DUJASE Vol. 7 (1) 58-65, 2022 (January) *https://doi.org/10.3329/dujase.v7i1.62887*

A Vertex-extension based Algorithm for Frequent Pattern Mining from Graph Databases

Md. Tanvir Alam, Chowdhury Farhan Ahmed* and Md. Samiullah

Department of Computer Science and Engineering, University of Dhaka, Bangladesh

**E-mail: farhan@du.ac.bd*

Received on 30 November 2021, Accepted for publication on 17 May 2022

ABSTRACT

Frequent pattern mining is a core problem in data mining. Algorithms for frequent pattern mining have been proposed for itemsets, sequences, and graphs. However, existing graph mining frameworks follow an edge-growth approach to building patterns which limits many applications. Motivated by real-life problems, in this work, we define a novel graph mining framework that incorporates vertex-based extensions along with the edge-growth approach. We also propose an efficient algorithm for mining frequent subgraphs. To deal with the exploding search space, we introduce a canonical labeling technique for isomorphic candidates as well as downward closure property-based search space pruning. We present an experimental analysis of our algorithm on real-life benchmark graph datasets to demonstrate the performance in terms of runtime.

Keywords: Graph mining, Knowledge graphs.

1. Introduction

Frequent pattern mining from a given collection of data is one of the core problems of data mining. Frequent patterns can be used for classification [1-2], clustering, data summarization, outlier analysis, etc. Research has been conducted on developing methods for mining frequent itemsets [3], sequences [4]. Frequent itemset mining algorithms take a collection of unordered itemsets as input and extract itemsets that co-appear frequently. These algorithms are useful for market basket analysis, recommendation systems, etc. Frequent pattern mining algorithms for sequential data such as DNA sequences, protein sequences take a collection of sequences as input and find frequent subsequences from those.

A graph is a data structure that can represent more complex associations than itemsets and sequences. Graphs are useful for encoding information such as web data [5], social network data [6], protein structures, chemical compounds, etc. Analogous to frequent itemset and subsequence mining, algorithms have been proposed for mining frequent subgraphs from a collection of graphs [7-9].

However, the existing frequent pattern mining frameworks have a significant limitation. It builds patterns following the edge-growth approach. A pattern is extended by adding an edge with both vertices already determined. This results in the loss of many interesting patterns and restricts the performance of many applications. To demonstrate the limitation, we can consider a motivational example based on knowledge graphs [10], [11], [12]. A knowledge graph is a special kind of graph where each edge encodes facts by representing a relationship between the data entities represented by the corresponding vertices. For example, in Fig. 1, we present two example knowledge graphs, G_1 and G_2. The knowledge graph G_1 states the facts that *"Pei is an actor who owns a cat"* and *"cat eats fish."*

Similarly, in G_2, the stated facts are *"Han is a clerk who owns a cat"* and *"the cat eats fish"*. In Fig. 2, we present three subgraphs, g_1, g_2, and g_3, from the knowledge graphs presented in Fig. 1. Here, g_1 represents "*Pei owns a cat"*, and g_2 represents "*Han owns a cat*". Now, the subgraph g_3 represents the fact "*Someone owns a cat"*. According to the existing graph mining frameworks, both g_1 and g_2 are considered as patterns mined from the collection of graphs containing G_1 and G_2 but not g_3. However, the fact that someone owns a cat expressed by g_3 is present in both G_1 and G_2. Thus g_3 is more frequent than g_1 and g_2. Ignoring patterns like g_3 can be crucial in many real-life applications. For example, we can use frequent subgraphs as features for fake news article detection by representing the articles as knowledge graphs. In our example, feature patterns like g_3 will increase the probability of authenticity of a new fact that Chen owns a cat which is not possible with feature patterns g_1 or g_2 as they specify the persons to be Han and Pei exactly.

To solve the problem, in this paper, we propose a new flexible framework for frequent subgraph mining. In our framework, we build patterns by adding a vertex or an edge only in each extension. It results in mining a higher number of interesting patterns, as we have described earlier. We also propose a novel algorithm for mining frequent subgraphs according to our modified framework.

