공학박사 학위논문

Mining Traversal Patterns from Weighted Traversals and Graph

가중치 순회 및 그래프로부터의 순회 패턴 마이닝

지도교수 박 휴 찬

2007년 8월

한국해양대학교 대학원

컴 퓨 터 공 학 과

이 성 대

본 논문을 이성대의 공학박사 학위논문으로 인준함

위원장 공학박사 류 길 수 인 

위 원 공학박사 신 옥 근 인 

위 원 공학박사 김 재 훈 인 

위 원 공학박사 양 희 재 인 

위 원 공학박사 박 휴 찬 인 

2007년 6월 27일

한국해양대학교 대학원

컴퓨터공학과

Contents

List of Figures

List of Tables

가중치 순회 및 그래프로부터의 순회 패턴 마이닝

이 성 대

한국해양대학교 대학원 컴퓨터공학과

지도교수 박 휴 찬

요 약

실세계의 많은 문제들은 그래프와 그 그래프를 순회하는 트랜잭션으로 모델링될 수 있다. 예를 들면, 웹 페이지의 연결구조는 그래프로 표현될 수 있고, 사용자의 웹 페이지 방문경로는 그 그래프를 순회하는 트랜잭션으로 모델링될 수 있다. 이와 같이 그래프를 순회하는 트랜잭션으로부터 중요하고 가치 있는 패턴을 찾아내는 것은 의미 있는 일이다. 이러한 패턴을 찾기 위한 지금까지의 연구에서는 순회나 그래프의 가중치를 고려하지 않고 단순히 빈발하는 패턴만을 찾는 알고리즘을 제안하였다. 이러한 알고리즘의 한계는 보다 신뢰성 있고 정확한 패턴을 탐사하는 데 어려움이 있다는 것이다.

본 논문에서는 순회나 그래프의 정점에 부여된 가중치를 고려하여 패턴을 탐사하는 두 가지 방법들을 제안한다. 첫 번째 방법은 그래프를 순회하는 정보에 가중치가 존재하는 경우에 빈발 순회 패턴을 탐사하는 것이다. 그래프 순회에 부여될 수 있는 가중치로는 두 도시간의 이동 시간이나 웹 사이트를 방문할 때 한 페이지에서 다른 페이지로 이동하는 시간 등이 될 수 있다. 본 논문에서는 좀 더 정확한 순회 패턴을 마이닝하기 위해 통계학의 신뢰 구간을 이용한다. 즉, 전체 순회의 각 간선에 부여된 가중치로부터 신뢰 구간을 구한 후 신뢰 구

간의 내에 있는 순회만을 유효한 것으로 인정하는 방법이다. 이러한 방법을 적용함으로써 더욱 신뢰성 있는 순회 패턴을 마이닝할 수 있다. 또한 이렇게 구한 패턴과 그래프 정보를 이용하여 패턴 간의 우선순위를 결정할 수 있는 방법과 성능 향상을 위한 알고리즘도 제시한다.

두 번째 방법은 그래프의 정점에 가중치가 부여된 경우에 가중치가 고려된 빈발 순회 패턴을 탐사하는 방법이다. 그래프의 정점에 부여될 수 있는 가중치로는 웹 사이트 내의 각 문서의 정보량이나 중요도 등이 될 수 있다. 이 문제에서는 빈발 순회 패턴을 결정하기 위하여 패턴의 발생 빈도뿐만 아니라 방문한 정점의 가중치를 동시에 고려하여야 한다. 이를 위해 본 논문에서는 정점의 가중치를 이용하여 향후에 빈발 패턴이 될 가능성이 있는 후보 패턴은 각 마이닝 단계에서 제거하지 않고 유지하는 알고리즘을 제안한다. 또한 성능 향상을 위해 후보 패턴의 수를 감소시키는 알고리즘도 제안한다.

본 논문에서 제안한 두 가지 방법에 대하여 다양한 실험을 통하여 수행 시간 및 생성되는 패턴의 수 등을 비교 분석하였다.

본 논문에서는 순회에 가중치가 있는 경우와 그래프의 정점에 가중치가 있는 경우에 빈발 순회 패턴을 탐사하는 새로운 방법들을 제안하였다. 제안한 방법들을 웹 마이닝과 같은 분야에 적용함으로써 웹 구조의 효율적인 변경이나 웹 문서의 접근 속도 향상, 사용자별 개인화된 웹 문서 구축 등이 가능할 것이다.

Mining Traversal Patterns from Weighted Traversals and Graph

Seong-Dae Lee

Department of Computer Engineering,
Graduate School, Korea Maritime University

Advised by:
Prof. Hyu-Chan Park

ABSTRACT

A lot of real world problems can be modeled as traversals on graph. Mining from such traversals has been found useful in several applications. However, previous works considered only unweighted traversals and graph.

This thesis generalizes this to the cases where traversals and vertices in a graph are given weights to reflect their importance. Two new methods are proposed to discover frequent patterns from the weighted traversals and vertices in a graph.

The first proposes the mining algorithm for discovering the frequent patterns from the weighted traversals on a unweighted graph. we adopts the notion of confidence interval to distinguish between confident traversals and outliers. By excluding the outliers, more reliable frequent

patterns can be obtained. Furthermore, we propose a performance enhancement by traversal split, and then verify through experiments. In addition, they are further ranked according to their priority.

The second proposes the mining problem to the discovery of weighted frequent patterns from the unweighted traversals on a weighted graph. Under such weight settings, traditional mining algorithms can not be adopted directly any more. To cope with the problem, this paper proposes new algorithms to discover weighted frequent patterns from the traversals. Specifically, we devise support bound paradigms for candidate generation and pruning during the mining process.

The proposed methods can be applied to the practical applications such as Web mining.

Chapter 1 Introduction

1.1 Overview

Data mining is the process of extracting valuable information or patterns from large information repositories such as relational database, data warehouses, XML repositories. It refers to the process of analyzing large databases to discover useful patterns. It is also known as one of the core processes of knowledge discovery in database [1-4].

Several data structures and mining algorithms have been proposed and successfully applied to many applications [4]. Recently, the data mining on the graph becomes a center of interest. The graph has been widely used to model several classes of real world problems, such as design of network, scheduling for operating system, bio-informatics, and GIS. The structure of Web site can be modeled as a graph, for example, in which vertices represent Web pages, and edges are for hyperlinks between the Web pages. User navigations on the Web site can be modeled as traversals on the graph. Each traversal can be represented as a sequence of vertices, or equivalently a sequence of edges.

Once the graph and its traversals are given, valuable information can be discovered. Most common form of the valuable information may be frequent patterns, i.e., the sub-traversals that is contained in a large ratio of traversals. In previous works, traversals on a graph are treated uniformly without considering their importances, such as mining patterns from traversals without weights and base graph [5-7].

Therefore, it is necessary to develop new data mining algorithms based on a graph with weights. This thesis focuses on the problem of finding weighted frequent patterns from a database of user traversals through a given graph structure. We propose two new methods for discovering frequent traversal patterns from weighted traversals and weighted graph respectively. The first method proposes an algorithm to find frequent patterns from the weighted traversals on a graph by excluding outliers of traversals, and then presents another algorithm to enhance the performance. The second method describes two algorithms for discovering weighted frequent traversal patterns from the weighted graph.

1.2 Motivations

Graphs and traversals on them are widely used to model several classes of real world problems [8-12]. The structure of a Web site, for example, can be modeled as a graph in which vertices represent Web pages, and edges represent hyperlinks between the pages. Furthermore, user navigations on the Web site can be modeled as traversals on the graph. Once a graph and its traversals are given, valuable information can be discovered. Most common form of the information may be frequent patterns, i.e., the sub-traversals that are contained in a large ratio of traversals [5-7, 13-15]. However, a drawback of these approaches is to discover only frequent patterns without considering the weights in the traversals and graphs to reflect their importance. Therefore, it is required new approaches to discover frequent patterns from the weighted traversals and weighted graph.

In this thesis, the weights are attached to the traversals and vertices in a graph to reflect their importance. For instance, when the users navigate a Web site, they may have different interest in each Web page, and therefore stay for different duration. Each edge, which represents a transition between Web pages, can be assigned with a weight standing for the user stay duration. In addition, each vertex can be also assigned with a weight according to the amount of information or the importance of each Web page.

This thesis generalizes the mining problem to the case where traversals and vertices in a graph are weighted. This problem can be directly applied to *Web Usage Mining* problem. In *Web Usage Mining*, because the number of web pages and the complexity of Web sites increase, Web service providers and online business want to track user browsing habits to improve their services better and get more profits. Therefore, the structure of Web sites has to be designed effectively for more efficient access between highly correlated Web pages, and better customer classification and behavior analysis.

In this thesis, we assign the weights to the edges of traversals and the vertices in a graph, and the distribution of weights follows the normal distribution.

1.3 Approach

This thesis addresses two new approaches to discover frequent traversal patterns from the traversals on a graph. One is to discover frequent traversal patterns from the weighted traversals on a graph, the other

from traversals on a weighted graph.

For mining patterns from the weighted traversals on a graph, we adopt the notion of confidence interval to classify the weights into confident ones and outliers. The confidence interval is defined statistically according to the distribution of values. If a weight lies within the confidence interval, then it is considered as a confident one, but if it lies outside the confidence interval, then it is considered as an outlier. On top of the notion, we propose a level-wise algorithm for the discovery of frequent patterns. In each pass, candidate patterns are tested on the traversals to count their supports, and then evaluated with respect to the supports to become frequent patterns. The frequent patterns are joined together to generate one-step larger candidates. It proceeds until no more candidates are generated. The frequent patterns are further ranked according to their priority. The priority reflects other aspects of the patterns beside the support, such as the connectivity and vertex weights.

For mining patterns from the weighted graph, we extend previous works by considering weights attached to the vertices of graph. Such vertex weight may reflect the importance of vertex. For example, each Web page may have different importance which reflects the value of its content. With the weight setting, the mining algorithm can not be relied on the well-known *Apriori* paradigm any more. The reason why *Apriori* paradigm works is due to the *downward closure property*, which says all the subsets of a frequent pattern must be frequent. With the weight setting, however, it is not necessarily true that all the subpatterns of a weighted frequent pattern are weighted frequent. Therefore, we adopt the notion of support bound. On top of the notion, we propose a new mining

algorithm for the discovery of weighed frequent patterns.

1.4 Organization of Thesis

This thesis is organized as follows. In chapter 2, we review previous works on the itemset and traversal minings without and with weights, respectively. Chapter 3 proposes a method for the discovery of frequent patterns from weighted traversals on graph, and chapter 4 also proposes another method for the discovery of weighted frequent patterns from traversals on weighted graph. Chapter 3 and 4 also contain experiments and analyses of the algorithms on synthetic data, respectively. Finally, chapter 5 contains conclusions and further works.

Chapter 2 Related Works

Data mining is the core of knowledge discovery process to extract useful information from large datasets or databases [16-18]. In other words, it looks for interesting patterns and trends that exist in large datasets. Traditionally, major data mining problems have been association rule [3, 19-21], sequential pattern [22, 23], classification [24-27], clustering [28, 29]. Many of them are based on the function of itemset mining. Recently, another main stream is the mining problems on the traversals and graphs. This chapter reviews previous works related with such itemset mining and traversal mining without and with weights, respectively.

2.1 Itemset Mining

The itemset is a set of the data items. On such itemset, the itemset mining is the core of several mining problems, such as association rule mining. The association rule mining is to find association among a large set of itemsets in transaction databases, relational databases, and data warehouses. It discovers any rule of the form $X \Rightarrow Y$, where X and Y are sets of data items [1-4]. For example, "80% of customers who buy cheese and milk also buy bread, and 5% of customers buy all of them together".

In the mining, there are two important measures of interestingness, called support and confidence. Equation (2.1) and (2.2) describe the two measures [1-4].

(2.1)

$$confidence(A \Rightarrow B) = \frac{Pr(A \cup B)}{Pr(A)}. \quad (2.2)$$

Typically, the frequent itemsets, traditionally called the large itemsets, are considered interesting if they satisfy the minimum support threshold. In the itemset mining, the *Apriori* algorithm is the most basic and well-known algorithm to find frequent itemsets in a transactional database. Fig. 2.1 describes the *Apriori* algorithm with pseudo-code [2].

Algorithm. ***Apriori***

Input: candidate itemset of k size C_k,
minimum support threshold *minsup*,
transaction database D

Output: frequent itemset of k size L_k

```
begin
  L1 = {large 1 − emsets};
  for (k = 2; Lk ≠ 0; k++) {
    // join and prune candidates
    Ck = apriori_gen(Lk−1);

    // scan D for counts
    for each transaction t ∈ D {
      Ct = {c | c ∈ Ck, c is a subsets of t};
      ∀ c ∈ Ct, c.count++;
    }
    Lk = {c | c ∈ Ck, c.count ≥ minsup};
  }
end;
```

Fig. 2.1 Apriori algorithm

In the *Apriori* algorithm, there are the important property, called *downward closure property*, which is used for joining and pruning candidates. It describes that a pattern of length k is frequent only if its all subpatterns of length $k-1$ are also frequent. Fig. 2.2 is an example of *Apriori* algorithm.

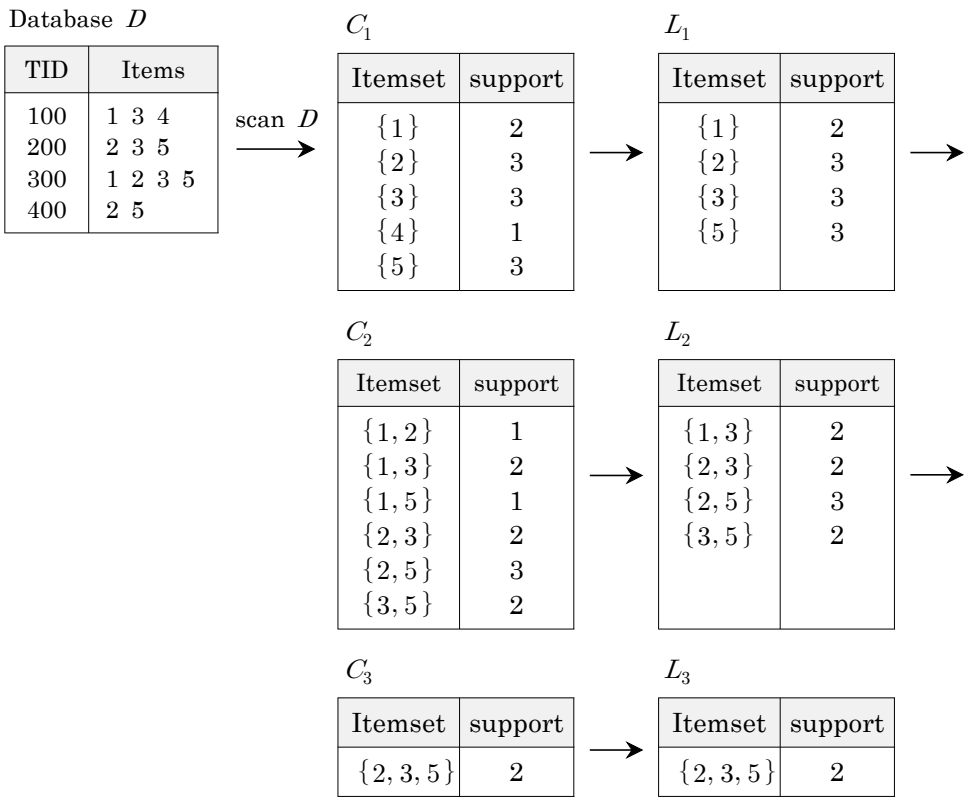

Fig. 2.2 An example of Apriori algorithm

In the example of Fig. 2.2, itemsets, $\{1,2,3\}$, $\{1,2,5\}$ and $\{1,3,5\}$ is not in candidates of length 3, called C_3, because of *downward closure property*. For instance, if $\{1,2,3\}$ becomes a candidate in C_3, then the all

subsets of length 2 of itemsets $\{1,2,3\}$ must be in L_2, but itemsets, $\{1,3\}$ and $\{2,3\}$ are included in L_2 while not $\{1,2\}$.

In addition, there are some modified methods for the enhanced performance of *Apriori*, such as *AprioriTid*, *Apriori-Hybrid* [2]. Although *AprioriTid* uses the same candidate generation function as *Apriori*, it does not use database for counting support after the first pass. And it only encodes the candidate itemsets used in the previous pass, in order to save reading effort of databases. *Apriori-Hybrid* uses *Apriori* in the initial passes and switches to *AprioriTid* when it expects that the candidate itemsets at the end of the pass will be in memory [2]. Other approaches on the itemset mining are partition technique [20], sampling technique [30], *DHP* algorithm based on hash technique [31] and multi-level or generalized association [32, 33].

