Structured Graph Decomposition Towards Proficient Exact Subgraph Matching

Aye NweThaing
University of Computer Studeies, Yangon
Ayenwethaing@gmail.com

Abstract

Structures that can be represented as graphs are based on graph theory. Graph databases apply graph theory to store information about the relationships between entries in terms of graphs. The study of graph decomposition has been one of the most important topics in graph theory and also plays an important role in the study of the combinatorics of experimental designs. The main idea of graph decomposing is that matching the smaller graph structure is easier and results in low complexity than matching the original large graph. In this paper, we propose the graph decomposition algorithm based on edge-based representation of the undirected connected graph to obtain its decomposed subgraphs. Finally, we conduct an extensive set of experiments on different type of graphs to demonstrate the efficiency of our approach for further efficient exact subgraph matching.

1. Introduction

Graphs are widely used to model data in various domains such as computer vision and the World Wide Web. In pattern recognition and computer vision, for example, graphs are used to represent hierarchical image features. On the World Wide Web, social networks naturally fit a graph data model. Graph theory is used to study molecules in chemistry and physics. In condensed matter physics, the three dimensional structure of complicated simulated atomic structures can be studied quantitatively by gathering statistics on graph-theoretic properties related to the topology of the atoms. In chemistry, a graph makes a natural model for a molecule, where vertices represent atoms and edges represent bonds. This approach is especially used in computer processing of molecular structures, ranging from chemical editors to database searching.

The development of algorithms to handle graphs is major interest in computer science. Graph decomposition is an important paradigm for many different areas of computer science ranging from graph theoretical to algorithmic applications. To reduce complex computations, it needs to consider building an effective index structure. The aim of the study of graph decompositions is to find tractable subgraphs for the subgraph isomorphism query (ie; exact subgraph matching) in databases and the constraint satisfaction problem in AI. Both these problems are equivalent and well known to be NP-complete. Our basic idea is to break graphs into subgraphs (a small substructure derived from an original graph).

There are many notions of graph decomposition which arise in the literature. Some decompositions involve decomposing a graph using separators of special types (star cutsets or clique cutsets), others involve identification of special sets (substitution of splits), while others involve tree decomposition (treewidth, cliquewidth, branchwidth) or tree composition (Cartesian product, lexicographic product).

These decompositions are fundamental importance for solving optimization and recognition problem on classes of graphs. For example, substitution decomposition is closely related to such problem as solving problems expressible in monadic second logic quantifying over vertices and edges and comparability graph recognition and optimization. Treewidth and its generalizations are of special importance due to the results on tree decomposition and existential proof of existence of algorithms.

Clique cutsets and star cutsets are fundamental tools used in the study of chordal and perfect graphs. Particular tools for working with these decompositions such as partition refinement and lexicographic breadth first search have recently been improved and generalized. The most prominent decomposition method is the tree decomposition of originally developed for graphs and also applicable to hypergraphs.

In this paper, we propose a new approach for graph decomposition for the undirected graphs based on edge-based representation to obtain induced subgraphs. We have to decompose the graph to strongly connected components.

The rest of the paper is organized as follow. Section 2 presents the related work of graph decomposition. Section 3discusses the key concept of graph decomposition comparing other proposed graph index structures. Section 4 discusses about our proposed work. Section 5 describes the algorithm of subgraph decomposition and illustrates the decomposed subgraphs of the given graph. In Section 6, we discuss the experimental result of our proposed subgraph decomposition strategy. Section 7concludes our paper.

2. Related Work

A number of graph decomposition algorithms have been proposed for processing subgraph queries and graph pattern matching. In graph theory, graph decomposition techniques are an instantiation of the divide and conquer model to overcome redundant work for further query processing problems[9][2][5][3].

The directed acyclic graph (DAG) is proposed for the purpose of graph decomposition in [2]. DAG

contains nodes and each node represents the unique, induced subgraphs of the database graphs. This technique is effective for processing dense graphs with labeled edges. The indexing structure using this decomposition allows more compact indexes when the graphs have a high degree of similarity. However, this proposed technique is only applicable to small graphs.

In [9], a hybrid approach of static and dynamic decomposition techniques in the presence of global constraints for solving the subgraph isomorphism problem is proposed. The basic idea is to preprocess a static heuristic on a subset of its constraint network to follow this static ordering until first problem decomposition is available and then switch to a fully propagated dynamically decomposing search. This also exploits the non-predictable reduction of the constraintgraph structure via constraint propagation and entailment but reduces the huge computational effort of a completely propagated search. This decomposition method beats the dedicated state-of-the-art algorithms for sparse graphs with high solution numbers. But this approach still needs to investigate more heuristics for SIP as it influences the quality of decomposition.

