

# **Efficient Large Scale Frequent Subgraph Mining Using MapReduce**

A Dissertation submitted in partial fulfillment of the requirement for  
the award of degree of

Master of Technology  
In Computer Science Engineering

Submitted By

**Vipul Kumar**

**2K15/CSE/21**

Under the guidance of

**Dr. Daya Gupta**



**Delhi Technological University**  
**Shahbad Daulatpur, Main Bawana Road**  
**Delhi – 110042**  
**June, 2017**

## CERTIFICATE



This is to certify that the dissertation entitled “**Efficient Large Scale Frequent Subgraph Mining Using MapReduce**” has been submitted by **Vipul Kumar, Roll No. 2K15/CSE/21**, in partial fulfillment of the requirement for the award of Master of Technology degree in Computer Science Engineering. This work is carried out by her under my supervision and has not been submitted earlier for the award of any degree or diploma in any university to the best of my knowledge.

**Dr. Daya Gupta**

**Project Guide**

**Department of Computer Science & Engineering**

**Delhi Technological University**

## DECLARATION

I hereby declare that the dissertation entitled “**Efficient Large Scale Frequent Subgraph Mining Using MapReduce**” which is being submitted to Delhi Technological University, in partial fulfillment of requirements for the award of degree of Master of Technology (Computer Science and Engineering) is a bona fide work carried out by me. The material contained in the report has not been submitted to any university or institution for the award of any degree.

**Vipul Kumar**

**2K15/CSE/21**

## ACKNOWLEDGEMENT

I would like to express my deep sense of respect and gratitude to my project supervisor Dr. Daya Gupta for providing the opportunity of carrying out this project and being the guiding force behind this work. I am deeply indebted to her for the support, advice and encouragement she provided without which the project could not have been a success.

Also, I would like to express gratitude to Mrs. Shruti Jaiswal (Research Scholar, Delhi Technological University) for providing me continuous support and guidance during this project.

I would also like to acknowledge Delhi Technological University library and staff for providing the right academic resources and environment for this work to be carried out. Last but not the least I would like to express sincere gratitude to my parents and friends for constantly encouraging me during the completion of work

**Vipul Kumar**

**2K15/CSE/21**

**M.Tech (Computer Science Engineering)**

**Department of Computer Engineering**

**Delhi Technological University Delhi – 110042**

## Abstract

Graph based data representations are getting popular in areas like bioinformatics, social networks, web data mining, etc. Over the years many algorithms have been created for analysis on graph data. One such challenging task in this field is Frequent Subgraph Mining (FSM). Extracting frequent subgraphs from a huge set of graphs is a fundamental task in numerous information mining applications.

There are existing frequent subgraph mining algorithms for unweighted graphs but they do not take into consideration the strength of relationships within the graph. In weighted graphs, some edges/vertices have more importance than others. In areas such as mobile communication networks, social networks, etc. weighted graphs are more useful. More relevant and specific subgraphs are generated through weighted frequent subgraph mining.

There has been only some little work done in the field of frequent subgraph mining on weighted graphs. Also most of the current techniques are memory-based and are not scalable. This work uses an existing distributed approach for Frequent Subgraph Mining using iterative MapReduce based framework and applies different weighing schemes over the current approach. This work uses two different weighing schemes, Average Total Weighing (ATW) scheme and Affinity Weighing (AW) scheme, and compares both approaches.

# Table of Contents

CERTIFICATE .....	i
DECLARATION .....	ii
ACKNOWLEDGEMENT .....	iii
Abstract .....	iv
Table of Contents .....	v
List of Figures .....	viii
List of Tables .....	ix
Chapter 1: Introduction .....	1
1.1. Overview .....	1
1.2. Basic Concepts .....	2
1.2.1. Graph.....	2
1.2.2. Graph Isomorphism .....	2
1.2.3. Support.....	3
1.2.4. Frequent Subgraph Mining .....	3
1.2.5. MapReduce .....	4
1.3. Motivation .....	7
1.4. Related Work.....	8
1.5. Problem Statement .....	8
1.6. Scope of Work.....	9
	v

1.7. Thesis Organization.....	9
Chapter 2: Literature Review .....	11
2.1. Frequent Subgraph Mining.....	11
2.2. Distributed Frequent Subgraph Mining.....	17
2.3. Frequent Subgraph Mining in Weighted Graphs .....	18
Chapter 3: Frequent Subgraph Mining .....	20
3.1. Simple in-memory frequent subgraph mining : gSpan .....	20
3.2. Basic Weighted Frequent Subgraph Mining Algorithm .....	21
3.3. Distributed Frequent Subgraph Mining .....	23
3.4. Graph Weighting Mechanisms.....	25
3.4.1. Average Total Weighting (ATW) Scheme .....	25
3.4.2. Affinity Weighting (AW) Scheme.....	26
Chapter 4: Proposed Method .....	28
4.1. Distributed weighted subgraph mining using ATW weighing scheme.....	28
4.2. Distributed weighted subgraph mining using AW weighing scheme .....	32
Chapter 5: Implementation and Results.....	36
5.1. Hardware Details.....	36
5.2. Software Details .....	36
5.3. Dataset.....	36
5.4. Results .....	38
Chapter 7: Conclusions and Future Work.....	41

References..... 42



## List of Figures

Figure 1.1: Graph Isomorphism [5] .....	2
Figure 1.2: Support Counting [5].....	3
Figure 1.3 : (a) A sample Graph database (b) Frequent subgraphs with support as 2.....	4
Figure 1.4: MapReduce pipeline [9] .....	5
Figure 1.5: MapReduce data flow [9].....	6
Figure 6.1: Number of frequent patterns for Unweighted, ATW and AW schemes .....	38
Figure 6.2: Number of patterns in ATW and AW for different $r(g)$ .....	39
Figure 6.3: Running time of ATW and AW for different different support values with $r(g) = 0.2$ .....	40

## List of Tables

Table 5.1: Description of Data.....	37
Table 5.2: Description of Graph Dataset .....	37

## Chapter 1: Introduction

This chapter provides the overview of frequent subgraph mining, motivation and problem statement of the thesis. The chapter explains some basic concepts of the domain and describes the scope of the work. The organization of this thesis is described at the end of this chapter.

### 1.1. Overview

Graph representation is expressive in nature and thus can be used to represent complex data and relationships in numerous real world scenarios. Over the years, many techniques have been designed to extract and discover useful knowledge from the complex data represented as graphs. Techniques like clustering, classification, searching, indexing, pattern finding, etc are used across different domains.

Graph mining techniques are used in a variety of domains such as chemoinformatics, bioinformatics, and social networks. Research on association rule mining has motivated many researchers in the field of frequent pattern mining [1].

Graph mining deals with pattern identification from data in graph form. Frequent subgraph mining is a form of graph mining used to find patterns (subgraphs) that occur frequently in a single large graph or a set of small graphs.

Due to its wide range of applications in the above mentioned domains, frequent subgraph mining has been studied by a lot of researchers [2] [3]. Different functions and relations can be understood through frequent patterns. Frequent patterns can show a close friend circle in the field of social networks. Similarly, frequent patterns can help discover unknown functions of a protein in a protein-protein interaction network (PPI).

In the case of weighted graphs, weighted support function is used to identify weighted frequent subgraph mining. The main issue with weighted subgraph mining is that the anti-monotone property would not be applicable if the weights are assigned in normal manner. Anti-monotone property states that if a subgraph of size  $K$  is not frequent, then all of its

supergraphs of size  $K+1$  would not be frequent. This is important for restricting search space size while matching patterns. Thus, the weighting strategy should be created in such a way that anti-monotone property holds true. [4]

## 1.2. Basic Concepts

### 1.2.1. Graph

A graph is defined as  $G = (V, E)$ , where  $V$  is a set of vertices and  $E \subseteq V \times V$  is a set of edges. We denote the label for vertex  $v \in V$ , by  $L(v)$  and the label for the edge  $(v_1, v_2) \in E$  by  $L(v_1, v_2)$ . In our context, we will use  $P = (V_P, E_P)$  for a pattern graph and  $G$  represents the input graph.

### 1.2.2. Graph Isomorphism

We say that the pattern  $P$  is subgraph isomorphic to  $G = (V, E)$ , denoted as  $P \subseteq G$ , if there exists an injective function,  $\phi : V_P \rightarrow V$  such that: 1)  $\forall v \in V_P, L(v) = L(\phi(v))$ , and 2)  $\forall (v_i, v_j) \in E_P, (\phi(v_i), \phi(v_j)) \in E$  and  $L(v_i, v_j) = L(\phi(v_i), \phi(v_j))$ . In this case, the isomorphic subgraph in  $G$  comprising the vertices  $\phi(v_1), \phi(v_2), \dots, \phi(v_p)$  (where  $p = |V_P|$ ) is also called an embedding of the pattern  $P$  in the input graph  $G$ . An example of graph isomorphism is showed in figure 1.1.

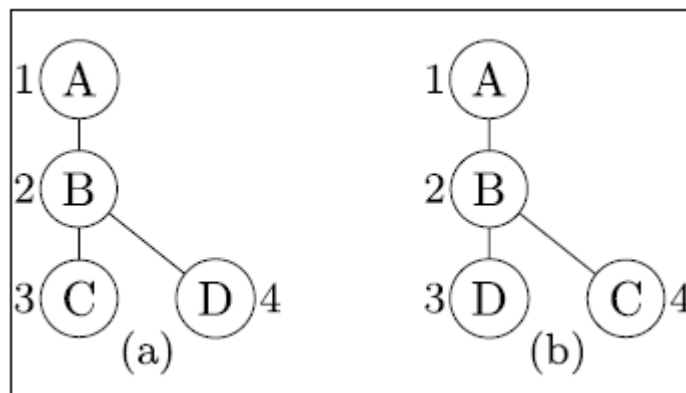


Figure 1.1: Graph Isomorphism [5]

### 1.2.3. Support

Let  $\Sigma(P) = \{\phi_1, \phi_2, \dots\}$  denote the set of all isomorphic graphs of pattern graph, P in input graph, G. Some function  $\Sigma(P)$  can be used to define the support of pattern, P, such as cardinality. However, in this case cardinality will not satisfy the anti-monotonic property as the support of subgraph cannot be less than the support of pattern. In order to tackle this problem, Kuramochi and Karypis (2005) proposed a technique using maximum independent set of overlapping in subgraphs [6]. But this technique is a NP-Hard problem. [7]. Figure 1.2 shows an example of support counting for the isomorphic graphs shown in figure 1.1.

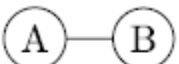
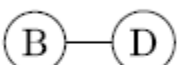
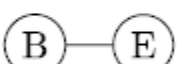


Pattern	Occerence List (OL)
	1 : [(1, 2)] ; 2 : [(1, 2)]
	1 : [(2, 4)] ; 2 : [(2, 3)] ; 3 : [(1, 2)]
	2 : [(2, 5)] ; 3 : [(1, 3)]
	1 : [(1, 2), (2, 4)] ; 2 : [(1, 2), (2, 3)]
	2 : [(1, 2), (2, 5)]

Figure 1.2: Support Counting [5]

### 1.2.4. Frequent Subgraph Mining

A graph database consisting of labeled, connected, simple and undirected graphs is given by  $G = \{G_1, G_2, \dots, G_n\}$ . The size of a graph, g is the number of edges it has. The *support-set* of the graph g is given by  $t(g) = \{G_i : g \subseteq G_i \in G_g\}, \forall i = \{1 \dots n\}$ . The *support-set* consists of all isomorphic subgraphs of g in the input graph database G. The *support* of g is given by the cardinality of the *support-set*. If  $support \geq \pi^{\min}$  then g is called frequent.

$\pi^{\min}$  is the *minimum support* threshold which can be user defined or predefined.  $F$  represents the set of frequent patterns.

