WeFreS: Weighted Frequent Subgraph Mining in a Single Large Graph

Nahian Ashraf[1], Riddho Ridwanul Haque[1], Md. Ashraful Islam[1], Chowdhury Farhan Ahmed[1], Carson K. Leung[2][0000−0002−7541−9127](✉), Jiaxing Jason Mai[2], and Bryan H. Wodi[2]

[1] University of Dhaka, Dhaka, Bangladesh
farhan@du.ac.bd
[2] University of Manitoba, Winnipeg, MB, Canada
kleung@cs.umanitoba.ca

Abstract. Considering edge weights during frequent subgraph mining can help us discover more interesting and useful subgraph patterns when compared to its unweighted counterparts. Although some recent works have proposed weight adaptation in frequent subgraph mining from transactional graph databases, the consideration of edge-weights in mining subgraph patterns from single large graphs is mostly unexplored. However, such graph structures appear frequently, with instances being found in social networks, citation and collaboration graphs, chemical and biological networks, etc. In this paper, we propose *WeFreS*, an efficient algorithm for mining weighted frequent subgraphs in edge-weighted single large graphs. *WeFreS* takes into consideration the weight, or significance of the interactions between different types of entities, and only outputs subgraphs whose weighted support is greater than a given user-defined threshold. The resulting subgraph patterns are both frequent and significant from the application perspective. Moreover, for efficiency, *WeFreS* is also equipped with various pruning techniques and optimizations.

Keywords: Single large graph · Weighted single large graph · Graph mining · Weighted frequent subgraph mining.

1 Introduction

Identifying frequently appearing patterns in large databases is an important domain of data mining [16]. In the modern world, graphs are being increasingly used to model data obtained from various real-life applications [2, 4, 8, 12, 14, 15, 18]. Weighted graphs have even more representational power than unweighted ones, and allow users to specify the relative significance of various edge-relations in the graph. Mining frequent subgraph patterns from weighted graph data can thus enable us to gain useful insights about the features and the nature of the data around us.

Graph mining approaches have traditionally focused on two different setups: (i) transactional graph database (which is viewed as a collection of small graphs)

and (ii) single large graph framework (which represents the entire dataset in a single graph). Although several approaches have been proposed for weighted frequent subgraph mining from transactional graph databases [9, 10], there exists a scarcity of efficient approaches addressing the same problem in the context of single large graphs. However, the single large graph representation is inevitable for many fields such as analyzing molecular fragments, image processing, software bug detection, text classification and social network analysis [6, 11]. Thus, considering edge weights during frequent subgraph mining in single large graphs can help us mine important subgraph patterns, which can be used in a variety of different applications [1, 12].

Consider the case of mining patterns of spam dispersion in a social network [7], in which the number of spammers is relatively low. Unweighted graph mining approaches will fail to mine patterns involving spammers and spam dispersion, for not being able to prioritize edge relations that include spammers. A weighted frequent subgraph mining approach, however, would do so if heavier weights were assigned to edges involving spammers. Such an approach could thus lead us to finding frequent patterns of spam dispersion across a community.

Recent literature involving single large graphs use *minimum image-based index* (*MNI*) [3] as the frequency support of a subgraph in a given large graph. Using the MNI measure, weighted support of a subgraph is defined as the product of the average of its edge weights and its MNI value.

In the current paper, we propose an algorithm that takes an edge-weighted single large graph as input and outputs all subgraphs whose weighted support satisfies a given user-defined threshold. Here, we consider graphs, where edge-weights are defined as a function of the labels of the nodes adjacent to an edge. This is a non-trivial task due to the absence of the Apriori property in weighted frequent subgraph mining. Traditional weighted pattern mining approaches [20] avoid extending a pattern if its support multiplied by $MaxW$ (highest weight value among all items) is less than the given threshold. Theoretically, extending the current subgraph with infinite edges can make the average weight at most $MaxW$. However, this is generally a rather high over-estimation, which leads to unacceptably high runtimes—especially when working on graphs having highly varying edge-weights.

Further challenges are caused by the computational difficulty of determining the exact value of the MNI. To avoid this costly operation, a *constraint satisfaction problem* (*CSP*) *model* has been applied to determine if the MNI of a subgraph is at least a given threshold (instead of determining its exact value) [5, 17]. During the mining process, the user defines a weighted support threshold value (instead of explicitly defining the required MNI). Our proposed algorithm, ***We****ighted* ***Fre****quent* ***S****ubgraph Miner* (*WeFreS*), efficiently overcomes these challenges by introducing the $MaxPosW$ measure, which is a tight upper bound of the highest weight value a subgraph can attain after being extended by one or more remaining edges from the given large graph. We also introduce a redesigned CSP model for the frequency evaluation of a subgraph in an edge-weighted framework. *Key contributions* of this paper include the following:

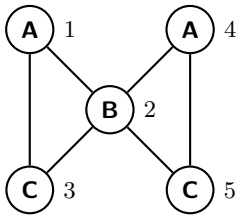

Fig. 1. An input graph G

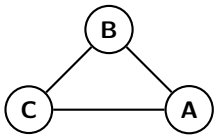

Fig. 2. A sample subgraph S

- Introduction of an efficient method to counter the absence of the *Apriori* property during weighted frequent subgraph mining.
- Proposal of an efficient technique for determining *MaxPosW* (i.e., maximum possible weight a subgraph can attain after being extended by one or more edges from the large graph).
- Reformulation of the *CSP* model so that it can fit in a weighted framework.
- Development of *WeFreS*, a weighted frequent subgraph mining algorithm that works on edge-weighted single large graphs.
- Demonstration of the efficiency of *WeFreS* in comparison with existing approaches and baseline algorithms using results obtained from experiments on several real-world datasets.

