WFSM-MaxPWS: An Efficient Approach for Mining Weighted Frequent Subgraphs from Edge-Weighted Graph Databases

Md. Ashraful Islam[1], Chowdhury Farhan Ahmed[1], Carson K. Leung[2](✉), and Calvin S. H. Hoi[2]

[1] Department of Computer Science and Engineering, University of Dhaka, Dhaka, Bangladesh
csedu.ashraful@gmail.com, farhan@du.ac.bd

[2] Department of Computer Science, University of Manitoba, Winnipeg, MB, Canada
{kleung,hoic}@cs.umanitoba.ca

Abstract. Weighted frequent subgraph mining comes with an inherent challenge—namely, weighted support does not support the downward closure property, which is often used in mining algorithms for reducing the search space. Although this challenge attracted attention from several researchers, most existing works in this field use either affinity based pruning or alternative anti-monotonic weighting technique for subgraphs other than average edge-weight. In this paper, we propose an efficient weighted frequent subgraph mining algorithm called WFSM-MaxPWS. Our algorithm uses the MaxPWS pruning technique, which significantly reduces search space without changing subgraph weighting scheme while ensuring completeness. Our evaluation results on three different graph datasets with two different weight distributions (normal and negative exponential) showed that our WFSM-MaxPWS algorithm led to significant runtime improvement over the existing MaxW pruning technique (which is a concept for weighted pattern mining in computing subgraph weight by taking average of edge weights).

1 Introduction

As *frequent pattern mining* has been an appealing area of data mining, many algorithms have been developed for general pattern mining [5,12,13] to specific pattern mining (e.g., mining sequential patterns, weighted patterns, web access sequences, data streams). To deal with complex data, graph mining [3,7] has emerged as an inevitable area. When compared to unweighted graphs, *weighted graphs* have strong representational power. *Weighted frequent subgraphs* describe underlying graph database more accurately, and thus contribute greatly in areas like feature extraction for graph classification, association rule mining, graph clustering. For example, it is impossible to identify sophisticated metamorphic malwares using signature based approaches. Runtime behavior, though very accurate, is hardly used due to its slower detection rate. As an alternative, malware call-graph analysis has been shown to be very effective. However, insertion of huge benign codes in malwares reduces the chance of identifying comparatively less frequent suspicious subgraphs when using traditional subgraph mining algorithms. So, the malware detection task can take advantage of weighted frequent subgraph mining by giving higher weights to those call sequences.

Frequent subgraph mining is a tedious task partially because of subgraph isomorphism checking and exponential growth of candidate patterns. Luckily, canonical ordering of graphs and anti-monotonicity of downward closure property have made frequent subgraph mining a feasible task. However, for weighted frequent subgraph mining, the downward closure property no longer holds. Existing algorithms handle general weighted frequent itemset mining in FP-tree based mining algorithms by considering GMaxW (weight of maximum weighted item from the initial global FP-tree) and then recursively considering LMaxW (weight of maximum weighted item in local conditional FP-trees) as the maximum possible itemset weight for pruning [1]. The completeness is ensured by sorting itemsets in weight ascending order. This approach generates a moderate number of candidate patterns because recursively lighter weights are considered for pruning. However, as MaxW is usually much heavier than the actual average weight of subgraph, a huge number of unnecessary candidate subgraphs are generated, and thus leading to long runtime. Although attempts to reduce the number of generated candidate sets have been made, most of the existing approaches either use affinity based pruning (which usually imposes extra conditions to measure interestingness of a subgraph) or other alternative weighting techniques (which redefines subgraph weights such that the downward closure property is satisfied).

In this paper, we aim to reduce the number of generated candidate sets via our non-trivial adoption of affinity-based conditions and subgraph-weighting techniques. The work is inspired by our observation that, when considering MaxW as the maximum possible weight of extended subgraph, those already-seen average weights of the subgraph are often ignored. So, we decided to make good use of (i) these already-seen weights of a subgraph and (ii) some statistical information about the dataset to calculate the maximum possible weight and frequency for extensions of subgraph up to many edges. Consequently, the **Max**imum **P**ossible **W**eighted **S**upport (**MaxPWS**) can be computed for a subgraph. Moreover, by making an intelligent change in canonical ordering for weighted subgraphs, the magnitude of MaxPWS can further be brought closer to the actual weighted support, which then leads to safe and effective pruning of unnecessary candidate subgraphs. Hence, *our key contributions* of this paper are as follows:

– a tighter pruning technique **MaxPWS** for weighted subgraph mining, and
– a canonical ordering for weighted subgraphs that makes MaxPWS tighter,
– a weighted frequent subgraph mining algorithm called **WFSM-MaxPWS**, which uses the MaxPWS pruning technique.