Efficient clique decomposition of a graph into its atom graph is proposed in [1]. Clique separator decomposition is useful for a divide and conquer approach for hard problems such as minimum fill-in, maximum clique, graph coloring and maximum independent sets. The process consists of repeatedly finding a clique separator S of a graph G and decomposing G by copying S into the different connected components of G-S, obtaining a set of induced subgraphs having no clique separator called atoms. This paper describes how to organize the atoms resulting from clique minimal separator decomposition into atom graph and give an efficient recursive algorithm to compute this graph at no extra cost than computing the atoms. However, deciding whether a graph is a clique graph or not at lower cost is still an open problem.

A new way for decomposing DAGs into spanning trees to compress transitive closures is proposed in [12]. Due to the very large size of many real world graphs, the computational cost and size of labels using existing methods are too expensive in practical. Therefore, this approach is introduced to decompose a graph into a series of spanning trees that share common edges to transform a reachability query over a graph into a set of queries over trees. Although the proposed method has efficiency and effectiveness over different kinds of graphs, the query time is still bounded by a constant.

3. Key Concepts of Graph Decomposition

In recent years, many efficient indexes have been developed to process subgraph isomorphism queries on graph databases [8] [11] [3] [4]. A subgraph query retrieves all the graphs in the database that are supergraphs of a given query graph. Query processing using these indexes has two main phases. Two phases are

- Filtering and
- candidate verification

First, filtering phase uses the index to eliminate false results and to produce a candidate answer set. Second, candidate verification tests whether each candidate is indeed a supergraph of the query.

As pointed out by the authors of the above-mentioned indexes, the cost of candidate verification is the dominating factor in the cost of processing a subgraph query. Therefore, many researchers have proposed the indexing structures aim at reducing the candidate answer set as much as possible. However, due to the high complexity of subgraph isomorphism testing, candidate verification is still the most expensive part in processing a subgraph query to get the exact answer set.

In this paper, we propose the graph decomposition scheme to construct graph index structure for subgraph query processing that does not require candidate verification. Eliminating of candidate verification is the main concept of our graph decomposition proposal. Our proposed work is designed mainly for processing databases containing smaller graphs such as chemical compound graphs.

4. Proposed Graph Decomposition Technique

Filtering candidate subgraphs for exact subgraph matching in other works[5][10][7][6]consumes more computational time. To avoid the difficulty of cleaning candidate subgraphs for subgraph isomorphism query, new graph decomposition system has been developed that generates all possible subgraphs of the original graph incoming into the graph database based on edge-based processing.

Graph decomposition is the technique of breaking down the graph of interest into smaller parts according to the operations of graph theory. In this paper, we study the concept of graph algebras introduced for this purpose. The graph operations such as union and intersection are simple and useful operations in graph theory. Therefore, the most basic ways of combining graphs are by union and intersection. These operations are associative and commutative and may be extended to an arbitrary number of graphs. The union of two graphs $\boldsymbol{G_1}$ and $\boldsymbol{G_2}$ is the graph ($\boldsymbol{G_1} \cup \boldsymbol{G_2}$) with vertex set ($\boldsymbol{V}(\boldsymbol{G_1}) \cup \boldsymbol{V}(\boldsymbol{G_2})$). The intersection of two graphs is defined analogously. When two graphs are disjoint, their intersection is the null graph.

Moreover, our graph decomposition work is the enumeration of all connected, induced subgraphs of a graph. All edges in $\boldsymbol{G}$ are partitioned into subsets so that each subset is the induced subgraph($\boldsymbol{g}$) of the given graph $\boldsymbol{G}$. The smallest subgraph contains two nodes and one edge exactly.

A graph of size $\boldsymbol{n}$ is decomposed into at most $\boldsymbol{2^n-2}$subgraphs in the case of complete graph containing unique label vertices. If all labels are identical, a complete graph of size $\boldsymbol{n}$ decomposes into $\boldsymbol{n-1}$subgraphs precisely. Figure 1 shows the proposed system

framework. The detail description of graph decomposition architecture is described as follows:

4.1. Preprocessing

In the preprocessing step, the vertex list of the input graph is generated. Every vertex in the graph is assigned with unique ***ID***. After that the edge list of the graph is computed. The edge in the graph is defined as (S_{id},E,D_{id}) where S_{id} is the source vertex ***id***, ***E*** is the edge label and D_{id} is the destination vertex ***id***. The source and destination ***ids*** in each edge are arranged in the ascending order because the graphs in the database are undirected connected graphs.