The remainder of this paper is organized as follows. The next section gives the formal problem definition and an overview of some related works. Section 3 presents our proposed method and relevant proofs. Evaluation results and conclusions are given in Sections 4 and 5, respectively.

2 Preliminary Concepts and Related Works

Definition 1. *For a* **weighted single large graph** *described by a five tuple* (V, E, L, l, w)*, (i)* V *and* E *are sets of vertices and edges, respectively, such that all* $e \equiv (u, v) \in E$ *where* $u, v \in V$ *and* e *connects nodes* u *and* v*; (ii)* L *is a set of node labels; (iii)* $l{:}V \rightarrow L$ *is a function that maps each node to a certain label; and (iv) the weight of each* $e \in E$ *is defined by the function* $w{:}(LXL) \rightarrow R$ *as* $w(l(u), l(v))$*.*

Definition 2. *For any subgraph* S *with vertex set* V_S*, let* $f(v)$ *denote the number of distinct nodes in an input graph* G *with a node* $v \in V_S$ *that can be mapped to in order to form at least one valid isomorphism. The* **minimum image-based (MNI) index** *of a subgraph* S *is* $\forall v \in V_S, \min(f(v))$*.*

For example, subgraph S in Fig. 2 has two isomorphisms in graph G in Fig. 1: (i) {1-2-3} and (ii) {4-2-5}. Here, nodes A and C in S can be mapped to two distinct nodes each to form valid isomorphisms, but node B can be mapped only to node 2 in G. Consequently, the MNI support of S in G is $\min(2, 1, 2) = 1$.

While alternative metrics exist for determining the frequency support of a subgraph in a single large graph, we use the MNI index as the support metric due to (i) the relative computational ease of determining its value and (ii) the mined subgraphs are supersets of those mined by other metrics. MNI is used as the support metric in several recent literature as well [5, 17].

Definition 3. *The* **weight $W(S)$ of a subgraph S** *is the average of the weights of the edges in it. The weighted support $WS(S)$ of S is the product of $W(S)$ and its MNI support $MNI(S)$, i.e., $WS(S) = W(S) \times MNI(S)$.*

For example, in Fig. 2, if the weights of edges connecting labels (A-B), (B-C) and (C-A) in S are 5, 10 and 15 respectively, then $W(S) = \frac{5+10+15}{3} = 10$ and $WS(S) = W(S) \times MNI(S) = 10 \times 1 = 10$. Weighted support is suitable for finding interesting patterns because it takes into account both the MNI support and weight (instead of only the MNI value).

Definition 4. *Given a weighted single large graph G and a threshold τ as input, the* **weighted frequent subgraph mining** *problem is defined as finding all subgraphs S such that $WS(S) \geq \tau$.*

In the context of relevant literature, gSpan [19] mines frequent subgraphs from transactional graph databases. Concepts introduced in gSpan regarding the canonical ordering of edges and subgraphs have been adopted in previous single large graph mining approaches [5, 17] and are used in *WeFreS* to avoid duplicate subgraph generation. GraMi [5] is a state-of-the-art approach for mining frequent subgraphs from single large graphs, with a CSP model to determine if the MNI of a subgraph satisfies a given threshold. GraMi does not take edge-weights into consideration, and thus risks mining subgraph patterns which are frequent but ultimately insignificant. It also misses out on mining subgraph patterns that are relatively less frequent, but nevertheless interesting due to higher weight values.

ReSuM [17] takes edge-weights into consideration and can mine weighted frequent subgraphs when the weights are within the interval [0,1]. ReSuM uses GraMi to identify all frequent subgraphs, and then filters out subgraphs from the output whose weighted support do not satisfy the given threshold. Since the weight of each subgraph is imposed to be within [0, 1], the weighted support is bounded by the MNI support, thus ensuring a complete search. Experiments in Section 4 show this approach to be inefficient compared to *WeFreS*. Furthermore, imposing weights to be within [0, 1] limits the representational power of the graph. As exemplified by the case of spam dispersion in Section 1, many applications require the weighted support of certain patterns to be scaled up from their MNI support, and that is not possible if edge-weights are restricted to being less than 1.

3 Our Proposed WeFreS Algorithm

Mining weighted single large graphs imposes several challenges. First, how do we determine if there is a possibility of an extension of a given subgraph to be

weighted frequent? Even though the MNI index of a subgraph maintains the *Apriori* property, the weight of the extensions of a subgraph may exceed its own weight. This paper introduces a novel approach for determining the **Max**imum **Pos**sible **W**eight (i.e., *MaxPosW*-measure) whether a subgraph can attain after being extended by one or more edges. Once the *MaxPosW* for a subgraph S is determined, we define $reqExt = \lceil \tau / MaxPosW \rceil$ as the minimum MNI value that S must have in order for an extension of S to be weighted frequent. Let (i) $W(S)$ the weight of the weighted frequent subgraph S, we define $reqFreq = \lceil \tau / W \rceil$ as the minimum required MNI value.