The remainder of this paper is organized as follows. The next section presents background and related works. Section 3 describes our MaxPWS pruning technique, canonical order modification, and WFSM-MaxPWS algorithm. Experimental results and conclusions are given in Sects. 4 and 5, respectively.

2 Background and Related Works

Let us give some background information about **weighted graph mining**, which aims to find weighted frequent subgraph. An *edge-weighted graph* G is a collection of nodes V, edges E, together with a mapping between edge-set E and weights. For a function $\boldsymbol{W}(\boldsymbol{e})$ that returns the weight of edge $\boldsymbol{e}$, the weight of a subgraph $\boldsymbol{g}$ can be defined as: $W(g) = \frac{1}{n}\sum_{i=1}^{n} W(e_i)$, where each e_i is an edge of g. Given a graph database GDB of weighted graphs and a minimum weighted support threshold τ, a subgraph g is said to be *weighted frequent* if $wsup(g) \geq \tau$ where $wsup(g) = W(g) \times sup(g)$. In this paper, we focus on the condition that "all edges with same edge label and end-point node label have the same weight".

As the base for our proposed MaxPWS pruning technique, the **gSpan algorithm** [16] represents each subgraph by using DFScode, and it uses an extended tuple comparison rule to rank the DFScode of a subgraph by following the right-most path extension. Among several isomorphism of a subgraph, the one with the lowest rank is said to be *canonical*. In this paper, we have modified such an extended tuple comparison rule for *weighted* graphs. A challenge is that such a modification affects the rank and changes the canonical DFScode of a subgraph.

In terms of **related works**, both GWF-mining and CWF-mining algorithms [15]—as extensions of utility based itemset mining (which considers non-weighted support)—mine internally and externally weighted graphs respectively by considering external weighted frequency for complex data. Along this direction, further extensions include closed and maximal subgraph mining [14]. In contrast, we consider individual subgraph weight calculated from edge weights and non-weighted frequency.

Eichinger et al. [4] showed that frequent subgraph mining task yields more precise results when considering weight-based constraints. As these weight-based constraints are not anti-monotonic, their algorithm returns approximate and thus *incomplete* results. In contrast, our proposed algorithm is *complete*.

Yang et al. [17] performed weighted subgraph mining on *single individual* weighted graphs. In contrast, our proposed algorithm focuses on mining weighted frequent patterns from a *set of weighted graphs*.

Three subgraph weighting techniques—namely, Average Total Weighting (ATW), Affinity Weighting (AW), and Utility Based Weighting (UBW) were proposed [9] and adapted for *longitudinal social network data* [8]. Among them, ATW requires redefining the subgraph weight as a ratio between total graph database weight and subgraph support set weight. Although ATW is anti-monotonic, it fails to discriminate between two subgraphs having the same support set. AW prunes a subgraph if its edges fail to satisfy some weight correlation condition, whereas UBW discards a subgraph if its weight-share falls below a weight-share threshold λ. On the contrary, our MaxPWS does not require redefining subgraph weight as it takes an average of edge weights. Hence, it can discriminate between two subgraphs even if they have exactly same support set. Moreover, MaxPWS can be applied to any affinity or utility conditions.

Lee and Yun [10] applied weighted-support affinity to reduce search space. Length-decreasing support constraints [11] were used as a weighted smallest-valid extension for frequent graph mining because graph patterns extracted from a given graph database can have various features (e.g., different pattern lengths). Moreover, a distributed approach of weighted frequent subgraph mining using ATW weighting technique [2] was proposed. Another approach [6] finds regular patterns from weighted-directed dynamic graphs with jitter. These four algorithms focus on *special or specific* weighted frequent subgraph mining tasks. In contrast, our algorithm focuses on the *general* weighted pattern mining task.

3 Our Proposed Algorithm

To reduce candidate subgraph generation, we propose (i) a tight pruning condition **MaxPWS** for weighted frequent subgraph mining and (ii) an algorithm called **WFSM-MaxPWS** for **W**eighted **F**requent **S**ubgraph **M**ining by using the MaxPWS condition. In this section, we first show our proposed canonical ordering for edge-weighted graphs, and then discuss details about MaxPWS.