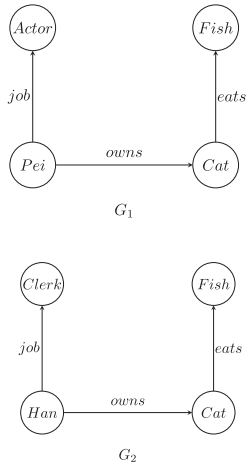

Fig. 1. Knowledge graphs.

The major challenge in graph mining is dealing with the exploding search space. We need to generate candidate patterns in such a way that the number of false candidates is minimized. For each candidate in the search space, we need to check for subgraph isomorphisms in all of the graphs. Subgraph isomorphism checking is an NP-hard problem. A higher number of false candidates involves more subgraph isomorphism checking and slows down the mining process significantly. To deal with this problem, we employ downward closure property and avoid extending any infrequent candidates as all of them will also be infrequent. To skip testing and extending isomorphic candidates, we introduce a canonical labeling technique for all the isomorphism classes, and we prune all the non-canonical candidates. Both of the pruning methods help to reduce the runtime significantly. To prove the efficiency of our proposed algorithm, we have conducted experiments on real-life graph databases.

Our key contributions are to:

- tailor a novel flexible framework for frequent subgraph mining motivated by real-life applications,
- devise a complete algorithm for mining frequent subgraph patterns as defined in our framework,
- develop a canonical labeling technique for pruning isomorphic candidates.
- perform extensive empirical analysis over real-life graph databases.

In the rest of this paper, Section 2 discusses the related works. Then, in Section 3, we present our proposed framework and algorithms. Next, experimental results and analysis are presented in Section 4. Finally, we conclude the paper in Section 5.

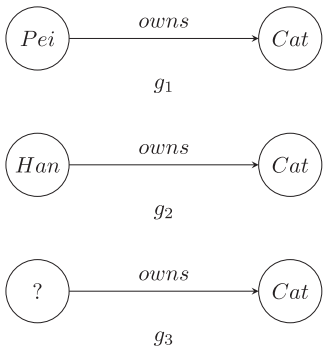

Fig. 2. Example subgraphs.

2. Background and Related Works

Frequent pattern mining problem was first introduced to mine association rules from itemset data [13], [3]. Frequent itemset mining extracts frequent patterns from a set of transactions, $D = \{T_1, T_2, ..., T_n\}$. Let I be the set of all items. A transaction T_i is a subset of I. An example transaction database is shown in Table 1. The database contains four transactions T_1, T_2, T_3, and T_4. Here, $I = \{p, q, r\}$. The transaction T_1 consists of items p, q, and r. Similarly, T_2 consists of p and r. The support of an itemset pattern $P \subseteq I$ in a transaction database D can be defined as,

$$sup(P, D) = \sum_{T \in D} \begin{cases} 1, & \text{if } P \subseteq T \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

For example, the support of the itemset $P = \{p, q\}$ in the database D of Table I is two because P is a subset of transactions T_1 and T_4. Frequent itemset mining from a transaction database D discovers itemsets P, such that $sup(P, D) \geq minsup$ where $minsup = |D| \times \delta$. Here, δ is a user-defined threshold. If we take $\delta = 0.50$, then $minsup = 4 \times 0.50 = 2$ for the example database D. Now, as the support of the pattern $P = \{p, q\}$ is 2, it is a frequent pattern.

Apriori [13] is a frequent itemset mining algorithm following level-wise candidate generation and testing. It uses frequent itemsets with n items to generate candidates containing $n+1$ items. Then candidates with $n + 1$ items are pruned with infrequent itemsets as a subset using the downward closure property. According to the downward closure property, itemsets containing infrequent itemsets as a subset will not be frequent. Finally, the Apriori algorithm finds the frequent patterns by checking the candidates. However, the Apriori algorithm has some significant limitations. The amount of false candidates generated is high, which turns out to be computationally costly. Besides, if the longest frequent pattern contains n items, it requires n database scans. FP-growth [3] has been proposed to address and resolve this problem, requiring only two database scans.