We begin extending subgraphs after deleting nodes and edges having 'infertile' labels from the large graph. Thus, the steps that are needed to be defined to fully express the mining process are as follows:

- Defining a method to determine *MaxPosW* for a subgraph S.
- Defining a method to determine if the MNI support of a subgraph S is at least *reqExt*.
- Defining a method to determine if the MNI support of a subgraph S is at least *reqFreq*.
- Defining a method to determine if a node or an edge is *infertile*.

Detailed descriptions of all these methods along with necessary mathematical proofs and illustrations are provided in the remainder of this section.

3.1 Calculation of MaxPosW

WeFreS works by (i) initiating a subgraph for each undeleted edge in the input graph and (ii) evaluating its weighted support to determine if it is weighted frequent itself and if it is extendable. Each extendable graph is then recursively extended by adding a single-edge to it, to form all canonical 1-edge extensions of the current subgraph, whose weighted supports are subsequently evaluated. Thus, the entire search space of *WeFreS* can be viewed as a collection of *DFS code trees*, each node of which represents a subgraph whose weighted support was evaluated. *WeFreS* functions by executing a depth first traversal of the tree.

During this traversal, each time we extend a subgraph with an edge, the weight calculations are effected. If there are $|G|$ edges in the entire graph and $|g|$ edges in the subgraph g, we can extend subgraph g by at most $rem(g) = |G| - |g|$ edges. The maximum possible weight an existing subgraph can attain after being extended by exactly i-edges is equal to the weight attained after being extended by the i-edges with the highest edge weights in $rem(g)$ and can be calculated as:

$$h_i(g) = \frac{cur\_sum(g) + sum_i(g)}{|g| + i} \tag{1}$$

where (i) *cur_sum* gives the summation of edge weights of g and (ii) sum_i returns weight summation of remaining i highest weighted edges in $rem(g)$. *MaxPosW(g)* can be calculated by taking the maximum of all $h_i(g)$ measures:

$$MaxPosW(g) = \max\{h_i(g) : 1 \leq i \leq rem(g)\} \tag{2}$$

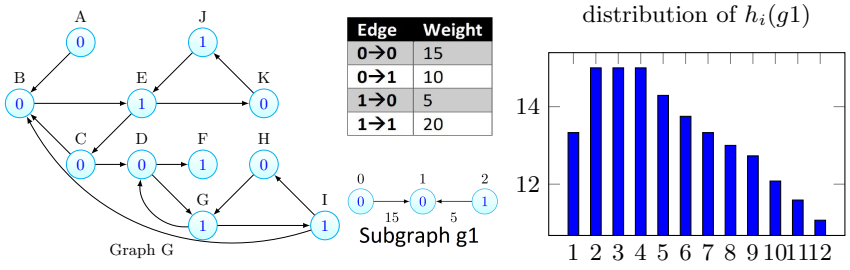

Fig. 3. A sample single large graph G, a subgraph $g1$ of G, and its h_i distribution

We can conclude from the definition of $MaxPosW$ that for any extension of the current subgraph to be frequent, the product of $MaxPosW$ and MNI of the current subgraph must be at least the weighted support threshold τ, i.e., $MaxPosW \times MNI \geq \tau$. So, for a subgraph g, the minimum required MNI for being frequent (i.e., *reqFreq*) and the minimum required MNI for being extendable (i.e., *reqExt*) can be determined by the following:

$$reqFreq(g) = \lceil \tau / weight(g) \rceil \tag{3}$$

$$reqExt(g) = \lceil \tau / MaxPosW(g) \rceil \tag{4}$$

For example, in the single-large graph of Fig. 3, where edge-weights are defined in the given table, weight of subgraph $g1$ is $\frac{15+5}{2} = 10$. It contains one edge connecting the label pair $(0, 0)$, and another connecting the label pair $(1, 0)$. Thus, the list of remaining edges contains two $(1, 1)$-edges of weight 20 each, two 15-weighted $(0, 0)$-edges, five 10-weighted $(0, 1)$-edges, and four 5-weighted $(1, 0)$-edges. As such, the maximum possible weight after extending by one edge, $h_1(g1) = \frac{15+5+20}{2+1} = 13.33$. The other values of $h_i(g1)$ are plotted in Fig. 3. Observed from the bar chart, the maximum weight attainable by $g1$ after extension is $maxPosW(g1) = 15$. If the threshold is 30 and the weight of the subgraph is 10, then the subgraph must have a MNI value of at least 3 to be weighted frequent. Again, since $MaxPosW$=15, the subgraph will be extendable if it has a MNI value of at least 2. Now, the valid isomorphisms of $g1$ are (C, D, G), (A, B, I) and (C, B, I). Thus, each node of the subgraph can be mapped with two distinct nodes of the main graph to form valid isomorphisms, and thus the MNI of the subgraph is 2. Hence, although $g1$ is not weighted frequent, it is extendable.

Theorem 1. *Maximum possible weight a subgraph can attain after being extended by i edges, denoted as (h_i), follows a unimodal distribution.*

Proof. We first prove the following statement is sufficient for unimodality:

$$\forall i \in 1 \leq i \leq rem, h_i \geq h_{i+1} \implies h_{i+1} \geq h_{i+2} \tag{5}$$

Let i be the least value, for which $h_i \geq h_{i+1}$. Thus, $\forall 1 \leq j < i, h_j < h_{j+1}$. This indicates that the portion of the distribution before the decreasing part starts shall be increasing. Now, we prove that, if the aforementioned statement is true, the rest of the distribution shall stay non-increasing once the non-increasing part appears.