3.1 WFSM-MaxPWS Canonical Ordering of Subgraph

For edge-weighted graphs, we add weight-property in the extended edge-tuple representation used in gSpan. Consequently, the new extended tuple for an edge (u, v) is of the following form as a 6-tuplet:

$$\langle dis_u, dis_v, L(u), L(v), L(u,v), W(u,v)\rangle,$$

where $W(u, v)$ is the weight of edge (u, v). To give a rank of a DFScode of subgraph, the WFSM-MaxPWS canonical order gives the highest priority to (dis_u, dis_v) as in gSpan. The second highest priority is given to edge weight $W(u, v)$. The higher the weight, the smaller is the tuple for the same discovery times. The third priority is given to a lexicographic comparison on node and edge label trio. To elaborate, let

$$t_1 = \langle v_i, v_j, L(v_i), L(v_j), L(v_i, v_j), W(v_i, v_j)\rangle; \text{ and}$$
$$t_2 = \langle v_x, v_y, L(v_x), L(v_y), L(v_x, v_y), W(v_x, v_y)\rangle.$$

Then, $t_1 < t_2$ if and only if

1. $(v_i, v_j) <_e (v_x, v_y)$; or
2. $(v_i, v_j) = (v_x, v_y)$ and $W(v_i, v_j) > W(v_x, v_y)$; or
3. $(v_i, v_j) = (v_x, v_y)$ and $W(v_i, v_j) = (v_x, v_y)$ and $\langle L(v_i), L(v_j), L(v_i, v_j)\rangle <_l \langle L(v_x), L(v_y), L(v_x, v_y)\rangle$.

Here, $<_e$ is an ordering on edge, and $<_l$ is an ordering on vertex and edge labels. Note that $<_l$ follows lexicographic order. The edge order ($<_e$) rule is derived from the *rightmost path extension* sequence. The edge that extended earlier is smaller.

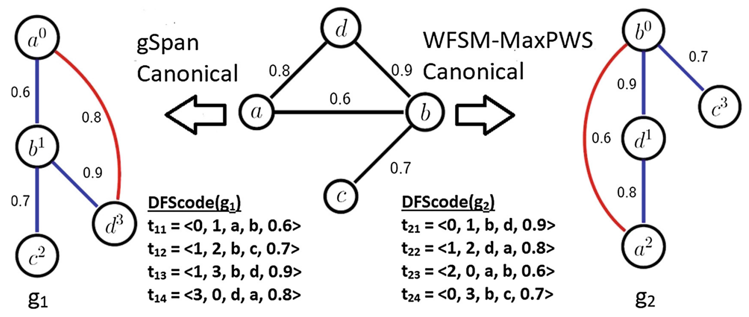

Fig. 1. Canonical code comparison with gSpan (without edge label)

Figure 1 shows a comparison between the canonical representations in gSpan [16] and our WFSM-MaxPWS algorithm. For simplicity and readability, tuples are shown as 5-tuplets by omitting the edge labels $L(u, v)$. As gSpan puts an edge ordering during the mining time of endpoint nodes and compares the (node labels-edge label)-trio in lexicographic order, tuple t_{11} becomes the smallest for graph in the figure. On the other hand, after performing edge ordering during the node mining time, the second importance of our WFSM-MaxPWS is put on edge weight. The *higher the weight, the smaller is the tuple.* As the first tuple of any DFScode has node discovery pair $(0, 1)$, the highest weighted edge would be the smallest. So, WFSM-MaxPWS would consider tuple $\langle 0, 1, b, d, 0.9\rangle$ as the smallest. DFScode with other edges as first tuple would not be canonical.

Lemma 1. *No tuple can have a weight higher than the weight of the first tuple in a canonical WFSM-MaxPWS DFScode.*

Proof. (By induction on edge count m) For the base case (when $m = 1$), first tuple is the only tuple. Hence, there is no other tuple with a heavier weight.

For the inductive step, let us assume that the first tuple in DFScode C_x for $1 < m \leq x$ is $t_{x0} = \langle 0, 1, L(v_1), L(v_2), L(v_1, v_2), W(v_1, v_2)\rangle$. If we extend the subgraph with a tuple having a weight lower or equal to the first tuple, then the condition continues to hold. However, if we extend with a higher weighted tuple $t_{xk} = \langle *, *, L(u_1), L(u_2), L(u_1, u_2), \mathrm{W}(u_1, u_2)\rangle$, then there exists a new possible DFScode C_y having $t_{y0} = \langle 0, 1, L(u_1), L(u_2), L(u_1, u_2), W(u_1, u_2)\rangle$ as first tuple. Here, $t_{x0} > t_{y0}$ implies that $Rank(C_y) < Rank(\{C_x, t_{xk}\})$. So, $\{C_x, t_{xk}\}$ cannot be canonical. Thus, we cannot extend a subgraph with a higher weighted tuple while preserving canonicity. $\square$

3.2 MaxPWS Pruning Technique

We divide the entire graph database into partitions $p_1, p_2, p_3, \ldots, p_x, \ldots, p_M$. Partition p_1 is the set of graphs in the database having the minimum number of edges; partition p_M is the collection of graphs having the maximum number of edges. Graphs in the same partition have the same number of edges. Each graph