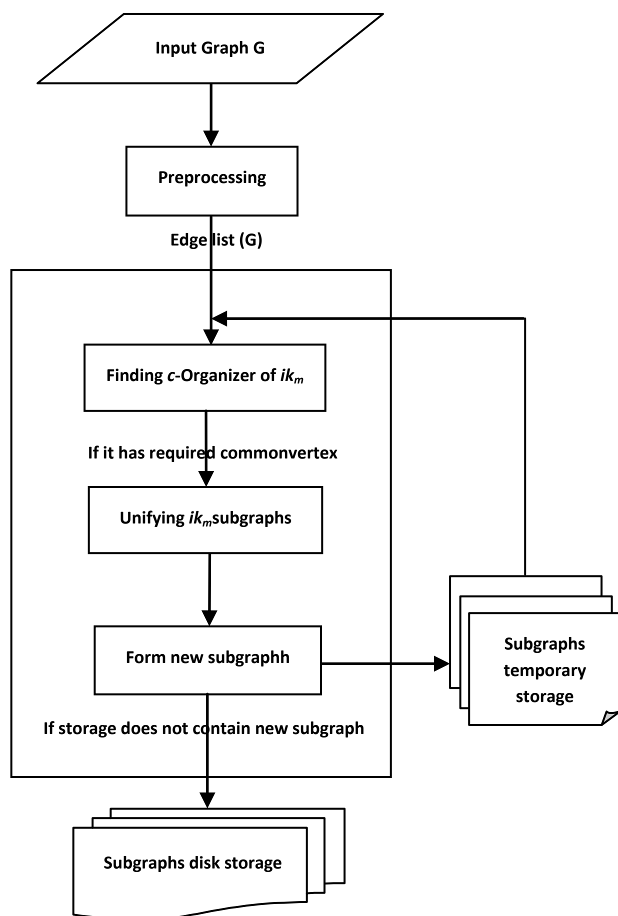


Figure 1. System architecture

4.2. Finding c-organizer Between Subgraphs

In this step, *c*-organizer is determined between two subgraphs. To compute c-organizer, the vertices of two subgraphs are the ids of corresponding vertices. The smallest subgraphs of the original graph are all edges in this graph. The number of subgraphs with their respective vertices is defined as $ik_m$where ***i*** is the number of vertices and ***m*** is the number of subgraphs with***i*** vertices. To compute $3k_m$subgraphs for the given graph, we need to examine whether $2k_2$subgraphs have *c*-organizer to merge these two graphs. In that case, we have some restrictions: to merge $2k_2$subgraphs, *c*-organizer must be '***1***' (i.e; the two subgraphs has 1 common organizer), and to combine $3k_2$subgraphs, the organizer must be '***2***' etc. Thus, *c*-organizer needs (***i-1***) to combine two ik_msubgraphs. The graph (G_C) in figure 2 has the following edges: {<1,s,2>,<2,s,3>,<3,d,4>,<3,s,5>, <5,d,6>, <5,s,7>,<7,s,8>}. Therefore G_C has $2k_7$smallest subgraphs (12, 23, 34, 35, 56, 57, and 78). For demonstration, we use $4k_2$subgraphs to get $5k_1$subgraph of the graph (G_C) shown in figure 2. Figure 3 demonstrates finding *c*-organizer of two subgraphs 2345 and 1235 and tests whether *3*-organizer or not. In that case, depending on the result, the two subgraphs can be combined.