Recently, there are various algorithms based on *Apriori* algorithm as in the itemset mining. One of them is sequential pattern mining, and it is the algorithm for finding all frequent itemsets within a transactional database, introduced in [33]. In general, $A \Rightarrow B$ says that buying the item A will be immediately followed by buying the item B with a certain confidence. From a book store's transaction database history, we can find the frequent sequential purchasing patterns, for example, 80% customers who have bought the book "HTML" typically bought the book "Java Script Handbook" and then bought the book "Web Programming" with certain time gap. In this example, all those books need not to be bought at the same time or consecutively, the most important thing is the order in which those books are bought and they are bought by the same customer. 80% here represents the percentage of customers who comply this

purchasing habit. Based on this heuristic, a series of *Apriori*-like algorithms have been proposed, such as *AprioriAll*, *AprioriSome*, *DynamicSome* in [33], and *GSP* [23]. Later on another a series of data projection based algorithms were proposed, which includes *FreeSpan* [4] and *PrefixSpan* [34]. *SPADE* [35] is a lattice based algorithm, *MEMISP* [36] is a memory indexing based approach, and *SPIRIT* [37] integrates constraints by using regular expression.

2.2 Weighted Itemset Mining

For the weighted itemset mining, most of previous works are related to the mining of association rules and its sub-problems. Such weighted itemset mining is defined as the problem of finding itemsets which have both sufficient support and weight. This problem may be more complex than the simple itemset mining because there is not the *downward closure property*, also known as the *Apriori* property, between itemsets. This is due to the fact that the weighted support of itemset may increase or decrease as the itemset is extended by adding additional items.

To resolve this difficulty, Cai et al. [38] generalized the discovery of frequent itemsets to the case where each item is given an associated weight. They introduced new criteria to handle the weights in the process of finding frequent itemsets, such as weighted support for the measurement of support, and the support bound for pruning of candidates. They also found that there is the *downward closure property* between candidates. The weighted support of a rule $X \Rightarrow Y$ is calculated by the multiplied sum of the weight and the support of itemsets X and

Y. A candidate can be a solution, that is the weighted frequent pattern, if its weighted support is greater than the user-defined threshold, called weighted minimum support. Wang et al. [39] extended the problem by allowing weights to be associated with items in each transaction. Their approach ignores weights when finding frequent itemsets, but only considers during the association rule generation. Tao et al. [40] proposed an improved model of weighted support measurement and the *weighted downward closure property* using average weight of items in each transaction in the database. Yun et al. [41, 42] also considered the weighted items in the process of finding frequent itemsets, and the length-decreasing support constraints for the new measurement of support. Yao et al. [43, 44] introduced a new weighted mining paradigm, called *utility mining*, which finds all itemsets in a transaction database with utility values higher than the minimum utility threshold. They defined two types of utilities for items, transaction utility, such as the quantity of an item sold in the transaction, and external utility, such as the maximum profit for each item. Then, they proposed an algorithm for discovering frequent itemsets using the utility bound property and the support bound property. Although the above works take the notion of weight into account, they can not be adapted directly to our work because they only concerned on the mining from items, but not from traversals.

2.3 Traversal Mining

Generally, the traversal is movement from one object to another

through a relationship between them, as in crossing between vertices in graph theory. In the World Wide Web environment, for example, users access information of interest and travel from one page to another via the corresponding hyperlink provided. In this example, traversal pattern is a sequence of web pages to be visited commonly by users.

For the traversal pattern mining, there have been few works. Chen et al. [5, 45] addressed the problem of traversal pattern mining, and then proposed algorithms with hashing and pruning techniques. However, they did not consider graph structure on which the traversals occur. Lee et al. [15] proposed the efficient interactive web traversal pattern mining algorithm to reduce the mining time and make the mining results to satisfy the user' requirements. They especially used the extended lattice structure, *Iterative_Update_Lattice*, and *Interactive_ Generate_Candidate* algorithms to solve this problem. Hung et al. [14] proposed a projection-based sequential pattern-growth approach, called *PrefixUnion*, for mining traversal patterns efficiently. It is a mining mechanism based on *inter-pattern growth* and *intra-pattern growth*. These two pattern growth criteria are used to minimize useless pattern growth by finding projected-patterns. Mobasher et al. [46] proposed a framework for *Web mining*, the applications of data mining and knowledge discovery techniques to data collected in World Wide Web transactions. And they presented a Web usage mining system, called *WEBMINER*. Its main purpose is the revealing of usage patterns in the given Web site, based on the application of several data mining techniques. Although they discover association rule or sequential patterns from Web access logs, Web structure is not considered. Ezeife et al. [47] proposed a more

efficient approach by using the Web access pattern tree, called WAP-tree, to mine frequent sequences. The WAP-tree is a sequential pattern mining technique for Web log access sequences. It stores the original Web access sequence database on a prefix tree, which is similar to the frequent pattern tree, called FP-tree, for storing non-sequential data.

2.4 Graph Traversal Mining

Graphs are used to represent a collection of related objects such as networks, biological system, electronic systems, and parallel computer architectures. Graph traversal algorithms are important since graphs are a common data structure in which information is distributed [12].

For mining the traversal patterns based on the graphs, there have been few works. Nanopoulos et al. [6, 7] proposed the problem of mining patterns from graph traversals. They defined new criteria for the support and subpath containment, and then proposed algorithms with a trie structure. They considered the graph, on which traversals occur, as well as the traversal in the mining process. Jing et al. [13] presents an approach based on suffix array for frequent reference path generation in Web environment. Borges and Levene [48] addressed the extraction of composite association rules from the structured data of World Wide Web. In this work, the notion of confidence and support measures are formalized in the context of directed graphs, and two algorithms are proposed. The first is a modification of the Depth-First-Search algorithm and the other uses an incremental approach for mining association rules.

Although the above works dealt with the mining of traversal patterns,

to the best of our knowledge, there is no work which considers the notion of weight as our work.

Chapter 3 Mining Patterns from Weighted Traversals on Unweighted Graph

This chapter presents a new method for mining patterns from weighted traversals on a graph. The method proposed in this chapter is mainly composed of three phases. The graph augmentation phase is a pre-processing phase, in which each edge of the base graph is augmented with average and standard deviation of traversal weights. The frequent pattern discovery phase is the main phase, in which frequent patterns are discovered from the augmented graph and traversal database. The pattern priority phase is a post-processing phase, in which the frequent patterns are ranked according to their importance to users. We first define some related notations and concepts, formalize problem statement, and then propose algorithms for these phases.

3.1 Definitions and Problem Statements

Definition 3.1. A *simple directed graph* is a finite set of vertices and edges, in which each edge joins one ordered pair of vertices. The graph contains no self loop which joins a vertex with itself. A *base graph* is a simple directed graph, on which traversals occur.

For example, the base graph shown in Fig. 3.1 has 5 vertices and 9 edges, in which each vertex and edge have no weights.

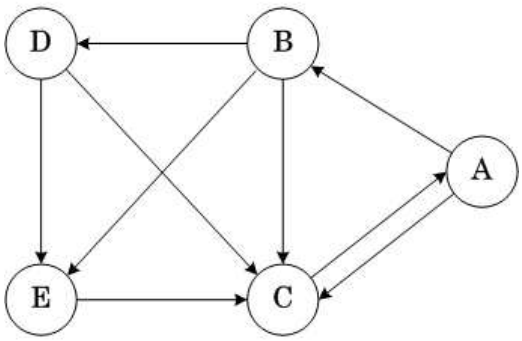

Fig. 3.1 An example of a base graph

Definition 3.2. A *traversal* is a sequence of consecutive edges of a base graph. It can be represented with a sequence of the connecting vertices of each edge, thus a traversal $t = < v_1, v_2, ..., v_n >$. A weighted traversal is a traversal, in which each edge in a traversal has an associated weight. Thus a traversal t with associated weights w is represented as $(t, w) = (< v_1, v_2, ..., v_n >, < w_1, w_2, ..., w_{n-1} >)$, where w_i is the weight of edge $< v_i, v_{i+1} >$. A *traversal database* is a set of weighted traversals.

Fig. 3.2 depicts an example of traversal database. In this database, there are 10 traversals which traverse the base graph shown in Fig. 3.1. The first traversal (TID=1), for example, visits a base graph in the order of vertices A, B and C through the edge $< A, B >$ with the weight 2.2, and $< B, C >$ with the weight 2.0.

TID	Traversal	Weight
1	<A, B, C>	<2.2, 2.0>
2	<B, D, E, C, A>	<3.0, 4.3, 3.5, 3.1>
3	<C, A, B, D>	<2.9, 2.0, 4.0>
4	<D, C, A>	<4.0, 3.0>
5	<B, C, A>	<2.2, 2.9>
6	<A, B, E, C>	<2.1, 3.4, 3.2>
7	<A, B, D, E, C>	<1.4, 3.9, 4.4, 3.2>
8	<B, E, C>	<2.3, 3.4>
9	<B, D, C>	<3.8, 3.1>
10	<C, A, B, D>	<2.5, 2.2, 4.1>

Fig. 3.2 An example of a traversal database

Definition 3.3. A *subtraversal* is any subsequence of consecutive vertices in a traversal. If $t = < v_1, v_2, ..., v_n >$ is a traversal, then $s = < s_1, s_2, ..., s_m >$ is a subtraversal of t when there exists a $k \geq 0$ such that $v_{j+k} = s_j$ for all $1 \leq j \leq m$, and $j + k \leq n$. If an arbitrary pattern is a subtraversal of a traversal, then we say that the pattern is contained in the traversal, or the traversal contains the pattern.

For Example, consider the traversals shown in Fig. 3.2. In the first traversal (TID=1), $< A, B, C >$, we have two kinds of subtraversal, $< A, B >$ and $< B, C >$ of length 2. If there is the pattern $< A, B >$, then we can say that a pattern $< A, B >$ is contained in the traversal (TID=1), or the traversal (TID=1) contains the pattern $< A, B >$.

Definition 3.4. Let $G = (V, E)$ be a base graph, and D be a traversal database, then an *augmented graph* G_w is defined as follows. Each node

$v_i \in V$ is assigned with a weight w_i. Each edge, $< v_i, v_j > \in E$, is labeled with a pair of average weight and standard deviation, (μ_{ij}, σ_{ij}), which are obtained from the weights of the corresponding edges of traversals in D.

Definition 3.5. A *confidence interval* is an interval between two numbers, within which a random variable X lies with a *confidence level*. In our problem, if a weight lies within the confidence interval, then it is considered as a confident one, but if it lies outside the confidence interval, it is considered as an outlier.

For example, in the Gaussian distribution, the 95% confidence level is given by

$$Pr(\mu - 1.960 \times \sigma \leq X \leq \mu + 1.960 \times \sigma) = 0.95 \quad (3.1)$$

In Equation (3.1), μ and σ are average and standard deviation of weights of edges in an augmented graph G_w, respectively. And the constant value 1.960 is used to calculate confidence interval for the confidence level 95%. Fig. 3.3 depicts the confidence interval for confidence level 95% and Table 3.1 presents constant values multiplied by standard deviation corresponding to various confidence levels.

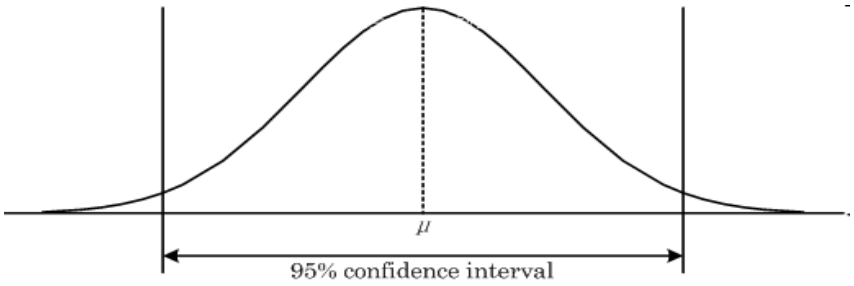

Fig. 3.3 The confidence interval for 95% confidence level

Table 3.1 Constant values for confidence levels

Confidence level	Constant value
80%	1.282
90%	1.645
95%	1.960
98%	2.326
99%	2.576

Defining the support and the ratio as the problem in this chapter is stated as follows. Given a base graph and weighted traversals on the graph, find all patterns contained in the traversals whose ratio is larger than *minsup*. The ratio is called *support*, and a pattern with the support larger than *minsup* is also said to be *frequent*. When counting the support, the weights of traversals should lie within a specified *confidence interval*. In addition, we determine the priority of frequent patterns according to their importance criteria besides the support.

3.2 Mining Frequent Patterns

3.2.1 Augmentation of Base Graph

When a base graph and weighted traversals are given, first phase of the algorithm is to augment the base graph with supplementary information. The supplementary information includes average and standard deviation of weights for each edge, and those for each vertex.

Fig. 3.1 and 3.2 depict an example of base graph and traversal database. On the base graph, all the traversals traverse the vertices through the edges. The traversal (TID=1), for example, traverses consecutively the vertices A, B and C through the edge $< A, B >$ with the weight 2.2, and $< B, C >$ with 2.0.

Given the base graph and traversal database, the base graph can be augmented as follows. For each edge of the base graph, we can collect corresponding weights of the edge from the traversal database, and then calculate average and standard deviation. For the edge $< A, B >$ in Fig. 3.2 as an example, the collected weights are 2.2, 2.0, 2.1, 1.4 and 2.2. Then average 2.0 and standard deviation 0.3 are calculated. Resulting augmented graph is obtained as in Fig. 3.4. Each vertex are also assigned with an arbitrary weight, which may reflect the importance of the vertex.

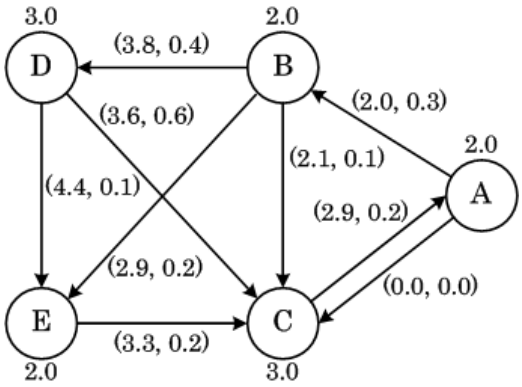

Fig. 3.4 An example of an augmented graph

3.2.2 In-Mining Algorithm

Main phase of the algorithm is to find frequent patterns from given traversal database and augmented graph. To derive the algorithm, we first investigate an important property of patterns. Let the length of a pattern be the number of vertices contained. On the augmented graph, any pattern $P = < p_1, p_2, ..., p_k >$ of length k has exactly two subpatterns of length $k-1$, i.e., $< p_1, p_2, ..., p_{k-1} >$ and $< p_2, p_3, ..., p_k >$. For example, a pattern $< A, B, D, E, C >$ in Fig. 3.2 has two subpatterns, $< A, B, D, E >$ and $< B, D, E, C >$. Therefore, a pattern of length k is frequent only if its two subpatterns of length $k-1$ are also frequent. Such downward closure property allows us to develop a level-wise algorithm like the *Apriori* algorithm [3].

Fig. 3.5 shows the In-Mining Traversal Patterns (IMTP) algorithm proposed in this chapter, which performs in a level-wise manner. The candidate patterns of length 1 are initialized with all vertices of the augmented graph. In each pass of the algorithm, the traversal database

is scanned to count the supports of all candidates. The supports are then adjusted according to the specified confidence interval. Next, frequent patterns are determined from candidates whose supports are larger than the specified minimum support. Finally, new candidates are obtained from the frequent patterns for next pass. The procedure repeats until no more candidates are generated.