**Example.** A graph database is showed in Fig. 1.3(a) with graphs ( $G_1$ ,  $G_2$  and  $G_3$ ). In Fig. 1.3(b), the frequent subgraphs of the database with minimum support threshold of 2,  $\pi^{\min}=2$  are shown.

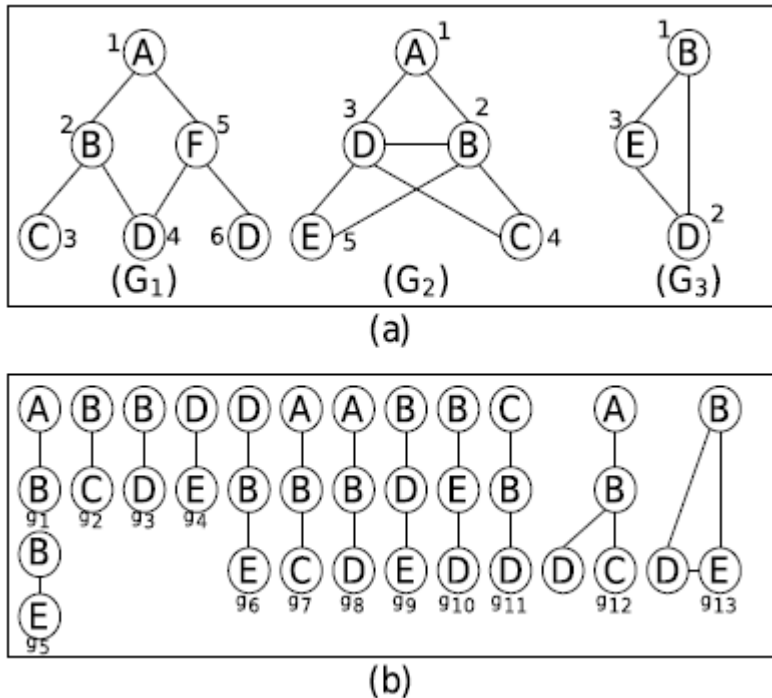


Figure 1.3 : (a) A sample Graph database (b) Frequent subgraphs with support as 2.

### 1.2.5. MapReduce

MapReduce [8] is a programming model initially created by Google for parallel and distributed execution of code across multiple clusters. It is composed of Map() and Reduce() methods. A worker node in a cluster can be either a mapper or a reducer, as per its role in the computation. The mapper nodes filter and sort the data and output key-value pairs. The reducer nodes summarizes the output of mapper nodes by aggregating the values

with same key and writes the output to a file. A distributed file system manages all the files of MapReduce.

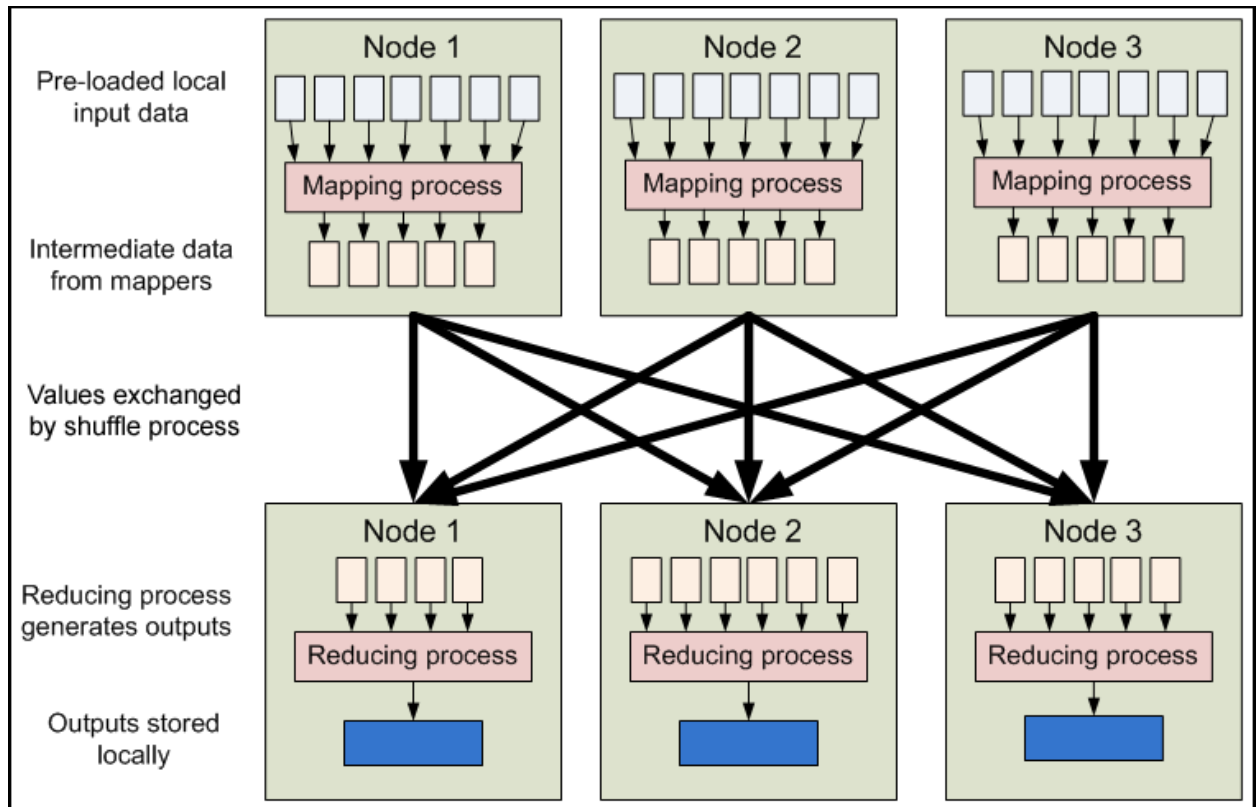


Figure 1.4: MapReduce pipeline [9]

Figure 1.4 shows the high-level pipeline of MapReduce. The input comes from files loaded in the HDFS. The mapper task runs on each node. Each mapper loads and processes the files local to it. After mapping, the intermediate values are shuffled and distributed among all the reducer nodes. This is the only communication step in MapReduce. Individual mappers do not communicate with each other. [9]

The detailed data flow of Hadoop MapReduce pipeline is given below in figure 1.5. Only two nodes are shown in the figure but the same pipeline can be replicated for a large number of nodes.

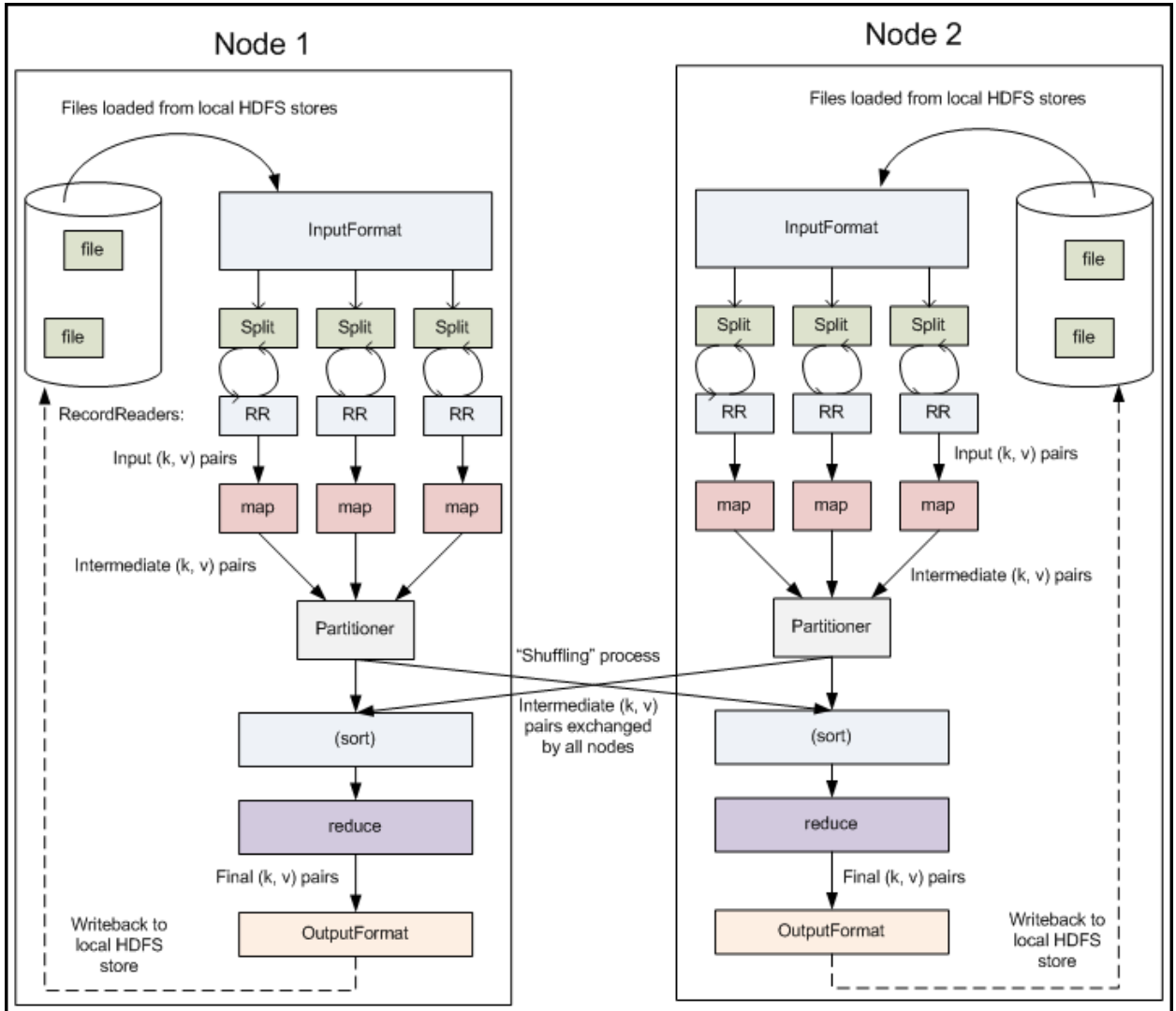


Figure 1.5: MapReduce data flow [9]

### 1.2.6. Iterative MapReduce

In iterative MapReduce [10] the mapper and reducer are run multiple times with some minor changes every time. The mapper in each iteration uses the output of the reducer from previous iteration as input. The termination of the loop is governed by an external condition. The pseudo code of the method is shown in the algorithm given below.



### Algorithm: Iterative MapReduce Algorithm

#### **MapReduce\_Itr():**

1. As long as condition is true:
2.     Perform MapReduce
3.     Record output on DFS
4.     Update counters and the condition

### 1.3. Motivation

Frequent pattern mining is a fundamental task in many areas of chemistry, biology and networks. For example, social scientists can find closely related communities through frequent subgraph mining. Similarly, bioinformatics researchers can use it to find common subgraphs in protein structures. Graph indexing, clustering and classification are some other graph applications.

Frequent subgraph mining has been implemented by many researchers [11] [12] using MapReduce framework [13], as well as in grid environments [14]. Most of the popular subgraph mining algorithms assume that the memory size is enough to fit the whole graph. This approach works on graphs of small size but proves inefficient as the graph size increases. To solve this issue, some database based implementations [15] [16] [17] were proposed. However, it is not scalable and computation time increases as the dataset size increases.

There are many frequent subgraph mining algorithms for unweighted graphs but very less work has been done in the field of weighted graphs. There are numerous fields where the relationship between entities is important such as social networks, transportation networks, etc. For example, the users of a social network are all connected to each other but still some relationships are stronger than others. In such situations, weighted graphs prove to be more useful than unweighted graphs. The use of weighted frequent subgraph mining in weighted-graph classification, logistics and software defect localization has been showed in [18]. However, in the field of weighted graphs, there are still very few implementations of frequent subgraph mining [4].