Frequent graph pattern mining algorithms from graph databases mine frequent subgraphs from a collection of graphs. For example, AGM [14] is an Apriori-based frequent subgraph mining algorithm. It generates subgraph candidates of size $k + 1$ from frequent subgraphs of size k. Here, a subgraph of size k contains k vertices. FSG [8] is a similar approach, but it defines the subgraph sizes by the number of edges. Both the algorithms employ downward closure property to prune candidates. However, in both approaches, duplicate isomorphic candidates are generated. This problem has been addressed by gSpan [7] algorithm. Following the edge-growth approach, it starts from an empty subgraph and extends each candidate by adding an edge to it. It represents each candidate subgraph using a sequence of extensions named DFS code. An order is defined among the DFS codes of all the isomorphic forms of a subgraph. By extending only the canonical DFS codes, gSpan avoids testing and extending duplicate isomorphic subgraphs. GASTON [20] is another frequent subgraph mining algorithm that introduces the "quickstart principle". It divides the mining process into multiple phases. The algorithm finds frequent paths first, and then it finds the frequent trees and cyclic graphs. PMFS-IB [21] introduces parallel processing for faster frequent subgraph mining. The algorithm adapts gSpan for large databases with parallel computing. GE-FSG [22] utilizes frequent subgraphs for entire graph classification. However, regardless of the methods, all these existing works define subgraphs or build patterns where all the vertices are determined. As a result, many interesting associations are

ignored in the mining process. The missing patterns due to rigidness in the definition of subgraph can be crucial to many real-life applications such as classification, clustering, and etc.

Table 1: A Transaction Database

Transaction Id	Transaction
T_1	*p, q, r*
T_2	*p, r*
T_3	*q, r*
T_4	*p, q*

3. Proposed Method

Let D be a set of labeled graphs and L be the set of vertex and edge labels. A graph G can be denoted as a 3-tuple, $<V_G, E_G, l_G>$, where V_G is a set of vertices, $E_G \subseteq V$ x V is a set of edges, l_G: $V_G \cup E_G \rightarrow L$ is a function that defines the labels of the vertices and edges. In Fig. 3, we present an example graph database consisting of three graphs G_1, G_2 and G_3. All the vertices and edges are labeled. For example, the vertex v_1 in G_1 is labeled as *"a"* and the edge e_1 is labeled as *"q"* respectively. Let $g = <V_g, E_g, l_g>$ be a subgraph where V_g is a set of vertices, $E_g \subseteq V \cup V$ x V is a set of edges, l_g:$V_g \cup E_G \rightarrow L$ is the labeling function. A subgraph isomorphism from g to a graph G holds if there exists a function ϕ: $V_g \cup E_g \rightarrow V_G \cup E_G$, such that,

- $\forall v \in V_g$, $l_g(v) = l_G(\phi(v))$.
- $\forall e \in E_g$, $l_g(e) = l_G(\phi(e))$.
- $\forall e \in E_h$, $\{\phi(v) : v \in e\} \subseteq \phi(e)$.

Table 2: Subgraph Isomorphisms

Vertex/Edge	***ϕ_1(in G_1)***	***ϕ_2(in G_2)***
v_1	v_1	v_1
v_2	v_2	v_2
e_1	e_1	e_1
e_2	e_4	e_2