Algorithm. ***IMTP***

Input: augmented graph G_w,
traversal database D,
minimum support *minsup*,
confidence level CL

Output: frequent patterns L_k

```
begin
  // initialize candidate patterns of length 1
  C_1 ← set of all vertices;
  k = 1;

  // while candidates exist
  while (| C_k| > 1) {
    // count supports for candidate patterns
    for each traversal t ∈ D {
      P= {p|p ∈ C_k, p is a subtraversal of t};
      ∀ p ∈ P, p.count++;
    }
    // prune candidate patterns w.r.t confidence level
    if (k ≥ 2)
      C_k ← PruneCandidates(C_k, G_w, CL);

    // generate frequent patterns
    L_k = {p|p ∈ C_k, p.count ≥ minsup};

    // generate candidate patterns for next pass
    C_{k+1} ← GenCandidates(L_k, G_w);
    k++;
  }
end;
```

Fig. 3.5 Algorithm to discover frequent traversal patterns (IMTP)

In the algorithm, *PruneCandidates*() adjusts the supports of candidate patterns as follows. Let a pattern $P = < p_1, p_2, ..., p_k >$ be a subtraversal of a weighted traversal $(t, w) = (< v_1, v_2, ..., v_n >, < w_1, w_2, ..., w_{n-1} >)$. If there is an edge $< v_i, v_j >$ in the part of the traversal coincided with the pattern, whose weight w_i lies outside the confidence interval, then the traversal can not contribute for the support of the pattern. For example, even though the pattern $< A, B, D >$ is contained in the traversal (TID=7), $(< A, B, D, E, C >, < 1.4, 2.3, 4.4, 3.2 >)$ in Fig. 3.2, the traversal can not contribute for the support because its edge $< A, B >$ has the weight 1.4 which lies outside the confidence interval 1.41 ~ 2.59. For determination of the confidence interval for each edge of the augmented graph, we assume that the distribution of weight values follows the normal distribution. As in almost applied practices, if the confidence interval corresponds to the 95% confidence level, then $Pr(\mu - 1.960 \times \sigma \leq X \leq \mu + 1.960 \times \sigma) = 0.95$, where μ is the average and σ is the standard deviation. In other words, 95% of weight values are considered to exist within the confidence interval, $(\mu - 1.960 \times \sigma)$ ~$(\mu + 1.960 \times \sigma)$, and the other 5% resides outside the interval. For example, the edge $< A, B >$ in Fig. 3.2 has the confidence interval, (2.0 − 1.960 × 0.3) ~ (2.0 + 1.960 × 0.3) ≡ 1.41 ~ 2.59. If a weight value lies outside this interval, then it can be considered as an outlier. Therefore, traversals whose edges have such weight values can not contribute for the support of patterns.

In the algorithm, *GenCandidates*() generates new candidate patterns for next pass. By the downward closure property, new candidates of length $k+1$ can be obtained by joining the frequent patterns of length k. If

there are two frequent patterns of length k, $< p_1, p_2, ..., p_k >$ and $< p_2, p_3, ..., p_{k+1} >$, a new candidate pattern of length $k+1$, $< p_1, p_2, ..., p_{k+1} >$ can be obtained. For example, $< A, B, C >$ and $< B, C, D >$ result in $< A, B, C, D >$. Note that $< A, B, C >$ and $< C, D, E >$ can not be joined to make $< A, B, C, D, E >$.

An example of the algorithm is shown in Fig. 3.6, which is derived from the traversal database D in Fig. 3.2, and the augmented graph in Fig. 3.4. We assume the minimum support as 2, and the confidence level as 95%. The algorithm initializes the candidates C_1 of length 1 with all the vertices. By scanning the database, the support of each candidate is determined as shown in C_1. The candidates, whose support is larger than 2, become the frequent patterns of length 1 as in L_1. By joining the frequent patterns, new candidates of length 2 are obtained as in C_2, after deleting non-existing edges in the augmented graph. The database is scanned again to count the support of the candidates. The supports are then adjusted by using the confidence interval. For example, the support of the pattern $< A, B >$ is 5 initially, and is decreased to 4. This is due to the fact that the weighted traversal (TID=7), $(< A, B, D, E, C >, < 1.4, 2.3, 4.4, 3.2 >)$ can not contribute for the support since the weight 1.4 of the edge $< A, B >$ lies outside the confidence interval 1.41 ~ 2.59. From the adjusted candidates, the frequent patterns L_2 are obtained. Again, the candidates of length 3, C_3, are obtained by joining the L_2. For example, $< A, B >$ and $< B, C >$ result in $< A, B, C >$. The algorithm proceeds similarly up to the L_3, and then terminates as no candidate of length 4 can be generated by joining L_3's.

C_1

Candidate Pattern	Pruned Support
<A>	8
<B>	9
<C>	10
<D>	6
<E>	4

L_1

Frequent Pattern	Pruned Support
<A>	8
<B>	9
<C>	10
<D>	6
<E>	4

C_2

Candidate Pattern	Pruned Support
<A, B>	4
<A, C>	0
<B, C>	2
<B, D>	4
<B, E>	2
<C, A>	4
<D, C>	2
<D, E>	2
<E, C>	4

L_2

Frequent Pattern	Pruned Support
<A, B>	4
<B, C>	2
<B, D>	4
<B, E>	2
<C, A>	4
<D, C>	2
<D, E>	2
<E, C>	4

C_3

Candidate Pattern	Pruned Support
<A, B, C>	1
<A, B, D>	2
<A, B, E>	1
<B, C, A>	1
<B, D, C>	1
<B, D, E>	1
<B, E, C>	2
<C, A, B>	1
<D, C, A>	1
<D, E, C>	2
<E, C, A>	1

L_3

Frequent Pattern	Pruned Support
<A, B, D>	2
<B, E, C>	2
<D, E, C>	2

Fig. 3.6 An example of discovering frequent patterns

3.2.3 Pre-Mining Algorithm

The algorithm *IMTP* described in the previous section examines the confidence interval in all steps to generate candidate patterns C_{k+1} from L_k. This causes the complexity of the algorithm to increase because the algorithm applies the confidence interval over and over again. To cope with this difficulty, we propose an algorithm, called Pre-Mining Traversal Patterns (PMTP). *PMTP* divides a traversal including outlier into split traversal over 2.

Definition 3.6. If a traversal $(t, w) = (< v_1, v_2, ..., v_n >, < w_1, w_2, ..., w_{n-1} >)$ includes an edge $< v_i, v_{i+1} >$ outside the confidence interval, then it can be split into 2 sub-traversals, $(t, w)' = (< v_1, v_2, ..., v_i >, < w_1, w_2, ..., w_{i-1} >)$ and $(t, w)'' = (< v_{i+1}, v_{i+2}, ..., v_n >, < w_{i+1}, w_{i+2}, ..., w_{n-1} >)$. A *split traversal database* is defined as a set of split traversals thus obtained.

Fig. 3.7 is a split traversal database converted from traversal database in Fig. 3.2 using augmented graph of Fig. 3.4, and Definition 3.6. For instance, in traversal (TID=2), $(< B, D, E, C, A >, < 3.0, 4.3, 3.5, 3.1 >)$ in Fig. 3.2, the edge $< B, D >$ lies outside the confidence interval 3.02 ~ 4.58 because the edge $< B, D >$ has the weight 3.0. Therefore, traversal (TID=2) is split into 2 sub-traversals as $< B >$ and $< D, E, C, A >$.

TID	Traversal	Weight
1	<A, B, C>	<2.2, 2.0>
2′	<B>	<0.0>
2″	<D, E, C, A>	<4.3, 3.5, 3.1>
3	<C, A, B, D>	<2.9, 2.0, 4.0>
4	<D, C, A>	<4.0, 3.0>
5	<B, C, A>	<2.2, 2.9>
6	<A, B, E, C>	<2.1, 3.4, 3.2>
7′	<A>	<0.0>
7″	<B, D, E, C>	<3.9, 4.4, 3.2>
8	<B, E, C>	<2.3, 3.4>
9	<B, D, C>	<3.8, 3.1>
10′	<C>	<0.0>
10″	<A, B, D>	<2.2, 4.1>

Fig. 3.7 An example of split traversal database

Fig. 3.8 presents the algorithm *PMPT* which discovers the frequent traversal patterns from the split traversal database, D'. In Fig. 3.8, the function *SplitTraversals*() splits traversal database D into D' by Definition 3.6. In this example, each traversals of Fig. 3.2 is converted into split traversal database D' of Fig. 3.7 by applying *SplitTraversals*(). Hence, we can expect performance enhancement in discovering frequent traversal patterns, because there are no outliers in the split traversals, and no examinations in confidence interval of each mining step like *IMTP*. But there may be the incremental costs, such as pre-process for traversal database, and size of traversal database. Therefore, totally enhanced performance will be presented by experiments.

Algorithm. ***PMTP***

input: weighted base graph G_w,
traversal database D,
minimum support *minsup*,
confidence level CL

output: frequent patterns L_k

```
begin
  // split traversals into sub-traversals
  D' = SplitTraversals(G_w, D, CL);

  C_1 ← set of all vertices;
  k = 1;

  // while candidates exist
  while (|C_k| > 0) {
    // count supports for candidate patterns
    for each traversal t' ∈ D' {
      P = {p | p ∈ C_k, p is a sub-traversal of t'};
      ∀ p ∈ P, p.count++;
    }

    // obtain frequent patterns
    L_k = {p | p ∈ C_k, p.count ≥ minsup};

    // generate candidate patterns for next step
    C_{k+1} ← GenCandidates(L_k, G_w);
    k++;
  }
end;
```

Fig. 3.8 Enhanced algorithm to discover frequent traversal patterns (PMTP)

3.2.4 Priority of Patterns

When mining a large database, the number of patterns discovered can easily exceed the capabilities of a human user to identify interesting results. To address this problem, various techniques have been suggested to reduce or order the patterns prior to presenting them to the user.

The algorithm estimates the importance of each pattern, as in the previous works, according to the number of their occurrences in the traversals. Although such support is concerned as the primary criterion for the most problems, variety of supplementary information can be adopted as secondary criteria. This thesis proposes a possible criterion shown in Equation (3.2).

$$PP(P) = {}^{port(P)} + \frac{|I_P|}{|E|} + \frac{\sum_{e_i \in P} w_i}{\sum_{e_j \in E} w_j} + \frac{\sum_{v_k \in P} w_k}{\sum_{v_l \in V} w_l} \tag{3.2}$$

In Equation (3.2), P denotes a pattern, E total edges, V total vertices, I_P the number of edges incident into P. The priority of any pattern P, called $PP(P)$, is determined by combining support, ratio of incident edges, ratio of edge weights, and ratio of vertex weights. The reason behind the combination is that a pattern becomes more important as it occurs more often, more referred from other vertices, and edges and vertices with higher weights. Fig. 3.9 shows the pattern priority of the frequent patterns from . Although the three patterns have the same support, they can be further ranked according to their priority.

Frequent Pattern	Pruned Support	Pattern Priority	Rank
<A, B, D>	2	2.93	3
<B, E, C>	2	3.28	2
<D, E, C>	2	3.42	1

Fig. 3.9 An example of pattern priority

3.3 Experimental Results

We conducted several experiments on the algorithms, specifically to evaluate the effect of confidence interval. For the experiments, base graphs are generated synthetically according to the parameters, i.e., the number of vertices V and the number of edges E leaving from each vertex, called the out-degree or fanout of vertex. And all vertices have at least one fanout. We then generate traversals, each of which traverses on the base graph. During the generation, weights are assigned to the edges in the traversals, and have the normal distribution.

The goal of the experiments is to examine the usefulness of confidence interval in the mining process. We will also verify that *PMTP* algorithm is faster than IMTP algorithm for the execution times in various experimental environments. The experimental environments are shown in Table 3.2. And Table 3.3 presents all the symbols used in the experiments of this chapter.

Table 3.2 Experimental environments

Resource	Description
Operating System	Windows XP Professional, SP 2
Database	Microsoft SQL Server 2000
Programming Language	Microsoft Visual C++ 6.0
PC machine	Pentium Ⅳ 3 GHz with 1 GB main memory

Table 3.3 Symbols representing the parameters of synthetic data

Symbol	Description
V	the number of vertices in base graph
E	the number of edges in base graph
D	the average number of fanout per vertex
T	the number of traversals
M	the maximum length of traversals
S	threshold (minimum support, %)
C	confidence level (%)

For example, V=100, E=300, T=10K, M=50, S=5, and C=95 represent a group of experimental data with 100 vertices, 300 edges, 10,000 traversals, the maximum length of traversals as 50, 5% minimum support, and 95% confidence level, which means that roughly 95% of edge weights are confident, and remaining 5% are outliers. In most experiments, we could clearly see that the *PMTP* is much more efficient than the *IMTP*, because the *IMTP* needs more time to classify the weights on traversals into confident ones and outliers.

Experiment 1: Execution times for different numbers of traversals

The experiment compares the execution times of two algorithms, *IMPT* and *PMTP*, for different numbers of traversals. For this experiment, the dataset have V=100 · E=300 · S=5 · M=50 · C=95, and the number of traversals vary from 10,000 to 50,000.

Table 3.4 Execution times for dataset at different numbers of traversals

	Runtime (in seconds) at different numbers of traversals				
Algorithms	10,000	20,000	30,000	40,000	50,000
IMPT	28	41	63	82	101
PMTP	10	19	29	38	47

From Table 3.4 and Fig. 3.10, the gap between execution times of two algorithms becomes larger as the number of traversals increases. We can verify the *IMTP* algorithm is more time-consuming algorithm. This is because the cost for testing outliers using confidence interval increases as the number of traversals increases.

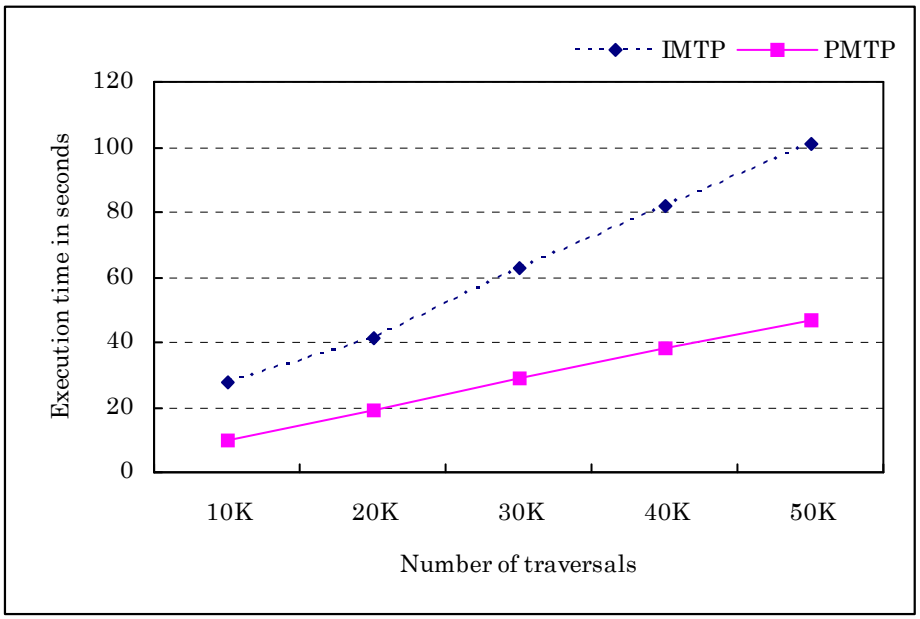

Fig. 3.10 Execution times w.r.t the number of traversals
(V=100 · E=300 · S=5 · M=50 · C=95)

Experiment 2: Execution times for different numbers of vertices

The experiment compares the execution times of two algorithms for different numbers of vertices. For this experiment, the dataset have 10,000 traversals, 5% minimum support, 95% confidence level, maximum length of traversals as 50, and the number of vertices varies from 100 to 500. Table 3.5 and Fig. 3.11 show the performance of *IMTP* and *PMTP* algorithms. As the experiment 1, the results of this experiment verifies that *PMTP* is more good algorithm than *IMTP*.

Table 3.5 Execution times for dataset at different numbers of vertices

	Runtime (in seconds) at different numbers of vertices				
Algorithms	100	200	300	400	500
IMPT	28	39	43	46	47
PMTP	10	15	15	13	15

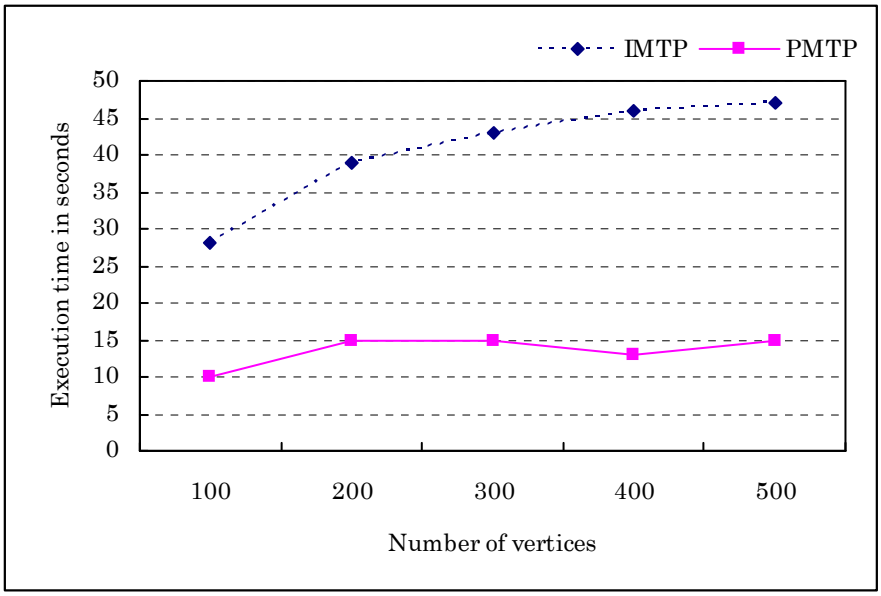

Fig. 3.11 Execution times w.r.t the number of vertices (T=10K · D=3 · S=5 · M=50 · C=95)

Experiment 3: Execution times for different numbers of edges

The experiment compares the execution times of two algorithms for different numbers of edges. The difference of the number of edges in a graph with fixed number of vertices means that the graph density is different. The graph density is defined as $D = |E| / (|V| \times |V - 1|)$, where

$|E|$ denotes the number of edges and $|V|$ the number of vertices. For this experiment, the dataset have V=100 · T=10K · S=5 · M=50 · C=95, and the number of edges varies from 150 to 500. It means that graph density changes from about 0.015 to 0.051. From Table 3.6 and Fig. 3.12, we can know that *PMTP* is usually faster than *IMTP* when the graph density varies.