#### 1.4. Related Work

In [19], the authors have proposed an iterative MapReduce based method for extracting significant patterns in labeled graphs. The method worked for both directed and undirected graphs. Using the work in [19], Bhuiyan and Al Hasan created a framework for frequent subgraph mining using iterative MapReduce called FSM-H [5]. This framework is ten times more efficient than the work done in [19].

Babu and John in 2016 [20] applied FSM-H on weighted graphs. The authors used Average Total Weighting method for mining frequent patterns in the PubChem database [21].

#### 1.5. Problem Statement

Frequent subgraph mining over MapReduce framework is a challenging problem. The support of the subgraph needs to be calculated over the complete graph database. Also, in a distributed environment, the input graphs would be distributed across various machines. In such a case, the node storing the graph can only compute its local support which is not enough to determine if the subgraph is frequent.

In MapReduce framework, there is not built-in method for accessing global data structures. Thus, the local support computed cannot be stored in global variables. It is also not feasible to delay the support computation. According to Apriori principle [22], only frequent patterns can generate future candidate patterns.

Most of the existing FSM algorithms are for unweighted graphs. They assume that all the frequent subgraphs are equally important. However, the work done in this thesis assumes that some subgraphs have higher importance due to the weight associated with the edges.

In this thesis, weighting techniques are applied in an unweighted frequent subgraph mining approach. Non-negative real values are assigned to the edges of the input graphs. By doing this, a subset of more significant subgraphs can be identified.

The research question posed in this thesis is thus:

**“Proposing a weighted frequent subgraph mining method using Affinity Weighting technique in a distributed environment.”**

## 1.6. Scope of Work

In this thesis, a MapReduce based approach is presented for weighted-frequent subgraph mining. The presented method is efficient than current methods as both data and computation are distributed. The method takes a weighted graph dataset as input and finds the significant weighted frequent subgraphs.

The system runs in an iterative way. The output of the previous iteration is the input for the next iteration. The input of the mapper of the current step comes from the output of the reducer of the previous step.

At each  $n^{\text{th}}$  iteration, the mapper creates subgraphs of size  $n$ . The local support of the subgraph is also calculated by the mapper during each iteration.

At each  $n^{\text{th}}$  iteration, the globally frequent subgraphs of size  $n$  are identified by the reducer using their local supports.

The performance is evaluated using a Facebook-like Social Network [65] database. The performance of different weighting techniques is compared.

The scope of work can be summarized as:

- Proposing a distributed method for mining weighted frequent subgraphs
- Designing mapper and reducer functions for affinity based weighting techniques
- Performance evaluation on a real biological dataset
- Performance comparison of different weighting techniques

## 1.7. Thesis Organization

Further thesis is organized as follows:

Chapter 2 presents the literature review of existing frequent subgraph mining methods.

Chapter 3 explains different graph weighting techniques used in this thesis.

Chapter 4 provides the detailed description of proposed method for Frequent Subgraph Mining using iterative MapReduce method.

Chapter 5 describes the implementation details of this research work.

Chapter 6 lists the results and evaluates the proposed system. It also compares the performance of the proposed system with other weighting techniques.

Chapter 7 concludes the thesis and discusses the possible improvements in this research work in future.

## Chapter 2: Literature Review

In this section, first we discuss the work in the field of frequent subgraph mining and distributed frequent subgraph mining followed by work related to frequent subgraph mining in weighted graphs.

### 2.1. Frequent Subgraph Mining

gSpan [12], AGM [23], FSG [24], Gaston [25], and DMTL [26] are some notable frequent subgraph mining algorithms. These algorithms are in-memory based. They work on small datasets. The time taken to complete the mining task is also not too much.

There are mainly two types of FSM algorithms: (a) Pattern Growth based approach, or (b) Apriori based approach.

Frequent subgraph mining algorithms based on pattern growth approach are discussed below:

Subdue was introduced by Nikhil S Ketkar et al in 2005 [27]. It takes a single large graph as input represented by adjacency matrix. It searches level-wise and generates subgraphs. It uses minimum description code length string for frequency counting and generates complete set of frequent subgraphs. The only limitation to this approach is that the number of patterns is extremely small.

gSpan [12] takes a graph dataset as input which are represented by adjacency list. It uses DFS order for frequency counting and generates the frequent graphs. The limitation to this approach is that it is not scalable.

In the year 2003, Yan proposed Close Graph [28] method. It takes a graph dataset as input which are represented by adjacency list. It uses DFS order for frequency counting and

generates close connected frequent graphs. The limitation to this approach is that failure detection takes a lot of time overhead.

Gaston is a popular subgraph mining tool created by Nijssen in 2004 [25]. It takes a graph dataset as input which are represented by hash table. It uses embedding lists for frequency counting and generates maximal frequent subgraphs. The limitation to this approach is that some interesting patterns may be lost.

TSP takes a graph dataset as input which are represented by adjacency list. It uses TSP tree for frequency counting and generates closed temporal frequent subgraphs. But there is an extra overhead to check whether temporal patterns are closed [29].

MOFA [30] was proposed by Berthold for mining molecular fragments. It takes a graph dataset as input which are represented by adjacency list. It uses DFS order for frequency counting and generates all frequent subgraphs. There is always a possibility of error in this method. The frequent subgraphs generated may not be exactly frequent.

RP-FP method proposed by Li [31] takes a graph dataset as input which are represented by adjacency list. It uses DFS order for frequency counting and generates representative graphs. The limitation to this approach is that the time summarizing the patterns is more than that for mining.

RP-GD is just like RP-FP. It takes a graph dataset as input which are represented by adjacency list. It uses DFS order for frequency counting and generates representative graphs. The limitation to this approach is that the time summarizing the patterns is more than that for mining [31].

JPMiner takes a graph dataset as input which are represented by adjacency list [32]. It uses DFS order for frequency counting and generates frequent jump patterns. Sometimes the set of jump patterns is too small.

Mspan takes a graph dataset as input which are represented by adjacency list. It uses DFS order for frequency counting and generates frequent subgraphs. [33]

Frequent subgraph mining algorithms based on apriori based approach are discussed below:

Table 2.1: Pattern growth based frequent subgraph mining algorithms

<u>Name</u>	<u>Input</u>	<u>Graph Representation</u>	<u>Frequency Counting Technique</u>	<u>Output</u>	<u>Limitation</u>
Subdue	Single large graph	Adjacency matrix	Minimum description code length string	Set of frequent subgraphs	Extremely small number of patterns
gSpan	Set of graphs	Adjacency list	DFS order	Frequent subgraphs	Not scalable
CloseGraph	Set of graphs	Adjacency list	DFS order	Close connected frequent graphs	Overhead time
Gaston	Set of graphs	Hash Table	Embedding lists	Maximal frequent subgraphs	Some interesting patterns may be lost
TSP	Set of graphs	Adjacency list	TSP tree	Closed temporal frequent subgraphs	Extra overhead to check if temporal patterns are closed
MOFA	Set of graphs	Adjacency list	DFS order	Frequent subgraphs	Frequent subgraphs generated may not be exactly frequent

RP-FP	Set of graphs	Adjacency list	DFS order	Representative graphs	Time spent summarizing the patterns is more than that for mining
RP-GD	Set of graphs	Adjacency list	DFS order	Representative graphs	Time spent summarizing the patterns is more than that for mining
JPMiner	Set of graphs	Adjacency list	DFS order	Frequent jump patterns	Sometimes the set of jump patterns is too small

In 2001, Nijssen proposed another subgraph mining method called Farmer [34]. It takes a graph dataset as input which are represented by a trie structure. Potential candidates are generated through level-wise search. It uses the trie data structure for frequency counting and generates frequent subgraphs. But this method is not efficient.

At the same time in 2001, Karypis [24] proposed another method called FSG. It takes a graph dataset as input which are represented by adjacency list. Potential candidates are generated through one edge extension. It uses transaction identifier (TID) lists for frequency counting and generates frequent connected subgraphs. This is a NP-complete algorithm.

HSIGRAM [35] takes a single large graph as input which is represented by adjacency matrix. Potential candidates are generated through iterative merging. It uses maximal independent sets for frequency counting and generates frequent subgraphs. But this approach is not much efficient.



Again in 2004, Karypis proposed another apriori based subgraph mining method, GREW [36]. It takes a single large graph as input which is represented by sparse graph representation. Potential candidates are generated through iterative merging. It uses maximal independent sets for frequency counting and generates maximal frequent subgraphs. The limitation to this approach is that it misses many interesting patterns.

FFSM, proposed by Huan in 2003 [37] is a very popular subgraph mining algorithm. It takes a graph dataset as input which are represented by adjacency matrix. Potential candidates are generated through merging and extension. It uses sub-optimal canonical adjacency matrix tree for frequency counting and generates frequent subgraphs. This is a NP-complete algorithm.

ISG is itemset based subgraph mining [38]. It takes a graph dataset as input which are represented by edge triplets. Potential candidates are generated through edge triplet extension. It uses transaction identifier (TID) lists for frequency counting and generates maximal frequent subgraphs. The set of graphs generated is incomplete.

Spin, also proposed by Huan [39], takes a graph dataset as input which are represented by adjacency matrix. Potential candidates are generated through join operations. It uses canonical spanning tree for frequency counting and generates maximal frequent subgraphs. Non-maximal graphs can also be found but needs an entire database scan.

Dynamic GREW [36] is a modification of GREW. It takes dynamic graphs as input which are represented by sparse graph representation. Potential candidates are generated through iterative merging. It uses suffix trees for frequency counting and generates dynamic patterns in frequent subgraphs. The limitation to this approach is that there is an extra overhead for identifying dynamic patterns.

AGM [23] is an apriori based graph mining method. It takes a graph database as input which is represented by adjacency matrix. Potential candidates are generated through

vertex extension. It uses canonical labelling for frequency counting and generates frequent subgraphs.

MUSE was proposed in 2010 by Zou [40]. It takes an uncertain set of graphs as input which are represented by adjacency matrix. Potential candidates are generated through disjunctive normal forms (DNF). It uses DFS order for frequency counting and generates frequent subgraphs. The limitation to this approach is that the frequent subgraphs are not exact.

DB-FSG [16], OOFSG [17], and DB-Subdue [41] have also been proposed for mining large datasets. These algorithms are traditional database based.

Table 2.2: Apriori based frequent subgraph mining algorithms

<u>Name</u>	<u>Input</u>	<u>Graph Representation</u>	<u>Candidate Generation</u>	<u>Frequency Counting Technique</u>	<u>Output</u>	<u>Limitation</u>
Farmer	Set of graphs	Trie structure	Level-wise search	Trie data structure	Frequent subgraphs	Not efficient
FSG	Set of graphs	Adjacency list	One edge extension	Transaction identifier (TID) lists	Frequent connected subgraphs	NP-complete algorithm
HSIGRAM	Single large graph	Adjacency matrix	Iterative merging	Maximal independent sets	Frequent subgraphs	Not efficient
GREW	Single large graph	Sparse graph representation	Iterative merging	Maximal frequent subgraphs	Maximal frequent subgraphs	Misses many interesting patterns
FFSM	Set of graphs	Adjacency matrix	Merging and extension	Sub-optimal canonical adjacency matrix tree	Frequent subgraphs	NP-complete algorithm
ISG	Set of graphs	Edge triplets	Edge triplet extension	Transaction identifier (TID) lists	Maximal frequent subgraphs	Incomplete set of graphs generated

Spin	Set of graphs	Adjacency matrix	Join operations	Canonical spanning tree	Maximal frequent subgraphs	Entire database scan needed for not maximal graphs
Dynamic GREW	Dynamic graphs	Sparse graph representation	Iterative merging	Suffix trees	Dynamic patterns in frequent subgraphs	Extra overhead for identifying dynamic patterns
MUSE	Uncertain set of graphs	Adjacency matrix	Disjunctive normal forms (DNF)	DFS order	Frequent subgraphs	Frequent subgraphs are not exact

Shared memory parallel algorithms have also been designed by researchers for large-scale graph mining. A parallel version of the frequent subgraph mining algorithm, Subdue [42] was presented by Cook et al. A parallel toolkit [43] for Motif-Miner [44] algorithm was also developed by Wang et al.