Induction base: For $k = i + 1$, as $h_i \geq h_{i+1} \implies h_{i+1} \geq h_{i+2}$ and $h_i \geq h_{i+1}$ are true, $h_{i+1} \geq h_{i+2}$ is true by modus ponens. This implies that the non-increasing part continues at least upto $i + 2$.

Induction step: Suppose the non-increasing portion continues upto k, i.e., $h_i \geq h_{i+1} \implies h_{i+1} \geq h_{i+2} \implies \ldots \implies h_{k-1} \geq h_k$. Again, $(h_{k-1} \geq h_k \implies h_k \geq h_{k+1}) \wedge (h_{k-1} \geq h_k) \implies (h_k \geq h_{k+1})$. Thus, if the non-increasing part extends upto k, it shall extend upto $k + 1$ as well. By principle of mathematical induction, we can conclude that, if the statement in Eq. (5) is proved, it stays non-increasing once the non-increasing part of the distribution of h_i starts. In simple terms, if statement in Eq. (5) is true, we can say that upto a certain value of i, h_i increases and stays non-increasing afterwards.

Now, we prove the statement in Eq. (5), given that $w_i \geq w_{i+1} \geq w_{i+2}$, h_i can be calculated using Eq. (1). Here, w_k denotes the weight of the k-th edge, when the edges are sorted in decreasing order of edge weights. From Eq. (1), $h_i = \frac{cur\_sum+sum_i}{k+i} = \frac{(cur\_sum+sum_i)(k+i+1)}{(k+i)(k+i+1)} = \frac{(cur\_sum+sum_i)(k+i)+(cur\_sum+sum_i)}{(k+i)(k+i+1)}$ $= \frac{cur\_sum+sum_i}{k+i+1} + \frac{cur\_sum+sum_i}{(k+i)(k+i+1)} = \frac{cur\_sum+sum_i+\frac{cur\_sum+sum_i}{k+i}}{k+i+1}$.

$$\therefore h_i = \frac{cur\_sum + sum_i + h_i}{k + i + 1} \tag{6}$$

Eq. (1) also implies that maximum possible weight after extending the subgraph in consideration by $(i+1)$-edges can at most be $h_{i+1} = \frac{cur\_sum+sum_{i+1}}{k+(i+1)} = \frac{cur\_sum+sum_i+w_{i+1}}{k+i+1}$. As $h_i \geq h_{i+1}$, using Eq. (6), we have $\frac{cur\_sum+sum_i+h_i}{k+i+1} \geq \frac{cur\_sum+sum_i+w_{i+1}}{k+i+1} \implies h_i \geq w_{i+1}$. Then, $h_{i+1} = \frac{cur\_sum+sum_i+w_{i+1}}{k+i+1} = \frac{\frac{(cur\_sum+sum_i)}{k+i}(k+i)+w_{i+1}}{k+i+1} = \frac{h_i(k+i)+w_{i+1}}{k+i+1}$. As $h_i \geq w_{i+1}$, we have $h_{i+1} \geq \frac{w_{i+1}(k+i)+w_{i+1}}{k+i+1} = \frac{w_{i+1}(k+i+1)}{k+i+1}$. Therefore, $h_{i+1} \geq w_{i+1}$. Again, from Eq. (6) and using relations, $h_{i+1} \geq w_{i+1}$ and $w_{i+1} \geq w_{i+2}$, we get $h_{i+1} = \frac{cur\_sum+sum_{i+1}+h_{i+1}}{k+(i+1)+1} \geq \frac{cur\_sum+sum_{i+1}+w_{i+1}}{k+i+2}$. Thus, $h_{i+1} \geq \frac{cur\_sum+sum_{i+1}+w_{i+2}}{k+i+2}$ because $sum_{i+1}+w_{i+2} = sum_{i+2}$, $\frac{cur\_sum+sum_{i+1}+w_{i+2}}{k+i+2} = \frac{cur\_sum+sum_{i+2}}{k+i+2}$ Therefore, using Eq. (1), it immediately follows that $\frac{cur\_sum+sum_{i+2}}{k+i+2} = h_{i+2}$ and $h_{i+1} \geq h_{i+2}$, which completes proving Eq. (5). Thus, distribution of h_i is unimodal. □

We proved Theorem 1 that distribution of highest possible weight on remaining edges (h_i) is unimodal. The classical technique of ternary search can be used to find the maximum of a unimodal function in $O(\log_2(rem))$—instead of naively applying a linear search that works in $O(rem)$—where rem is the number of remaining edges for a current subgraph. Thus, we take advantage of the unimodal property and efficiently calculate $MaxPosW$ using ternary search.

To calculate sum_i in Eq. (1) efficiently, we need to maintain a sorted list of remaining edges. While recursively searching for larger subgraphs, we add an edge to the current subgraph and remove that edge from the sorted list and do the opposite during backtracking. Thus, we need a data structure capable of supporting fast insertion, deletion and answering queries regarding the sum of the first i values in a list. Using a *segment tree*, all these operations can be achieved in logarithmic time. To reduce the size of the segment tree, we group edges with equal weights together, and keep count of how many edges map to each distinct weight. Each node of the segment tree shall contain the sum of the weights and the number of 'unremoved' edges present in the interval it represents. Thus, the value of sum_i in Eq. (1) can be determined in $O(\log_2(D))$ where D is the number of distinct edge weights. Thus, $MaxPosW$ calculation is achieved in $O(\log_2(rem) \times \log_2(D))$ time, making it feasible to calculate it in large graphs.