Table 3.6 Execution times for dataset at different numbers of edges

	Runtime (in seconds) at different numbers of edges							
Algorithms	150	200	250	300	350	400	450	500
IMPT	52	23	26	20	22	26	28	29
PMTP	15	9	12	10	11	12	15	14

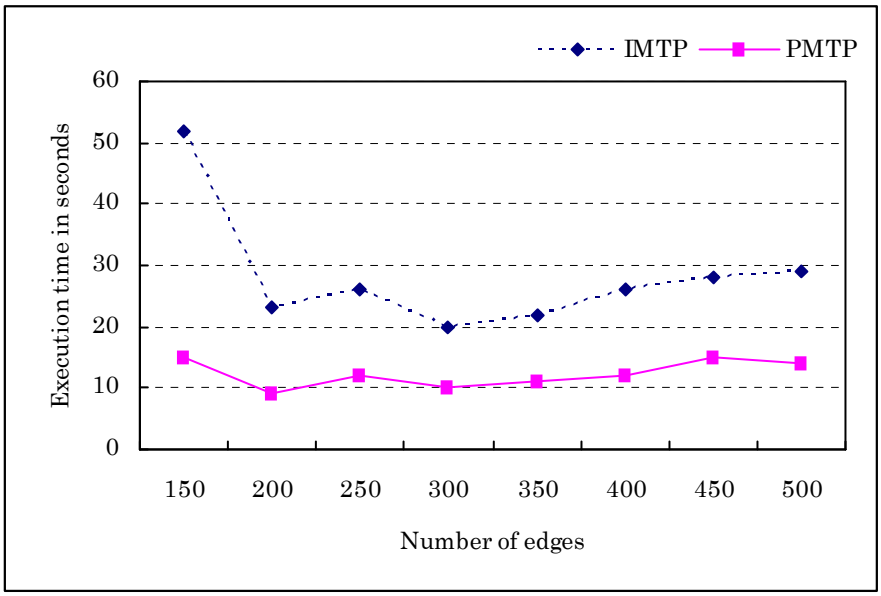

Fig. 3.12 Execution times w.r.t the number of edges (V=100 · T=10K · S=5 · M=50 · C=95)

Experiment 4: Execution times for different minimum supports

The experiment compares the execution times of two algorithms, IMTP and PMTP, for different minimum supports. For this experiment, the dataset have V=100 · E=300 · T=50K · M=50 · C=95, and minimum supports vary from 1% to 10%. From Table 3.7 and Fig. 3.13, we can see that the execution times of all algorithms decrease as the minimum support increases and the gap between the execution times of two algorithms becomes smaller due to the decrease of target traversals.

Table 3.7 Execution times for dataset at different minimum supports

	Runtime (in seconds) at different numbers of thresholds									
Algorithms	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
IMPT	290	185	138	113	99	92	80	72	62	58
PMTP	141	85	60	51	45	41	35	34	30	26

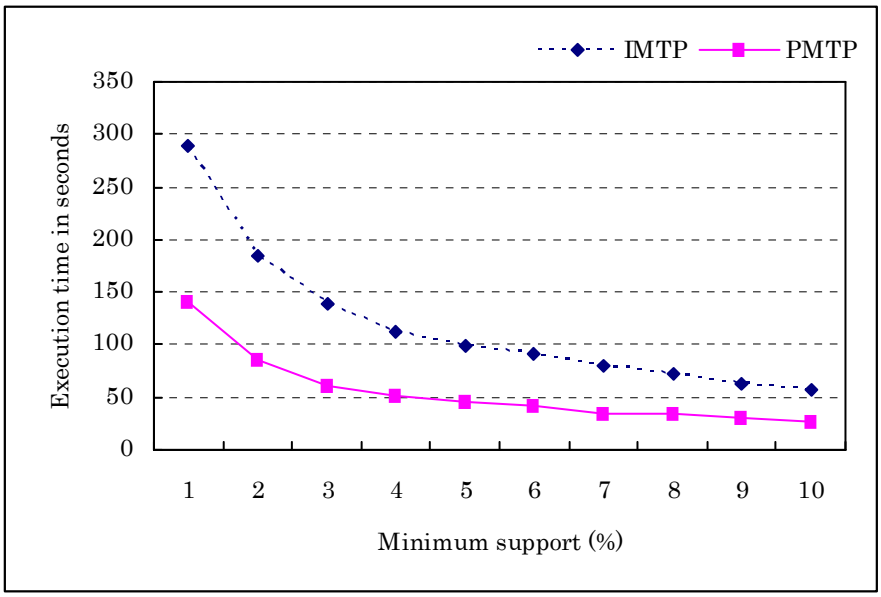

Fig. 3.13 Execution times w.r.t minimum supports
(V=100 · E=300 · T=50K · M=50 · C=95)

Experiment 5: Execution times for different average lengths of traversals

The experiment compares the execution times of two algorithms for different average lengths of traversals. For this experiment, the dataset have V=100 · E=300 · S=5 · T=10K · C=95, and average lengths of traversals varies from 8 to 45. From Table 3.8 and Fig. 3.14, we can see that when the average length of traversals is shorter, the gap between execution times of two algorithms becomes smaller, because the number of edges to be tested by confidence interval becomes fewer.

Table 3.8 Execution times for dataset at different average lengths of traversals

	Runtime (in seconds) at different average lengths of traversals								
Algorithms	8	14	17	22	26	31	35	39	45
IMPT	5	8	10	13	16	18	18	20	21
PMTP	4	5	6	7	9	9	9	10	10

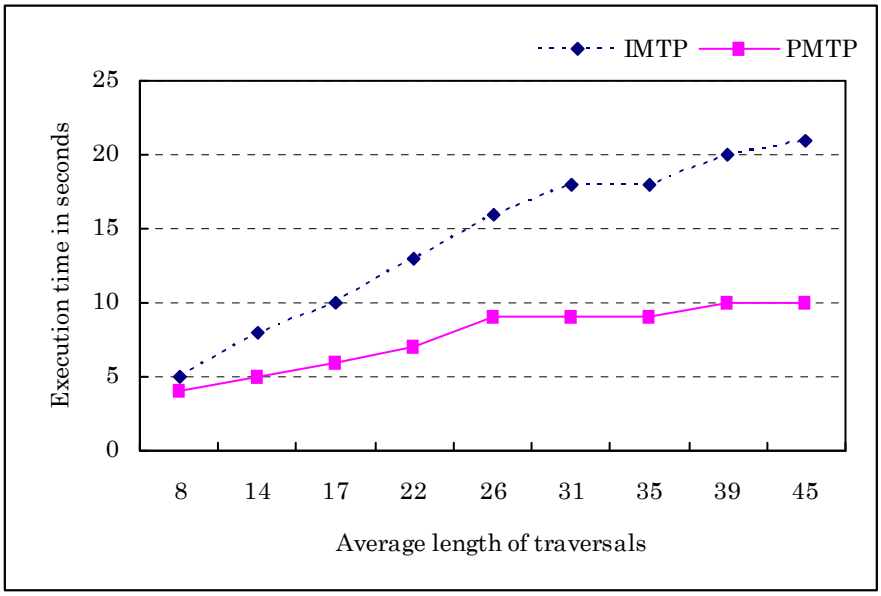

Fig. 3.14 Execution times w.r.t average length of traversals (V=100 · E=300 · S=5 · T=10K · C=95)

Experiment 6: Execution times for different confidence levels

The experiment compares the execution times of two algorithms for different confidence levels. For this experiment, the dataset have V=100 · E=300 · S=5 · M=50 · T=10K, and confidence levels varies from 80% to 99%.

From Table 3.9 and Fig. 3.15, the gap between execution times of two algorithms becomes larger, when the specified confidence level becomes larger. This is because the number of edges to be tested by confidence interval becomes larger.

Table 3.9 Execution times for dataset at different confidence levels

	Runtime (in seconds) at different confidence levels				
Algorithms	80%	90%	95%	98%	99%
IMPT	86	94	99	101	102
PMTP	33	43	46	48	49

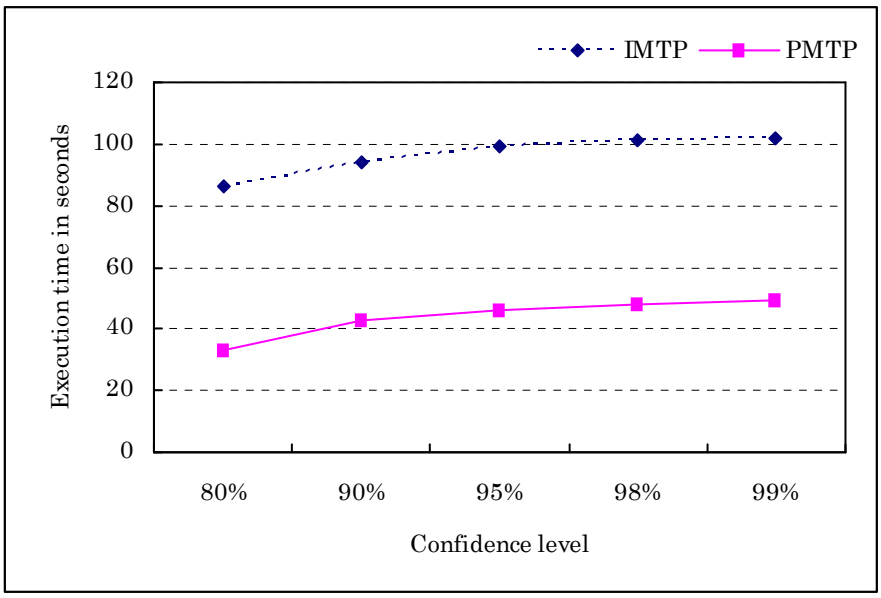

Fig. 3.15 Execution times w.r.t confidence levels
(V=100 · E=300 · S=5 · M=50 · T=10K)

Experiment 7: Execution times with different confidence levels and without confidence level

The experiment compares the execution times of *PMTP* for different confidence levels and without confidence level. For this experiment, the dataset have V=100 · E=300 · T=10K · S=5 · M=50, confidence level varies from 80% to 100%, where 100% confidence level means that the algorithm don't consider confidence interval. From Table 3.10 and Fig. 3.16, the runtime changes from 33 up to 52 in seconds as the confidence level varies from 80% to 100%. We can see that the execution time becomes larger, when the confidence level becomes larger. This is because the number of traversals for testing outliers increases as the confidence level increases.

Table 3.10 Execution times for dataset at different confidence levels in PMTP

	Runtime (in seconds) with different confidence levels					
Algorithm	80%	90%	95%	98%	99%	100%
PMTP	33	43	46	48	49	52

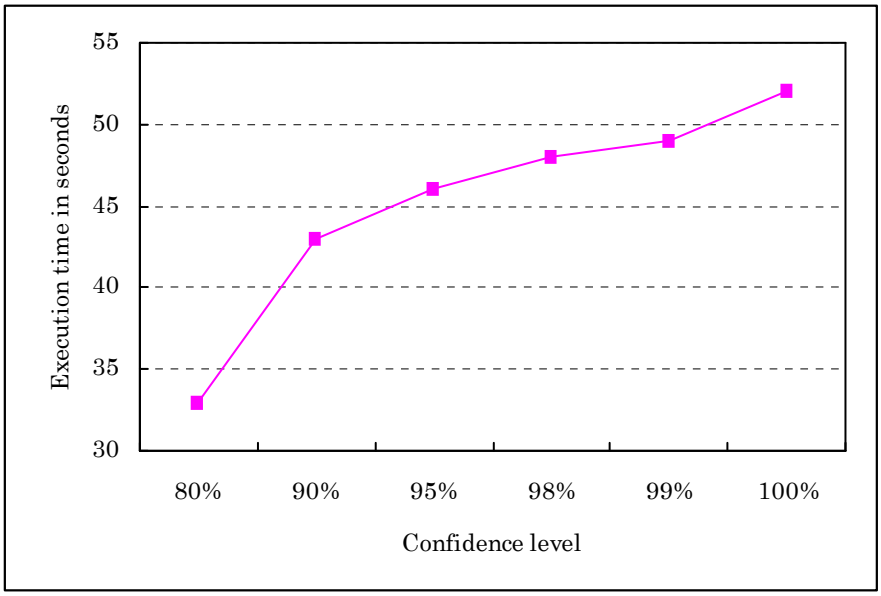

Fig. 3.16 Execution times w.r.t confidence levels in PMTP (V=100 · E=300 · T=10K · S=5 · M=50)

Experiment 8: The number of patterns for different confidence levels and different numbers of traversals

The experiment compares the number of patterns for different confidence levels and different numbers of traversals. For this experiment, the dataset have V=100 · E=300 · S=5 · M=50, and confidence levels vary from 80% to 100% and the number of traversals varies from 10,000 to 50,000. Table 3.11 and Fig. 3.17 show the number of patterns according to the confidence levels and the number of traversals. In this figure, the number of patterns becomes fewer, when the confidence level becomes smaller. This means that the detection of outliers by the confidence interval allows us to discover more reliable patterns. Therefore, we need to select the confidence level with intention.

Table 3.11 The number of patterns for dataset at different confidence levels and numbers of traversals

Confidence levels	The number of patterns for different numbers of traversals				
	10,000	20,000	30,000	40,000	50,000
80%	250	187	189	187	188
90%	296	256	254	254	251
95%	318	297	297	297	297
98%	327	317	318	318	317
99%	330	326	324	324	324
100%	339	336	334	334	334

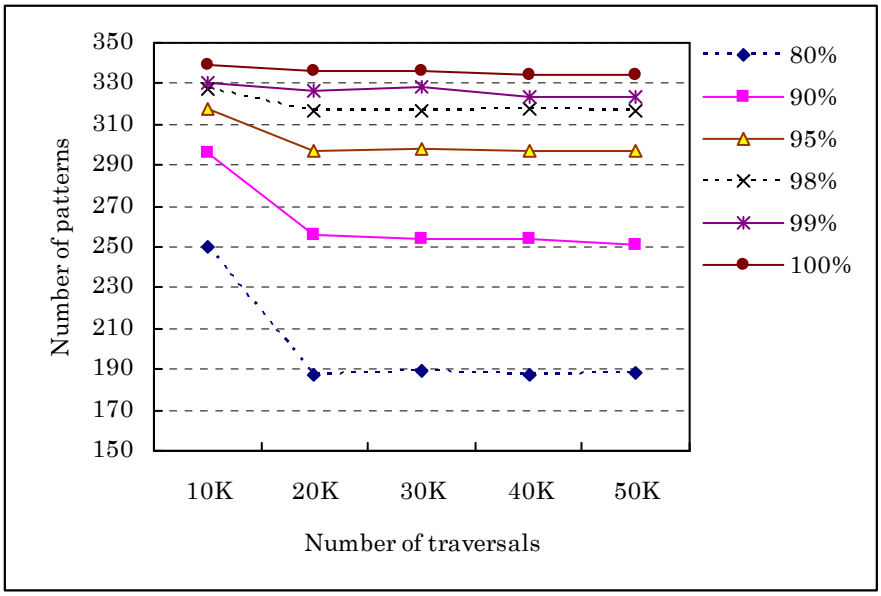

Fig. 3.17 The number of patterns w.r.t the number of traversals and confidence levels
(V=100 · E=300 · S=5 · M=50)

Experiment 9: The number of patterns for different confidence levels and minimum supports

The experiment compares the number of patterns for different confidence levels and minimum supports. For this experiment, the dataset have V=100 · E=300 · S=5 · M=50, and the confidence level varies from 80% to 100% and the minimum support varies from 1% to 10%.

