Journal of KONBiN 2021
Volume 51, Issue 1
DOI 10.2478/jok-2021-0006

Varsha MITTAL, Durgaprasad GANGODKAR, Bhaskar PANT
Graphic Era Deemed to be University, Dehradun, Uttarakhand, India

K-GRAPH: KNOWLEDGEABLE GRAPH FOR TEXT DOCUMENTS

Abstract: *Graph databases are applied in many applications, including science and business, due to their low-complexity, low-overheads, and lower time-complexity. The graph-based storage offers the advantage of capturing the semantic and structural information rather than simply using the Bag-of-Words technique. An approach called Knowledgeable graphs (K-Graph) is proposed to capture semantic knowledge. Documents are stored using graph nodes. Thanks to weighted subgraphs, the frequent subgraphs are extracted and stored in the Fast Embedding Referral Table (FERT). The table is maintained at different levels according to the headings and subheadings of the documents. It reduces the memory overhead, retrieval, and access time of the subgraph needed. The authors propose an approach that will reduce the data redundancy to a larger extent. With real-world datasets, K-graph's performance and power usage are threefold greater than the current methods. Ninety-nine per cent accuracy demonstrates the robustness of the proposed algorithm.*

Keywords: subgraph mining, graph database, text classification

1. Introduction

The data available everywhere is either news, tweets, blogs, comments or documents. All the mentioned sources contain data in the textual format, having many facts, observations and insights which can be a breakthrough in various business decisions [3]. In its inherent form, the data is of no utilization as it does not provide any value, so no knowledge can be extracted from it. To extract the knowledge from the huge number of text documents, an appropriate structure to store the documents is required [21]. The perilous metric used in text analysis is to identify the context and existing relationships between the words [20]. The mechanism to store the data should provide a connected way to store the extracted meaning. The appropriate model to store such highly connected data is a graph database. The knowledge extracted from documents is represented through nodes and edges between them [6]. But building the knowledgeable graph manually for a large set of documents like legal case files, books is a challenging task.

Previously, a technique like Bag-of-words was proposed to capture the frequency of words and phrases, but the structural information was ignored [18]. But it has been shown that structural information affects classification accuracy [13]. So, graph databases appeared to be more expressive in storing text documents. But it introduces the complexity of comparing two similar graphs [16]. It is computationally more expensive when the graphs are large [1].

In this paper, an approach is proposed to address the challenge of constructing knowledgeable graphs (K-graphs). The work also focuses on reducing the computational cost so that the expressive power of graph databases can be fully utilized. To generate the k-graphs, the documents are retrieved from the URLs. Before storing, the documents are pre-processed using the standard dictionaries. The extracted headings and subheadings are organized in a subgraph structure. The general headings appear close to the top, and the more specific subheadings appear at the lowest level. To reduce the computational cost, the weighted subgraphs are used, and a referral table is maintained, which will be updated incrementally. To include the new documents in k-graphs, the referral table is used. Thus, the search space for matching the headings and subheadings is reduced.

The rest of the paper is organized as follows: Section II gives the literature survey of the work done by different researchers, Section III discusses some preliminary concepts and the proposed approach for building K-graphs and the evaluation and comparison of the proposed algorithm with other baseline techniques is given in section IV. Finally, the paper is concluded with the future scope in section V.

2. Related work

Previous approaches that use graph-based document representation were mainly meant for web documents. The method proposed by Geibel, Krumnack, Pustylnikov and Mehler

[5] showed that web documents could be classified using only structural information. However, excluding linguistic and semantic structure in the case of text documents affects classification accuracy. Schenker [22] proposed an approach by combining semantic information along with structural information. And for classification, the graph similarity-based algorithm was used. But the approaches that work on graph similarity-based methods are computationally expensive [19]. Instead of using pure graph representation, a hybrid representation was proposed to reduce computational cost [17]. However, even after using a hybrid method, the computational overhead was still high due to many nodes and edges and exponential growth in search space.

Gee and Cook [4] proposed to include both semantic information and word order in the graph that was used for text classification. But this was limited to small text only. In our work, we use structural, linguistic and semantic information to generate K-graphs for text classification, even for large text.