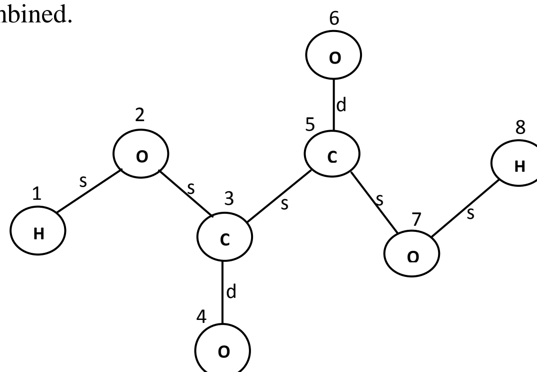
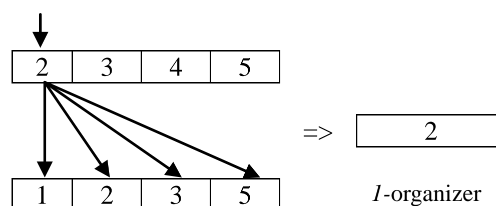
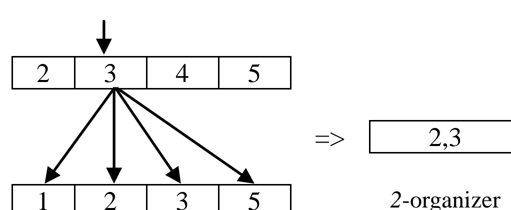


Figure 2. A chemical compound graph (G_C)

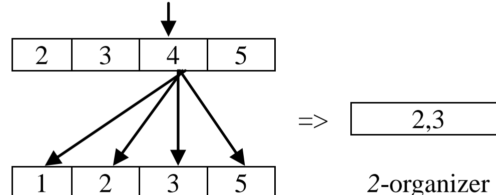
Step.1.



Step.2.



Step.3.



Step.4.

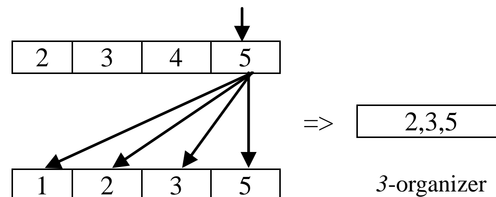


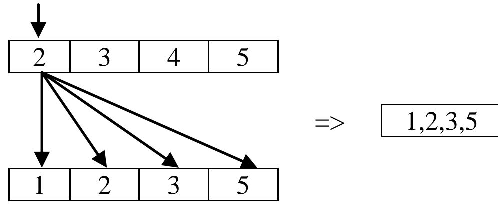
Figure 3.Finding *c-organizer* between two subgraphs

4.3. Forming a New Subgraph by Unifying Two Subgraphs

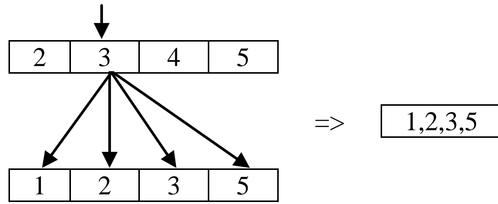
In this step, ik_m subgraphs are merged to form new $(i+1)k_m$ subgraphs depending on the c-organizer in the previous step. Figure 4 shows the unifying between $4k_2$ subgraphs (2345 and 1235) to get a new $5k_1$ subgraph (12345). In the proposed system, two storages are used. The subgraph temporary storage is used to temporarily store all possible ik_m subgraphs of the given graph.Subsequently, ik_m subgraphs in the temporary storage are used to generate $(i+1)k_m$ subgraphs. After that, ik_m subgraphs are cleaned in the temporary storage and $(i+1)k_m$ subgraphs are stored in this storage again.

In the subgraph disk storage, all subgraphs are inserted with their corresponding vertex IDs. If more than one subgraph has the same vertex list, only one subgraph is stored in the subgraph disk storage.

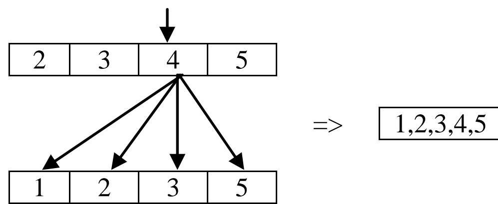
Step.1.



Step.2.



Step.3.



Step.4.

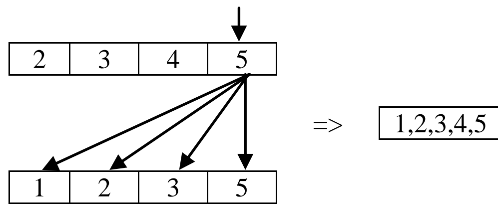


Figure4. Unifying two subgraphs to form new subgraph

5. Subgraph Decomposition Algorithm

In this section, we express the subgraph decomposition algorithm (***SGD***) in figure 5 and GraphMatching algorithm in figure 6 to decompose the given graph by using step-by-step processing described in section IV. Table 1 describes the notations used in the algorithms for our proposed system.