From Table 3.12 and Fig. 3.18, the number of patterns becomes also smaller as minimum support and confidence level increase. This is due to decrease the number of target traversals

Table 3.12 The number of patterns for dataset at different minimum supports and confidence levels

Confidence levels	The number of patterns for different thresholds									
	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
80%	619	377	294	235	187	162	145	117	99	88
90%	992	542	377	312	256	208	186	158	137	111
95%	1,256	668	443	354	297	251	207	181	158	135
98%	1,493	789	526	389	317	279	229	197	173	152
99%	1,606	839	566	401	326	289	243	204	180	159
100%	1,733	889	590	420	336	294	257	208	183	164

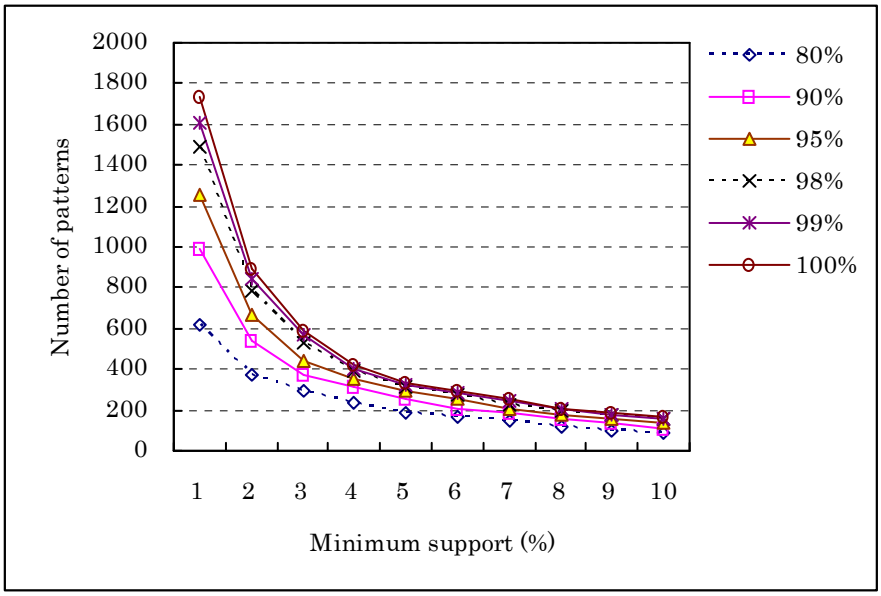

Fig. 3.18 The number of patterns w.r.t minimum supports and confidence levels (V=100 · E=300 · S=5 · M=50)

Experiment 10: The maximum length of patterns for different minimum supports and different confidence levels

The experiment compares the maximum length of patterns for different minimum supports and confidence levels. For this experiment, the dataset have V=100 · E=300 · T=10K · M=50, and the minimum support varies from 1% to 10% and the confidence level varies from 80% to 100%. Table 3.13 and Fig. 3.19 show the effect of confidence level on the length of patterns. As previous experiments, the maximum length of patterns decreases as confidence level decreases. It is because the number of traversals containing pattern decreases if confidence level decreases from 98% to 90%.

Table 3.13 Maximum length of patterns for dataset at different minimum supports and confidence levels

Confidence levels	The maximum length of patterns for different minimum supports									
	1%	2%	3%	4%	5%	6%	7%	8%	9%	10%
80%	5	4	4	4	4	4	3	3	3	3
90%	7	6	5	5	5	4	4	4	4	4
95%	7	6	6	6	5	5	5	4	4	4
98%	8	7	6	6	6	5	5	5	5	4
99%	8	7	7	6	6	6	5	5	5	5
100%	8	8	7	6	6	6	6	5	5	5

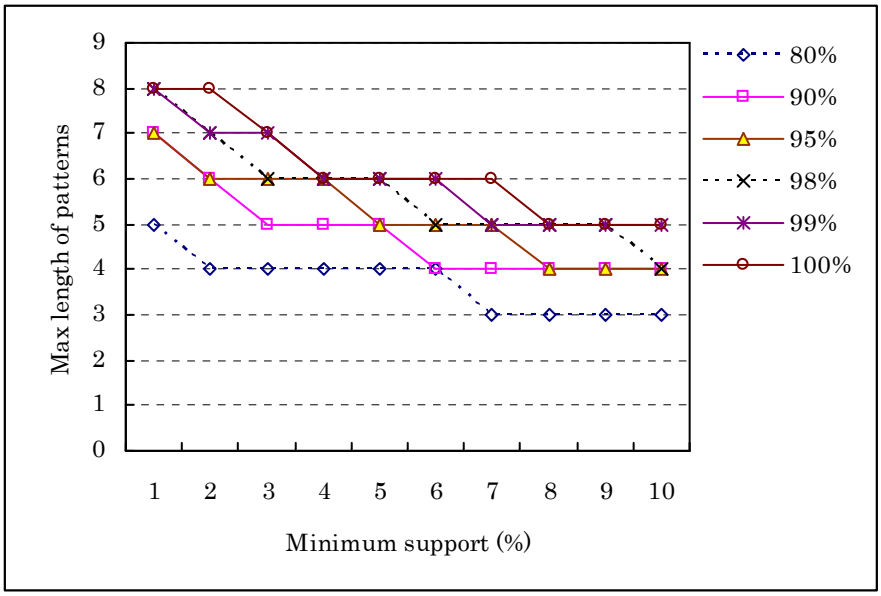

Fig. 3.19 Maximum length of patterns w.r.t minimum supports and confidence levels (V=100 · E=300 · S=5 · M=50)

In above experiments, we examined the execution times for the two algorithms, *IMTP*, and *PTMP*, using synthetic datasets. In *IMTP*, as

stated above, it is necessary to examine confidence interval in each mining step for frequent traversal patterns. On the contrary, *PMTP* executes a preprocessing stage for split traversal database, but no examination of confidence interval. In the most of experiments, the processing time of *PMTP* is decreased about 46.8% when comparison *IMTP* with *PMTP*.

Chapter 4 Mining Patterns from Unweighted Traversals on Weighted Graph

This chapter proposes a new mining method for the discovery of weighted traversal patterns from unweighted traversals on weighted graph. For the weighted graph, this thesis only focused on the weights attached to the vertices.

4.1 Definitions and Problem Statements

Definition 4.1. A *weighted directed graph* is a finite set of vertices and edges, in which each vertex is attached with a weight value, and each edge joins an ordered pair of vertices. A *weighted base graph* is a weighted directed graph, on which traversals occur.

For example, the following base graph has 6 vertices and 8 edges, in which each vertex is associated with a weight in Fig. 4.1.

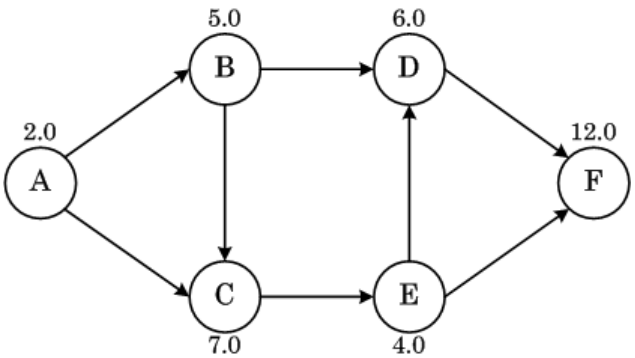

Fig. 4.1 An example of a weighted base graph

Definition 4.2. A *traversal* is a sequence of consecutive vertices along a sequence of edges on a weighted base graph. We assume that every traversal is a path, which has no repeated vertices and edges. The *length* of a traversal is the number of vertices in the traversal. The *weight* of a traversal is the sum of vertex weights in the traversal. A *traversal database* is a set of traversals.

TID	Traversal
1	<A, B>
2	<B, C, E, F>
3	<A, C>
4	<B, C, E>
5	<A>
6	<A, C, E, D>

Fig. 4.2 An example of a traversal database

Definition 4.3. A *subtraversal* is any subsequence of consecutive vertices in a traversal. If a pattern P is a subtraversal of a traversal T, then we say that P is *contained* in T, and T *contains* P.

There is a well known property on such subtraversal [4, 5] as follows.

Property 4.1. Given a traversal of length k, there are only two subtraversals of length $k-1$.

For example, given a traversal of length 4, $< B, C, E, F >$, there are only two subtraversals of length 3, $< B, C, E >$ and $< C, E, F >$. Note that

non-consecutive sequences, such as $< B, C, F >$, are not subtraversals.

Definition 4.4. The *support count* of a pattern P, $scount(P)$, is the number of traversals containing the pattern. The *support* of a pattern P, , is the fraction of traversals containing the pattern. Given a traversal database D, let $|D|$ be the number of traversals.

(4.1)

There is a well known property on such support count and support as follows.

Property 4.2. The support count and the support of a pattern decrease monotonically as the length of the pattern increases. In other word, given a k-pattern P and any l-pattern containing P, denoted by (P, l), where $l > k$, then $scount(P) \geq scount(P, l)$ and .

Given a weighted base graph with a set of vertices $V = \{v_1, v_2, ..., v_n\}$, in which each vertex v_j is assigned with a weight $w_j \geq 0$, we will define the weighted support of a pattern.

Definition 4.5. The *weighted support* of a pattern P, $wsupport(P)$, is

$$wsupport(P) = \left(\sum_{v_j \in P} w_j \right)^{(port(P))} \tag{4.2}$$

Definition 4.6. A pattern P is said to be *weightedly frequent* when the

weighted support is greater than or equal to a given minimum weighted support (*minwsup*) threshold,

$$wsupport(P) \geq minwsup \tag{4.3}$$

For example, given a weighted base graph and a traversal database of Fig. 4.1 and 4.2, and *minwsup* of 5.0, then the pattern $< B, C, E >$ is weightedly frequent since (5.0 + 7.0 + 4.0) × 2/6 = 5.3 ≥ 5.0, but the pattern $< B, C >$ is not since (5.0 + 7.0) × 2/6 = 4.0 < 5.0.

From Equation (4.1), (4.2) and (4.3), a pattern P is weightedly frequent when its support count satisfies:

$$scount(P) \geq \frac{minwsup \times |D|}{\sum_{v_j \in P} w_j} \tag{4.4}$$

We can consider the right hand side of Equation (4.4) as the lower bound of the support count for a pattern P to be weightedly frequent. Such a lower bound, called a support bound, is given by

$$sbound(P) = \left\lceil \frac{minwsup \times |D|}{\sum_{v_j \in P} w_j} \right\rceil \tag{4.5}$$

We take the ceiling of the value since the function $sbound(P)$ is an integer. From Equation (4.4) and (4.5), we can say a pattern P is weighted frequently when the support count is greater than or equal to the support bound.

$$scount(P) \geq sbound(P) \tag{4.6}$$

Note that $sbound(P)$ can be calculated from the weighted base graph without referring the traversal database. On the contrary, $scount(P)$ can be obtained by referring traversal database.

The problem concerned in this chapter is stated as follows. Given a weighted directed graph, called a weighted base graph, and a set of path traversals on the graph, called a traversal database, find all weighted frequent patterns.

4.2 Mining Weighted Frequent Patterns

We propose a method for the mining of weighted frequent patterns. An efficient algorithm for mining large itemsets has been the *Apriori* algorithm [1, 4, 21, 33]. The reason why the *Apriori* algorithm works is due to the *downward closure property* [49], which says all the subsets of a large itemset must be also large. For the weighted setting, however, it is not necessarily true for all subpatterns of a weighted frequent pattern to be weighted frequent. For example, although a pattern <*B*, *C*> is a subpattern of the weighted frequent pattern <*B*, *C*, *E*>, it is not weighted frequent. Therefore, we can not directly adopt Apriori algorithm. Instead, we will extend the notion of the support bound [38], which can be applied to pruning and candidate generation in the mining process.

4.2.1 Pruning by Support Bounds

One of the cornerstones to improve the mining performance is to devise a pruning method which can reduce the number of candidates as many

as possible. We must prune such candidates that have no possibility to become weighted frequent in the future. On the contrary, we must keep such candidates that have a possibility to become weighted frequent in the future. Main concern is how to decide such possibility.

Definition 4.7. A pattern P is said to be *feasible* when it has a possibility to become weighted frequent in the future if extended to longer patterns. In other words, when some future patterns containing P will be possibly weighted frequent.

Now, the pruning problem is converted to the feasibility problem. For the decision of such feasibility, we will first devise the weight bound of a pattern. Let the maximum possible length of weighted frequent patterns be u, which may be the length of the longest traversal in the traversal database. Given a k-pattern P, suppose l-pattern containing P, denoted by (P,l), where $k< l \leq u$. For the additional $(l-k)$ vertices, if we can estimate upper bounds of the weights as $w_{r_1}, w_{r_2}, \cdots, w_{r_{l-k}}$, then the upper bound of the weight of the l-pattern (P,l) is given by

$$wbound(P,l) = \sum_{v_j \in P} w_j + \sum_{j=1}^{l-k} w_{r_j} \tag{4.7}$$

We call this upper bound as l-*weight bound* of P. The first sum is the sum of the weights for the k-pattern P and the second one is the sum of the $(l-k)$ estimated weights, which can be estimated in several ways. We will propose three estimation methods in the following section.

From Equation (4.5) and (4.7), we can derive the lower bound of the

support count for l-pattern containing P to be weighted frequent. Such lower bound, called l-support bound of P, is given by

$$sbound(P,l) = \left\lceil \frac{minwsup \times |D|}{wbound(P,l)} \right\rceil \tag{4.8}$$

Lemma 4.1. A pattern P is feasible if $scount(P) \geq sbound(P,l)$ for some $k < l \leq u$, but not feasible if $scount(P) < sbound(P,l)$ for all $k < l \leq u$.

Proof. Let l_i be that out of l. If $scount(P) \geq sbound(P,l_i)$, then because $scount(P) \geq scount(P,l_i)$ by Property 4.2, there is a possibility to be $scount(P,l_i) \geq sbound(P,l_i)$. It means that (P,l_i) will possibly be weighted frequent. On the contrary, if $scount(P) < sbound(P,l_i)$, then $scount(P,l_i) < sbound(P,l_i)$ because $scount(P) \geq scount(P,l_i)$ by Property 4.2. It means that (P,l_i) will definitely not be weighted frequent.

If a pattern P is feasible then some l-patterns containing P will be possibly weighted frequent. In other word, P has a possibility to be subpatterns of some weighted frequent l-patterns. Therefore, P must be kept to be extended to longer patterns for possible weighted frequent patterns in the coming passes. On the contrary, if a pattern P is not feasible, then all l-patterns containing P will not be weighted frequent. In other word, P certainly has no possibility to be subpattern of any weighted frequent l-patterns. Therefore, P must be pruned.

For example, referring to Fig. 4.1 and Fig. 4.2, given a 2-pattern $<B, C>$, suppose 3-pattern $<B, C, ->$. For the additional vertex '$-$', we

can estimate a possible upper bound of the weight as 12.0, which is the greatest weight among the remaining vertices besides B and C. Therefore, the 3-support bound of $< B, C >$ is

$$sbound(< B, C >, 3) = \left\lceil \frac{5.0 \times 6}{(5.0 + 7.0) + (12.0)} \right\rceil = 2$$

It means if the support count of $< B, C >$ is greater than or equal to 2, some 3-patterns will be possibly weighted frequent. In other word, $< B, C >$ has a possibility to be subpatterns of some weighted frequent 3-patterns. Because the support count of the pattern $< B, C >$ is actually 2, the pattern must be extended to 3-patterns for possible weighted frequent patterns.

Corollary 4.1. A pattern P is feasible if $scount(P) \geq sbound(P)$.

Proof. From Equation (4.5), (4.7) and (4.8), $sbound(P) \geq sbound(P, l)$ for all $k < l \leq u$. Therefore, $scount(P) \geq sbound(P, l)$ for all $k < l \leq u$, which means P is feasible by Lemma 4.1.

In this case, we don't need to estimate $sbound(P, l)$ to decide the feasibility of P. On the contrary, in case of $scount(P) < sbound(P)$, we can not decide the feasibility, and therefore we need to estimate $sbound(P, l)$ to decide the feasibility by Lemma 4.1.

According to Lemma 4.1 along with Corollary 4.1, we can devise a pruning algorithm, called 'pruning by support bounds', as follows.

Algorithm. ***Pruning-SB***

Input: Candidate pattern P,
Unpruned candidate patterns set C_k,
Maximum possible length of pattern l

Output: Pruned candidate patterns set C_k

```
begin
  for each pattern P in candidates set C_k {
    if (scount(P) ≥ sbound(P))
      continue;                    // P is feasible. keep
    for each l from k+1 to u {
      estimate sbound(P, l);
      if (scount(P) ≥ sbound(P, l))
        break;                     // P is feasible. keep
    }
    if (l > u)
      C_k = C_k − {P};             // P is not feasible. prune
  }
end;
```

Fig. 4.3 Algorithm Pruning-SB pruning by support bounds

We can devise another pruning algorithm by using the minimum of l -support bounds.