Parmol [45] is a parallel implementation of Mofa [30], gSpan [12], FFSG [37] and Gaston [25]. gSpan also has a parallel implementation called ParSeMis [46].

PartMiner [47] and PartGraphMining [48] partition the graph dataset in order to scale up the size of input dataset. For CMP Architectures, there is a work [49] on adaptive parallel graph mining.

## 2.2. Distributed Frequent Subgraph Mining

Frequent patterns can be mined through MapReduce framework. It has been used for input datasets of sets [50], [51], [52], [53], and sequences [54].

In [19], the approach used by the author for mining using MapReduce is not efficient. It has various shortcomings. There is no way to generate unique patterns in their method. There may be duplicate patterns. Thus, leading to an increase in the size of search space. Also, there will be duplicates in output. Such output is hard to unify. A separate subgraph isomorphism method would be required for this. This method does not determine the number of MapReduce iterations by itself. The number of iterations are fed by the user.

Induced occurrences of subgraphs in a single large graph is also used to mine subgraphs in some existing works [55], [56].

### 2.3. Frequent Subgraph Mining in Weighted Graphs

In the area of weighted frequent subgraph mining, very little work has been done yet. However, a lot of work has been done in the field of weighted sequence mining (WSM) and weighted association rule mining (WARM) [57].

The concept of “support confidence” and “weight confidence” was introduced by Yun and Leggett [57] in 2005. In their work, they integrated weight and support confidences in the mining method. They identified patterns with similar weight and support levels through this approach.

A study done by Eichinger et al [18] showed that frequent subgraph mining with weights can provide more accurate results. They have explained the constraints of weight based work. Integration of weights in mining algorithms has also been demonstrated in this paper.

Real-world problems like weighted-graph classification, logistics and software defect localization are discussed in this study. The precision increased and the runtime decreased in this study. The results displayed better explorative logistics.

Jiang et al [4] proposed another work on weighted frequent subgraph mining. They introduced weighting mechanisms namely: Average Total Weighting (ATW), Affinity Weighting (AW) and Utility Based Weighting (UBW). By identifying the most significant subgraphs, these weighting schemes led to a reduction in the overall search space and improved efficiency and accuracy.

In order to verify the relevance of the results, the weighting techniques were applied to classification problems. The experiment was conducted on two data sets. A text mining data set [58] and an MRI scan data set [59] were used in the experiments. The results proved that the classification precision improved by using the weighting techniques.

Conclusion: A lot of work has been done in the field of Frequent Subgraph Mining. Weighting techniques have been explored to mine weighted graphs. MapReduce based techniques have been implemented which have made mining scalable and more efficient. However distributed mining of weighted graphs is still an unexplored territory. Thus, this work proposes a MapReduce based technique for mining patterns in weighted graphs.

## Chapter 3: Frequent Subgraph Mining

Frequent subgraph mining has been a hot topic of interest for many years. Different researchers have proposed different algorithms and approaches for frequent subgraph mining. In this chapter, some basic frequent subgraph mining approaches are discussed.

### 3.1. Simple in-memory frequent subgraph mining : gSpan

gSpan is one of the most popular frequent subgraph mining algorithm. It runs on a single machine and uses a sparse adjacency list representation to store graphs. It uses DFS lexicographical ordering for frequency counting. The algorithm is discussed below:

**Algorithm:** gSpan algorithm – single node, in-memory approach

```
gSpan(DataSet, ResultSet):  
1.   Sort DataSet by frequency  
2.   Eliminate non-frequent edges/vertices from DataSet  
3.   ResultSet1 = all frequent single edge graphs in DataSet  
4.   Sort ResultSet1 in DFS order  
5.   ResultSet = ResultSet1  
6.   For each edge in ResultSet1  
7.       sol = all graphs that contain edge  
8.       MineSubgraph(DataSet, ResultSet, sol)  
9.       DataSet = DataSet - edge  
10.      If |DataSet| < minimumSupport  
11.          break
```

**Algorithm:** Subprocedure for mining in gSpan algorithm

```
MineSubgraph(DataSet, ResultSet, sol):  
1.   if sol ≠ min(sol)  
2.       return  
3.   ResultSet = ResultSet U {sol}  
4.   Find sol in each graph in DataSet and count its children  
6.   For each child of sol  
7.       If support(child) > minimumSupport  
8.           sol = child  
9.           MineSubgraph(DataSet, ResultSet, sol)
```

Consider a graph with edges {A, B, C, ....}. Lines 6-11 will discover all the frequent subgraphs containing an edge A---A in the first iteration. All frequent subgraphs containing edge A---B, but not A---A will be identified in second iteration. The loop continues until all frequent subgraphs are identified. As the loop continues the size of the DataSet decreases (line 9). MineSubgraph is called recursively. It expands the graph and finds its children. It ends when the support of graph is less than minimum Support or the DFS code is not minimum. [12]

### 3.2. Basic Weighted Frequent Subgraph Mining Algorithm

There are two main steps in unweighted frequent subgraph mining. First is to construction of candidate subgraphs. Second is the frequency counting of candidate subgraphs. But in weighted graphs, the candidates generated also have weights which tell their relevance. In weighted frequent subgraph mining, the importance of subgraphs is identified by a weighting function. It means that even though a subgraph may have higher frequency, it might not be that much relevant for the results. Similarly, a subgraph having lower frequency may be more relevant.

A generic weighted frequent subgraph mining algorithm is described below:

**Algorithm:** Basic weighted frequent subgraph mining algorithm

```

Weighted_FSM(WG, support_threshold):
0. //n = 1
1. Populate WFP1
2. While WFPn has elements
3.     Cn+1 = Generate_Candidate_Set (WFPn, WG)
4.     Forall candidate, c in candidate set Cn+1
5.         If Isomorphism_Check(c) = true
6.             Weighted_Support_Counting(c, WG)
7.             If support ≥ support_threshold
8.                 Add candidate, c to WFPn+1
9.     Increment n by 1
10. Return WFP

```

The algorithm uses breadth-first candidate enumeration. It is a candidate-generation-and-test based algorithm. The algorithm starts with single edge patterns (frequent patterns of size 1) denoted as  $WFP_1$  (Line 1). Using a while loop (Line 3-9), the algorithm finds  $WFP_2$ ,  $WFP_3$  and so on. The loop continues till the complete weighted frequent pattern set (WFP) is created. The loop repeats if at the end of iteration  $WFP_n$  is non-empty. It generates candidate subgraphs of size  $n+1$  from each of the frequent patterns in  $WFP_n$  (Line 3). The candidate patterns are denoted by  $C$ . The algorithm calculates the candidate's support against the dataset  $WG$  (Line 6). The generated pattern is frequent if its support is greater than  $support\_threshold$  and is stored in the set  $WFP_{n+1}$  (Line 8). The algorithm makes sure that it only processes unique candidate patterns (Line 5). The while loop (line 2 to 9) continues till all frequent patterns of size  $n+1$  are obtained. During  $i^{th}$  iteration, frequent patterns of size  $i+1$  are obtained, and it repeats until all the frequent subgraphs are generated.

The algorithm has three main steps:

### **1. Generation of Candidate Set**

Suppose  $c$  is a frequent pattern of size  $n$ . During candidate generation, a frequent edge (from  $WFP_1$ ) is added to  $c$  and a new candidate  $d$  of size  $n+1$  is created. If a new vertex is present in the new candidate, then the new edge is a forward edge, otherwise a backward edge. Backward edge connects existing vertices of the candidate subgraph. In case of a forward edge, the new vertex is assigned an integer id. This id is larger than ids of any other vertices. Thus the vertex id represents the order of their addition to the candidate subgraph.

### **2. Isomorphism Check**

There can exist multiple paths by which a candidate is generated. But only one path should be processed during the mining process to avoid duplicity. Thus, isomorphism check is required during subgraph mining so that the duplicate copies of the candidate subgraphs can be ignored. Canonical coding scheme is a popular



way of checking graph isomorphism. In this method, the edges of the graph are serialized in a specific order and a string is generated. All isomorphic graphs generate same strings. *min-dfscode* is one such canonical coding scheme [12]. *min-dfscode* checks the generation path of pattern. If the edge ordering and insertion ordering in the generation path are same, then it is considered to be correct. Otherwise it is a duplicate and is ignored.

### **3. Support Counting**

Support counting is a fundamental task in frequent subgraph mining. It determines whether the generated subgraph is frequent. We need to count the occurrence of the subgraph in all the input graphs for finding its support. This a NP-complete problem. Support functions differ for different weighting schemes.

#### **3.3. Distributed Frequent Subgraph Mining**

Distributed weighted FSM algorithm [5] has four phases:

##### **1. Input Refining**

During the refining stage, the pre-processing is done. Firstly, the graph weights are computed. Then weight frequent edges are determined. As per the anti-monotone property, non-frequent edges are discarded, as non-frequent subgraphs cannot be used to generate frequent subgraphs. For further processing, only frequent edges are kept.

##### **2. Input Splitting**

During the splitting phase, the input data is split into chunks and distributed across Hadoop Distributed File System (HDFS). The input file is split in such a way that almost equal number of edges are present across all the chunks. This helps in load balancing.

### 3. Initialization

The data structures are initialized during initialization step. Data structures required for support computation and map and reduce jobs for candidate generation are initialized. A data structure is needed to maintain the list of all the vertices and all the possible extensions from those vertices for candidate generation.

### 4. Mining

This is the actual mining phase. Iterative process is used to find weighted frequent subgraphs. The first iteration starts with the single length weighted frequent subgraphs (size=1).

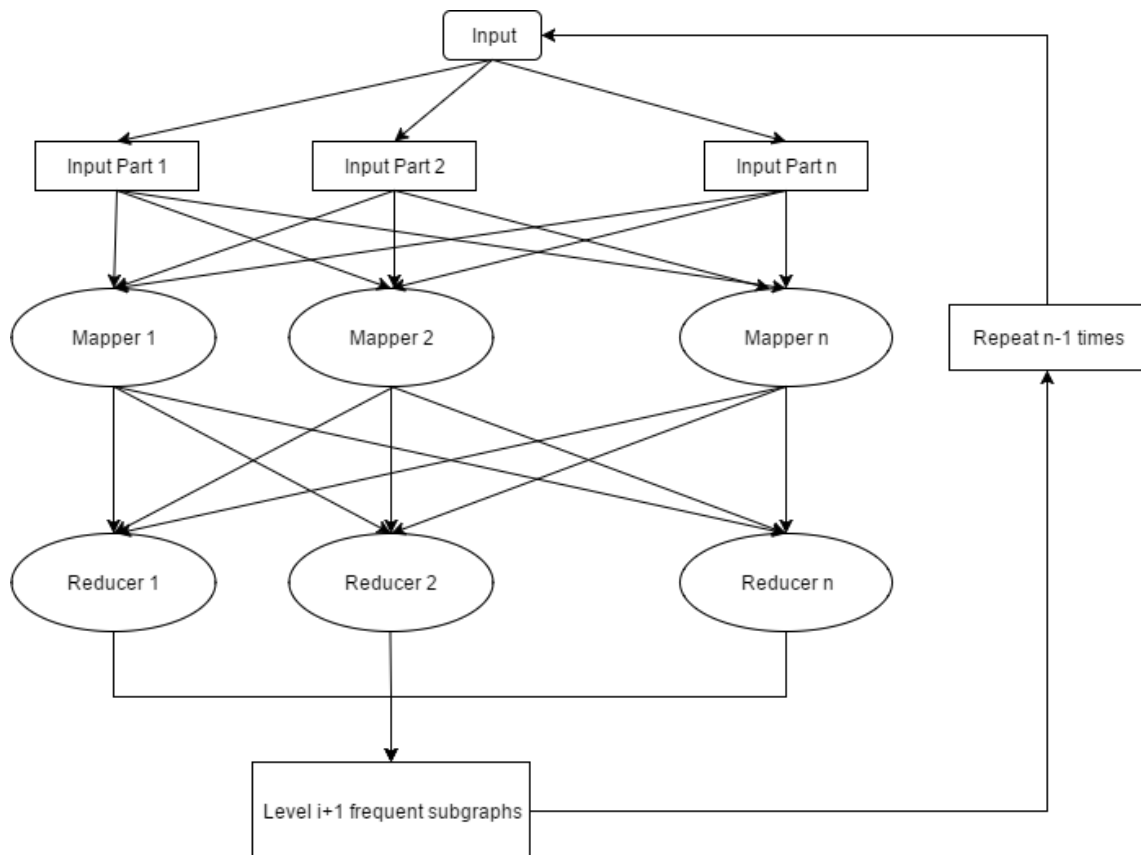


Figure 3.1: Workflow of complete mining process

Figure 4.1 presents the complete workflow of all the steps. Input data is first refined. The weighted frequent edges are identified. Then the input goes through the splitting phase and is filtered and split across various chunks.