For each graph G_k in the database, ***SGD*** gets the edge list of G_k and each edge of G_k is the smallest subgraph of G_k. Then, these subgraphs are checked using the GraphMatching algorithm to find duplicated subgraph. In GraphMatching algorithm, subgraph represents the ids of vertices contained in thissubgraph. If the subgraph does not already exist in GS, this subgraph is stored in GS. In ***SGD***, to form a new subgraph, c-organizer is computed between two subgraphs if these two subgraphs are not identical. If the subgraphs are duplicated, ***SGD*** stores only one subgraph for further processing.

Figure 7 provides an example of our graph decomposition strategy. For the purpose of illustration, we use the graph G_C to generate all possible induced subgraphs.

Table 1. Notations used in the system

Notation	Definition
GDB	Graph database
G_k	Graph in the GDB
$EL(G_k)$	Edge list of G_k
ik_m	m subgraphs with i vertices
N	no. of vertices in the graph
GS	Graphs storage

Algorithm SubGraphDecomposition (SGD)

Input: GDB $\leftarrow\{G_1,G_2,\ldots,G_n\}$, $EL(G_k)\leftarrow G_k$, EDict
Output: GS
$n=|G_k|$, i=1, j=2
For each $G_k \in$ GDB
  For each e $\in EL(G_k)$
    g:=e
    $ik_m := ik_m + g$
  End for
  GS:=GraphMatching(ik_m)
Return GS

While (n>3)
  For each $g_a \in ik_m$
    For each $g_b \in ik_m$
    If ($g_a \neq g_b$) **then**
    If($|g_a \cap g_b|$) = $|g_a|$-1 **then**
      $g_{ab} := \{g_a\} \cup \{g_b\}$
      $jk_m = jk_m + \{g_{ab}\}$
      End if
    End if
  End for

End for
i:=j
j:=j+1
GS:=GraphMatching(ik_m)
// **End while**
End for
Return GS

Figure 5.Subgraph decomposition algorithm

Algorithm GraphMatching (ik_m)

For each $g_{ab} \in ik_m$
$N_{id}(g_{ab}) \leftarrow \{v_1, v_2, \ldots, v_n\}$
If $N_{id}(g_{ab})$ does not already exist in GS **then**
GS:= GS+ $N_{id}(g_{ab})$
End if
End for
Return GS

Figure 6.Graph matching algorithm

6. Experimental Results

The analysis of the computational time of **SGD** algorithm is described as follows. The algorithm takes $\boldsymbol{m}$ comparisons to generate the edge lists of the graph. The edges of the graph are the smallest induced subgraphs of the given graph. Therefore, the computational time complexity to generate all possible smallest subgraphs is $\boldsymbol{m}$. The algorithm takes $\boldsymbol{m(m-1)}$ comparisons to unify two decomposed subgraphs for ($\boldsymbol{n-3}$) processing steps.

Thus, the total time complexity of our proposed algorithm is $\boldsymbol{O(m+m(m-1)(n-3))}$ in worst case. The space complexity of SGD algorithm is $\boldsymbol{O(2^n-2)}$. Table 2 describes the total time and space complexity of **SGD** using different type of graphs such as sparse, dense and complete graphs.

Table 2. Analysis of computational time and space complexity using SGD

No. of Vertices	No. of Edges	TimeComplexity $O(m+m(m-1)(n-3))$	Space Complexity $O(2^n-2)$
Sparse Graphs			
8	9	369	254
9	11	671	510
10	12	936	1022
Dense Graphs			
8	20	1920	254
9	28	4564	510
10	35	8365	1022
Complete Graphs			
8	28	3808	254
9	36	7596	510
10	45	13905	1022

The study shows that our graph decomposition (**SGD**) is based on the edge-based representation and the computational time complexity varies depending on the different type of graphs such as sparse, dense and complete graphs. Moreover, our method reduces the computational time complexity more efficiently in sparse graph than dense and complete graphs.