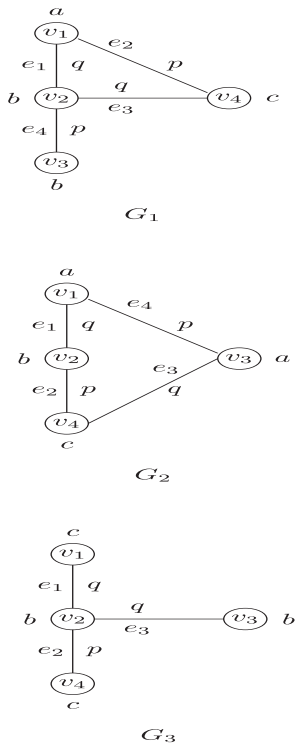

Fig. 3. A graph database D

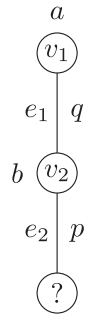

Fig. 4. A subgraph g

We present an example subgraph in Fig. 4. Here, the edge e_1 is labeled as q. The other edge, e_2 represents an association between v_2 and any other vertex where the edge label is p. Table 2 presents the subgraph isomorphism from the subgraph g of Fig. 4 to the graphs in the database D of Fig. 3. ϕ_1 represents a subgraph isomorphism from g to G_1 and ϕ_2 represents a subgraph isomorphism from g to G_2. For G_3, there is no subgraph isomorphism from g.

Let us denote the set of all subgraph isomorphism from a subgraph g to a graph G as $\Phi(g, G)$. Now, we can define the frequency support of g in a G as,

$$sup(g, G) = \begin{cases} 1, & \text{if } |\Phi(g, G)| \geq 1 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

In database D of Fig. 3, we find a subgraph isomorphism from the subgraph g of Fig. 4 to graph G_1, as shown in Table 1. So, the frequency support of g in G_1, $sup(g, G_1) = 1$. For the same reason, $sup(g, G_2) = 1$. However, $sup(g, G_3) = 0$ as, from g to G_3, we find no subgraph isomorphism. We define the frequency support of a subgraph g in a graph database D as,

$$sup(g, D) = \sum_{G \in D} sup(g, G) \quad (3)$$

In our example database D of Fig. 3, we calculate the frequency support of the subgraph g of Fig. 4 as, $sup(g, D) = sup(g, G_1) + sup(g, G_2) + sup(g, G_3) = 1 + 1 + 0 = 2$.

Frequent subgraph Mining: Given a graph database D and a frequency threshold δ, frequent subgraph mining extracts the subgraphs g where $sup(g, D) >= minsup$. Here, $minsup = |D| \times \delta$. For database D of Fig. 3, let $\delta = \frac{2}{3}$. Now, $minsup = \frac{2}{3} \times 3 = 2$. We find that $sup(g, D) = 2$. That is $sup(g, D) >= minsup$. So, g is a frequent subgraph in D.

Now, we present our proposed algorithm for the frequent subgraph mining problem as defined in Section 2. Our objective is to mine patterns so that no frequent subgraph is missed and the number of candidates is minimized. Our algorithm starts from an empty subgraph candidate and extends the candidates in each step like a depth-first search instead of Apriori [13] based level-wise candidate generation. We assign all the vertices and edges in a candidate subgraph with a unique identification number following the order of them being included in the corresponding candidate.

For example, let the vertex added last to the candidate g be $last_v(g)$. Similarly, let $last_e(g)$ be the edge that has been added last. In our algorithm, we extend each candidate by employing any of the following strategies:

- **Vertex-append:** Adding an existing or new vertex to the edge that has been added last.
- **Edge-append:** Adding a new edge with precisely one existing vertex.

Extending candidates in this manner reduces the search space compared to the brute force approach. However, still, many isomorphic candidates will be generated. Testing and extending these isomorphic candidates is redundant and costly. We propose a canonical labeling technique for isomorphic candidates to deal with this challenge. The idea is to define an order among all the isomorphic forms of a candidate subgraph. Whenever a candidate is generated, we can determine if it is the minimum one among all the isomorphic forms of the candidate according to the defined order. If it is the minimum one, we call it a canonical candidate. Our algorithm only avoids redundant and costly subgraph isomorphism checking by testing and extending the canonical candidates only.

Now, we formally define our proposed canonical labeling technique. We represent each subgraph candidate as an ordered sequence of extensions. Let us denote an extension using a tuple <type, vertex, vertex label, edge label>. For example, a Vertex-append extension by appending a vertex with identification number 2 and label b is represented as $<v, 2, b, ->$. An Edge-append extension that adds an edge with label p containing a vertex with identification number 1 and label c is denoted as $<e, 1, c, p>$. Let $ext_1 = <t_1, d_1, l_{v1}, l_{e1}>$ and $ext_2 = <t_2, d_2, l_{v2}, l_{e2}>$ be two extensions. We define the order among extensions such that $ext_1 < ext_2$ if and only if one of the following holds true.