3.2 Our Proposed CSP Model

As described previously, for each subgraph in the search space, *WeFreS* determines if the *MNI* of the subgraph is at least equal to *reqFreq* and *reqExt*. Let $max\tau$=max(*reqFreq*, *reqExt*) and $min\tau$=min(*reqFreq*, *reqExt*). In effect, instead of taking the time-consuming route of determining the exact MNI of a subgraph, it suffices to determine if the MNI is at least equal to $max\tau$, and failing that, if it satisfies $min\tau$.

The problem of determining a lower bound of the MNI of a subgraph can be modelled as a *constraint satisfaction problem* (*CSP*). The subgraph pattern itself represents the constraint graph, with its nodes representing the variables and its edges and labels symbolizing the constraints. The values in the domain of each node in a subgraph are initially all nodes in the input graph having the same labels as it. A valid solution to the CSP is a valid subgraph isomorphism and must satisfy the following constraints.

1. No two different nodes in the subgraph can be assigned to the same node in the large graph.
2. The label of each node in the subgraph must match the label of the node in the large graph it is assigned to.
3. For each edge (u_1, u_2) in the subgraph, if v_1 is the large graph node mapped to u_1 and v_2 is mapped to u_2, there must exist an edge (v_1, v_2) in the large graph.

Initially, we seek to find if the MNI of the subgraph satisfies $max\tau$. Thus, we iterate over each variable, and try to find $max\tau$ values in its domain from which a valid subgraph isomorphism can be found. If for any variable, such $max\tau$ values do not exist, the searched MNI value is changed to $min\tau$. Thus, for the current variable, and every variable onwards, we seek to find $min\tau$ values leading to valid isomorphisms. Again, if we can ascertain that $min\tau$ values do not exist for a certain variable, then neither $max\tau$ nor $min\tau$ shall be satisfied. The MNI_LOWER_BOUND procedure terminates immediately and returns 0. To

keep track of the required MNI value to be satisfied, we propose using a *statusFlag*. Initially, the status flag is set to 2, indicating the value in question is $max\tau$. When searching for $min\tau$, the *statusFlag* is updated to 1. After iterating through all the variables for *statusFlag* values 2 and 1, $max\tau$ and $min\tau$ are returned respectively. The procedure is described in Algorithm 1.

Algorithm 1 MNI_LOWER_BOUND

Input Subgraph S, weighted graph G, min MNI req. minτ, & a max MNI req. maxτ
Output Lower bound of the MNI of S in G

```
1: statusFlag ← 2
2: if (min size of domain for nodes in S) <maxτ then
3:     statusFlag ← 1
4: if (min size of domain for nodes in S) <minτ then
5:     return 0
6: for each node v ∈ S do
7:     satisfiedValues ← 0
8:     for each value x ∈ domain(v) do
9:         if a valid isomorphism is found by assigning x to v then
10:            satisfiedValues ← satisfiedValues +1
11:        else
12:            if statusFlag=2 & (satisfiedValues+(#remaining values)) <maxτ then
13:                statusFlag ← 1
14:            if statusFlag=1 & (satisfiedValues+(#remaining values)) <minτ then
15:                return 0
16: if statusFlag = 1 then
17:     return minτ
18: return maxτ
```

If the number of nodes in the subgraph is V_S, the number of nodes in the large graph is V_G, the probabilities of the MNI of a subgraph satisfying maxτ and minτ are p_1 and p_2 respectively and the probability of an assignment of a value to a variable leading to a valid subgraph isomorphism is p, the complexity of Algorithm 1 (for the MNI lower bound) is $\mathrm{O}(V_S \cdot (p_1(\frac{max\tau}{p}) + (1-p_1)\frac{min\tau}{p}) \cdot V_G^{V_S-1})$. However, various pruning and optimization measures defined in existing literature for unweighted single large graph mining are preserved here, making the actual runtime much shorter in practice than what is suggested by the time complexity. Use of the *statusFlag* allows for parallel checking for satisfaction of *reqFreq* and *reqExt*, further increasing the efficiency of the model.

3.3 Subgraph Extension

As discussed earlier, *WeFreS* functions through a depth first traversal of some DFS code trees. The traversal occurs according to the recursive procedure outlined in Algorithm 2 (for subgraph extension). Before starting the extension process, *WeFreS* takes some pre-pruning measures to reduce the search space.

WeFreS defines a node label or a label pair as 'infertile' when it is mathematically impossible for any subgraph with that node label or the label pair to be weighted frequent and they are deleted from the node label and label pair list before starting the extension process. The mechanism for detection of infertility of a label pair is similar to determining if a subgraph containing a single edge with that label pair should be extended or not. Node labels not belonging to any 'fertile' label pair are deleted. The remaining labels are then sorted in decreasing order of the sum of the weights of the edges adjacent to nodes of each label, since such labels are more likely to output a higher number of weighted frequent subgraphs. Also, edge relations containing these labels are hence removed earlier, thus reducing the number of edges with high weight values earlier, and this in turn helps reduce the maximum possible weight estimation for subsequent subgraphs. Afterwards, the recursive SUBGRAPH_EXTENSION procedure shown in Algorithm 2 is called after initiating a single-edge subgraph with each of the remaining distinct edges. This function returns a list of weighted frequent subgraphs derived after extending the subgraph S.