An extensive study of different Frequent Subgraph Mining (FSM) techniques has been done [12, 23, 24,. Most of the FSM techniques work on the generation of candidate keys. The generation of unnecessary candidate keys is a major limitation of FSM techniques [14]. To generate the candidate keys, the database is scanned recursively, and it is computationally expensive for large datasets. A lot of work was done to design an efficient FSM approach by improving search strategies [11, 15]. A constraint-based FSM approach is proposed so that unwanted subgraph with infrequent patterns can be eliminated [27]. Another technique called Closegraph is proposed that mines only the subgraphs which are closed [28]. Pruning only the close subgraphs increases the closegraph's performance in comparison to other techniques, but some important subgraphs are also ignored [25]. In our work, we have focused on a weighted subgraph that combines the constraint of weight with FSM. With weighted subgraphs, the search space is reduced as the subgraphs which are insignificant are ignored. Moreover, in our work, we also introduce the concept of maintaining a Fast Embedding Referral Table (FERT) to reduce the search complexity and make the graphs knowledgeable by storing all important headings and subheadings at different levels.

3. K-Graphs

3.1. Overview of K-Graphs approach

An approach called K-graphs is proposed in this section to generate knowledgeable graphs for large text documents. K-graphs uses two important concepts. It assigns weight to each keyword stored in a node according to its relative frequency with respect to a document to be stored in a graph database. And it maintains a referral table called FERT that have the list of all frequent subgraphs along with the embedding of each subgraph root node.

The proposed approach works in the following steps:

1. Pre-processing- For a set of documents to be stored, extract all the keywords. Since all the extracted keywords are not to be stored, to improve the model's accuracy, the text documents are pre-processed by removing extra words.

2. Storing in a graph database- All the retained words are stored in the graph database along with the document_id.

3. The important subheadings are assigned a weight. A subgraph is created using the subheading as the root node, and referral table FERT is updated accordingly. The FERT trims the search space before generating the new subgraphs; then, the FERT is accessed. If the subheadings already exist, then only the embedding information is updated. Otherwise, the subgraph is created, and FERT is updated.

3.2. Proposed methodology

Initially, two graphs $Dict_{eliminate}$ and $Dict_{retained}$ are created from the pre-existing dictionaries according to the application context. $Dict_{retained}$ is a dictionary of retained words from a larger graph database. The model creates graph databases of benchmark dictionaries of verbs, positive, negative, common nouns and prepositions. This graph is named $Dict_{eliminate.}$ These graphs preprocess the text and retain only the keywords. Further, a graph database based on the dictionary of legal words (taken as an example for the paper) filters the words and a graph database with legal words in a document is created. $Dict_{retained}$ is a benchmark graph database of legal words created by authors.

1. The dictionaries are created to preprocess the documents
   $Dict_{eliminate}$ and $Dict_{retained}$
   After step 1, two graphs will be created.
2. Pre-process the document that is to be tested
   a. Eliminate the unnecessary words using the graph created in step one using $Dict_{retained}$.
   Read a line of the document, split the words, and remove the unwanted words using the dictionary graph.
   b. Retain relevant words using Retained words' graph database in step 1.
   Let W_n is the set of all retained words in a document. It can be explained as follows:

$$W_n = \left\{ \left(Doc_{w_{n_i}}, f_{n_i} \right) \;\middle|\; Doc_{w_{n(i)}} \in Dict_{retained} \right\}_{n \in \overline{(1,l_n)}} \tag{1}$$

where $Doc_{w_{n_i}}$ is the *i*-th word of the *n*-th document, f_{n_i} is its (word) frequency in the *n*-th document, $Dict_{retained}$ is the dictionary of retained words, and l_n is the length of the *n*-th document.

Using equation (1), finally obtained retained words are stored using a graph database. All the retained words are the nodes, and the relationship with the document are the edges in the graph database. The figures given in section 4.5 show the snapshot of the graph database.

3. Create Subgraphs based on the important attributes using (2). For example, if we store a legal case in a graph database, the important attributes like the category of the case: Indian Penal Code (IPC), Civil Procedure Code (CPC), Criminal Procedure Code (CrPC) forms the head nodes. The properties of the data are stored as the attributes (or properties) of the node. Sections related to the above-mentioned categories form the leaf nodes. A relation is set between the category and the section. Further, the cases related to the section fall within the relevant sections and a relationship is established between the case and the section. The graph is further split from the section to form the subgraphs. The subgraphs are split using equation (2).

$$Attr_n = \{Atrrib_i \mid Attrib_i \in Doc_n\}_{i \in \overline{(1,C)}} \tag{2}$$

where Atrrib_i is the pre-given attribute of the database, $Attrib_i \in Doc_n$ means that given attribute characterises the document (i.e. if the document involves Indian Penal Code(IPC), then IPC is stored as an attribute in the node) and C is the total number of all attributes.