Definition 4.8. The *maximum l-weight bound*, $wbound(P,+)$, and the *minimum l-support bound* of a pattern P, $sbound(P,+)$, are defined as follows.

$$wbound(P,+) = max(wbound(P,l)),$$

$$sbound(P,+) = min(sbound(P,l)), k < l \leq u.$$

Corollary 4.2. A pattern P is feasible if $scount(P) \geq sbound(P,+)$, but not feasible if $scount(P) < sbound(P,+)$.

Proof. If $scount(P) \geq sbound(P,+)$, then there is at least one l_i such that $scount(P) \geq sbound(P,l_i)$, where $sbound(P,l_i) = sbound(P,+)$. On the contrary, if $scount(P) < sbound(P,+)$, then $scount(P) < sbound(P,l)$ for all $k < l \leq u$.

According to Corollary 4.2 along with Corollary 4.1, we can devise another pruning algorithm, called 'pruning by minimum support bound', as follows.

Algorithm. ***Pruning-MSB***

Input: Candidate pattern P,
Unpruned candidate patterns set C_k

Output: Pruned candidate patterns set C_k

```
begin
  for each pattern P in candidates set C_k {
    if (scount(P) ≥ sbound(P))
      continue;                    // P is feasible. keep
    estimate sbound(P,+);
    if (scount(P) ≥ sbound(P,+))
      continue;                    // P is feasible. keep
    C_k = C_k - {P};               // P is not feasible. prune
  }
end;
```

Fig. 4.4 Algorithm Pruning-MSB pruning by minimum support bound

4.2.2 Candidate Generation

This thesis devises candidate generation algorithms by defining downward closure properties between feasible patterns. If there is a downward closure property between feasible patterns, new candidates can be generated from current feasible patterns.

Definition 4.9. We say that there is *partial downward closure property* when the $(k-1)$-subpattern $< p_1, p_2, ..., p_{k-1} >$ of a feasible k-pattern $< p_1, p_2, ..., p_k >$ is also feasible. We say that there is *full downward closure property* when two $(k-1)$-subpatterns $< p_1, p_2, ..., p_{k-1} >$ and $< p_2, p_3, ..., p_k >$ of a feasible k-pattern $< p_1, p_2, ..., p_k >$ are also feasible.

Note that there are only two $(k-1)$-subpatterns of a k-pattern by Property 4.1. When there is the partial downward closure property, we can generate candidate $(k+1)$-patterns, C_{k+1}, from feasible k-patterns, C_k, as follows.

Algorithm. ***Gen-PDC***

Input: Candidate patterns set C_k,
Weighted base graph G

Output: Joined candidate patterns set C_{k+1}

```
begin
  C_{k+1} = ∅;
  for each P = < p_1, p_2, ..., p_k > in C_k {
    for each edge <p_k, v> in G
      if v is not already in P {        // not repeated vertex
        P is extended to P' = < p_1, p_2, ..., p_k, v >;
        C_{k+1} = C_{k+1} ∪ P';
      }
  }
end;
```

Fig. 4.5 Algorithm Gen-PDC for candidate generation

When there is a full downward closure property, we can generate C_{k+1} in a similar way.

Algorithm. ***Gen-FDC***

Input: Candidate patterns set C_k,
Weighted base graph G

Output: Joined candidate patterns set C_{k+1}

```
begin
  C_{k+1} = ∅;
  for each P = < p_1, p_2, ..., p_k > in C_k {
    for each edge < p_k, v > in G
      if (v is not already in P) and (Q = < p_2, ..., p_k, v > is in C_k) {
        P is extended to P' = < p_1, p_2, ..., p_k, v >;
        C_{k+1} = C_{k+1} ∪ P';
      }
  }
end;
```

Fig. 4.6 Algorithm Gen-FDC for candidate generation

This algorithm will generate less number of candidates than algorithm *Gen-PDC*.

When there is the full downward closure property, C_{k+1} can be alternatively obtained by self-joining C_k. That is, two k-patterns $P = < p_1, p_2, ..., p_k >$ and $Q = < q_1, q_2, ..., q_k >$ will be joined if $p_2 = q_1$, $p_3 = q_2$, ⋯, $p_k = q_{k-1}$, and $p_1 \neq q_k$. This results in a new candidate pattern $< p_1, p_2, ..., p_k, q_k >$. For example, the join of $< A, B, C >$ and $< B, C, D >$ results in $< A, B, C, D >$. This method need not refer to the weighted base graph G, besides for C_2 generation. For C_2 generation, each generated 2-pattern must be excluded if there is no corresponding edge in G.

Algorithm. ***Gen-SQL***

Input: Candidate patterns set C_k,

Output: Joined candidate patterns set C_{k+1}

```
begin
  C_{k+1} = ∅
  CSet = select   P.p_1, P.p_2, ..., P.p_k, Q.q_k
         from     C_k P, C_k Q
         where    P.p_2 = Q.q_1 and P.p_3 = Q.q_2 ...
                  and P.p_k = Q.q_{k-1} and P.p_1 ≠ Q.q_k
  while (CSet ≠ ∅) {
    P' = CSet{< P.p_1, P.p_2, ..., P.p_k, Q.q_k >}
    C_{k+1} = C_{k+1} ∪ P'
  }
end;
```

Fig. 4.7 Algorithm Gen-SQL for candidate generation

In Fig. 4.7, algorithm *Gen-SQL* need not refer to the weighted base graph G, besides for C_2 generation. For C_2 generation, each generated 2-pattern must be excluded if there is no corresponding edge in G.

4.2.3 Mining Algorithm

By combining the pruning and candidate generation algorithms as a whole, we can devise an algorithm for mining weighted frequent patterns. Fig. 4.8 shows the algorithm proposed in this thesis, which performs in a level-wise manner.

Algorithm. ***Mining weighted frequent patterns***

Inputs: Weighted base graph G,
Traversal database D,
Minimum weighted support $minwsup$

Output: Set of weighted frequent patterns L_k

```
begin
  // 1. maximum length of weighted frequent patterns
  u = max(length(t)), t ∈ D;
  // 2. initialize candidate patterns of length 1
  C_1 = V(G);

  for (k = 1; k ≤ u and C_k ≠ ∅; k++) {
    // 3. obtain support counts
    for each traversal t ∈ D {
      for each pattern p ∈ C_k
        if p is contained in t, then p.scount++;
    }
    // 4. determine weighted frequent patterns
    L_k = {p | p ∈ C_k, p.weightedSupport ≥ minwsup};
    //                    equivalently, p.scount ≥ p.sbound

    if (k < u) {
      // 5. prune candidates
      C_k = PruneCandidates(C_k, G, u);
      // 6. generate new candidates for next pass
      C_{k+1} = GenCandidates(C_k, G);
    }
  }
end;
```

Fig. 4.8 Algorithm for mining weighted frequent patterns

In the algorithm as shown in the Fig. 4.8, each step is outlined as follows. Step 1 is to find out the maximum possible length of weighted frequent patterns, which is limited by the maximum length of traversals. Step 2 initializes candidate patterns of length 1 with the vertices of weighted base graph. In Step 3, traversal database is scanned to obtain the support counts of candidate patterns. Step 4 is to determine weighted frequent patterns if the weighted support is greater than or equal to the specified minimum weighted support. Equivalently, if the support count is greater or equal to the support bound. In Step 5, the subroutine *PruneCandidates*() is to prune candidate patterns by checking their feasibility. The algorithm *Pruning-SB* or *Pruning-MSB* can be used according to their efficiency. The remaining patterns are feasible patterns. In Step 6, the subroutine *GenCandidates*() generates new candidate patterns of length $k+1$ from the feasible patterns of length k for the next pass. The algorithm *Gen-PDC*, *Gen-FDC* or *Gen-SQL* can be used according to its applicability and efficiency.

4.3 Estimations of Support Bounds

In this section, we propose two methods for the estimation of weight and support bound.

4.3.1 Estimation by All Vertices

Given a k-pattern P, suppose l-pattern containing P, where $k < l \leq u$. Let V be set of all vertices in the weighted base graph. Among

remaining vertices $(V-P)$, let the vertices with the $(l-k)$ greatest weights be $v_{r_1}, v_{r_2}, ..., v_{r_{l-k}}$. Then, the l-weight bound, $wbound(P,l)$, and the l-support bound, $sbound(P,l)$, of P are defined same as Equation (4.7) and (4.8), respectively.

For example, refer to Fig. 4.1 and Fig. 4.2, the 3-support bound for the pattern $< A >$ is

$$sbound(< A >, 3) = \left\lceil \frac{5.0 \times 6}{(2.0) + (12.0 + 7.0)} \right\rceil = 2$$

Corollary 4.3. $wbound(P,l)$ increases monotonically, and accordingly $sbound(P,l)$ decreases monotonically as l increases.

Let the upper limit of the length of possible weighted frequent patterns be known as u. By Corollary 4.3, the minimum support bound of P is the u-support bound of P,

$$sbound(P,+) = sbound(P,u) \tag{4.9}$$

By Equation (4.9) along with Corollary 4.2, if $scount(P) \geq sbound(P,u)$, then P is feasible. On the contrary, if $scount(P) < sbound(P,u)$, then P is not feasible. This means that we do not need to calculate l-support bounds of P for $k < l \leq u$. Therefore, the pruning algorithm *Pruning-MSB* is more efficient than *Pruning-SB*.

Corollary 4.4. For any p_i in $P = < p_1, p_2, ..., p_k >$, $wbound(P - \{p_i\}, l) \geq$

$wbound(P, l)$, and accordingly $sbound(P-\{p_i\}, l) \leq sbound(P, l)$.

Proof. $wbound(P-\{p_i\}, l)$ is the sum of the vertex weights of P excluding p_i and $(l-k+1)$ greatest vertex weights among the vertices including p_i. This sum is always greater than or equal to the sum of the vertex weights of P and $(l-k)$ greatest vertex weights.

Lemma 4.2. There is the full downward closure property among feasible patterns. That is, if a k-pattern $P=<p_1, p_2, ..., p_k>$ is feasible, then the two $(k-1)$-subpatterns $P_a=<p_1, p_2, ..., p_{k-1}>$ and $P_b=<p_2, p_3, ..., p_k>$ are also feasible.

Proof. The *if* condition means $scount(P) \geq sbound(P, u)$. For P_a, $scount(P_a) \geq sbound(P)$ by Property 4.2, and $sbound(P_a, u) \leq sbound(P, u)$ by Corollary 4.4. Therefore $scount(P_a) \geq sbound(P_a, u)$, which implies P_a is feasible. This is similar for P_b.

Therefore, the candidate generation algorithm *Gen-FDC* or *Gen-SQL* can be applied.

Consider an example.

From the Fig. 4.1 and 4.2, we will show how the weighted frequent patterns are generated from the traversal database. Suppose that the *minwsup* (minimum weighted support) is 5.0.

1. In the *upperLimit*() subroutine, the algorithm will scan the length of

traversals, and returns the maximum length, which is 4 in this example. The maximum length is the upper limit of the length of weighted frequent patterns.

2. During the initialization step, the candidate patterns of length 1 are generated with all vertices of the weighted base graph.

$$C_1 = \{< A >, < B >, < C >, < D >, < E >, < F >\}$$

3. The algorithm repeats as followings.

Pattern *(P)*	*scount(P)*	*sbound(P)*	*wbound(P, 4) / sbound(P, 4)*	Weightedly frequent	Feasible
<A>	4	15	27 / 2		✓
<B>	3	6	30 / 1		✓
<C>	4	5	30 / 1		✓
<D>	1	5	30 / 1		✓
<E>	3	8	29 / 2		✓
<F>	1	3	30 / 1		✓

Candidates for next pass are generated by *Gen-FDC* or *Gen-SQL*.

Pattern *(P)*	*scount(P)*	*sbound(P)*	*wbound(P, 4) / sbound(P, 4)*	Weightedly frequent	Feasible
<A, B>	1	5	26 / 2		
<A, C>	2	4	27 / 2		✓
<B, C>	2	3	30 / 1		✓
<B, D>	0	–	–		
<C, E>	3	3	–	✓	✓
<D, F>	0	–	–		
<E, D>	1	3	29 / 2		
<E, F>	1	2	29 / 2		

Pattern *(P)*	*scount(P)*	*sbound(P)*	*wbound(P, 4) / sbound(P, 4)*	Weightedly frequent	Feasible
<A, C, E>	1	3	25 / 2		
<B, C, E>	2	2	–	✓	✓

In the above example, '−' denotes 'not need'.

The weighted frequent patterns are $\{<C,E>,<B,C,E>\}$.

4.3.2 Estimation by Reachable Vertices

To prune unnecessary candidates as many as possible, support bounds need to be estimated as high as possible. It means that we must estimate weight bounds as low as possible. The previous method, however, has a tendency to over-estimate the weight bounds. This is why the topology of a weighted base graph is not considered. Specifically, the vertices with greatest weights are chosen one after one, even though they can not be reached from the corresponding pattern.

Definition 4.10. Given a weighted base graph G, *k-reachable vertices* from a vertex v is all the vertices reachable from v within the distance k.

k-reachable vertices can be regarded as the vertices within the radius k from v. Therefore, k-reachable vertices include all the $(k-1)$-reachable vertices.

Given a k-pattern P, let $R(P,l)$, $k<l\leq u$, be the $(l-k)$-reachable vertices from the head vertex of P, but not in P and not through the vertices in P. They can be obtained by a level wise manner.

Algorithm. ***Reachable,*** $R(P, l)$

Inputs: Weighted base graph G,
Candidate pattern P,
The number of vertices appended for extending pattern l

Output: Set of weighted frequent patterns L_k

```
begin
  T = {head vertex of P} if (l = k + 1), otherwise N;
  N = ∅;
  for each vertex v in T
    for each edge < v, w > in G
      if w is not in P and R(P, l − 1) and N, then append w to N;
  R(P, l) = R(P, l − 1) ∪ N;
end;
```

Fig. 4.9 Algorithm Reachable for searching reachable vertices from candidate pattern

For example, from Fig. 4.1, $R(<A>, 2)$ is $\{B, C\}$, and $R(<A>, 3)$ is $\{B, C, D, E\}$.

Among the vertices in $R(P, l)$, let the vertices with the $(l-k)$ greatest weights be $v_{r_1}, v_{r_2}, ..., v_{r_{1-k}}$. Then, the l-weight bound, $wbound(P, l)$, and the l-support bound, $sbound(P, l)$, of P are obtained by Equation (4.7) and (4.8), respectively. For example, refer to Fig. 4.1 and Fig. 4.2, the 3-support bound for the pattern $<A>$ is

$$sbound(<A>, 3) = \left\lceil \frac{5.0 \times 6}{(2.0) + (7.0 + 6.0)} \right\rceil = 2$$

Corollary 4.3'. $wbound(P, l)$ increases monotonically, and accordingly $sbound(P, l)$ decreases monotonically as l increases.

By Corollary 4.3', the minimum support bound of P is the u-support bound of P,

$$sbound(P,+) = sbound(P,u) \tag{4.9'}$$

In spite of Equation (4.9'), however, the pruning algorithm *Pruning-SB* may be more efficient than *Pruning-MSB* because the pruning can be decided before u due to the level wise characteristic of the algorithm $R(P,l)$.

Corollary 4.4'. For any p_k in $P = < p_1, p_2, ..., p_k >$, $wbound(P - \{p_k\}, l) \geq wbound(P,l)$, and accordingly $sbound(P - \{p_k\}, l) \leq sbound(P,l)$.

Proof. $wbound(P - \{p_k\}, l)$ is the sum of the vertex weights of P excluding p_k and $(l-k+1)$ greatest vertex weights among the vertices of $R_{l-k+1}(P - \{p_k\})$ which includes all the vertices of $R_{l-k}(P)$ and p_k. This sum is always greater than or equal to the sum of the vertex weights of P and $(l-k)$ greatest vertex weights among the vertices of $R_{l-k}(P)$.

Lemma 4.2'. There is the partial downward closure property among feasible patterns. That is, if a k-pattern $P = < p_1, p_2, ..., p_k >$ is feasible, then the $(k-1)$-subpattern $P_a = < p_1, p_2, ..., p_{k-1} >$ is also feasible.

Proof. The necessary condition means $scount(P) \geq sbound(P,l)$. For P_a, $scount(P_a) \geq sbound(P)$ by Property 4.2, and $scount(P_a, u) \leq sbound(P,u)$ by

Corollary 4.4'. Therefore $scount(P_a) \geq sbound(P_a, u)$ which implies P_a is feasible.