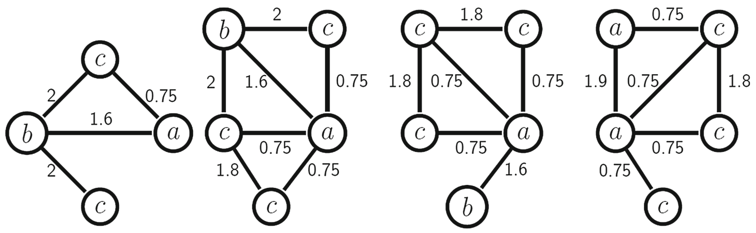

Fig. 2. Sample graph database *GDB*

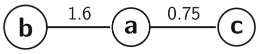

Fig. 3. Sample subgraph g_x

in partition p_x contain at least one more edge than any graphs in partition p_{x-1} and contain at least one fewer edge than graphs in partition p_{x+1}. We called these partition-collection **Edge-Class (EC)**. For the sample graph database in Fig. 2, there are three partitions. Hence, $EC = \{p_1 = \{G_1\}_4;\ p_2 = \{G_3, G_4\}_6;\ \ p_3 = \{G_2\}_7\}$, where the subscript after each curly brackets indicates the edge count of the partition. For any subgraph of consideration, we calculate its *occurrence list*, which is a collection of subsets from each Edge-Class partition entry where each member graph of the collection is a superset for that subgraph.

Definition 1. Let g be a subgraph. Its **occurrence list (OL)** is defined as $OL(g) = \{q_1, q_2, q_3, \ldots\}$ where $\forall q_i \in OL(g)[q_i \subseteq p_i$ and g is a subgraph of each member of $q_i]$. We call q_i an *occurrence list member* (*OLM*).

For example, for the database in Fig. 2, the occurrence list for subgraph g_x shown in Fig. 3 is $OL(g_x) = \{q_1 = \{G_1\}_4;\ q_2 = \{G_3\}_6;\ q_3 = \{G_2\}_7\}$.

We can also calculate the **possible occurrence list (pol)** and **maximum possible frequency (mpf)** for a subgraph based on its OL for extension of that subgraph up to a different edge count. For example, from the OL of g_x, if we extend the subgraph g_x up to 4 edges, then the corresponding *pol* of the extended subgraph $g_{x\to4}$ is still $pol(g_{x\to4}) = \{\{G_1\}_4; \{G_3\}_6; \{G_2\}_7\}$. Similarly, the mpf of $g_{x\to4}$ becomes $mpf(g_{x\to4}) = 3$. However, if we extend g_x beyond 4 edges, graph G_1 can no longer be a part of its OL because it contains only 4 edges in total. For extension up to 6 edges, $pol(g_{x\to6}) = \{\{G_3\}_6;\ \{G_2\}_7\}$ and $mpf(g_{x\to6}) = 2$. Similarly, for extension up to 7 edge, $pol(g_{x\to7}) = \{\{G_2\}_7\}$ and $mpf(g_{x\to7}) = 1$. Now, the heaviest possible extension of g_x up to 4, 6 and 7 edges can contain 2, 4 and 5 more MaxW-weight edges, respectively. For the database in Fig. 2, it is 2.0 (edge b–c). According to Lemma 1, this extension can be canonical if and only if the first tuple in the canonical DFScode of g_x has a weight at least 2.0. However, canonical DFScode of g_x is $\{\langle 0, 1, a, b, 1.6\rangle, \langle 0, 2, a, c, 0.75\rangle\}$. (For simplicity, we omitted edge labels here.) Thus, to be canonical, any extended subgraph of g_x cannot have an edge with a weight heavier than 1.6.

In general, to preserve canonicity, the heaviest possible extension of any subgraph is its **first tuple weight (FTW)**. With this concept, for any subgraph g, we can calculate its **possible weighted support (PWS)** after extending up to m-edges by the following equation:

$$PWS(g_{\to m}) = \frac{|g| \times W(g) + (m - |g|) \times FTW}{m} \times mpf(g_{\to m}), \tag{1}$$

where $|g|$ is the number of edges in g. In this calculation, g is assumed to be extended up to m edges with each new edge having the FTW weight to ensure that this imaginary extended supergraph has the maximum possible weight w.r.t. the graph database (because extension with a heavier weighted edge would not be canonical). We calculate MaxPWS by taking the maximum among all PWS values for g. If MaxPWS fails to satisfy the minimum weighted support threshold τ, then we can safely prune g. Otherwise, we need to extend g because its extended subgraph have potential to be weighted-frequent.