4. Allocate weight to the nodes by calculating the arithmetic mean by using equation (3)

$$\bar{f} = \frac{\sum_{i=1}^{l} f_i}{\sum_{i=1}^{l} tf_i} \tag{3}$$

where f_i is the frequency of a keyword in a document and tf_i is the total number of extracted words from the document. Thus, the properties stored with a node are specified in equation (4):

$$Key_n = \{W_n, D_n, Attr_n, \bar{f}\} \tag{4}$$

where W_n , $Attr_n$, *and* $\bar{f}$ are described above in equation (1), (2) and (3) and D_n is the document id.

3.3. Algorithm of K-Graph

The algorithm for the proposed methodology is specified in three parts. The first part describes the creation of the main graph specified by algorithm 1 in a tab. 1. Fast Embedding Referral Table (FERT) is created when the main graph database is split into sub-graphs. FERT has the references for the main and subgraphs with keywords. It is an index of the entire database. FERT is created so that the model can refer directly to the related sub-graphs instead of traversing the entire database. This helps in cutting the traversing time considerably. Algorithm 2 in a tab. 2 describes the creation of subgraph and updating the FERT table. The algorithm 3 in a tab. 3 defines the traversal and searching the graph using the referral table.

Table 1

Creating the main graph

Algorithm 1. Creating Main Graph

Create a Graph
ID← set to zero
Root ← ID, Name
While the end of the folder:
File ← read file
Title ← read the title
ID ← +1
SubNode ← [ID, Title]
[Contains]← set relation with Root
Ltable← ID, Title
While not end of File:
If Heading1:
ID ← +1
Sub1←Create node
Properties←{Lev1_ID, Current_ID,Heading1}
[Contains]← set relation with SubNode
Key_w← Applying equations (1) to (2)
Ltable ← append [ID, Heading1, Key_w]
if Heading2:
ID ← +1
Sub2←Create node
Properties←{Lev2_ID, Current_ID,Heading2}
[Contains]← set relation with Sub1
Key_w← Applying equations (2) to (4)
Ltable2 ← append [Lev1_ID ,ID, Heading2, Key_w]
End if
End while
End while

Algorithm 1 shows the process of creating graphs. The process of creating the graphs is simultaneously followed by creating the subgraphs. The referral table is updated concurrently on the creation of the subgraph. The graph is created to store any document with its headings and subheadings. For storing any document, SubNode is created with the document's title and the document Id as node embedding. The relation of this SubNode is established with the root node. For every heading in the document, another SubNode is created with its Lev_ID, Current_ID, Heading as node embedding. The referral table is updated by appending the key weight as calculated in (3) represented by wt. The process continues for all the headings in the document.

Table 2

Creating Subgraphs

Algorithm 2. Creating Subgraphs

Sub-Graph – I
Read FERT_table
Create subGraph
ID← set to zero
n←1
Root ← ID, 'Level{n}'
Read FERT table
LID ← Level ID //store Level ID of first record
While not end of FERT_table:
If LID== Level ID
Keyword ← {Key_w}
While key in Keyword:
If unique(kw):
Sub←create node
ID←+1
Properties←{ID,Key, wt}
Keyword← set relation with Root
Else
wt← mean(Previous_wt ,current_wt)
Update Properties
End if
End while
Else:
Create subGraph
ID← set to zero
n=n+1;
Root ← ID, 'Level{n}'
LID ← Level ID
End while

Sub-Graphs Until the FERT table has records sub-graphs will be created starting from Level1.

Algorithm 2 in tab. 2 shows the creation of subgraphs. The subgraphs are created by taking level id from the referral table FERT. Before inserting a new node, the node for the corresponding keyword is searched in the referral table; if found, the node embeddings and the referral table are updated. Otherwise, a new node is created, and its relation is established with the root node. The structure of the referral tables produced is shown below in FERT_table1 in a tab. 3 and FERT_table2 in a tab. 4.

Table 3

FERT_table 1

Level 0	Level 1	Level 2	Keywords
0	1	{2,Name}	2.1
0	1	4	4.1
0	1	6	6.1

Table 4

FERT_table2

Level 3	Keywords
{2,3,Name,Kws}	3.1
{4,5,Name,Kws}	5.1

Algorithm 3 in tab. 5 shows the process of traversing the subgraphs whenever searched for a particular keyword. All the subgraphs are traversed, and the target subgraph is displayed.