Then during the preparation phase, the data structures are initialized. Finally, the mining starts. At each  $i^{\text{th}}$  iteration, subgraphs of size  $i+1$  are generated. The mining process repeats until all frequent subgraphs of size  $i+1$  are identified.

### 3.4. Graph Weighting Mechanisms

In this section, the graph weighing mechanisms used in this thesis are described. In the context of weighted FSM, the weights associated with a subgraph pattern,  $g$  can be defined in a number of ways.

#### 3.4.1. Average Total Weighting (ATW) Scheme

Inspired by the work in Tao et al. [60], in the Average Total Weighting (ATW) scheme, given a graph data set  $GD = \{G_1, G_2, \dots, G_n\}$ , the weight for a subgraph  $g$  is calculated by dividing the sum of the avg weights in the graphs that contain  $g$  with the sum of the average weights across the entire graph dataset  $GD$ . It can be used for both edge and vertex weighted graphs.

Given a graph data set  $GD = \{G_1, G_2, \dots, G_n\}$ , if  $G_i$  is edge weighted by  $\{w_1, w_2, \dots, w_k\}$ , then the average weight of  $G_i$  is defined as:

$$w_{avg}(G_i) = \frac{\sum_{j=1}^k w_j}{k} \quad (1)$$

Where  $w_j$  is either defined by the user or calculated by some weighting methods.

The total weight of  $GD$  is defined as:

$$w_{sum}(GD) = \sum_{i=1}^n w_{avg}(G_i) \quad (2)$$

Given a graph data set  $GD = \{G_1, G_2, \dots, G_n\}$ , and an arbitrary subgraph  $g$ , let the set of graphs where  $g$  occurs equal  $GD(g)$ . Then, the weight of  $g$  with respect to  $\delta_{GD}$  is:

$$W_{GD(g)} = \frac{\sum_{G_i \in \delta_{GD(g)}} W_{avg}(G_i)}{W_{sum(GD)}} \quad (3)$$

The actual importance of subgraph  $g$  is told by  $W_{GD(g)}$ . The weighted support of a subgraph  $g$  is given by the product of the support of  $g$  and the importance factor of  $g$ :

$$wsup_{GD(g)} = W_{GD(g)} \times sup_{GD(g)}$$

A subgraph  $g$  is weighted frequent with respect to  $GD$ , if  $wsup_{GD(g)} \geq \tau$ , where  $0 < \tau \leq 1$  is a weighted support threshold.

In equation 3, the anti-monotone property is satisfied by the function  $W_{GD(g)}$ . This means that if a subgraph of size  $k$  is not frequent, then all the supergraphs of size  $k+1$  containing  $k$  won't be frequent too and they can be pruned during candidate generation. This approach is best suited when the graphs in the dataset are of similar size. The reason behind this is that this approach will tend to be more favored towards larger graphs as compared to smaller graphs.

#### 3.4.2. Affinity Weighting (AW) Scheme

In Affinity Weighting (AW) scheme, the search space growth is restricted by two factors: (a) graph distance measure, and (b) weighting ratio measure. For a subgraph  $g$  to be weighted frequent, both must be greater than specified user thresholds. Let  $g$  be a candidate pattern for a graph dataset  $GD = \{G_1, G_2, \dots, G_n\}$ . We define graph distance as:

$$W_{GD(g)} = \frac{\sum_{G_i \in \delta_{GD(g)}} 1 - \frac{|V(g)|}{|V(G_i)|}}{|V(g)|} \quad (4)$$

Where  $V(G_i)$  is the set of vertices in transaction graph  $G_i$  and  $V(g)$  is the set of vertices in the sub-graph  $g$ .

The value of (4) will only decrease on adding vertexes to  $g$  because  $|\delta_{GD(g)}|$  cannot be increased.

The graph distance measure is based on the number of vertices in a graph. The weighting ratio is concerned with the edge weights.  $r(g)$  is the weighting ratio of an edge-weighted subgraph  $g$ . It returns a value between zero and one which is non-increasing in the number of edges of  $g$ . Given an edge weighted subgraph  $g$  with weights  $S = \{w_1, w_2, \dots, w_n\}$ , the weighting ratio function  $r(g)$  which is similar to [61], is defined as:

$$r(g) = \frac{\text{MIN}_{w_i \in S} \{w_i\}}{\text{MAX}_{w_j \in S} \{w_j\}} \quad (5)$$

Given an edge-weighted graph data set  $GD = \{G_1, G_2, \dots, G_n\}$ , a weighted support threshold  $\tau \in (0, 1]$ , and a weighting ratio threshold  $\lambda \in [0, 1]$ , a subgraph  $g$  is weighted frequent only if the following two conditions (C1 and C2) are satisfied:

$$(C1) \text{wsup}_{GD(g)} = \text{sup}_{GD(g)} \times W_{GD(g)} \geq \tau;$$

$$(C2) r(g) \geq \lambda$$

Thus, we get another pruning method. This method may be used in any frequent subgraph mining algorithm. The weighted support and weighting ratio are tracked during mining. While selecting candidates, the candidates which do not satisfy at least one of the conditions will be discarded.

## Chapter 4: Proposed Method

The proposed system for weighted frequent subgraph mining works on weighted graph dataset. The input dataset contains graphs of medium size. The graphs have undirected weighted edges. The goal of weighted frequent subgraph mining is to find weighted-frequent subgraphs in the input database occurring more than the provided weighted-support threshold value.

The proposed system uses two different weighting mechanisms for frequent subgraph mining, namely Average Total Weighting (ATW) and Affinity Weighting (AW). The mining algorithm for both ATW and AW weighting schemes are presented below:

### 4.1. Distributed weighted subgraph mining using ATW weighing scheme

**Algorithm:** Mapper for ATW scheme based frequent subgraph mining

```
ATW_Map(key, value):  
1. Subgraph = getSubGraph(value)  
2. Reconstruct_DataStructures(value)  
3. C = Generate_Candidate_Set(Subgraph)  
4. For each graph, c in C  
5.     If Isomorphism_Check(c) = true  
6.         If any graph of this division contains c  
7.             outkey = min_dfs_code(c)  
8.             out1 = ATW_weighted_support_factor(c)  
9.             out2 = serialized value of c  
10.            outval = out1 + out2  
11.            Output the values outkey, outval
```

A key-value pair is provided as input to the AWT\_Map function. In order to uniquely identify subgraphs, min-dfscode is used for key. The serialized value of the subgraph is used as the value. Initially, the data structures are initialized. The candidate is generated in line 3. Only one isomorphic subgraph will be used for each candidate generated. If at least one graph contains a subgraph, then it should be sent for processing to reducer function.

outkey is set as min\_dfs\_code of this new candidate subgraph. Along with min\_dfs\_code, weighted\_support\_factor is also calculated and appended to the serialized value of the subgraph. The weighted\_support\_factor and serialized value of subgraph are set as outval. The function emits the (outkey, outval) pair. This pair is then shuffled and sorted with other pairs before sending to the reducer.

**Algorithm:** Algorithm for weighted support factor for ATW scheme

**ATW\_weighted\_support\_factor(c):**

1. for each edge in c
2.     Total += weight(c)
3.     Counter = counter + 1
4. Return total/counter

The ATW\_weighted\_support\_factor method takes a candidate graph as input and returns the average weight of the candidate graph. It traverses all edges and adds their weights. It then divides the total weight by the number of edges. This gives us the average weight of the candidate graph.

**Algorithm:** Reducer for ATW scheme based frequent subgraph mining

**ATW\_Reduce(key, values):**

1. For each value in values
2.     total\_support += getATWWeightedSupport(value)
3. If total\_support / total\_weight  $\geq$  support\_threshold
4.     For each value in values
5.     write\_output(key, value)

The intermediate outputs are shuffled and sorted before going to the reducer. One reducer function will receive the outputs from mapper functions with same outkey. In

ATW\_Reduce, the weighted-support factor from all mapper functions are aggregated and total support is calculated. Total support is then divided by total weight of input dataset. This result is the total weighted support of that subgraph pattern. Then, it checks if the subgraph is weight frequent or not by verifying the support threshold. If it is frequent then it is written to the HDFS. The number of output files will be equal to the number of reducer nodes.