For example, consider subgraph g_x in Fig. 3 with $W(g_x) = \frac{1.6+0.75}{2} = 1.175$. For graph database in Fig. 2, PWS values are as follows:

- $PWS(g_{x\to 4}) = \frac{2\times 1.175+(4-2)\times 1.6}{4} \times 3 = 1.3875 \times 3 = 4.1625$
- $PWS(g_{x\to 6}) = \frac{2\times 1.175+(6-2)\times 1.6}{6} \times 2 \approx 1.4583 \times 2 = 2.9166$
- $PWS(g_{x\to 7}) = \frac{2\times 1.175+(7-2)\times 1.6}{7} \times 1 \approx 1.4786 \times 1 = 1.4786$

Hence, $MaxPWS(g_x) = 4.1625$. In contrast, the MaxW measure for g_x is $2\times 3 = 6$ (because $MaxW = 2$ and frequency = 3), which is greater than MaxPWS.

Note that, if we do not use the modified canonical ordering for weighted graphs as proposed in Sect. 3.1, then the PWS calculation would have to use MaxW (instead of FTW as shown in Eq. (2)):

$$PWS(g_{\to m}) = \frac{|g| \times W(g) + (m - |g|) \times MaxW}{m} \times mpf(g_{\to m}) \tag{2}$$

The resulting algorithm (which uses MaxW) is called **MaxPWS-gSpan algorithm**, which can be considered as a variant of our WFSM-MaxPWS algorithm (which uses FTW).

Lemma 2. *MaxPWS-measure is anti-monotonic.*

Proof. (By contradiction) Suppose that MaxPWS is not anti-monotonic. Then, there exists an extended subgraph g_x of g for which $MaxPWS(g_x) \geq \tau$ even though $MaxPWS(g) < \tau$. If $|g_x| - |g| = k$, then

$$W(g_x) \times |g_x| \leq W(g) \times |g| + k \times FTW. \tag{3}$$

If MaxPWS of g_x occurs in m-edge extension, then $PWS(g_{x\to m}) > PWS(g_{\to m})$. This means $\frac{|g_x|\times W(g_x)+(m-|g_x|)\times FTW}{m} \times mpf(g_{x\to m}) > \frac{|g|\times W(g)+(m-|g|)\times FTW}{m} \times$

$mpf(g_{\rightarrow m})$. Due to the downward closure property of frequency, $mpf(g_{x\rightarrow m})$ must be less than or equal to $mpf(g_{\rightarrow m})$. So, by removing the mpf terms from both side, $|g_x| \times W(g_x) + (m - |g_x|) \times FTW > |g| \times W(g) + (m - |g|) \times FTW$. By using Eq. (3) and $|g_x| = |g| + k$, we get $W(g) \times |g| + k \times FTW + (m - |g| - k) \times FTW > |g| \times W(g) + (m - |g|) \times FTW$. Consequently, $k \times FTW + (m - |g| - k) \times FTW > (m - |g|) \times FTW$, and thus $(m - |g|) \times FTW > (m - |g|) \times FTW$, which is impossible. Thus, $MaxPWS(g_x)$ cannot be greater than $MaxPWS(g)$. So, MaxPWS-measure is anti-monotonic. □

Corollary 1. *If $MaxPWS(g) < \tau$, then g has no potential weighted frequent extension.* □

Corollary 2. *Due to MaxPWS-measure $\leq$ MaxW-measure, MaxPWS pruning technique prunes more unnecessary patterns.* □

3.3 The WFSM-MaxPWS Algorithm

A pseudocode for WFSM-MaxPWS is shown in Algorithm 1. Here, our WFSM-MaxPWS algorithm takes the following four input parameters: (i) the canonical DFScode of a graph C, (ii) graph database D, (iii) weighted support threshold τ, and (iv) occurrence list OL_C of C. With the initial call, $C = \emptyset$ and $OL_C = EC$ (edge class). WFSM-MaxPWS puts all frequent weighted subgraphs in a *result* set.

Algorithm 1. Algorithm *WFSM-MaxPWS*

```
1: procedure WFSM-MaxPWS($C, D, \tau, OL_C$)
2:     $OL\_vec$ = rightmost-path-extension($C, OL_C, D$)
3:     for each $(t, OL_t) \in OL\_vec$ do
4:         $C' = C \cup t$
5:         if IS_CANONICAL(C') = false then continue.
6:         Set $sup_{C'} = 0$ and $MaxPWS = 0$
7:         for each $q_i \in OL_t$ in reverse order (last to first) do
8:             $m_i$ = edge count of $q_i$
9:             $sup_{C'} = sup_{C'} + |q_i|$   //cardinality of $q_i$
10:            $PWS_i = \frac{|C'| \times W(C') + (m_i - |C'|) \times FTW}{m_i} \times sup_{C'}$
11:            if $MaxPWS < PWS_i$ then $MaxPWS = PWS_i$
12:        $wsup_{C'} = W(C') \times sup_{C'}$
13:        if $wsup_{C'} \geq \tau$ then
14:            $result = result \cup C'$   //Enlist $C'$ as frequent weighted subgraph
15:            WFSM-MaxPWS($C', D, \tau, OL_t$)
16:        else if $MaxPWS \geq \tau$ then WFSM-MaxPWS($C', D, \tau, OL_t$)
```

The function "rightmost-path-extension" in line 2 enumerates all possible extensions on rightmost path of the given DFScode and returns a vector of those extensions as new edge tuples and their corresponding occurrence list OL_vec.