Table 5

Traversing a Subgraph

Algorithm 3. Traversing a subgraph

SubGraph1 ← read Sub-Graph – I
SubGraph2 ← read Sub-Graph – II
Key_search ← read keyword to be searched
If (match(keyword,SubGraph1)):
Display {node name, weight}
Else If (match(keyword,SubGraphII)):
Display {node name, weight}
Else
Display ← errors message
End If

3.4. Fast Embedding Referral Table (FERT)

The proposed referral table (FERT) maintains the list of extracted subgraphs headers at a different level. It provides an efficient mechanism by reducing redundancy, memory overhead and retrieval time. It supports fast update, searching and access to the target node. The referral table components will be the subgraph heading (node id of the root of the extracted subgraph) and the predecessors and successors of the corresponding root node id of the stored subgraph. Along with the IDs, it would have keywords as well. This will make the graphs Knowledgeable as each subheading maintains a link to its adjacent node and its predecessor node. FERT provides an efficient method for inserting and deleting edge in the

graph. A new edge is initially searched for in the FERT. If found, it will not be inserted again as FERT does not allow for multiple entries for the same edge. A relationship would be established between the existing keyword and the level; also, the weight would be incremented by one. Adding a new entry in the FERT includes the storage of the node id of the subgraph node and also the storage of the embedding of all nodes at its next level. The deletion of every node will result in the removal of all the relationships to the node, and the counter is decremented by one. Maintenance of the FERT will eliminate the need for the entire database retrieval and will reduce the retrieval time, as shown in the results.

4. Experimental evaluation

In this section, the proposed approach K-Graph is compared with the three basic approaches Gspan [27], WARM [26] and GastonFSM [2].

Gspan technique carries out the iteration of subgraph generation from scratch after each update in the input graph. The methodology of WARM works on the concept of weighted association rule mining.

The GastonFSM keeps a list of each node's embedding. After evaluating the existing embedding list, the new nodes are created. It has improved the performance of the embedding generation system but has increased the traverse time as even for single embedding; the whole chain has to be traversed.

In the further section, the proposed technique is compared with the above-mentioned techniques with respect to efficiency, memory overhead and accuracy. The memory overhead is computed in terms of the consumption of memory. Finally, the output of the proposed algorithm in the form of a graph database is shown.

4.1. Experimental environment and datasets

The experiments are performed on Intel® Core TM i5-8th Gen 8250u CPU with 16 GB RAM with NVIDIA GeForce and 256 CUDA cores GPUs. Graph Database Neo4j has been installed on Windows 10. Python and Pytorch 0.4.0 are taken as a programming language.

The four actual datasets are used to conduct the experiments. The studies are carried out on four different types of documents. The graphs are developed for the storage of academic papers, thesis, books, and legal cases. “The number of documents considered is 10000. For the sake of the experiment, legal judgments are downloaded directly from the web site of the Supreme court (indiankanoon.org). They are in PDF formats.

To assess the efficiency of the proposed model on other domains, we have also tested on the thesis (library.stanford.edu), research papers (shodhganga.inibnet.ac.in) and books (read.gov)”.

4.2. Efficiency

The efficiency of the proposed algorithm is compared with Gspan, WARM and GastonFSM. Efficiency is evaluated in terms of time required for generating the graph along with all the subgraphs and mining the entire graph. Figure 1 illustrates the result of the efficiency comparison. The X-axis shows the different datasets used, and Y-axis shows the elapsed time for the graph generation. The estimated time is calculated for a small subset of data and then extrapolated for the entire dataset. The graph clearly shows that K-graph outperforms in comparison to all the other three techniques. The result's improvement is due to the use of a referral table (FERT) since it eliminates the need for accessing the entire graph again and again.