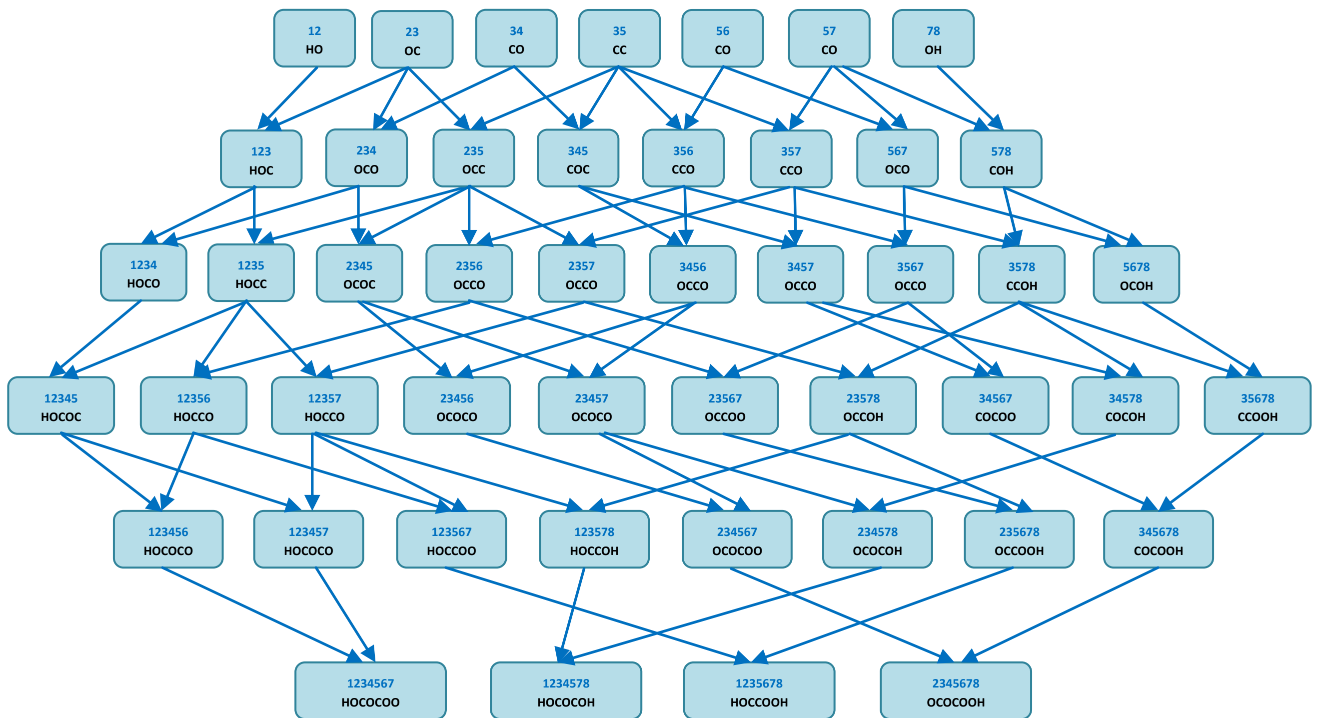


Figure 7.Graph Decomposition using graph (G_c)

To evaluate the performance of our proposed work presented in section 4 and 5, we developed a graph decomposition engine named **SGD**. **SGD** was implemented and run in Java. We tested our graph decomposition engine using different type of graphs such as sparse, dense and completed graphs. Our proposed work is designed mainly for processing databases consisting of large set of smaller graphs. It can be applied effectively in applications such as chemical informatics, protein interaction. We used chemical compound dataset from http://pubchem.ncbi.nlm.nih.gov/ to test **SGD**.

The graph decomposition time was measured for different type of graphs in millisecond. All experiments were made using a 3GHz Intel Core 2 Duo CPU with 1 GB memory and Microsoft Windows XP. Decomposition times were measured while maintaining index in memory.

Figure 8 shows a comparison of graph decomposition time for three types of graphs: sparse, dense and complete graphs. The results are obtained on chemical graph data sets by varying the graph size from 5 to 25. Each graph encodes the structure of a molecule where the vertices are used to represent atoms with edge labels representing bonds. From the empirical analysis, it is found that the time of execution varies for sparse, dense, and complete graphs. This is because our graph decomposition work is based on edge based representation. It takes considerable time for decomposition complete graphs comparable to sparse and dense graphs.

Figure 9 shows the decomposition times for various database sizes. We tested our proposed algorithm using 100 database graphs with an average of 7 vertices for each graph. Most of the graphs in the database are dense graphs. Chemical graphs were obtained randomly from the set of all molecular structures that are represented in the dataset. From our experimental result, it shows that the execution time varies for different number of database graphs. Moreover, our proposed work is edge-based. Hence, the execution time is more effective and efficient for sparse and non-sparse graphs than complete graphs with irrespective of vertices in the graphs.