- $t_1 = v, t_2 = e.$
- $t_1 = v, t, = v, d_1 < d_2.$
- $t_1 = v, t_2 = v, d_1 = d_2, l_{v1} < l_{v2}.$
- $t_1 = e, t_2 = e, d_1 < d_2.$
- $t_1 = e, t_2 = e, d_1 = d_2, l_{e1} < l_{e2}.$
- $t_1 = e, t_2 = e, d_1 = d_2, l_{e1} = l_{e2}, l_{v1} < l_{v2}.$

Finally, by comparing extension tuples, we define the order among candidate subgraphs, and the minimum candidate, according to the order, is canonical. Besides canonical-labeling-based pruning, our algorithm also utilizes the downward closure property of frequent patterns. According to the property, patterns extended from an infrequent pattern will always be infrequent. By using this property, our algorithm only extends frequent candidates.

In Algorithm 3, we present the pseudocode of our frequent subgraph mining algorithm. We find all the possible extensions of a given candidate first (line 2). Then, for each extension in Algorithm 3, we check if the extended candidate is canonical or not and if it is a frequent pattern (line 5).

Given that both conditions are satisfied, we output the candidate as a frequent pattern and mine further extended patterns recursively (lines 6-7). The procedure for finding the extensions is presented in Algorithm 1. It takes a candidate subgraph and the graph database as input. All the graphs in the database are considered to generate extensions (line 3). If the candidate subgraph is empty, all the vertices in the edges are considered, and an edge-append extension is generated for the vertex in the edge (lines 4-7). Otherwise, all the subgraph isomorphism are considered to find extensions (lines 9-24). The pseudocode for determining whether a candidate is canonical or not is shown in Algorithm 2. It starts with an empty subgraph and keeps extending it in the input graph. If any sequence of extension tuple representation is found lower according to the defined order, it outputs the candidate to be non-canonical.

Runtime Complexity Analysis: In our algorithm, for each

candidates generated, the support count and canonicity is checked, and the set of possible extensions is found. Let N_v and N_e be the highest number of vertices and edges in any graph in the database. For finding the support count, we need to find subgraph isomorphisms from the candidate to each graph in the database, which costs O(|D|(N_v + N_e)^(N_v + N_e)). Now, the cost of each extension finding process is O(N_e(N_v + N_e)^(N_v + N_e)) as all the edges are considered for all the subgraph isomorphisms (Algorithm 1, line 9-24). For checking canonicity, we need to find extensions in the candidate N_v + N_e times (Algorithm 2, line 4) which costs O(N_e(N_v + N_e)^(N_v + N_e + 1)). Let N_c be the number of candidates generated. So, the overall runtime complexity of the algorithm is O(N_c|D|(N_v + N_e)^(N_v + N_e) + $N_c$$N_e$($N_v$ + N_e)^(N_v + N_e + 1)).

Algorithm 1: Find Extensions

```
Input  : g : a candidate subgraph,
         D: a graph database
Output : E: set of all possible extensions
1  begin
2      E ← ∅;
3      for G ∈ D do
4          if g = ∅ then
5              for (u, v) ∈ E_G do
6                  E ← E ∪ {<e, 0, l(u), l(u,v)>};
7                  E ← E ∪ {<e, 0, l(v), l(u,v)>};
8          else
9              for φ ∈ Φ(g, G) do
10                 for (u,v) ∈ E_G do
11                     if (u,v) ∉ φ^-1 and then
12                         if u ∈ φ^-1 then
13                             E ← E ∪ {<e, discovery(φ^-1(u)), l(u), l(u,v)>};
14                         if v ∈ φ^-1 then
15                             E ← E ∪ {<e, discovery(φ^-1(v)), l(v), l(u,v)>};
16                 for (u,v) ∈ φ(last_e(g)) do
17                     if u ∉ φ^-1 then
18                         E ← E ∪ {<v, discovery(last_v(g)) + 1, l(u), ->};
19                     else if φ^-1(u) ∉ last_e(g) then
20                         E ← E ∪ {<e, discovery(φ^-1(u)), l(u), ->};
21                     if v ∉ φ^-1 then
22                         E ← E ∪ {<e, discovery(last_v(g)) + 1, l(v), ->};
23                     else if φ^-1(v) ∉ last_e(g) then
24                         E ← E ∪ {<e, discovery(φ^-1(v)), l(v), ->};
25     return E;
```