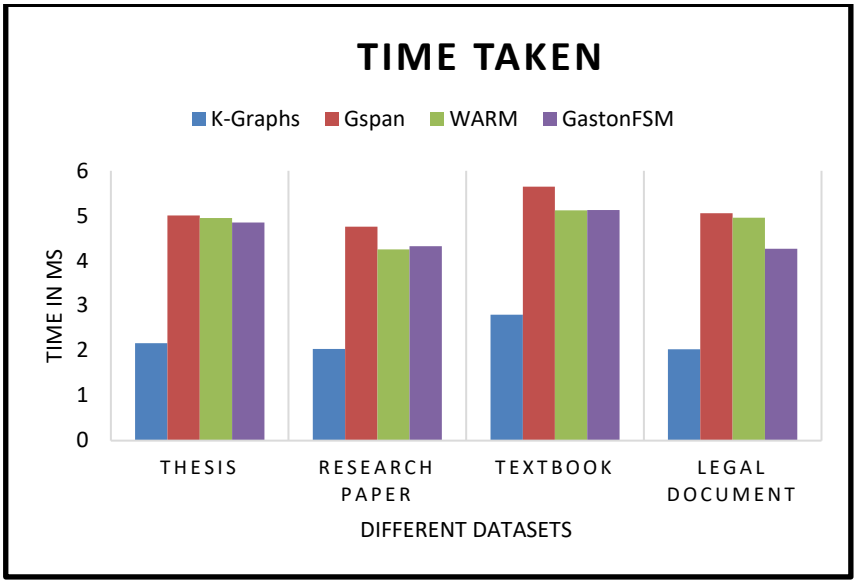

Fig. 1. Time comparison for graph generation

Gspan takes the maximum time as every subgraph generation is done from scratch. Gaston FSM & WARM perform well when the graph is small. But with larger graphs, the performance is depreciated since all the node embeddings are to be stored. And for larger graphs, the embeddings become too large to store. This increases the memory requirement and time taken; hence the efficiency is reduced.

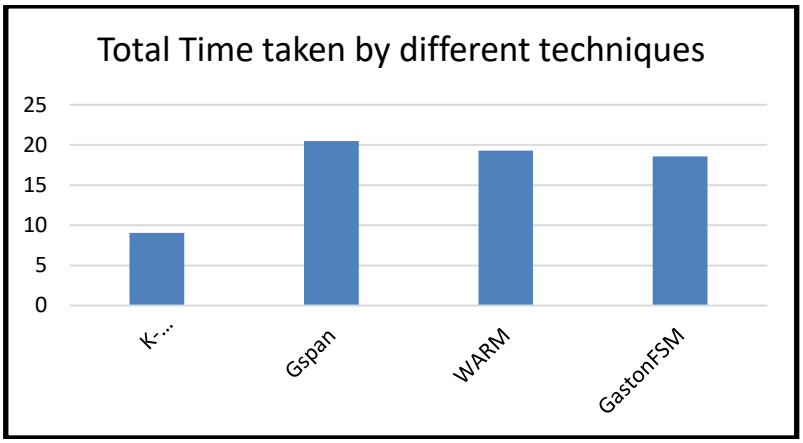

Fig. 2. Comparison of K-graph to other techniques

Figure 2 shows the comparison of K-graph to other techniques on improved computation time. It shows that the K-graph performs 55.85% better than Gspan, 53.11% better than WARM and 51.5% better than GastonFSM.

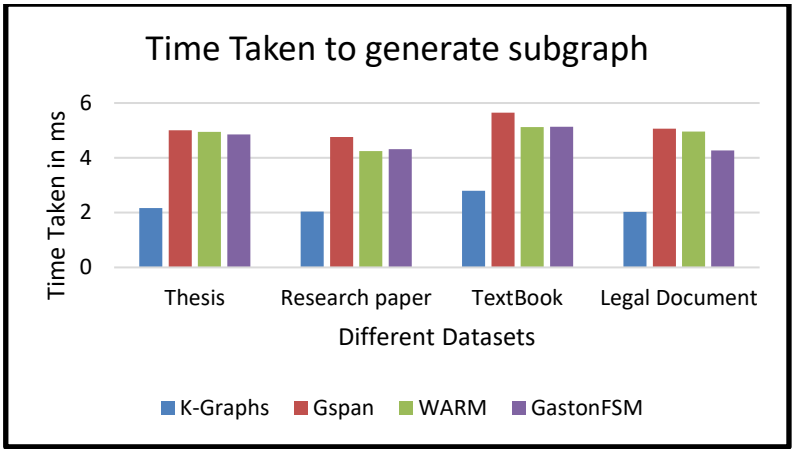

Fig. 3. Time taken to generate subgraphs

Efficiency can also be evaluated in terms of the time taken to generate the subgraphs at different levels where the levels are taken from the FERT. Figure 3 shows the time taken by different techniques to generate the subgraphs on the different datasets.

In K-graphs, the subgraphs are generated after searching the referral table at a previous level, which reduces the time required to mine the entire workload. So it outperforms the other three techniques. Figure 3 shows that K-graphs performs 56.42% better than Gspan, 48.9% better than GastonFSM and 44.07% better than WARM.