Therefore, the candidate generation algorithm *Gen-PDC* can be only applied.

Consider an example.

Pattern *(P)*	*scount(P)*	*sbound(P)*	*wbound(P, l) / sbound(P, l)*			Weightedly frequent	Feasible
			l = 2	l = 3	l = 4		
<A>	4	15	9 / 4	–	–		✓
<B>	3	6	12 / 3	–	–		✓
<C>	4	5	11 / 3	–	–		✓
<D>	1	5	18 / 2	×	×		
<E>	3	8	16 / 2	–	–		✓
<F>	1	3	×	×	×		

In the above example, '×' denotes 'not applicable'.

Candidates for next pass are generated by *Gen-PDC*.

Pattern *(P)*	*scount(P)*	*sbound(P)*	*wbound(P, l) / sbound(P, l)*		Weightedly frequent	Feasible
			l = 3	l = 4		
<A, B>	1	5	14 / 3	26 / 2		
<A, C>	2	4	13 / 3	27 / 2		✓
<B, C>	2	3	16 / 2	–		✓
<B, D>	0	–	–	–		
<C, E>	3	3	–	–	✓	✓
<E, D>	1	3	22 / 2	×		
<E, F>	1	2	×	×		

Pattern *(P)*	*scount(P)*	*sbound(P)*	*wbound(P, l)* / *sbound(P, l)*	Weightedly frequent	Feasible
			l = 4		
<A, C, E>	1	3	26 / 2		
<B, C, E>	2	2	−	✓	✓
<C, E, D>	1	2	29 / 2		
<C, E, F>	1	2	×		

Pattern *(P)*	*scount(P)*	*sbound(P)*	Weightedly frequent	Feasible
<B, C, E, D>	0	−		
<B, C, E, F>	1	2		

The weighted frequent patterns are $\{< C, E >, < B, C, E >\}$.

4.4 Experimental Results

This section describes experimental results of the mining algorithms, and compares two estimation algorithms, *All* vertices and *Reachable* vertices, by using synthetic dataset. For the experiments, a weighted base graph is generated synthetically according to the parameters, i.e., number of vertices and average number of edges per vertex. And then, we assigned distinctive weight to each vertex in the weighted base graph. Traversal datasets are also generated randomly according to the parameters, i.e., number of traversals and maximum length of traversals.

By these experiments, we compare the running times of two algorithms, *All Estimation Algorithm* and *Reachable Estimation Algorithm*. And then, we examine the number of feasible patterns generated during the mining process. The experimental environments are shown in Table 4.1.

Table 4.1 Experimental environments

Type	Description
Operating System	Windows XP Professional, SP 2
Database	Microsoft SQL Server 2000
Programming Language	Microsoft Visual C++ 6.0
PC Machine	Pentium Ⅳ 3 GHz with 1 GB main memory

Table 4.2 presents all the symbols used in the experiments of this chapter. In the experiments, for example, V=100, E=300, T=10K, M=10, D=3, and S=5 mean a group of data with 100 vertices, 300 edges, 10,000 traversals in the database, the maximum length of traversal as 10, the average number of fanout as 3, and the minimum weighted support as threshold as 5.

Table 4.2 Symbols representing the parameters of synthetic data

Symbol	Descriptions
V	the number of vertices in weighted base graph
E	the number of edges in base graph
D	the average number of fanout per vertex
T	the number of traversals
M	the maximum length of traversals (u)
S	threshold (minimum weighted support)

Experiment 1: Execution times for different numbers of traversals

This experiment compares the execution times of two algorithms for different numbers of traversals. This experiment uses the different

number of traversals to compare the running times of two estimation algorithms, *All* and *Reachable*. The dataset have V=1,000 · E=3,000 · S=5 · M=10, and the number of traversals varies from 10,000 to 50,000.

Table 4.3 Execution times for dataset at different numbers of traversals

	Runtime (in seconds) at different number of traversals				
Algorithms	10,000	20,000	30,000	40,000	50,000
All	41	81	165	270	408
Reachable	744	1,051	1,272	1,484	1,691

From Table 4.3 and Fig. 4.10, it can be seen that the gap between the execution times of two algorithms becomes larger as the number of traversals increases. We can find that the *Reachable* algorithm is more time-consuming. This is because the cost of finding reachable vertices increases when the number of traversals increases.

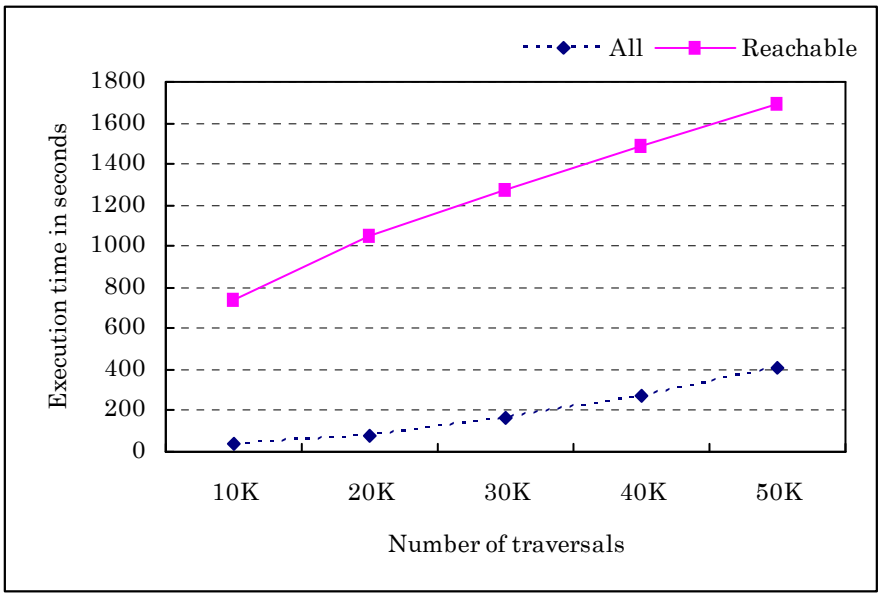

Fig. 4.10 Execution times w.r.t the number of traversals
(V=1,000 · E=3,000 · S=5 · M=10)

Experiment 2: Execution times for different number of edges

This experiment compares the execution times of two algorithms for different number of edges. The difference of the number of edges in a graph with fixed number of vertices means that the graph density is different. The density of a directed graph is defined as $D = |E|/(|V| \times |V-1|)$, where $|E|$ denotes the number of edges and $|V|$ the number of vertices. In generally, there is an inverse relationship between the density and the radius of a graph, then the graph radius becomes smaller as the graph density becomes larger. This experiment compares the running times of two algorithms when the graph density varies.

Table 4.4 Execution times for dataset at different numbers of edges

	Runtime (in seconds) at different numbers of edges							
Algorithms	150	200	250	300	350	400	450	500
All	12	12	14	13	13	14	13	14
Reachable	11	13	15	16	16	18	17	19

For this experiment, the dataset have 100 vertices, 10,000 traversals, minimum weighted support as 5, the maximum length of traversals as 5, and the number of edges varies from 150 to 500. This experiment tests performance of two estimation algorithms when the graph density varies from 0.015 to 0.051. We can verify that if graph density becomes smaller, i.e, 0.015, *Reachable Estimation Algorithm* becomes more fast than *All Estimation Algorithm*. This is because *Reachable Estimation Algorithm* is less time-consuming for searching reachable vertices from terminal vertex in a pattern, when the number of edges in a graph becomes smaller.

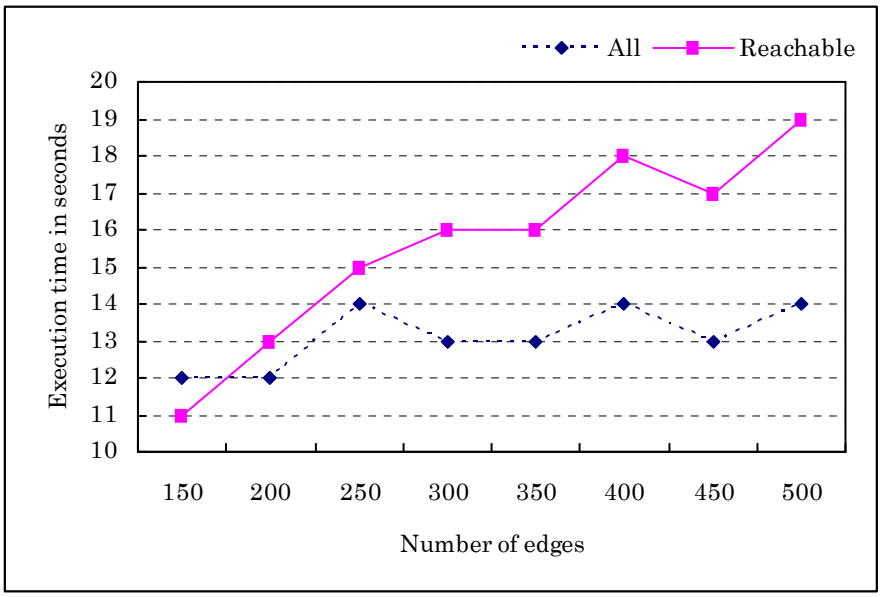

Fig. 4.11 Execution times w.r.t the number of edges
(V=100 · T=10K · S=5 · M=5)

Experiment 3: Execution times for different minimum weighted supports

This experiment compares the execution times of the two algorithms for varying minimum weighted supports. For this experiment, the dataset have V=100 · E=300 · T=10K · M=10, and the minimum weighted supports varies from 1 to 10. Table 4.5 and Fig. 4.12 show the performance of two estimation algorithms for mining weighted frequent patterns. As shown in Fig 4.12, we observe that the difference of execution times between two estimation algorithms becomes smaller when the specified minimum weighted support becomes larger. This is because the number of target traversals in traversal database becomes relatively smaller, when the specified minimum weighted support used for finding weighted frequent

patterns becomes larger.

Table 4.5 Execution times for dataset at different minimum weighted supports

Algorithms	Runtime (in seconds) at different minimum weighted supports									
	1	2	3	4	5	6	7	8	9	10
All	163	91	65	51	41	35	31	27	24	21
Reachable	1,451	1,270	1,137	1,061	988	896	848	831	775	744

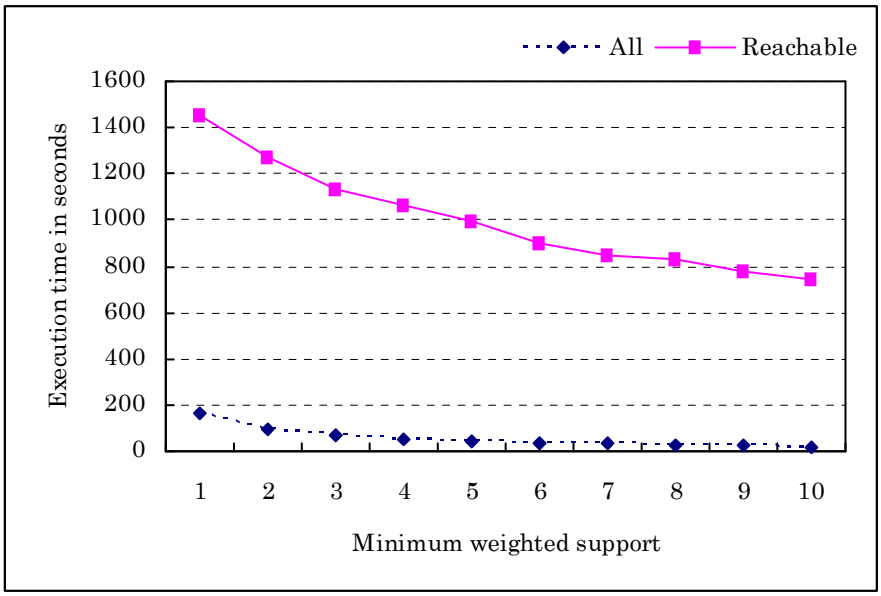

Fig. 4.12 Execution times w.r.t minimum weighted supports (V=100 · E=300 · T=10K · M=10)

Experiment 4: Execution times for different maximum lengths of traversals

The experiment compares the execution times of two algorithms for

different maximum lengths of traversals. The maximum length of traversals is an important parameter used for detecting vertices to have possibility to be patterns later. It becomes longer, the consuming time to search reachable vertices in especially *Reachable* becomes much more than that of *All*. For this experiment, the dataset have V=100 · E=300 · S=5 · T=10K, and the maximum length of traversals varies from 4 to 10.

From Table 4.6 and Fig. 4.13, when the maximum length of traversals becomes shorter, i.e., 4, *Reachable* is more efficient than *All*. On the other hand, when the maximum length of traversals becomes longer, *Reachable* is less efficient. This is because *Reachable* spends more time to find reachable vertices as the maximum length of traversals increases.

Table 4.6 Execution times for dataset at different maximum lengths of traversals

	Runtime (in seconds) at different maximum lengths of traversals						
Algorithms	4	5	6	7	8	9	10
All	11	12	15	17	19	23	26
Reachable	10	12	15	19	22	29	32

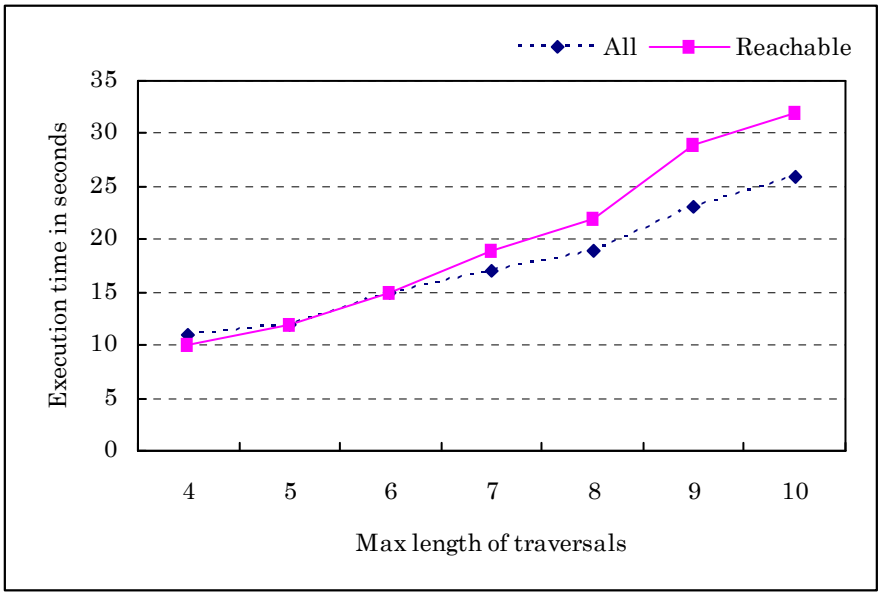

Fig. 4.13 Execution times w.r.t maximum length of traversals (V=100 · E=300 · S=5 · T=10K)

Experiment 5: The number of feasible patterns for different numbers of traversals

The experiment illustrates the impact of *Reachable Estimation Algorithm* on mining weighted frequent patterns for varying number of traversals. This experiment uses different number of traversals to compare the number of feasible patterns of two estimation algorithms. The dataset have V=1,000 · E=2,000 · S=5 · M=10, and the number of traversals varies from 10,000 to 50,000. As mentioned before, *All Estimation Algorithm* leads to the over-estimated weight bounds for the feasible patterns, due to the non-consideration of the topology of a weighted base graph. Therefore, *Reachable* is usually more good estimation algorithm than *All* for the number of feasible patterns.

