting semi-structured data available on the current web into structured data. The data coverage of proposed system can be expanded by taking unstructured text data into consideration and converting it into structured data using 'gate tools'.

Acknowledgment

I would like to thank (Late) Dr. A. K. Sharma for his guidance.

Author's Contributions

Ranjna Jain: This work has been accomplished as a Ph.D thesis of this author. She is the core contributor of this work.

Neelam Duhan: This author was one of the supervisors of the thesis and work extensively towards preparing the article.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Berners-Lee, T., Y. Chen, L. Chilton, D. Connolly and R. Dhanaraj *et al*., 2006. Tabulator: Exploring and analyzing linked data on the semantic web. Proceedings of the 3rd International Semantic Web User Interaction Workshop (UIW' 06).

Brin, S. and L. Page, 1998. The anatomy of a large-scale hypertextual Web search engine. Proceedings of the 7th International Conference on World Wide Web, (WWW' 98), Brisbane, Australia, pp: 107-117. DOI: 10.1016/S0169-7552(98)00110-X

Delbru, R., S. Campinas and G. Tummarello, 2011. Searching web data: An entity retrieval and high performance indexing model. J. Web Semant., 10: 33-58. DOI: 10.1016/j.websem.2011.04.004

Jain, R., N. Duhan and A.K. Sharma, 2017. Design of building automatic global concept indexer for ontology alignment. Int. J. Eng. Technol., 9: 1532-1541. DOI: 10.21817/ijet/2017/v9i3/170903503

Jain, R., N. Duhan and A.K. Sharma, 2018a. A novel method for building indexer for aligning ontologies. Int. J. Inform. Retrieval Res., 8: 67-86. DOI: 10.4018/IJIRR.2018100105

Jain, R., N. Duhan and A.K. Sharma, 2018b. Comparative study on ontology management approaches in semantic web. Int. J. Comput. Sci. Eng., 6: 132-140. DOI: 10.26438/ijcse/v6i1.132140

Mcguinness, D.L., R. Fikes, J. Hendler and L.A. Stein, 2002. DAML+OIL: An ontology language for the Semantic Web. IEEE Intell. Syst., 17: 72-80. DOI: 10.1109/MIS.2002.1039835

O'Connor, M.J. and A. Das, 2007. Querying the semantic web with SWRL. Proceedings of the International Conference on Advances in Rule Interchange and Applications, Oct. 25-26, Springer, Orlando, Florida, pp: 155-159. DOI: 10.1007/978-3-540-75975-1_13

Sander, M., U. Waltinger, M. Roshchin and T. Runkler, 2014. Ontology-based translation of natural language queries to SPARQL. Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI) Symposium on) Natural Language Access to Big Data, (ABD' 14).

Shekarpour, S., 2011. DC Proposal: Automatically transforming keyword queries to SPARQL on Large-Scale Knowledge Bases. Proceedings of the 10th International Conference on The Semantic Web, Oct. 23-27, Springer, Bonn, Germany, pp: 357-364. DOI: 10.1007/978-3-642-25093-4_29

Weiand, K.A., 2011. Keyword-based querying for the social semantic web: The KWQL language: Concept, algorithm and system. Dissertation, Faculty of Mathematics, Computer Science and Statistics, LMU München.

Yahya, M., K. Berbrich, S. Elbassuoni, M. Ramanath and V. Tresp *et al*., 2012. Natural language questions for the web of data. Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Learning, Jul. 12-14, Association for Computational Linguistics, Jeju Island, Korea, pp: 379-390.

Zhou, Q., C. Wang, M. Xiong, H. Wang and Y. Yu, 2007. SPARK: Adapting keyword query to semantic search. Proceedings of the 6th International Semantic Web Conference on the Semantic Web and 2nd Asian Semantic Web Conference, Nov. 11-15, Busan, Korea, pp: 694-707. DOI: 10.1007/978-3-540-76298-0_50

APPENDIX- I

QID	SPARQL Query	Filter_count

O1Q1	"PREFIX rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p:
	<http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?Job ?title ?skill ?location where" {
	"?Job p:hasloc ?location."+ "?Job p:hasskill ?skill."+
	“FILTER(?skill=”Python”)." + “FILTER(?skill=”Java”)."+
	“FILTER(?skill=”XML”)."+ “FILTER(?location=”Delhi”)."
	}
		4

O1Q2	"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p:
	http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?Job ?title ?skill ?location where" {
	"?Job p:hasloc ?location."+ "?Job p:hasskill ?skill."+
	“FILTER(?skill=”Python”)." + “FILTER(?skill=”Java”)."+
	“FILTER(?skill=”XML”)."+ “FILTER(?location=”Noida”)."
	}
		4

O1Q3	"PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>\n" +
	"PREFIX p: <http://www.semanticweb.org/ranjna/ontologies/2018/1/test2.owl#>\n"+
	"SELECT ?Job ?title ?skill ?location where" {
	{
	"?Job p:hasloc ?location."+ "?Job p:hasskill ?skill."+
	“FILTER(?skill=”Python”)." + “FILTER(?skill=”Java”)."+
	“FILTER(?location=”Delhi”)."
	}
		3