4.3. Memory overhead

The proposed algorithm is evaluated in terms of memory overhead. The memory overhead is the memory required to store the subgraphs generated. The subgraphs are generated according to the different subheadings of the target document.

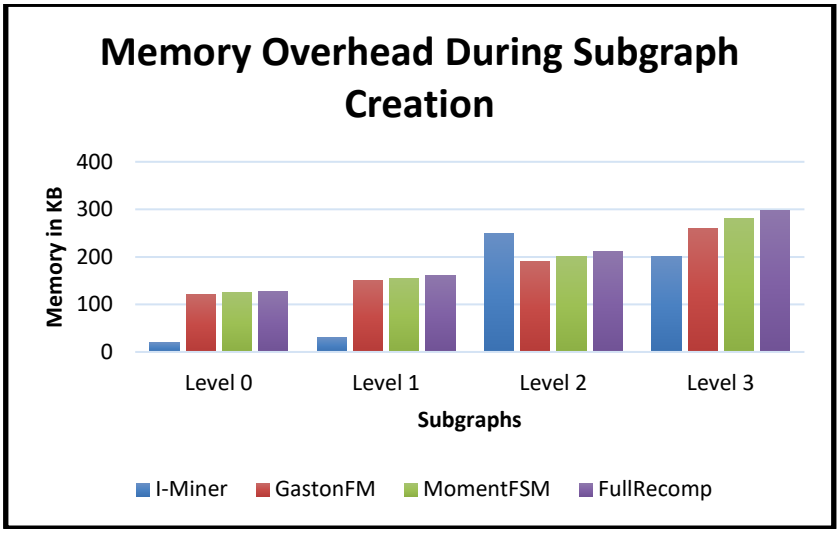

Fig. 4. Memory overhead of different techniques

The graph in figure 4 shows the memory consumption for storing the subgraphs in different techniques. The interesting point is that the memory consumption of the proposed algorithm is not large at level 0, and level 1 since the subgraphs at level 0,1 are according to headings and subheadings. At level 2, where the keywords are stored, the memory consumption increases. But at level 4, only the relation to the existing node is established, so finally, the memory consumption decreases and improves the K-graph's performance. The WARM has maximum memory consumption since the subgraph is always generated from scratch. The histogram in fig. 5 below shows the comparison of the K-graph with the other three techniques. It is observed that the memory consumption of the K-graph is less since the FERT is maintained at different levels, which not only reduces the data redundancy but also affects the memory requirement to a larger extent. As it maintains the list of embeddings, the memory requirement for the FERT is insignificant compared to other techniques. In the proposed approach, the list of embeddings is carefully reduced.

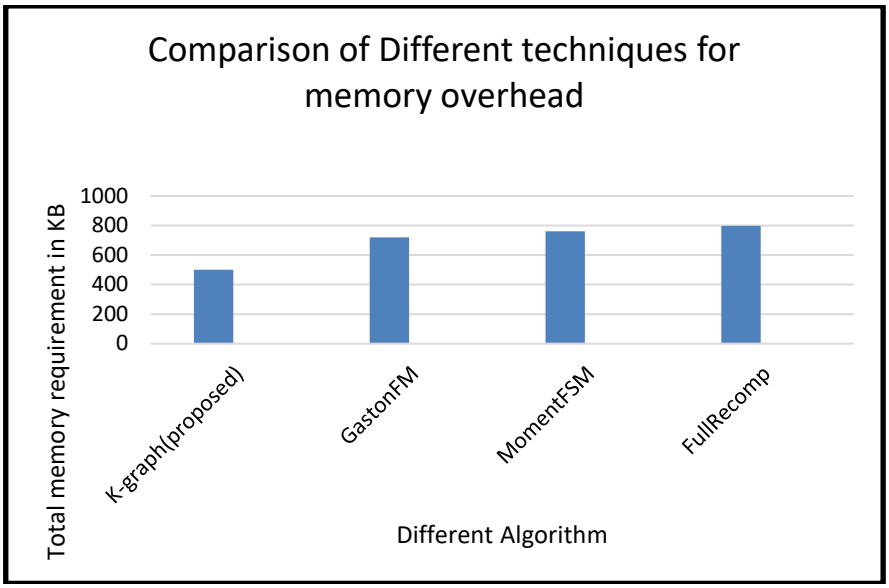

Fig. 5. Improvement of K-graph over other techniques in memory overhead

The improvement of the proposed algorithm is shown in tab. 4 below, which signifies the number of subgraphs reduced. Table 6 clearly shows the number of subgraphs that can be created according to the subheadings in the legal case files and the number of subgraphs that are actually created.