The flowchart of the algorithm is shown below in figure 4.1



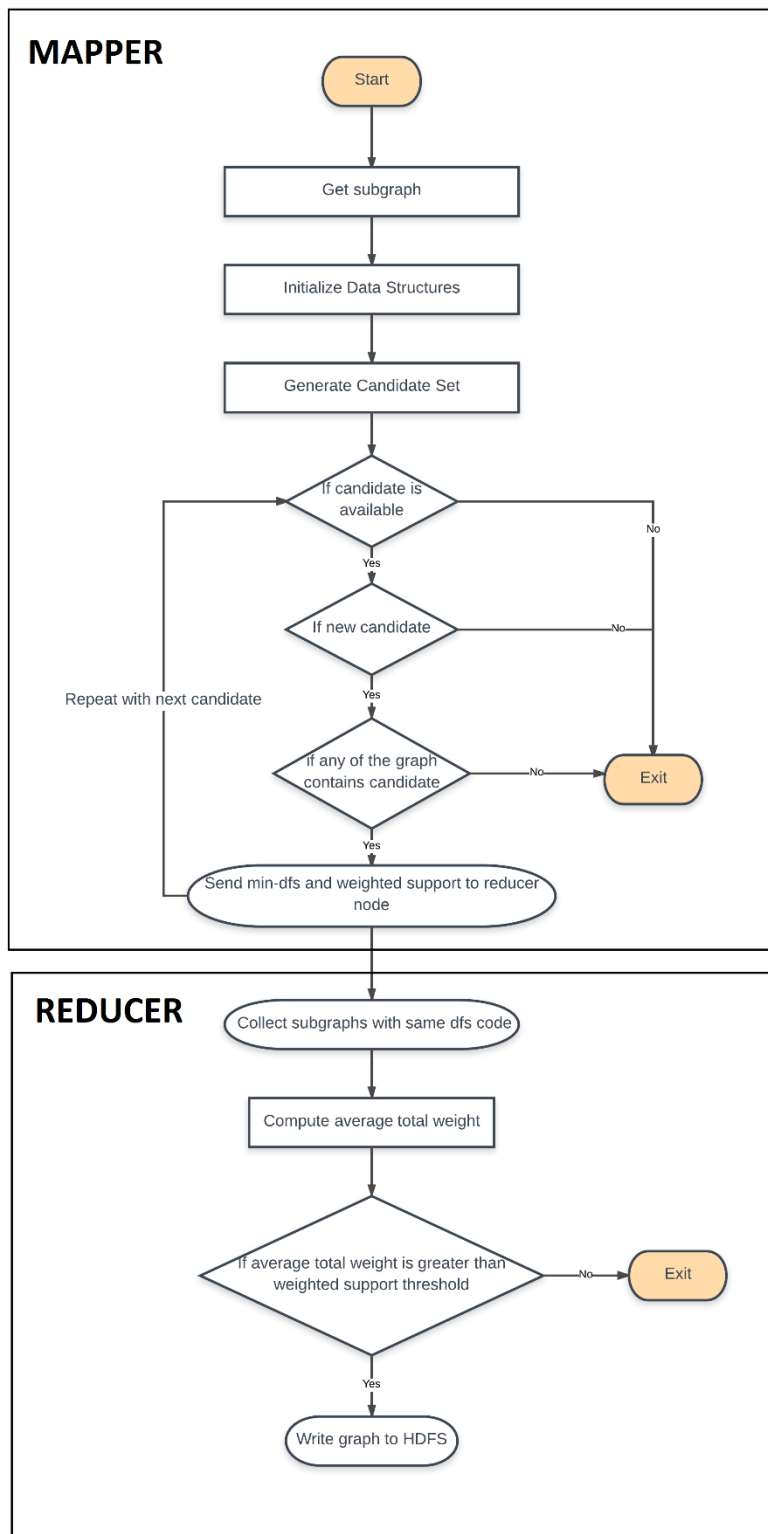


Figure 4.1: Flowchart of distributed FSM using ATW scheme

## 4.2. Distributed weighted subgraph mining using AW weighing scheme

### **Algorithm:** Mapper for AW scheme based frequent subgraph mining

```
AW_Map(key, value):  
1. Subgraph = getSubGraph(value)  
2. Reconstruct_DataStructures(value)  
3. C = Generate_Candidate_Set(Subgraph)  
4. For Each graph, c in C  
5.     If Isomorphism_Check(c) = true  
6.     If any graph of this division contains c  
7.         outkey = min_dfs_code(c)  
8.         out1 = AW_weighted_support_factor(Subgraph,c)  
9.         out2 = serialized value of c  
10.        outval = out1 + out2  
11.        Output the values outkey, outval
```

A key-value pair is provided as input to the AW\_Map function. In order to uniquely identify subgraphs, min-dfscode is used for key. The serialized value of the subgraph is used as the value. Initially, the data structures are initialized. The candidate is generated in line 3. Only one isomorphic subgraph will be used for each candidate generated. If at least one graph contains a subgraph, then it should be sent for processing to reducer function. outkey is set as min\_dfs\_code of this new candidate subgraph. Along with min\_dfs\_code, weighted\_support\_factor is also calculated and appended to the serialized value of the subgraph. The weighted\_support\_factor and serialized value of subgraph are set as outval. The function emits the (outkey, outval) pair. This pair is then shuffled and sorted with other pairs before sending to the reducer.

**Algorithm:** Algorithm for weighted support factor for AW scheme

**AW\_weighted\_support\_factor(Graph, Candidate):**

1. For Each vertex in Graph
2.     g++
3. For Each vertex in Candidate
4.     c++
5. Return  $1 - g/c$

The AW\_weighted\_support\_factor method takes as input, the transaction graph and the candidate graph. It returns the ratio of number of vertices (of the transaction graph) not present in the subgraph to the total number of vertices in the candidate graph. It counts the number of vertices in the graph as well as the potential candidate solution. It returns the ratio of difference in number of vertices in transaction graph and candidate graph to the total number of vertices in the candidate subgraph.

**Algorithm:** Reducer for AW scheme based frequent subgraph mining

**AW\_Reduce(key, values):**

1. For each value in values
2.     total\_support += getAWWeightedSupport(value)
3. Vg = getVerticeSet(key)
3. If total\_support / Vg  $\geq$  support\_threshold
4.     For each edge in getEdges(key)
5.         If max < getEdgeWeight(edge)
6.             max = getEdgeWeight(edge)
7.         If min > getEdgeWeight(edge)
8.             min = getEdgeWeight(edge)
9.     ratio = max / min
10.     If ratio  $\geq$  weighting\_ratio
11.         For each value in values
12.             write\_ouput(key, value)

The intermediate outputs are shuffled and sorted before going to the reducer. One reducer function will receive the outputs from mapper functions with same outkey. In `AW_Reduce`, the weighted-support factor from all mapper functions are aggregated and total support is calculated. Total support is then divided by total number of vertices in the subgraph. This result is the total weighted support of that subgraph pattern. The `AW_Reduce` function checks the `support_threshold` and also computes the ratio of the maximum and minimum edge weights and checks if it is greater than the weight ratio required. If it is frequent then it is written to the HDFS. The number of output files will be equal to the number of reducer nodes.

The flowchart of the algorithm is shown below in figure 4.2

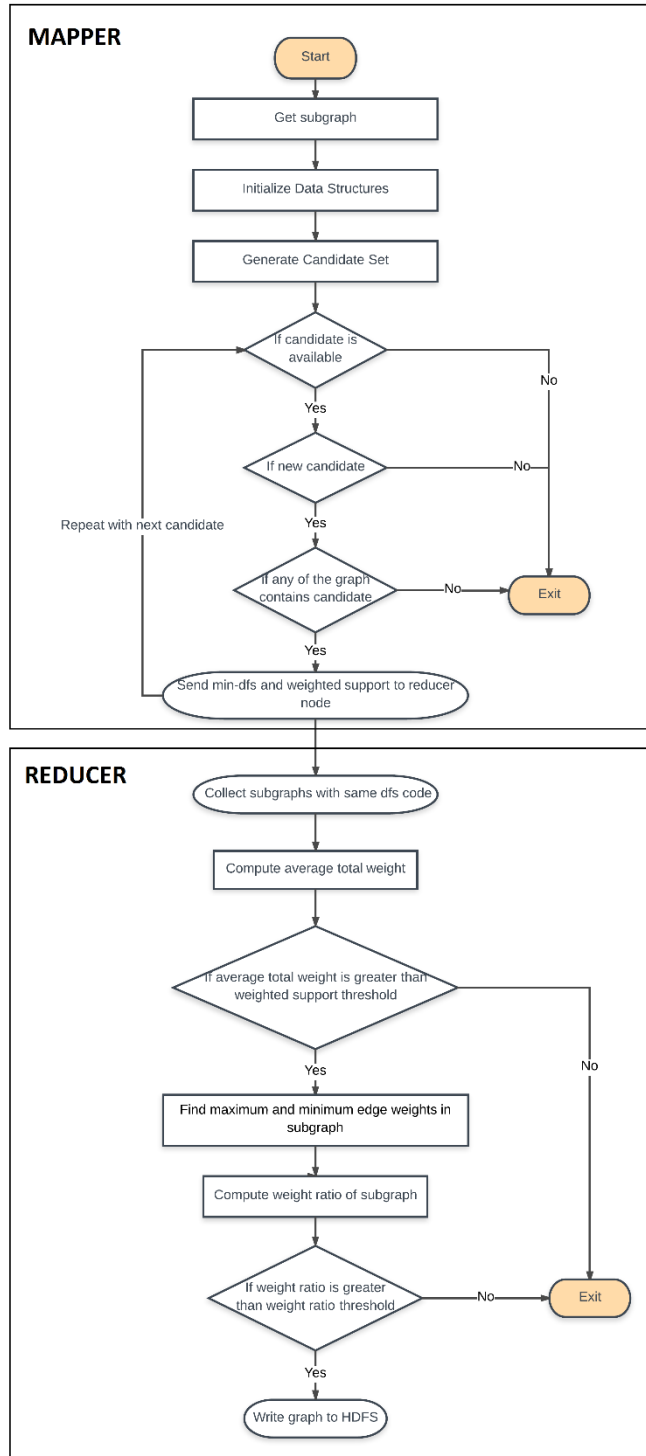


Figure 4.2: Flowchart of distributed FSM using AW scheme

## Chapter 5: Implementation and Results

This chapter discusses the implementation details of the method proposed in the previous chapter. The objective of implementation is to check whether the if fewer patterns are generated using the weighted frequent subgraph mining as compared to standard unweighted subgraph mining. Another goal is to compare and analyze both weighting schemes in the context of frequent subgraph mining and check weighting scheme is faster and which weighting scheme produces fewer patterns.

### 5.1. Hardware Details

The experiment was performed on Linux-based machines. A Hadoop cluster with 4 nodes was created. In the cluster one node was set as the master node which also worked as a data node. The rest of the three nodes were data nodes. Each node had a 3.2 GHz Intel Core i5 processor and 2GB of memory.

### 5.2. Software Details

The Hadoop version used was 2.7.3. The code for preparation and mining phase was written in Java. A 32-bit version of Java 7 was used for the experiments. The data was compressed while writing to HDFS. This saved execution time of the MapReduce job.

### 5.3. Dataset

For the purpose of evaluation, a real-life social network dataset was used. A dataset containing Facebook-like Social Network was acquired from an online source [65].

The Facebook-like Social Network was created using the data of students in University of California. The users that sent or received any message were included in the database. The message sent by a user to another message is represented as a list of edges.

Three types of datasets were used in this work.

Table 5.1: Description of Data

<b>Dataset Name</b>	<b>Description</b>
DS1	Weighted longitudinal one-mode network (weighted by number of characters)
DS2	Weighted static one-mode network (weighted by number of characters)
DS3	Weighted static one-mode network (weighted by number of messages)

Out of the complete dataset, the dataset for three types were used. The description of the data used is presented in table 5.1

Table 5.2: Description of Graph Dataset

<b>Name</b>	<b># of graphs</b>	<b># of edges</b>	<b>Avg # of edges</b>	<b>Total graph weight</b>	<b>Avg. graph weight</b>
DS1	9567	247684	28.9	18331	1.9160
DS2	2024	75487	37.3	8213	4.0578
DS3	1954	71971	36.8	7610	3.8945

Table 5.2 describes the complete description of the graph dataset. It specifies the number of graphs, the number of edges and the average number of edges in each dataset. It also states the total weight of the complete graph as well as the average weight of the graph.

#### 5.4. Results

The performance of the system proposed in chapter 4 is discussed in this section. The performance of both ATW and AW weighting techniques are compared. The number of patterns for varying support functions is depicted.