Algorithm 2 SUBGRAPH_EXTENSION

Input A subgraph S, a weighted graph G, the minimum weighted threshold τ
Output All subgraphs of G extending from S w/ product of avg weight & MNI $\geq \tau$

```
1: if DFSCode(S) ≠ min(DFSCode(S)) then
2:     return
3: reqFreq ← ⌈τ/current weight of subgraph⌉
4: reqExt ← ⌈τ/maxPosW(S)⌉
5: minτ ← min(reqFreq, reqExt)
6: maxτ ← max(reqFreq, reqExt)
7: mniLowerBound ← MNI_LOWER_BOUND(S, G, minτ, maxτ)
8: result ← ∅
9: if mniLowerBound ≥ reqFreq then
10:    result ← S
11: if mniLowerBound ≥ reqExt then
12:    for each edge e ∈ Edges and node u of S do
13:        if e can be used to extend u then
14:            Let ext be the extension of S with e
15:            Decrement the count of e in SegmentTree
16:            result ← result ∪ SUBGRAPH_EXTENSION(ext, G, τ)
17:            Increment the count of e in SegmentTree
18: return result
```

The procedure initially checks if the subgraph in question is lexicographically minimal. The concepts of minimum DFS code and lexicographical ordering of subgraphs are introduced in gSpan and are used here to counter duplicate subgraph generation. Afterwards, the values of *reqFreq* and *reqExt* are determined from the current weight and *maxPosW* of S respectively. If *reqFreq* is satisfied, S is added to the list of weighted frequent subgraphs. If *reqExt* is satisfied, S is

Table 1. Datasets

Dataset	#nodes	#edges	#labels	Directed	Distribution	MinW	MaxW	Normalized
MiCo	100k	108,029	29	No	negExp (λ=1.0)	25.75	70.0	No
Amazon	163k	296k	1,856	Yes	normal (μ=10, σ=1)	0.00	1.0	Yes
					negExp (λ=1)			
FreeAssoc	10,617	72,176	10	Yes	normal (μ=25, σ=1)	22.99	27.5	No

recursively extended, making necessary updates on the list of remaining edges, which is maintained using a *segment tree*.

With the time complexity involved with the determination of *maxPosW* being negligible in comparison, the time complexity of *WeFreS* is proportional to the product of the search space and the time complexity of determining the lower bound of the MNI of each subgraph. With the search space being $(2^{V_G})^2$ in the worst case, the worst case complexity of the algorithm is bounded by $O((2^{V_G})^2 \cdot V_S \cdot (p_1(\frac{\tau_1}{p}) + (1-p_1)\frac{\tau_2}{p}) \cdot V_G^{V_S-1})$ where τ_1 and τ_2 are maximums of all values of maxτ and minτ encountered, respectively, in the search space. However, the *maxPosW* pruning technique makes the search space much smaller for reasonable thresholds, making *WeFreS* feasible for use in real life applications, as demonstrated by experimental analysis presented in Section 4.

4 Evaluation Results

Experiments were conducted on the following three real-world graph datasets to evaluate the performance of our proposed approach in comparison with other existing approaches and baseline algorithms w.r.t. runtime and memory usage:

1. MiCo, a co-authorship and collaboration graph representing data from academic.research.microsoft.com;
2. Amazon [13], a co-purchase network consisting of electronic items found in the Amazon website; and
3. FreeAssoc [14], a dataset representing a word association network based on the English language.

Since none of these datasets had pre-specified edge-weights, we generated the weights using normal and exponential distributions. To test the performance of *WeFreS* in graphs containing high values of edge-weights, both the MiCo and FreeAssoc datasets were assigned weights using exponential and normal distributions respectively. For comparison with ReSuM [17], which requires edge-weights to be within the range [0, 1], the Amazon dataset was assigned normalized edge weights using both statistical distributions. Finally, to show that *WeFreS* can perform sufficiently well even in unweighted graphs, we compare *WeFreS* to GraMi [5] by assigning weight equal to 1 in all edges of all three datasets. Table 1 shows the quantitative specifications of each dataset.

All experiments were conducted on a device with 8GB RAM, an intel core i5 7th gen processor, with 2500 MHz clock speed and an Ubuntu 17.10 operating system. All approaches were implemented in Java by modifying a public

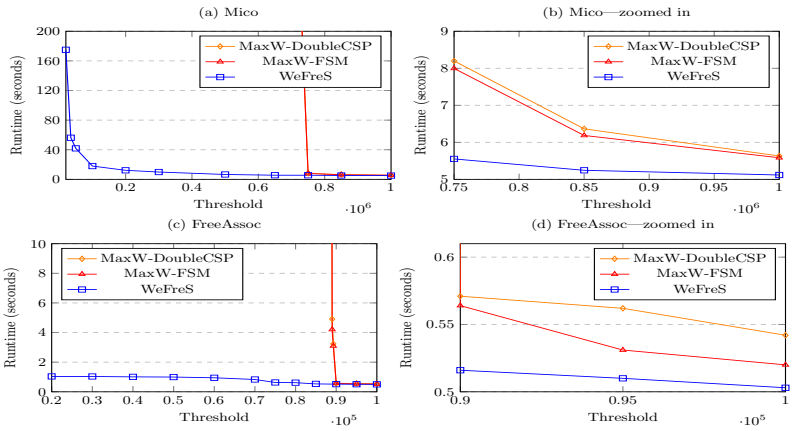

Fig. 4. Runtime comparisons with baseline algorithms

Table 2. Memory usage comparisons

Dataset	Threshold	Memory Usage (MaxW-FSM) (MB)	Memory Usage (WeFreS) (MB)
Mico	850,000	914.97	899.54
	650,000	1,111	907.36
	300,000	–	998.80
FreeAssoc	89,000	200.73	118.23
	88,500	5,231	118.23
	15,650	–	2153