Table 6

Comparison of the number of subgraphs created

Legal Case file	Number of subgraph in the document	Number of subgraph actually created
GianSingh2012.pdf	7	7
AbdulvsRSNaya1991.pdf	6	5
GMtankvsstofguj2005.pdf	12	8
Narinderstofpunjab2014.pdf	20	15
NoidaentreNoidaors2007.pdf	15	11
Ranbhaivsstofgujarat2000.pdf	14	14
VikramvsstofMaharash2014.pdf	13	13

4.4. Accuracy

The improvement can also be evaluated in terms of Accuracy. Table 7 shows the keywords extracted (TN) and the keywords stored in every subgraph (TP). The table shows the keywords stored under the subgraph for the dataset [26]. Using these values, the precision, recall, accuracy and F-score of the proposed algorithm is computed. From the

table, it can be perceived that the accuracy of the proposed algorithm is approximately 99 per cent. It proves that the proposed algorithm is optimized to generate the subgraphs and store the relevant keywords after removing redundancy.

Table 7

Computed values of Precision, Recall, Accuracy and F_score

Sub graph	TP	FP	TN	FN	Precision	Recall	Accuracy	F-score
1	1024	0	1024	0	1	1	1	1
2	1012	0	1012	0	1	1	1	1
3	9862	0	9861	1	0.999898	1	0.99994	0.9999493
4	8013	0	8011	2	0.999750	1	0.99987	0.9998752
5	9776	0	9771	5	0.999488	1	0.99974	0.9997443

4.5. Results

Figure 6 shows a screenshot of the Level 1 graph created. The title of the document is taken as the root node and Headers as its leaf nodes.

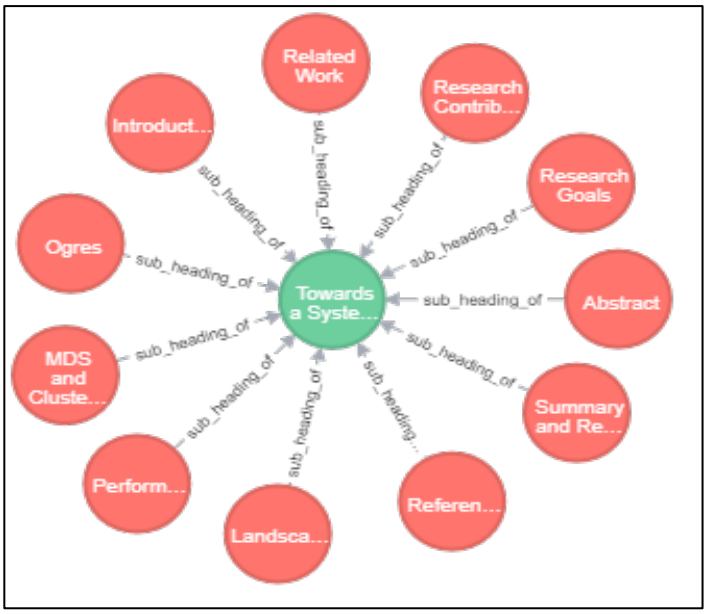

Fig. 6. Snapshot of graph database at level 1

Figure 7 shows a screenshot of the Level 2 graph. One of the headers has been considered with keywords. Similar headers will not be added, only weight would be incremented by one, and unique keywords would be added.

Fig. 7. Snapshot of level 2 graph database

5. Conclusions

The proposed algorithm allows for the careful classification of the text using the database of Graphs. The findings clearly show that redundancy is reduced by more than 50 per cent, and even space is reduced by about 50 per cent. Because time is directly proportional to the amount of space taken, lower space requirements resulted in lower traverse time needed. FERT is one more plus. It is only used to search the Database, and is updated periodically. K-graph can prove to be a revolution in the processing of natural languages (NLP). The authors plan to refine the algorithm a little more in the future and apply it to real-life case studies to determine its robustness. K-graph will help develop Bots, Plagiarism software, electronic keyword library, educational browsers, and even design new SAP software to generate fast queries.