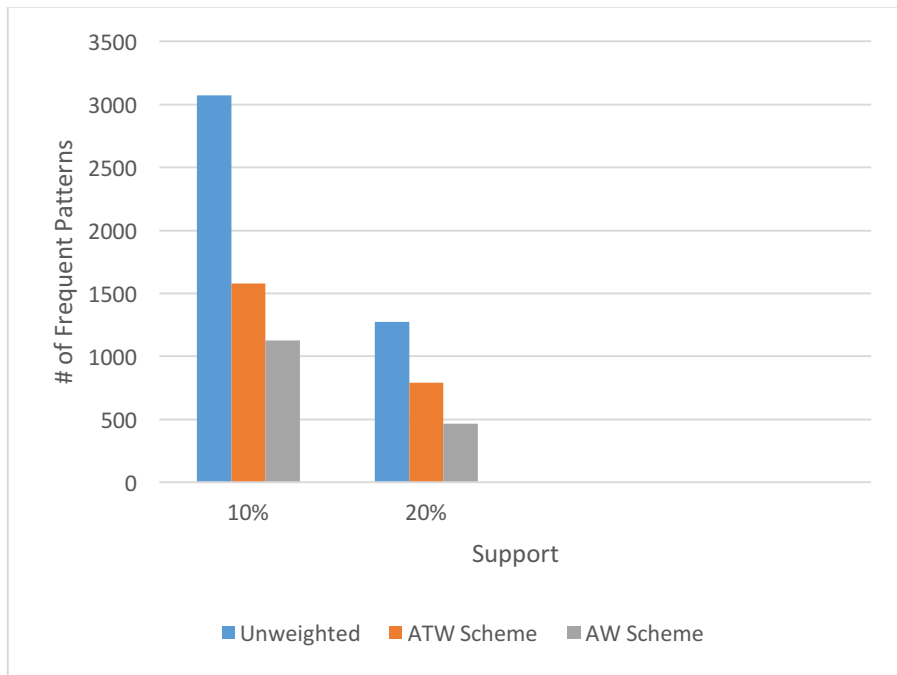


Figure 6.1: Number of frequent patterns for Unweighted, ATW and AW schemes

Figure 6.1 shows the comparison of unweighted and weighted frequent subgraph mining in terms of number of frequent patterns identified. There is a significant decrease in number of patterns in case of weighted frequent subgraph mining. Less number of pattern means that only the relevant and significant patterns are identified. Further within weighted mining, ATW scheme has more patterns than AW scheme. This clearly shows that AW scheme gives fewer and more relevant results as compared to ATW scheme.



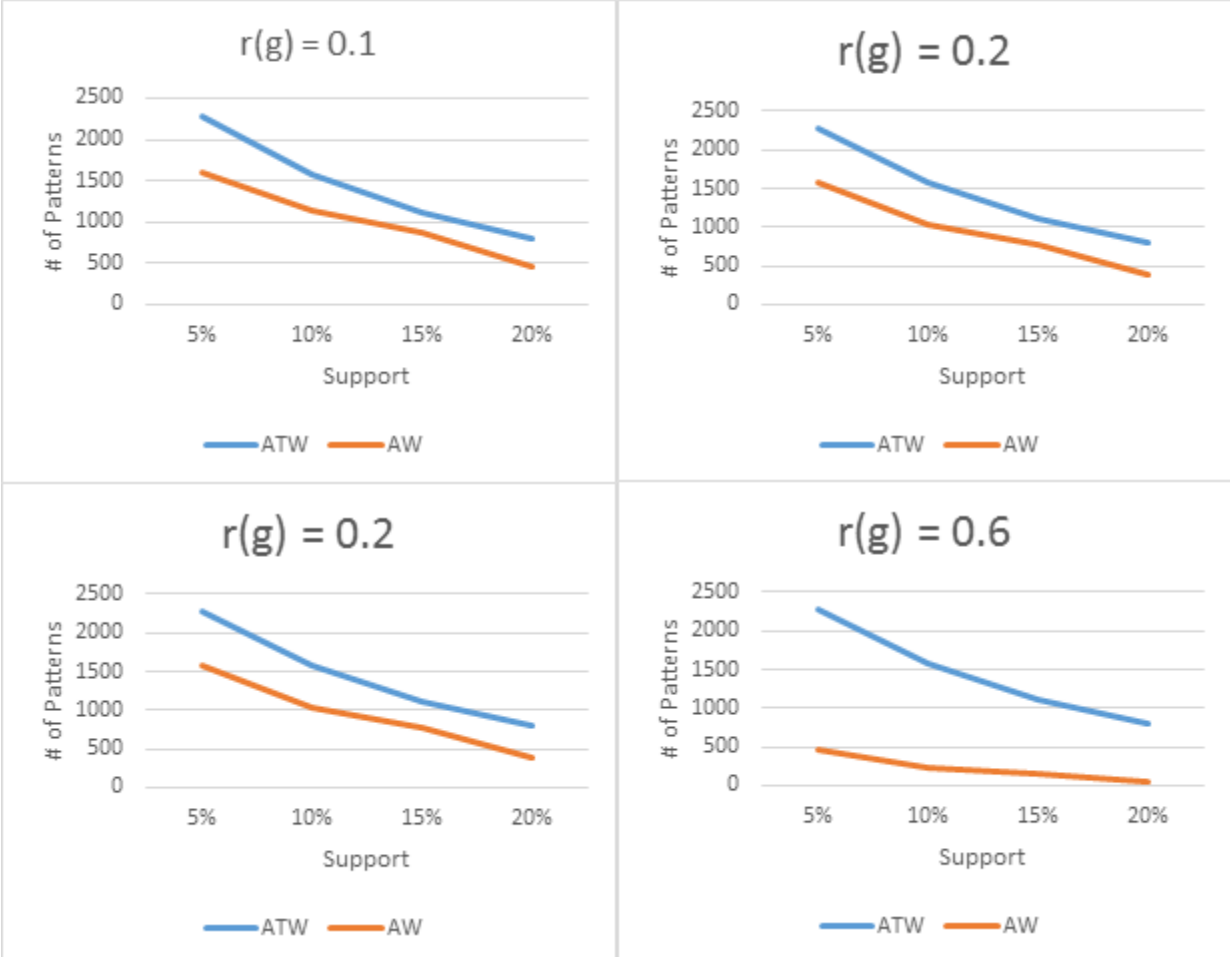


Figure 6.2: Number of patterns in ATW and AW for different  $r(g)$

Figure 6.2 shows the comparison of ATW and AW schemes for different weighting ratios,  $r(g)$ . It can be observed from the figure that the number of patterns decrease drastically on increasing the  $r(g)$  threshold for same support threshold. AW weighting scheme gives further advantage over ATW scheme by adding an additional constraint of weighting ratio. And thus provides only the specific and significant patterns.

However, on increasing the weighting ratio to a higher value, the number of patterns decrease drastically and on further increasing, there may be no results at all. One has to carefully choose the value of the weighting ratio in order to get the best and most accurate results.

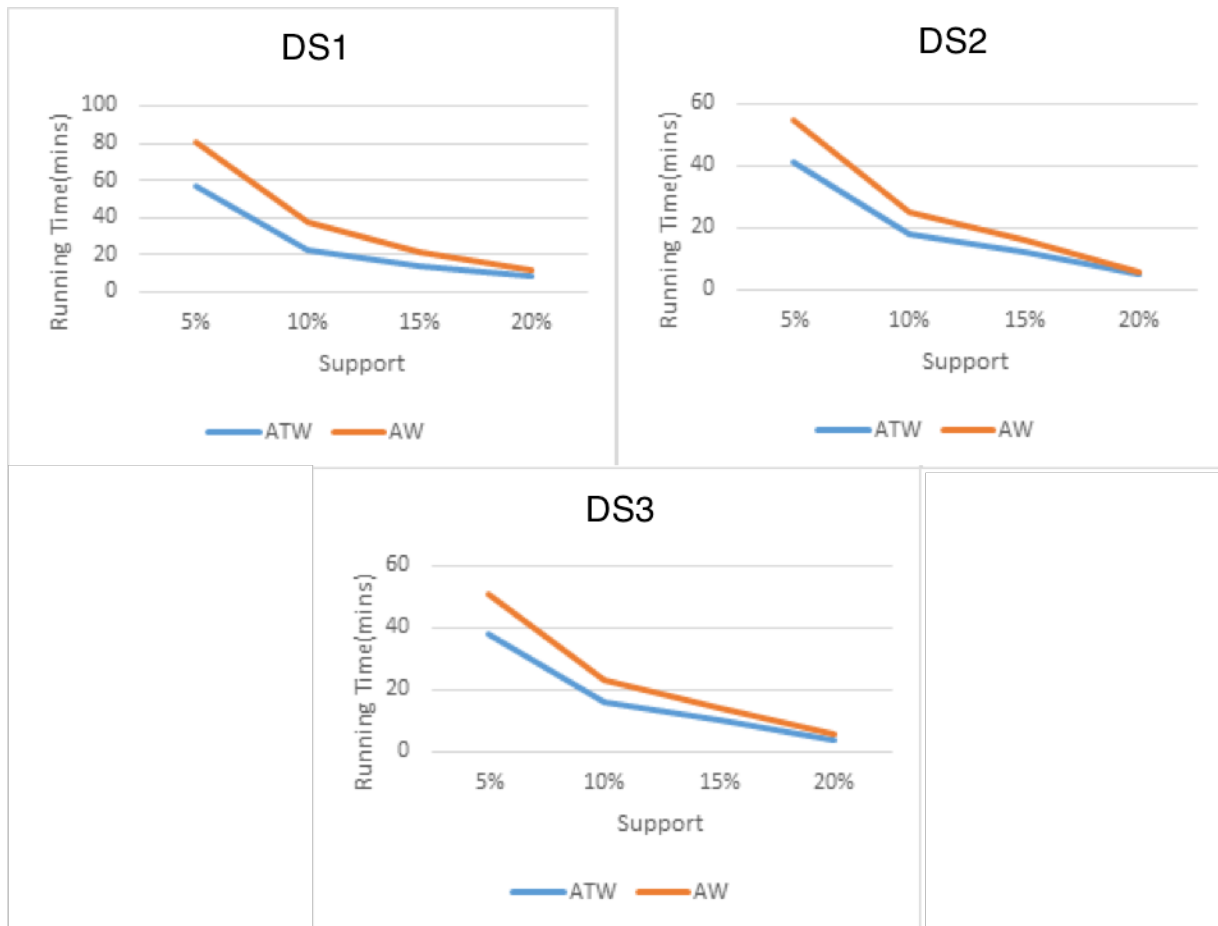


Figure 6.3: Running time of ATW and AW for different support values with  $r(g) = 0.2$

The running time of both ATW and AW schemes for different datasets is depicted in figure 6.3 shown above. It is clear that AW scheme has a slightly higher running time than ATW scheme. This is because more computation is required in case of AW. However, the running times become comparable on increasing the support threshold.

From the results shown in this chapter, it can be deduced that there is a tradeoff between ATW and AW weighting schemes. ATW has a slightly higher running time as compared to AW. But the number of patterns generated by ATW is significantly less than AW for same dataset, which leads to more significant and specific results. One can choose any of the schemes on the basis of the requirement. ATW should be used in the scenarios where running time is a crucial factor. Otherwise AW can be used when more filtered and relevant patterns are required.

## Chapter 7: Conclusions and Future Work

A distributed approach for weighted frequent subgraph mining is proposed in this work. The work uses ATW and AW schemes for finding more important subgraphs from weighted frequency subgraph dataset. In this work, edge weightings are assumed, but the same approach may be applied for vertices. The proposed method finds weighted frequent subgraphs that satisfy the weight threshold. The method

The comparison of results of weighted and unweighted frequent subgraph mining revealed that fewer and more significant patterns are extracted. A MapReduce based distributed system was created to manage large graph databases. The performance of the system was evaluated for four datasets using different parameters. The proposed method is efficient and scalable. It can handle huge graph datasets, which cannot be handled on a single machine.

It is found that the AW scheme consumes more time than ATW scheme. It is due to slightly extra complexity in calculations. However, AW scheme gives the least number of patterns. The patterns found are more significant, relevant and specific. In the case of ATW scheme, the number of patterns found is more than AW scheme but lower than the number of patterns in the case of unweighted frequent subgraph mining.

Thus, it can be concluded that ATW scheme has a better runtime than AW scheme, but the patterns produced in the case of AW scheme are more significant and relevant.

In the future, the algorithm can be optimized to use some sort of caching or intermediate values. Apache Spark framework [63] can be used to optimize the algorithm. Apache Spark will reduce the serialization and disk I/O overhead of writing results of each iteration to disk. It has been found that iterative algorithms are faster than MapReduce [64] based algorithms. Thus, Spark is an ideal platform for distributed graph algorithms.