implementation of GraMi [5]. Since there is no existing approach for mining weighted frequent subgraphs from single large graphs where edge-weights can have any numeric value, we defined two baseline algorithms for comparison: (i) *MaxW-DoubleCSP* (which applies the CSP model used in GraMi and issues two successive calls to determine if the MNI of a given subgraph satisfies *reqFreq* and *reqExt*) and (ii) *MaxW-FSM* (which applies the CSP model described in Section 3.2 and thus issues a single CSP call only). Both approaches differ from *WeFreS* in that, instead of using *MaxPosW* to determine the value of *reqExt*, they use the *MaxW* measure (which is applied in traditional weighted pattern mining approaches [20]). The value of *MaxW* is equal to the highest edge-weight present in the input graph.

Being a much tighter upper bound than *MaxW*, the *MaxPosW* estimate helps prune out many subgraphs and their extensions from the DFS code tree that is traversed, which were otherwise visited by algorithms adopting the *MaxW* measure. Thus, *WeFreS* has less search space than *MaxW-FSM* and *MaxW-DoubleCSP*. Furthermore, the reduced pruning tendency of these baseline algorithms mean that they traverse further down the DFS code tree, meaning that they need to evaluate the MNI of subgraphs containing a higher number of nodes, thus requiring longer runtime and more memory. Notably, the segment tree used in *WeFreS* introduces very little memory overhead, which is compen-

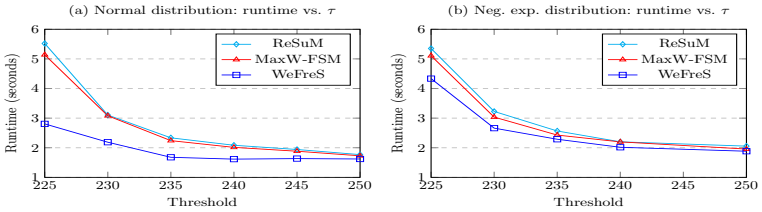

Fig. 5. Runtime comparison with *ReSuM* in the *Amazon* dataset

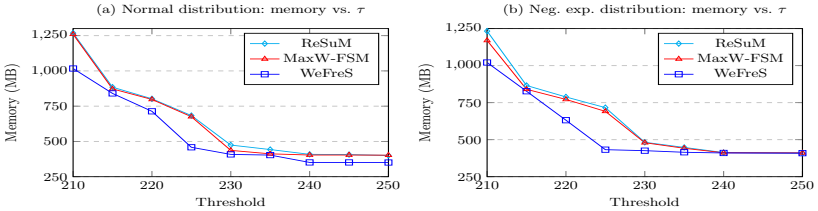

Fig. 6. Memory usage comparison with *ReSuM* in the *Amazon* dataset

sated by the benefits of reduced search space as shown in Fig. 4 and Table 2. For low threshold values, the baseline algorithms failed to produce results even after being run for hours, with such entries being marked X in Table 2.

Comparisons with *ReSuM* are done in the *Amazon* dataset, using both normal and exponential distributions for weight assignments. With $MaxW = 1$, *MaxW-DoubleCSP* behaves identical to *ReSuM* (while mining for subgraphs with high weighted support) in that it first finds frequent subgraphs and then filters out the weighted infrequent ones. Figures 5 and 6 show that *WeFreS* outperforms *ReSuM* in terms of runtime and memory.

Although designed for use in weighted graphs, Fig. 7 shows the runtimes of *WeFreS* are similar to that of *GraMi* in unweighted graph datasets. The additional pre-pruning measure of deleting node labels that are not part of any fertile label pair causes *WeFreS* to be more memory efficient. See Fig. 8.

5 Conclusions

In this paper, we explored an innovative direction of considering edge-weights in mining subgraph patterns from single large graphs. Our novel algorithm—namely, *weighted frequent subgraph miner* (*WeFreS*)—considers both the weight and significance of the interactions between different types of entities and only outputs weighted frequent subgraphs. Experimental results show the feasibility of using *WeFreS* in large graphs (where edge weights can have any real values) and its excellent performance over an existing state-of-the-art approaches (which require edge-weights to have values within the range of $[0, 1]$). Moreover, *WeFreS* is also feasible for use in unweighted frameworks, making it a truly general solution to the problem of frequent subgraph mining from single large graphs. The