6. References

1. Atastina I., Sitohang B., Saptawati G., Moertini V.S.: A Review of Big Graph Mining Research. IOP Conf. Ser. Mater. Sci. Eng., 180, 12-16 , 2017.
2. Abdelhamid E., Canim M., Sadoghi M., Bhattacharjee B., Chang Y., Kalnis P.: Incremental Frequent Subgraph Mining for Large Evolving Graphs. IEEE Transactions on Knowledge and Data Engineering, 29, 12, 2017.
3. Dhiman A., Jain S.K..: Frequent subgraph mining algorithms for single large graphs — A brief survey. International Conference on Advances in Computing, Communication, Automation (ICACCA) (Spring), Apr. 2016.
4. Gee K.R., Cook D.J.: Text Classification Using Graph-Encoded Linguistic Elements. In FLAIRS Conference, 487-492, 2005.
5. Geibel, Krumnack U., Pustylnikow O., Mehler A.: Structure-Sensitive Learning of Text Types. Advances in Artificial Intelligence ,4830, 642-646, 2007.
6. Giarelis N., Kanakaris N., Karacapilidis N.: On a Novel Representation of Multiple Textual Documents in a single Graph. Proceedings of International Conference on Intelligent Decision Technologies IDT 2020, Split, Croatia, 105-115, 2020.
7. https://shodhganga.inibnet.ac.in.
8. https://library.stanford.edu/spc/universityarchives/dissertations-and-theses.
9. https://indiankanoon.org/browse/supremecourt/
10. http://read.gov/books/
11. Huan J., Wang J., Prins J.: Efficient mining of frequent subgraphs in the presence of isomorphism. Third IEEE International Conference on Data Mining, 549–552, 2003.
12. Inokuchi A., Washio T., Motoda H.: An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery, London, UK, UK,13–23, 2003.
13. Kang U., Tsourakakis C.E., Faloutsos C.: PEGASUS: A Peta-Scale Graph Mining System Implementation and Observations. Ninth IEEE International Conference on Data Mining, Miami Beach, FL, USA, Dec. 2009.
14. Kuramochi M., Karypis G.: Frequent Subgraph Discovery. Proceedings - IEEE International Conference on Data Mining, ICDM, 313–320, 2010.
15. Kuramochi M., Karypis G.: GREW - a scalable frequent subgraph discovery algorithm. IEEE International Conference on Data Mining (ICDM'04), 439–442, 2004.
16. Markov A.: Efficient Graph-based Representation of web Documents. Proceedings of the Third International Workshop on Mining Graphs, Trees and Sequences, Potro Portugal 52-62, 2005.
17. Markov A., Last M., Kandel A.: A Fast Categorization of Web Documents represented by Graphs. Advances in Web Mining and Web Usage Analysis, 4811, 56-71, 2007.
18. Mukund D., Kuramochi M., Karypis G.: Frequent Sub-structur based Approaches for Classifying Chemical Compounds, In Proceedings of the Third IEEE International Conference on Data Mining, 2003.

19. Nijssen S., Kok J.N.: A Quickstart in Frequent Structure Mining Can Make a Difference. Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2004.
20. Paulheim H.: Knowledge graph refinement: A survey of approaches and evaluation methods. Semantic Web, vol. 8, no.3, 489–508, 2016.
21. Pokorny J.: Integration of Relational and Graph Database Functionally. Foundation of Computing and Decision Sciences, 44, 4, 427-441, 2019.
22. Schenker A.: Graph Theoretic Techniques for Web Content Mining, Phd Thesis, University of South Florida, 2003.
23. Ramraj T., Prabhakar R.: Frequent Subgraph Mining Algorithms – A Survey. Procedia Comput. Sci.,47, 197–204, 2015.
24. Rehman S.U., Khan A.U and Fong S.: Graph mining: A survey of graph mining techniques. Seventh International Conference on Digital Information Management (ICDIM 2012), 88–92, 2012.
25. Rehman S.U., Asghar S., Fong S.: An Efficient Ranking Scheme for Frequent Subgraph Patterns. Proceedings of the 2018 10th International Conference on Machine Learning and Computing, New York, NY, USA, 257–262, 2018.
26. Tao F., Murtagh F., Farid M.: Weighted Association Rule Mining Using Weighted Support and Significant Framework. Proceedings of ACM International Conference on Knowledge Discovery and Data Mining, USA, 2003.
27. Yan X., Han J.: gSpan: graph-based substructure pattern mining. IEEE International Conference on Data Mining Proceedings, pp. 721–724, 2002.
28. Yan X., Han J.: CloseGraph: Mining Closed Frequent Graph Patterns. Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 286–295, 2003.