Algorithm 2: Check Canonicity

```
Input  : g : a candidate graph in sequence of extension tuples form
1  begin
2      g_t ← ∅;
3      for i ← 1 to |g| do
4          E ← FindExtensions(g_t, {g});
5          if g[i] ≠ min(E) then
6              return False;
7          else
8              g_t.insert(g[i]);
9      return True;
```

Algorithm 3: Frequent Subgraph Mining

```
Input  : g : a candidate subgraph,
         D: a graph database,
         minsup: frequency support threshold // Initially g is an empty subgraph
1  begin
2      E ← FindExtensions(g, D);
3      for ext ∈ E do
4          g_t ← extend(g, ext);
5          if CheckCanonicity(g_t)= true and sup(g_t, D) ≥ minsup then
6              Output g_t as frequent pattern;
7              FrequentSubgraphMining(g_t, D, minsup);
```

4. Experiments

In this section, we present the experiment details that we have performed to demonstrate the efficiency of our proposed algorithm. Next, section 4.1 contains the details of the datasets we have used. Finally, Section 4.2 presents the experimental setup and performance analysis.

4.1 Dataset Description

In our experiments, we have used six real-life graph datasets namely Nci1 [15], Enzymes [16], IMDB-B [17], Mutag [18], Proteins [16], and PTC [19]. D&D, Enzymes, and Proteins are protein structure graph datasets. IMDB-B is a graph dataset from the social network domain. Mutag and PTC contain chemical compounds presented as graphs. In Table 3, we present the statistical description of the databases.

Table 3: Statistical Description of Datasets

Dataset	No. of graphs	Average no. of vertices	Average no. of edges
NCi1	4,110	29.87	32.30
Enzymes	600	32.63	62.14
IMDB-B	1,000	19.77	96.53
Mutag	188	17.93	19.79
Proteins	1,113	39.06	72.82
PTC	344	14.29	14.69