Each entry in OL_vec is then checked for canonicity (lines 4 & 5). The support of code C' is updated while iterating through each entry in OL (line 9). Simultaneously, possible weighted support from the highest to the lowest $OLM(q_i)$ is calculated to find $MaxPWS$ (lines 8–11). The actual weighted support $wsup$ of the code/subgraph is calculated in line 11. If it satisfies weighted support threshold condition (line 13), then it is enlisted as a frequent weighted subgraph (line 14) and sent for further extension (line 15). Otherwise, if $MaxPWS$ satisfies weighted support threshold condition, though it will not be enlisted as frequent weighted subgraph, then it will be sent to the WFSM-MaxPWS algorithm for further extension (line 16) as its extended graph still has a chance to be frequent weighted subgraph.

4 Experimental Results

To evaluate the performance of our proposed algorithm WFSM-MaxPWS, we conducted several experiments on a PC with an Intel Core i3-2100 CPU at 3.10 GHz and 4 GB RAM running MS Windows 10 operating system. We analyze performance of WFSM-MaxPWS w.r.t. runtime, search-space reduction efficiency, and memory requirement. For comparison, we used MaxW-gSpan (which uses MaxW pruning technique with gSpan) as the baseline algorithm. Both WFSM-MaxPWS and MaxW-gSpan were implemented in Python.

Regarding the test datasets, several graphs datasets[1] were selected from PubChem[2], which provides information on biological activities of small molecules and contains the bioassay records for anti-cancer screen tests with different cancer cell lines. Each dataset captures a certain type of cancer screen with the outcome active or inactive. In particular, we used **MCF-7**, **P388** and **Yeast** datasets. Since these datasets come with no weights, we added weights according to two different weight distributions—namely, *normal* and *neg*ative *expo*nential. Dataset statistics after adding weight is given in Table 1.

Table 1. Datasets

Dataset	#graphs	Distinct #edges	Avg #edges	Distribution	MinW	MaxW
MCF-7	2,293	54	36	normal ($\mu = 0.5$, $\sigma = 0.07$)	0.13	0.86
				negExpo (f = 1)	0.07	0.98
P388	2,297	64	30	normal ($\mu = 10$, $\sigma = 1.5$)	3.16	16.84
				negExpo (f = 18)	1.55	16.86
Yeast	9,567	125	26	normal ($\mu = 2$, $\sigma = 0.3$)	0.46	3.55
				negExpo (f = 3.6)	0.23	3.53

[1] http://www.cs.ucsb.edu/~xyan/dataset.htm.
[2] https://pubchem.ncbi.nlm.nih.gov/.