## References

- [1] R. A. a. R. Srikant, "Fast algorithms for mining association rules," in *20th International Conference on Very Large Data Bases (VLDB)*, September 1994.
- [2] H. T. a. R. D. K. L. Dehaspe, "Finding frequent substructures in chemical compounds," in *4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 1998.
- [3] D. J. C. a. S. D. L. B. Holder, "Substructure discovery in the subdue system," in *Workshop on Knowledge Discovery in Databases (KDD) Proceedings*, July 1994.
- [4] F. C. M. Z. Chuntao Jiang, "Frequent Sub-graph Mining on Edge Weighted Graphs," in *International Conference on Data Warehousing and Knowledge Discovery*, September 2010.
- [5] M. A. H. Mansurul A. Bhuiyan, "An Iterative MapReduce Based Frequent Subgraph Mining Algorithm," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 3, pp. 608-620, March 2015.
- [6] G. K. Michihiro Kuramochi, "Finding Frequent Patterns in a Large Sparse Graph," *Data Mining and Knowledge Discovery*, vol. 11, no. 3, pp. 243-271, 2005.
- [7] M. J. Z. N. Talukder, "A distributed approach for graph mining in massive networks," *Data Mining and Knowledge Discovery*, vol. 30, no. 5, p. 1024–1052, September 2016.
- [8] S. G. Jeffrey Dean, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113 , 2008.
- [9] "Hadoop Tutorial - YDN," Yahoo Developer Network, [Online]. Available: <https://developer.yahoo.com/hadoop/tutorial/module4.html>.

- [10] J. L. a. C. Dyer, *Data-Intensive Text Processing with MapReduce*, Morgan and Claypool Publishers, 2010.
- [11] X. J. H. C. J. M. X. Z. Yang Liu, "MapReduce-Based Pattern Finding Algorithm Applied in Motif Detection for Prescription Compatibility Network," in *8th International Symposium on Advanced Parallel Processing Technologies*, 2009.
- [12] J. H. Xifeng Yan, "gSpan: Graph-Based Substructure Pattern Mining," in *IEEE International Conference on Data Mining*, Maebashi City, Japan, 2002.
- [13] S. G. Jeffrey Dean, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107-113, January 2008.
- [14] G. D. F. a. M. Berthold, "Dynamic load balancing for the distributed mining of molecular structures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 8, pp. 773 - 785, August 2006.
- [15] R. B. a. R. B. S. Chakravarthy, "DB-Subdue: Database Approach to Graph Mining," in *8th Pacific-Asia Conference in Knowledge Discovery and Data Mining (PAKDD) Proceedings*, May 2004.
- [16] S. C. a. S. Pradhan, "Db-fsg: An sql-based approach for frequent subgraph mining," in *19th international conference on Database and Expert Systems Applications (DEXA) Proceedings*, 2008.
- [17] B. S. a. R. Sunderraman, "Oo-fsg: An object-oriented approach to mine frequent subgraphs," in *Australasian Data Mining Conference (AusDM) Proceedings*, December 2011.
- [18] F. M. H. a. K. B. Eichinger, "On the usefulness of weight-based constraints in frequent subgraph mining," in *Research and Development in Intelligent Systems XXVII*, October 2010.
- [19] B. S. a. R. S. S. Hill, "An iterative Mapreduce approach to frequent subgraph mining in biological datasets," in *Proc. ACM Conf. Bioinformat., Comput. Biol. Biomed*, 2012.

- [20] A. J. Nisha Babu, "A distributed approach to weighted frequent Subgraph mining," in *International Conference on Emerging Technological Trends (ICETT)*, 2016.
- [21] "The PubChem Project," NCBI - National Center for Biotechnology Information, 2004. [Online]. Available: <https://pubchem.ncbi.nlm.nih.gov/>.
- [22] R. S. Rakesh Agrawal, "Fast Algorithms for Mining Association Rules in Large Databases," in *20th International Conference on Very Large Data Bases*, 1994.
- [23] T. W. H. M. Akihiro Inokuchi, "An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data," in *4th European Conference on Principles of Data Mining and Knowledge Discovery*, 2000.
- [24] M. K. a. G. Karypis, "Frequent subgraph discovery," in *Proc. Int. Conf. Data Mining*, 2001.
- [25] a. J. K. S. Nijssen, "A quickstart in frequent structure mining can make a difference," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2004.
- [26] M. H. S. S. a. M. Z. V. Chaoji, "An integrated, generic approach to pattern mining: Data mining template library," *Data Mining and Knowledge Discovery*, vol. 17, no. 3, p. 457–495, 2008.
- [27] L. B. H. D. J. C. Nikhil S. Ketkar, "Subdue: compression-based frequent pattern discovery in graph data," in *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, 2005.
- [28] X. a. J. H. Yan, "CloseGraph: mining closed frequent graph patterns," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003.
- [29] C.-T. L. Hsun-Ping Hsieh, "Mining Temporal Subgraph Patterns in Heterogeneous Information Networks," in *IEEE Second International Conference on Social Computing*, 2010.

- [30] C. B. a. M. Berthold, "Mining molecular fragments: finding relevant substructures of molecules," in *Proc. IEEE Int. Conf. Data Mining*, 2002.
- [31] Y. L. a. H. G. Jianzhong Li, "Efficient Algorithms for Summarizing Graph Patterns," *IEEE Transactions On Knowledge And Data Engineering*, vol. 23, no. 9, September 2011.
- [32] J. L. H. G. Yong Liu, "JPMiner: Mining Frequent Jump Patterns From Graph Databases," in *Sixth International Conference on Fuzzy Systems and Knowledge Discovery*, 2009.
- [33] Q. L. G. Z. D. D. Y. J. W. B. Yuhua Li, "A Directed Labeled Graph Frequent Pattern Mining Algorithm Based on Minimum Code," in *Third International Conference on Multimedia and Ubiquitous Engineering*, 2009.
- [34] S. a. K. J. Nijssen, "Faster association rules for multiple relations," in *IJCAI'01: Seventeenth International Joint Conference on Artificial Intelligence*, 2001.
- [35] F. a. M. Z. Chuntao Jiang, "A Survey of Frequent Subgraph Mining Algorithms," *The Knowledge Engineering Review*, vol. 28, no. 1, pp. 1-31, 2004.
- [36] M. K. a. G. Karypis, "GREW A Scalable frequent subgraphdiscovery algorithm," in *Fourth IEEE International Conference on Data Mining*, 2004.
- [37] W. W. a. J. P. J. Huan, "Efficient mining of frequent subgraphs in the presence of isomorphism," in *Proc. 3rd IEEE Int. Conf. Data Mining*, 2003.
- [38] L. V. S. a. K. K. Thomas, "Isg: Itemset based subgraph mining," IIIT, Hyderabad, December 2009.
- [39] J. W. W. J. P. a. J. Y. Huan, "Spin: mining maximal frequent subgraphs from graph databases," in *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2004.
- [40] J. L. H. G. a. S. Z. Zhaonian Zou, "Mining Frequent Subgraph Patterns from Uncertain Graph Data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 9, pp. 1203-1218, September 2010.

- [41] R. B. a. R. B. S. Chakravarthy, "Db-subdue: Database approach to graph mining," in *Proc. Adv. Knowl. Discov. Data Mining*, 2004.
- [42] L. B. H. G. G. a. R. M. D. J. Cook, "Approaches to Parallel Graph-Based Knowledge Discovery," *Journal of Parallel and Distributed Computing*, vol. 61, no. 3, pp. 427-446 , 2001.
- [43] C. W. a. S. Parthasarathy, "Parallel algorithms for mining frequent structural motifs in scientific data," in *Proc. 18th Annu. Int. Conf. Supercomput.*, 2004.
- [44] S. P. a. M. Coatney, "Efficient discovery of common substructures in macromolecules," in *Proc. IEEE Int. Conf. Data Mining*, 2002.
- [45] M. W. O. U. I. F. a. M. P. T. Meinel, "The parmol package for frequent subgraph mining," *Electron. Commun. EASST*, vol. 1, p. 1–12, 2006.
- [46] M. A. D. a. T. W. M. Philippsen, "Parsemis - The parallel and sequential mining suite," 2011. [Online]. Available: <https://www2.cs.fau.de/EN/research/zold/ParSeMiS/index.html>.
- [47] W. H. M. L. L. a. C. S. J. Wang, "A partition-based approach to graph mining," in *Proc. 22nd Int. Conf. Data Eng.*, 2006.
- [48] M. E. O. a. X. L. S. N. Nguyen, "Graph mining based on a data partitioning approach," in *Proc. 19th Australasian Database Conf.*, 2008.
- [49] S. P. a. Y.-K. C. G. Buehrer, "Adaptive parallel graph mining for CMP architectures," in *Proc. 6th IEEE Int. Conf. Data Mining*, 2006.
- [50] Y.-B. Y. a. Y. Z. G.-P. Chen, "Mapreduce-based balanced mining for closed frequent itemset," in *Proc. IEEE 19th Int. Conf. Web Serv.*, 2012.
- [51] Y.-B. Y. Y. G. G.-P. C. a. Y. Z. S.-Q. Wang, "Mapreduce-based closed frequent itemset mining with efficient redundancy filtering," in *Proc. IEEE 12th Int. Conf. Data Mining Workshops*, 2012.



- [52] Z. Z. J. C. J. L. J. H. a. S. F. L. Zhou, "Balanced parallel FP-growth with Mapreduce," in *Proc. IEEE Youth Conf. Inf. Comput. Telecommun.*, 2010.
- [53] Y. W. D. Z. M. Z. a. E. Y. C. H. Li, "Pfp: parallel FP-growth for query recommendation," in *Proc. ACM Conf. Recommender Syst.*, 2008.
- [54] H.-J. C. M. A. H. M. M. R. a. M. R. K. B.-S. Jeong, "A MapReduce framework for mining maximal contiguous frequent patterns in large DNA sequence datasets," *IETE Technical Review*, vol. 29, no. 2, pp. 162-168, 2012.
- [55] G. K. Michihiro Kuramochi, "Finding Frequent Patterns in a Large Sparse Graph\*," *Data Mining and Knowledge Discovery*, vol. 11, no. 3, pp. 243-271, November 2005.
- [56] E. A. S. S. P. K. Mohammed Elseidy, "GraMi: frequent subgraph and pattern mining in a single large graph," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 517-528, March 2014.
- [57] L. J. Yun U, "Mining weighted support affinity patterns," in *18th International Conference On Computer Applications in Industry and Engineering*, 2005.
- [58] F. C. R. S. M. Z. Chuntao Jiang, "Text classification using graph mining-based," *Knowledge-Based Systems*, vol. 23, no. 4, pp. 302-308, 2010.
- [59] F. C. A. Elsayed, "Corpus callosum MR image classification," *Knowledge-Based Systems*, vol. 23, no. 4, pp. 330-336, 2010.
- [60] F. M. F. F. M. Tao, "Weighted Association Rule Mining using Weighted Support and Significance Framework," in *The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.
- [61] U. Yun, "WIS:Weighted Interesting Sequential Pattern Mining with a Similar Level of Support and/or Weight," *ETRI Journal*, vol. 29, no. 3, pp. 336-352, 2007.
- [62] "XIFENG YAN," University of California Santa Barbara, [Online]. Available: <https://www.cs.ucsb.edu/~xyan/dataset.htm>.

- [63] M. C. T. D. A. D. J. M. M. M. M. J. F. S. S. I. S. Matei Zaharia, "Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.
- [64] Y. Q. U. F. M. L. J. C. W. B. R. F. Ö. Juwei Shi, "Clash of the titans: MapReduce vs. Spark for large scale data analytics," *Proceedings of the VLDB Endowment*, vol. 8, no. 13, pp. 2110-2121, 2015.