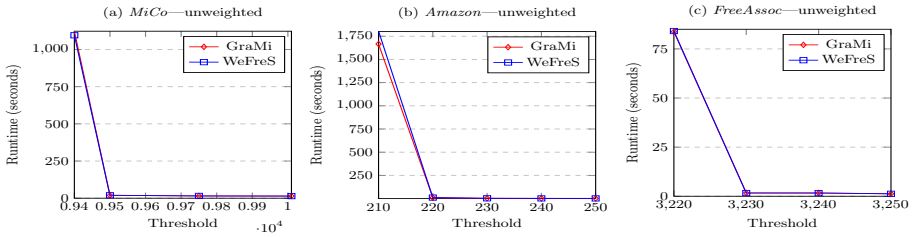

Fig. 7. Runtime comparison with *GraMi* in unweighted datasets

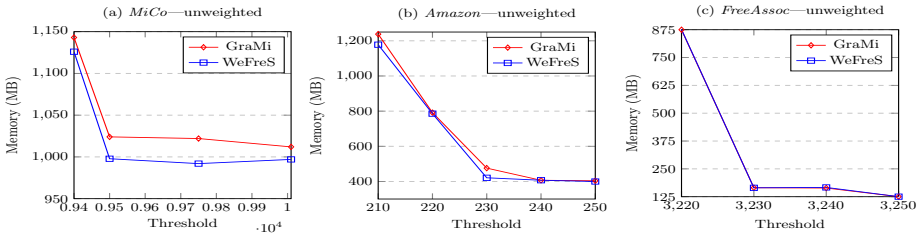

Fig. 8. Memory usage comparison with *GraMi* in unweighted datasets

subgraphs mined by *WeFreS* are both frequent and significant, and can be used in a variety of applications. In addition, we also introduced novel approaches for determining *MaxPosW* and proposed the constraint satisfaction problem (CSP) model. As ongoing and future work, we are extending our WeFreS algorithm, optimizing our determination of *MaxPosW*, and enhancing our CSP model.

Acknowledgements

This project is partially supported by NSERC (Canada) and University of Manitoba.

References

1. Ata, S.K., Fang, Y., Wu, M., Li, X., Xiao, X.: Disease gene classification with metagraph representations. Methods 131, 83–92 (2017)
2. Braun, P., Cuzzocrea, A., Leung, C.K., Pazdor, A.G.M., Tran, K.: Knowledge discovery from social graph data. Procedia Computer Science 96, 682–691 (2016)
3. Bringmann, B., Nijssen, S.: What is frequent in a single graph? In: PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 858–863 (2008)

4. El Bazzi, M.S., Mammass, D., Zaki, T., Ennaji, A.: A graph-based ranking model for automatic keyphrases extraction from arabic documents. In: ICDM 2017. LNCS (LNAI), vol. 10357, pp. 313–322 (2017)
5. Elseidy, M., Abdelhamid, E., Skiadopoulos, S., Kalnis, P.: GraMi: frequent subgraph and pattern mining in a single large graph. PVLDB 7(7), 517–528 (2014)
6. Fan, C., Hao, H., Leung, C.K., Sun, L.Y., Tran, J.: Social network mining for recommendation of friends based on music interests. In: IEEE/ACM ASONAM 2018, pp. 833–840 (2018)
7. Fakhraei, S., Foulds, J., Shashanka, M., Getoor, L.: Collective spammer detection in evolving multi-relational social networks. In: ACM KDD 2015, pp. 1769–1778 (2015)
8. Hoi, C.H.S., Leung, C.K., Tran, K., Cuzzocrea, A., Bochicchio, M., Simonetti, M.: Supporting social information discovery from big uncertain social key-value data via graph-like metaphors. In: ICCC 2018. LNCS (LNISA), vol. 10971, pp. 102–116 (2018)
9. Islam, M.A., Ahmed, C.F., Leung, C.K., Hoi, C.S.H.: WFSM-MaxPWS: an efficient approach for mining weighted frequent subgraphs from edge-weighted graph databases. In: PAKDD 2018, Part III. LNCS (LNAI), vol. 10939, pp. 664–676 (2018)
10. Jiang, C., Coenen, F., Zito, M.: Frequent sub-graph mining on edge weighted graphs. In: DaWaK 2010. LNCS, vol. 6263, pp. 77–88 (2010)
11. Leung, C.K., Middleton, R., Pazdor, A.G.M., Won, Y.: Mining 'following' patterns from big but sparsely distributed social network data. In: IEEE/ACM ASONAM 2018, pp. 916–919 (2018)
12. Liakos, P., Papakonstantinopoulou, K.: On the impact of social cost in opinion dynamics. In: ICWSM 2016, pp. 631–634 (2016)
13. McAuley, J., Targett, C., Shi, Q., Van Den Hengel, A.: Image-based recommendations on styles and substitutes. In: ACM SIGIR 2015, pp. 43–52 (2015)
14. Nelson, D.L., McEvoy, C.L., Schreiber, T.A.: The University of South Florida free association, rhyme, and word fragment norms. Behavior Research Methods, Instruments, & Computers 36(3), 402–407 (2004)
15. Perner, P.: Mining frequent subgraph pattern over a collection of attributed-graphs and construction of a relation hierarchy for result reporting. In: ICDM 2017. LNCS (LNAI), vol. 10357, pp. 323–344 (2017)
16. Phan, H., Le, B.: A novel parallel algorithm for frequent itemsets mining in large transactional databases. In: ICDM 2018. LNCS (LNAI), vol. 10933, pp. 272–287 (2018)
17. Preti, G., Lissandrini, M., Mottin, D., Velegrakis, Y.: Beyond frequencies: graph pattern mining in multi-weighted graphs. In: EDBT 2018, pp. 169–180 (2018)
18. Wang, Y., Cai, Y.: Message passing on factor graph: a novel approach for orphan drug physician targeting. In: ICDM 2017. LNCS (LNAI), vol. 10357, pp. 137–150 (2017)
19. Yan, X., Han, J.: gSpan: graph-based substructure pattern mining. In: IEEE ICDM 2003, pp. 721–724 (2002)
20. Yun, U., Leggett, J.J.: WFIM: weighted frequent itemset mining with a weight range and a minimum weight. In: SIAM SDM 2005, pp. 636–640 (2005)