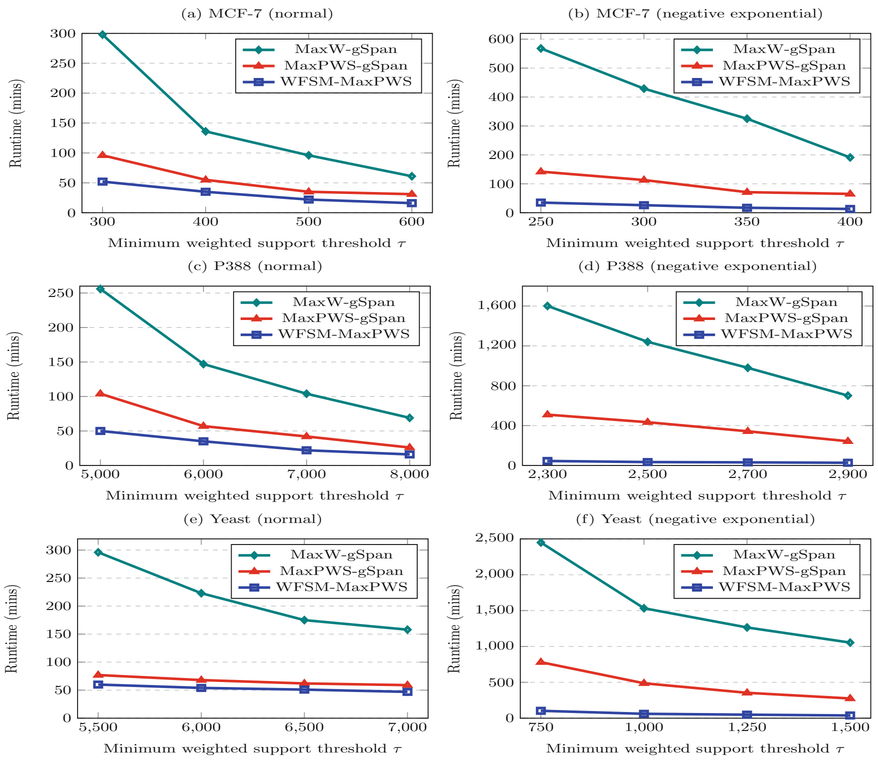

Fig. 4. Runtime

To analyze the performance of our proposed algorithm WFSM-MaxPWS (which uses Eq. (1) for the PWS calculation), we compared it with MaxPWS-gSpan (which uses Eq. (2) for the PWS calculation) and the baseline MaxW-gSpan algorithm (which simply uses the MaxW pruning technique) by using the three datasets in Table 1. We examined the following aspects: (i) runtime (ii) search-space reduction, and (iii) memory usage.

Figure 4 shows the **runtimes** on three datasets (each with two distributions). Here, both our WFSM-MaxPWS and MaxPWS-gSpan algorithms ran significantly faster than MaxW-gSpan. Between the former two, WFSM-MaxPWS ran faster than MaxPWS-gSpan, and the margin was wider for datasets having negative exponential weight distributions because WFSM-MaxPWS takes full advantage of distribution by considering FTW (instead of MaxW) in the MaxPWS calculation. When the weights follows positive exponential, WFSM-MaxPWS still ran faster than MaxPWS-gSpan or MaxW-gSpan, though the gap between the latter two was smaller.

A reason behind the runtime improvement of WFSM-MaxPWS over the other two algorithms is its efficiency in **search-space reduction**. Specifically, WFSM-MaxPWS generates a very small number of candidates when compared with the

Table 2. P388 candidate count

(a) Normal distribution

τ	WFS cnt	WFSM-MaxPWS	MaxPWS-gSpan	MaxW-gSpan
5,000	616	711	1,259	2,053
6,000	362	415	750	1,104
7,000	257	278	478	739
8,000	173	199	330	487

(b) Negative exponential

τ	WFS cnt	WFSM-MaxPWS	MaxPWS-gSpan	MaxW-gSpan
2,300	248	399	30,869	91,053
2,500	192	319	12,148	38,312
2,700	142	266	7,145	27,944
2,900	111	212	4,953	22,455

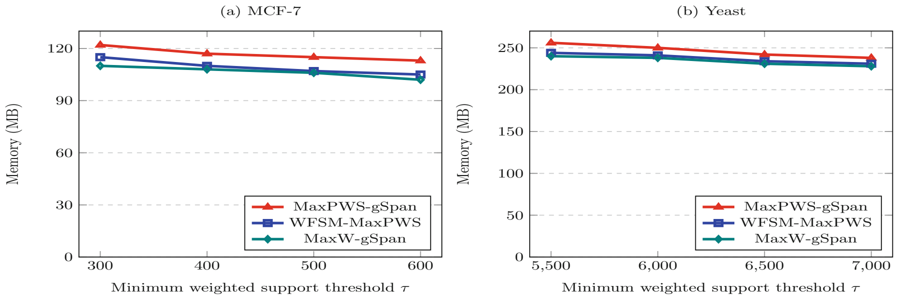

Fig. 5. Memory usage

other two algorithms. Table 2 shows the numbers of generated candidates for the P388 dataset. In the table, the second column shows the actual weighted frequent subgraph count (WFS cnt). The third to fifth columns show candidate counts for WFSM-MaxPWS, MaxPWS-gSpan and MaxW-gSpan. Our WFSM-MaxPWS generated the smallest number of candidates when compared with the other two algorithms, especially with negative exponential distribution. Such a reduction in search space effectively reduced runtime.

As for **memory usage**, both WFSM-MaxPWS and MaxPWS-gSpan required just slightly more memory than MaxW-gSpan due to the occurrence list storage. As shown in Fig. 5 on both MCF-7 and Yeast datasets, the slight increase in memory requirement was insignificant, especially when compared with fruitful benefits of reduction in both runtime and the number of generated candidates.

5 Conclusions

In this paper, we proposed a weighted frequent subgraph mining algorithm called WFSM-MaxPWS, which uses MaxPWS-measure to reduce search-space. MaxPWS is calculated using a modified canonical ordering for weighted graphs to achieve smallest possible upper bound of maximum possible weighted support for any extensions of a particular subgraph along with ensuring no loss of weighted frequent subgraph patterns. Experimental results and comparative analysis on three real datasets with normal and negative exponential distribution show that our WFSM-MaxPWS algorithm outperforms the existing MaxW-gSpan algorithm w.r.t. runtime and reduction in the number of generated candidates. Moreover, our modified canonical ordering for weighted graphs facilitates MaxPWS calculation to achieve even better performance. This concept of modified canonical ordering and MaxPWS pruning have potential to be further utilized in uncertain or utility-based graph databases.