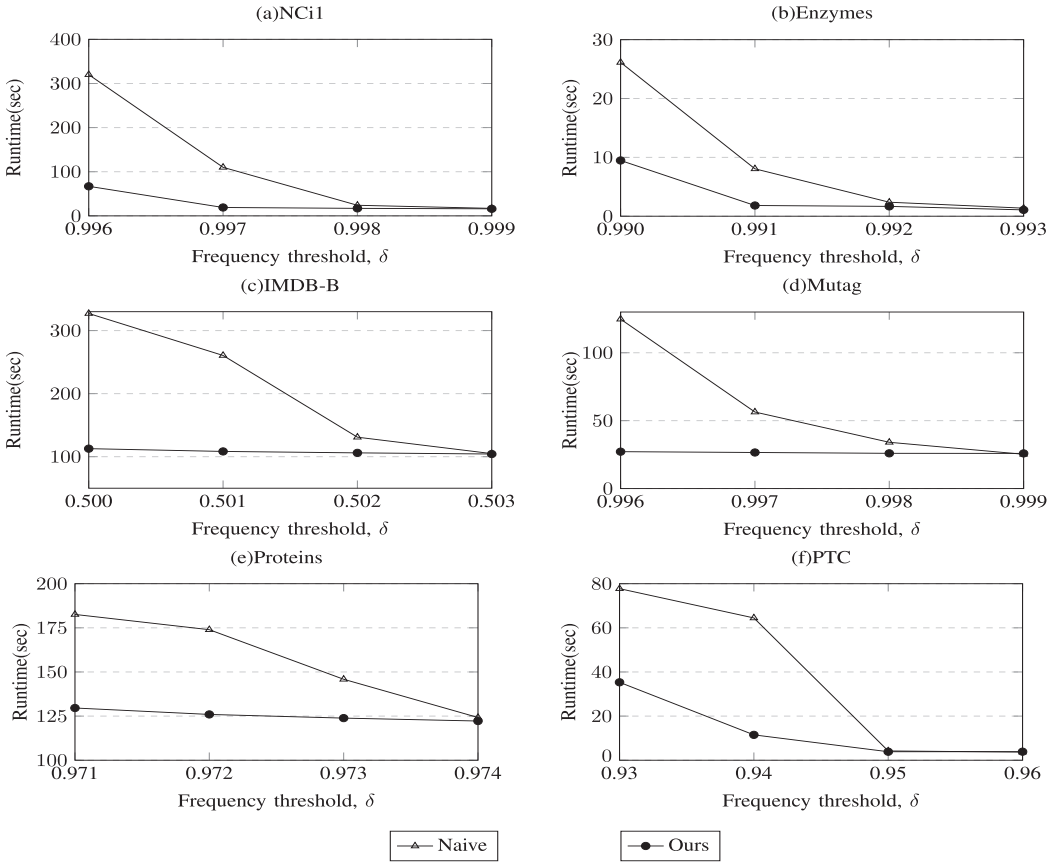

Fig. 5. Runtime Analysis.

4.2 Results and Discussions

Here we present the experiment settings and presents the performance analysis. To prove the efficiency of our algorithm, we have run our algorithm on real-life datasets as presented in Section 4.1. We have implemented our algorithm using the Python 3.7 programming language. We have run all the experiments on an Intel Core i7-6700k CPU @ 4.00GHz with 16GB RAM. Due to differences in the definition of the subgraph, we could not perform a comparative analysis against existing graph mining algorithms. So, as a baseline, we have taken a relatively naive version of our proposed algorithm that does not prune the search space using canonical labeling.

Instead, we consider runtime and the number of candidates as performance metrics. The number of candidates generated indicates the pruning capability of the algorithm. The fewer the candidates generated, the faster the mining will be.

In Fig. 5, we show the runtime analysis. We present our algorithm's runtime and that of the naive algorithm on all the six datasets we have collected. We have reported the runtime for different frequency thresholds. We observe that our algorithm with canonical labeling technique mines frequent patterns faster. For example, on dataset Nci1, from Fig. 5-a, we observe that the runtime of our algorithm is 67.87 seconds with $\delta = 0.996$, whereas the runtime without pruning is 320.34 seconds. That is, without pruning, it took 4.71 times more runtime. The reason behind lower runtime is that duplicate isomorphic candidates are not processed in our algorithm. The difference in runtime is more significant for lower frequency thresholds. For example, on dataset Nci1 (Fig. 5-a), the difference in runtime for $\delta = 0.999$ is only 1.18 seconds whereas for $\delta = 0.996$, it is 252.47 seconds. Note that pattern mining with a lower threshold is more challenging for any kind of pattern. We can also observe that the runtime increases with decreasing frequency threshold. For example, on dataset Nci1, the runtime rises from 16.21 seconds to 67.87 seconds from $\delta = 0.999$ to $\delta = 0.996$. As the frequency threshold decreases, more patterns are mined, and the search space gets larger. As a result, the runtime increases. Our proposed canonical labeling technique results in faster mining by avoiding extending duplicate isomorphic candidates.

5. Conclusion

This paper proposes a new framework for mining frequent subgraphs motivated by real-life problems. We devise an efficient algorithm that mines frequent subgraphs from graph databases defined in our proposed framework. To deal with isomorphic candidates, we employ a canonical labeling technique in our algorithm. By defining a representative of the whole isomorphic class of a candidate, called the canonical candidate, we avoid testing and extending any duplicate isomorphic subgraphs. We have conducted experiments to demonstrate the effectiveness and efficiency of our algorithm on real-life databases. Significantly reduced runtime and search space prove the effectiveness of our canonical labeling technique. We can consider incorporating weight into the new framework and introducing parallel processing for faster mining as a future research direction.