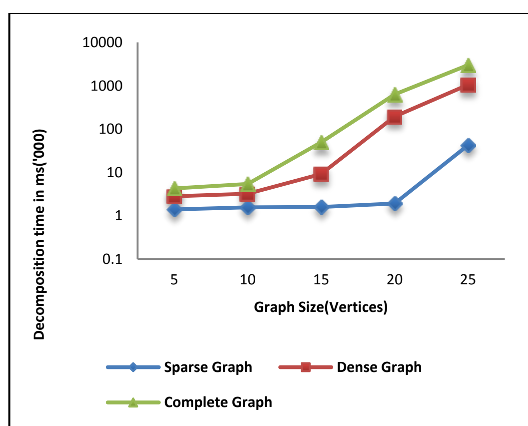


Figure 8.Graph decomposition time for different type of graphs

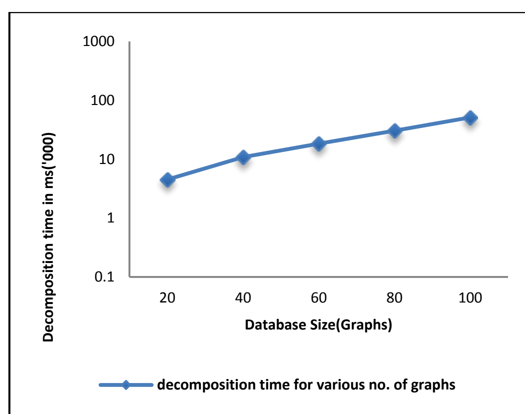


Figure 9.Graph decomposition time for various database sizes

7. Conclusion

Our goal is to find substructures of the given graph for further processing of exact graph matching. In order to restrict the search space, our proposed algorithm SGD considers only connected substructures, i.e., graphs having only one connected component. We have shown how the proposed method decomposes the graph into its subgraphs using chemical structure from the chemical compound database. Our proposed work breaks down the graph based on edge-based representation. For that reason, our proposed effort is more proficient for sparse and dense graphs rather than complete graphs. Consequently, it saves a good amount of search space when it is used in chemical graphs data set. This is because almost all chemical graphs are non-sparse but not complete. Future work will focus on making the presented approach more meaningful for the exact graph matching for subgraph isomorphism query.

References

[1] A. B. R. Pogorelcnik, G. Simonet. “Efficient Clique Decomposition of a graph into its atom graph”, Research Report LIMOS/PR-10-07, 10 March 2010.

[2] D. W. Williams, J. Huan, W. Wang. “Graph Database Indexing Using Structured Graph Decomposition”, 2007.

[3] H.He and A. K. Singh. “Closure-tree: An Index Structure for Graph Queries”. In ICDE. 38, 2006.

[4] H. Jiang, H. Wang, S. Zhou. “Gstring: A Novel Approach for Efficient Search in Graph Databases”. In ICDE. 566-575, 2007.

[5] J. Cheng, Y. Ke, W. NG. “Efficient Query Processing on Graph Databases”. ACM Trans. On Database Systems, Vol. V, No. N, 1-44, Sept 2008.

[6] J. Cheng, Y. Ke, W. NG, A. Lu. “FG-index: Towards Verification-free Query Processing on Graph Databases”. In SIGMOD Conf, 857-872, 2007.

[7] P. Zhao, J. X. Yu, P. S. Yu. “Graph Indexing: Tree+delta>= Graph”. In VLDB, 938-949, 2007.

[8] H. Jiang, H. Wang, S. Zhou. “Gstring: A Novel Approach for Efficient Search in Graph Databases”. In ICDE. 566-575, 2007.

[9] S. Zampelli, M. Mann, Y. Deville, R. Backofen. “Decomposition Techniques for Subgraph Matching”, 2008.

[10] S. Zhang, M. Hu, J. Yang. “Treepi: A Novel Graph Indexing Method”. In ICDE, 966-975, 2007.

[11] Yan, J. Han. “Graph Indexing based on Discriminative Frequent Structure Analysis”. ACM Trans. Database Syst. 30,4, 960-993, 2005.

[12] Y. Chen, Y. Chen2. “Decomposing DAGs into Spanning Trees: A New Way to Compress Transitive Closures”, in Proc. of Int. Conf. on Data Engineering (IDCE 2011), IEEE, 2011.