Acknowledgement. This project is partially supported by NSERC (Canada) and U. Manitoba.

References

1. Ahmed, C.F., Tanbeer, S.K., Jeong, B.S., Lee, Y.K., Choi, H.J.: Single-pass incremental and interactive mining for weighted frequent patterns. Expert Syst. Appl. **39**(9), 7976–7994 (2012)
2. Babu, N., John, A.: A distributed approach to weighted frequent subgraph mining. In: ICETT 2016. IEEE (2016). https://doi.org/10.1109/ICETT.2016.7873705
3. Cheng, Z., Flouvat, F., Selmaoui-Folcher, N.: Mining recurrent patterns in a dynamic attributed graph. In: Kim, J., Shim, K., Cao, L., Lee, J.-G., Lin, X., Moon, Y.-S. (eds.) PAKDD 2017. LNCS (LNAI), vol. 10235, pp. 631–643. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57529-2_49
4. Eichinger, F., Huber, M., Böhm, K.: On the usefulness of weight-based constraints in frequent subgraph mining. In: Bramer, M., Petridis, M., Hopgood, A. (eds.) Research and Development in Intelligent Systems, vol. XXVII, pp. 65–78. Springer, London (2011). https://doi.org/10.1007/978-0-85729-130-1_5
5. Fariha, A., Ahmed, C.F., Leung, C.K.-S., Abdullah, S.M., Cao, L.: Mining frequent patterns from human interactions in meetings using directed acyclic graphs. In: Pei, J., Tseng, V.S., Cao, L., Motoda, H., Xu, G. (eds.) PAKDD 2013. LNCS (LNAI), vol. 7818, pp. 38–49. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37453-1_4
6. Gupta, A., Thakur, H., Gupta, T., Yadav, S.: Regular pattern mining (with jitter) on weighted-directed dynamic graphs. JESTEC **12**(2), 349–364 (2017)
7. Huang, Z., Ye, Y., Li, X., Liu, F., Chen, H.: Joint weighted nonnegative matrix factorization for mining attributed graphs. In: Kim, J., Shim, K., Cao, L., Lee, J.-G., Lin, X., Moon, Y.-S. (eds.) PAKDD 2017. LNCS (LNAI), vol. 10234, pp. 368–380. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57454-7_29
8. Jiang, C., Coenen, F., Zito, M.: Finding frequent subgraphs in longitudinal social network data using a weighted graph mining approach. In: Cao, L., Feng, Y., Zhong, J. (eds.) ADMA 2010. LNCS (LNAI), vol. 6440, pp. 405–416. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17316-5_39
9. Jiang, C., Coenen, F., Zito, M.: Frequent sub-graph mining on edge weighted graphs. In: Pedersen, T.B., Mohania, M.K., Tjoa, A.M. (eds.) DaWaK 2010. LNCS, vol. 6263, pp. 77–88. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15105-7_7
10. Lee, G., Yun, U.: Mining strongly correlated sub-graph patterns by considering weight and support constraints. IJMUE **8**(1), 197–206 (2013)

11. Lee, G., Yun, U., Kim, D.: A weight-based approach: frequent graph pattern mining with length-decreasing support constraints using weighted smallest valid extension. Adv. Sci. Lett. **22**(9), 2480–2484 (2016)
12. Leung, C.K.-S., Mateo, M.A.F., Brajczuk, D.A.: A tree-based approach for frequent pattern mining from uncertain data. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 653–661. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68125-0_61
13. Leung, C.K.-S., Tanbeer, S.K.: PUF-tree: a compact tree structure for frequent pattern mining of uncertain data. In: Pei, J., Tseng, V.S., Cao, L., Motoda, H., Xu, G. (eds.) PAKDD 2013. LNCS (LNAI), vol. 7818, pp. 13–25. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37453-1_2
14. Ozaki, T., Etoh, M.: Closed and maximal subgraph mining in internally and externally weighted graph databases. In: IEEE AINA 2011 Workshops, pp. 626–631 (2011)
15. Shinoda, M., Ozaki, T., Ohkawa, T.: Weighted frequent subgraph mining in weighted graph databases. In: IEEE ICDM 2009 Workshops, pp. 58–63 (2009)
16. Yan, X., Han, J.: gSpan: graph-based substructure pattern mining. In: IEEE ICDM 2002, pp. 721–724 (2002)
17. Yang, J., Su, W., Li, S., Dalkilic, M.M.: WIGM: discovery of subgraph patterns in a large weighted graph. In: SIAM SDM 2012, pp. 1083–1094 (2012)