Acknowledgement

This work is funded by the centennial research grant of the University of Dhaka.

References

1. H. Cheng, X. Yan, J. Han, and C.-W. Hsu, "Discriminative frequent pattern analysis for effective classification," in 2007 IEEE 23rd international conference on data engineering. IEEE, pp. 716–725, 2007.
2. M. T. Alam, C. F. Ahmed, M. Samiullah, and C. K. Leung, "Discriminating frequent pattern based supervised graph embedding for classification," in Advances in Knowledge Discovery and Data Mining, pp. 16–28, 2021.
3. J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," vol. 29, no. 2. ACM, pp. 1–12, 2000.
4. J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Prefixspan: Mining sequential patterns efficiently by prefixprojected pattern growth," in ICDE. IEEE, 2001.
5. R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tompkins, and E. Upfal, "The web as a graph," in Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1–10, 2000,
6. L. Tang and H. Liu, "Graph mining applications to social network analysis," in Managing and Mining Graph Data. Springer, pp. 487–513, 2010.
7. X. Yan and J. Han, "gspan: graph-based substructure pattern mining," in ICDM, 2002.
8. M. Kuramochi and G. Karypis, "Frequent subgraph discovery," in ICDM. IEEE, 2001.
9. M. T. Alam, C. F. Ahmed, M. Samiullah, and C. K. Leung, "Mining frequent patterns from hypergraph databases," in Advances in Knowledge Discovery and Data Mining, pp. 3–15, 2021.
10. Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," IEEE Transactions on Knowledge and Data Engineering, vol. 29, no. 12, pp. 2724–2743, 2017.
11. J. Pujara, H. Miao, L. Getoor, and W. Cohen, "Knowledge graph identification," in International Semantic Web Conference. Springer, pp. 542–557, 2013.
12. X. Chen, S. Jia, and Y. Xiang, "A review: Knowledge reasoning over knowledge graph," Expert Systems with Applications, vol. 141, p. 112948, 2020.
13. R. Srikant, Q. Vu, and R. Agrawal, "Mining association rules with item constraints." in KDD, vol. 97, pp. 67–73, 1997.
14. A. Inokuchi, T. Washio, and H. Motoda, "An apriori-based algorithm for mining frequent substructures from graph data," in European conference on principles of data mining and knowledge discovery. Springer, 2000.
15. N. Wale, I. A. Watson, and G. Karypis, "Comparison of descriptor spaces for chemical compound retrieval and classification," Knowledge and Information Systems, vol. 14, no. 3, pp. 347–375, 2008.
16. K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel, "Protein function prediction via graph kernels," Bioinformatics, vol. 21, no. suppl_1, pp. i47–i56, 2005.
17. P. Yanardag and S. Vishwanathan, "Deep graph kernels," in ACM SIGKDD, pp. 1365–1374, 2015.
18. A. K. Debnath, R. L. Lopez de Compadre, G. Debnath, A. J. Shusterman, and C. Hansch, "Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity," Journal of medicinal chemistry, vol. 34, no. 2, pp. 786–797, 1991.

19. H. Toivonen, A. Srinivasan, R. D. King, S. Kramer, and C. Helma, "Statistical evaluation of the predictive toxicology challenge 2000–2001," Bioinformatics, vol. 19, no. 10, pp. 1183–1193, 2003.

20. S. Nijssen and J. N. Kok, "A quickstart in frequent structure mining can make a difference," in Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 647–652, 2004.

21. B. Vo, D. Nguyen, and T.-L. Nguyen, "A parallel algorithm for frequent subgraph mining," in Advanced Computational Methods for Knowledge Engineering. Springer, pp. 163–173, 2015.

22. D. Nguyen, W. Luo, T. D. Nguyen, S. Venkatesh, and D. Phung, "Learning graph representation via frequent subgraphs," in Proceedings of the 2018 SIAM International Conference on Data Mining. SIAM, pp. 306–314, 2018.