Table 4.7 The number of feasible patterns for dataset at different numbers of traversals

	Different changed traversal size				
Algorithms	10,000	20,000	30,000	40,000	50,000
All	1,273	1,305	2,001	2,582	3,307
Reachable	1,170	1,199	1,909	2,523	3,200

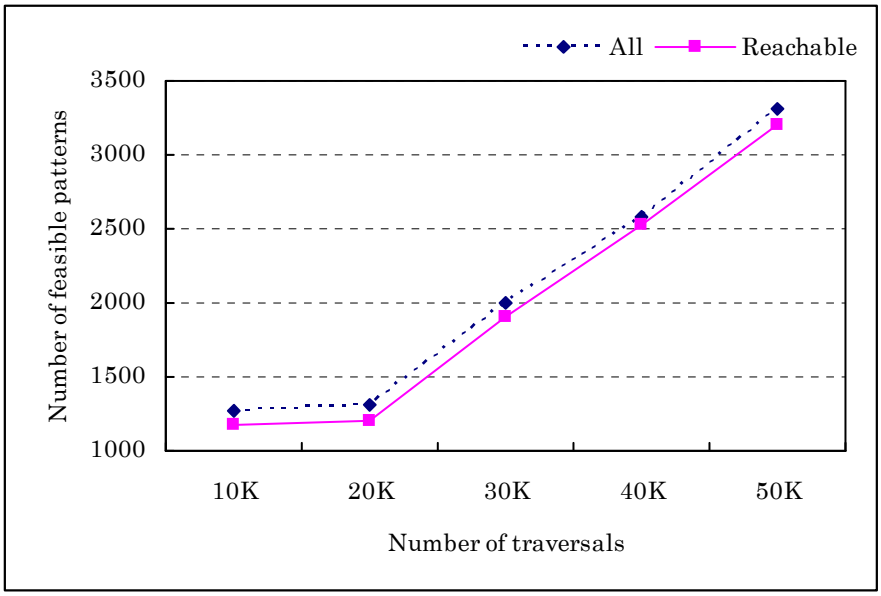

Fig. 4.14 The number of feasible patterns w.r.t the number of traversals (V=1,000 · E=2,000 · S=5 · M=10)

Experiment 6: The number of feasible patterns for different numbers of edges

In Experiment 2, we discussed the effect of graph density. This experiment tests the trend for the number of feasible patterns with two estimation algorithms when the graph density varies. For this experiment,

the dataset have 10,000 traversals, a weighted minimum support as 5, and the maximum length of traversals as 10, and the graph density varies as the total number of edges from 150 to 500 with the fixed number of vertices as 100.

Table 4.8 and Fig. 4.15 test the number of feasible patterns with respect to different graph densities. In the figure, we observe that graph density becomes larger, the number of feasible patterns between two estimation algorithms is analogously. It means that when estimating weight bounds, the number of vertices included for estimation is similar, because the radius of weighted base graph becomes shorter.

Table 4.8 The number of feasible patterns for dataset at different numbers of edges

	Different changed numbers of edges							
Algorithms	150	200	250	300	350	400	450	500
All	325	285	292	261	248	250	216	219
Reachable	270	262	272	252	238	241	212	219

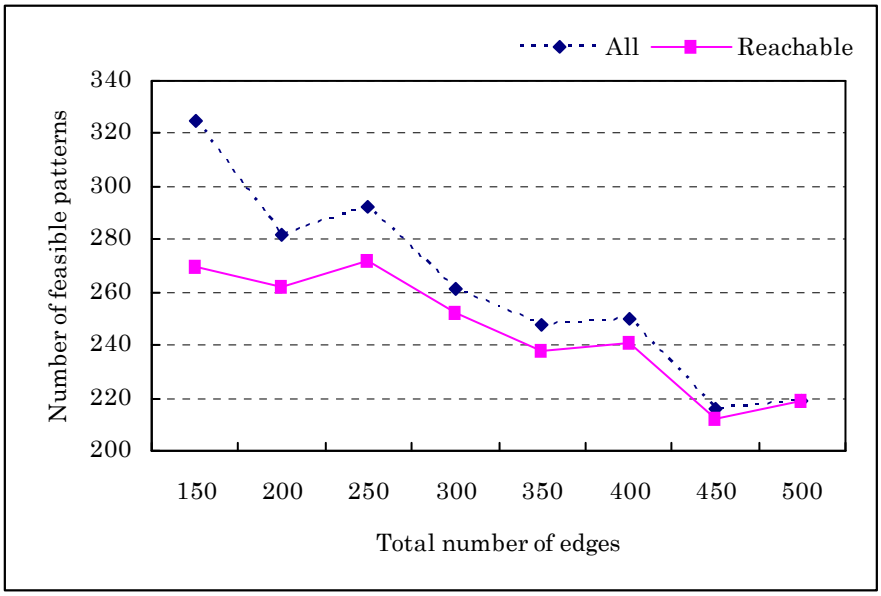

Fig. 4.15 The number of feasible patterns w.r.t the number of edges (V=100 · T=10K · S=5 · M=10)

Experiment 7: The number of feasible patterns for different minimum weighted supports

The experiment compares the number of feasible patterns with two estimation algorithms for different minimum weighted supports. For this experiment, the dataset have V=100 · E=300 · T=10K · M=10, and the minimum weighted support varies from 1 to 10. As previous experiment, *Reachable* generates less number of feasible patterns than that of *All*.

Table 4.9 The number of feasible patterns for dataset at different minimum weighted supports

Algorithms	Different changed minimum weighted supports									
	1	2	3	4	5	6	7	8	9	10
All	699	504	401	340	285	259	236	211	188	170
Reachable	652	460	362	296	262	238	207	173	158	143

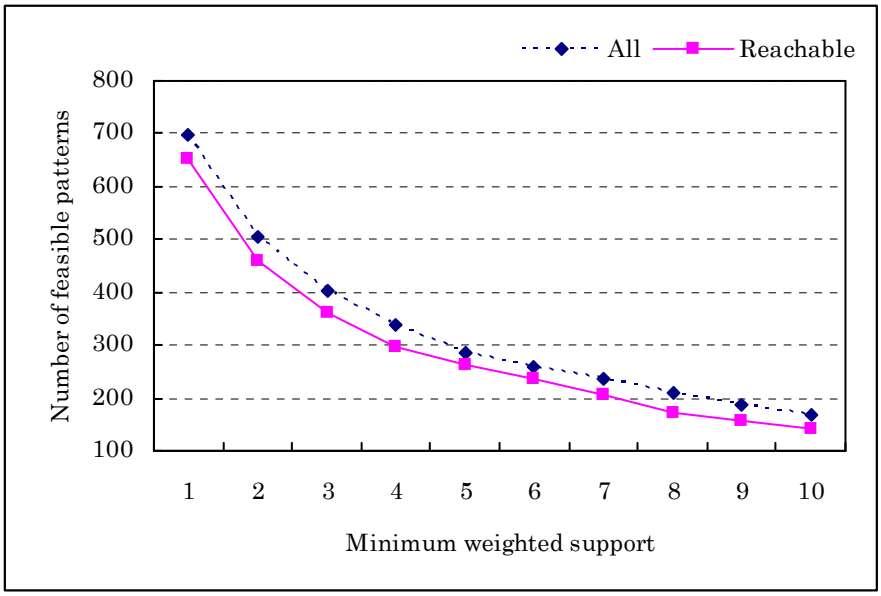

Fig. 4.16 The number of feasible patterns w.r.t minimum weighted supports (V=100 · E=300 · T=10K · M=10)

Experiment 8: The number of feasible patterns for different maximum lengths of traversals

The experiment shows the trend of the number of feasible patterns with respect to the maximum lengths of traversals. For this experiment, the dataset have V=100 · E=300 · S=5 · T=10K, and the maximum length of

traversals varies from 4 to 10. In this experiment, we measured the total number of feasible patterns for all mining stages. As shown in the figure, the number of feasible patterns for *Reachable* is smaller than that of *All*. The difference of the number of feasible patterns between two estimation algorithms becomes larger as the maximum length of traversals increases.

Table 4.10 The number of feasible patterns for dataset at different maximum lengths of traversals

	Different changed maximum length of traversals						
Algorithms	4	5	6	7	8	9	10
All	215	285	391	509	637	781	952
Reachable	176	258	353	463	595	738	869

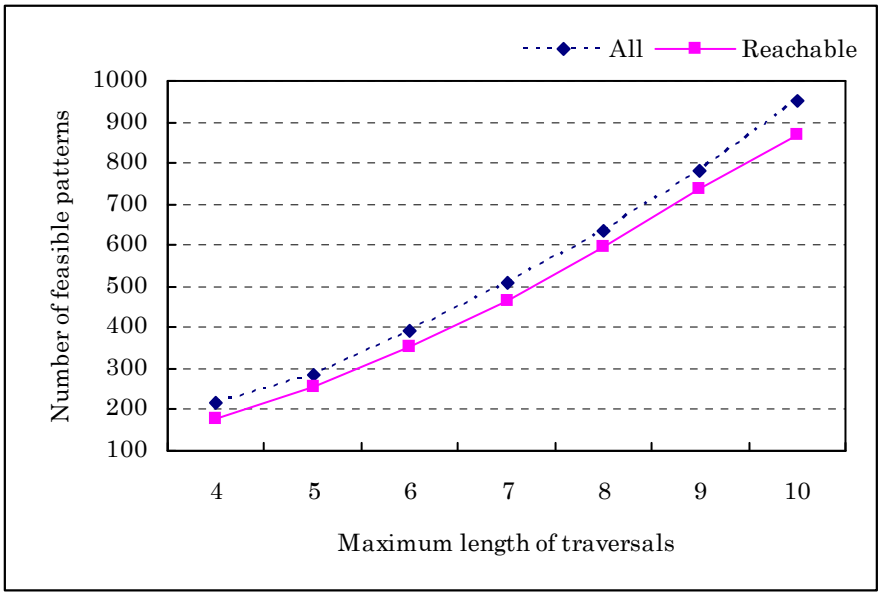

Fig. 4.17 The number of feasible patterns w.r.t maximum length of traversals (V=100 · E=300 · S=5 · T=10K)

Experiment 9: The number of feasible patterns for each mining stage

The experiment compares the number of feasible patterns with two estimation algorithms for each mining stage. For this experiment, the dataset have V=100 · E=300 · S=5 · M=10 · T=10K. From Table 4.11 and Fig. 4.18, we can see that the gap between the number of feasible patterns of *Reachable* and *All* becomes larger when the mining stage is 2, 3 or 4. This is because the feasible patterns are more generated in 2, 3 or 4 stage of mining process.

Table 4.11 The number of feasible patterns for each mining stage

	Mining stage						
Algorithms	1	2	3	4	5	6	7
All	100	172	238	243	136	52	11
Reachable	100	139	197	201	114	46	9

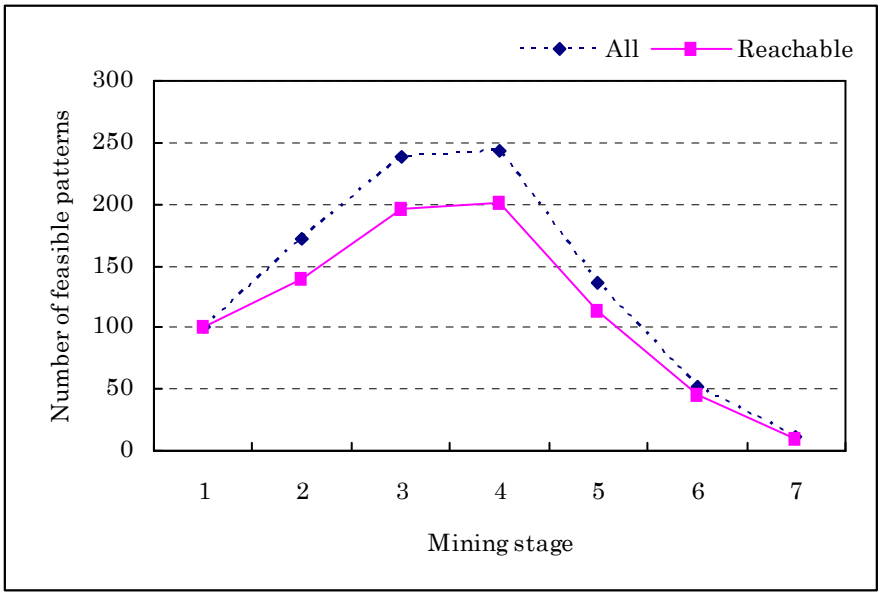

Fig. 4.18 The number of feasible patterns w.r.t each mining stage (V=100 · E=300 · S=5 · M=10 · T=10K)

In above experiments, we investigated the execution time and the number of feasible patterns between the two estimation algorithms, called *All* and *Reachable Estimation Algorithm*. For the execution time, *All* is more or less good algorithm than *Reachable*, but *All* leads to the over-estimated weight bounds for the feasible patterns, due to the non-consideration of the topology of a weighted base graph. For the performance for the number feasible patterns, therefore, *Reachable* is generally more efficient in the mining of weighted frequent patterns.

Chapter 5 Conclusions and Further Works

This thesis examined the mining problems of discovering valuable patterns from the weighted traversals and graph. Differently from previous approaches, the traversals and vertices of a graph are attached with the weights that reflect their importance. Such weights may depend on the problem domains. For example, the weight of a graph vertex may be the size of a Web page, and the weight of a traversal may be the navigation time between Web pages. On these weight setting, we presented two approaches which take the weights into account in the miming process.

First, we presented the mining algorithm for discovering the frequent patterns from the weighted traversals on a unweighted graph. In the algorithm, the traversals whose weights are outside the confidence interval are treated as outliers, and do not contribute to the support count. Through this approach, more reliable frequent patterns can be discovered. Furthermore, we also proposed the enhanced algorithm to improve the performance of this approach. The discovered patterns are further ranked according to their priority which reflects several criteria beside the support.

Second, we extended the mining problem to the discovery of weighted frequent patterns from the unweighted traversals on a weighted graph. This algorithm considers the weighted support instead of the traditional support, which requires the estimation of support bound. We presented two approaches for the estimation of the support bound. Through several experiments, the algorithms were evaluated and analyzed.

In the future, we will further extend the mining problems to the discovery of valuable patterns from other weight settings and criteria. We will also apply the algorithms to the practical applications such as Web mining.

References

[1] M. S. Chen, J. Han, and P. S. Yu, "Data mining: an overview from a database perspective," IEEE Transactions on Knowledge and Data Engineering, Volume 8, No. 6, pp. 866-883, December, 1996.

[2] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," In Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 207-216, May, 1993.

[3] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," In Proceedings of International Conference on Very Large Databases (VLDB), September, 1994.

[4] J. Han and M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufman, 2000.

[5] M. S. Chen, J. S. Park, and P. S. Yu, "Efficient Data Mining for Path Traversal Patterns", IEEE Transactions on Knowledge and Data Engineering, Volume 10, No. 2, pp. 209-221, March, 1998.

[6] A. Nanopoulos and Y. Manolopoulos, "Finding Generalized Path Patterns for Web Log Data Mining", In Proceedings of East-European Conference on Advanced Databases and Information Systems (ADBIS), pp. 215-228, September, 2000.

[7] A. Nanopoulos and Y. Manolopoulos, "Mining Patterns from Graph Traversals", Data and Knowledge Engineering, Volume 37, No. 3, pp. 243-266, June, 2001.

[8] J. L. Gross and J. Yellen, Graph Theory and its Applications, CRC Press LLC, 1998.

[9] R. Diestel, Graph Theory, Springer-Verlag, 1997.

[10] D. B. West, Introduction to Graph Theory, Prentice Hall, 1996.

[11] M. C. Golumbic, Algorithmic Graph Theory and Perfect Graphs, Academic Press, New York, 1980.

[12] J. J. Holdsworth, "Graph Traversal & Graph Transformation," International Journal of Theoretical Computer Science, Elsevier Science Publishers, Volume 321, Issue 2-3, pp. 215-231, August, 2004.

[13] T. Jing, W. L. Zuo, and B. Z. Zhang, "An efficient Web traversal pattern mining algorithm based on suffix array," In Proceedings of 2004 International Conference on Machine Learning and Cybernetics, Volume 3, pp. 1535-1539, August, 2004.

[14] S. S. Hung, T. C. Kuo, and D. S. M. Liu, "PrefixUnion: Mining Traversal Patterns Efficiently in Virtual Environments," International Conference on Computational Science (ICCS 2005), Lecture Notes in Computer Science (LNCS), Springer-Verlag, Volume 3516, pp. 830-833, May, 2005.

[15] Y. S. Lee, M. C. Hsieh, and S. J. Yen, "Efficient Approach for Interactively Mining Web Traversal Patterns," International Conference on Computational Science and Its Application (ICCSA 2005), Lecture Notes in Computer Science (LNCS), Springer-Verlag, Volume 3481, pp. 1055-1065, May, 2005.

[16] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus, "Knowledge Discovery in Databases: An Overview," International Journal of Knowledge Discovery in Databases, AI Magazine, Volume 13, Number 3, pp. 57-70, 1992.

[17] C. Matheus, P. Chan, and G. Piatetsky-Shapiro, "Systems for

knowledge discovery in databases," IEEE Transactions on Knowledge and Data Engineering, Volume 5, pp. 903-913, 1993.

[18] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smith, "From Data Mining to Knowledge Discovery: An Overview," Advanced in Knowledge Discovery and Data Mining, AAAI/MIT Press, pp. 1-34, 1996.

[19] M. A. W. Houtsma and A. N Swami, "Set-Oriented Mining for Association Rules in Relational Databases," In Proceedings of the 11th International Conference on Data Engineering, pp. 25-33, March, 1995.

[20] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," In Proceedings of the 21st International Conference on Very Large Data Bases, pp. 432-444, September, 1995.

[21] R. Srikant and R. Agrawal, "Mining Generalized Association Rules," In Proceedings of the 21st International Conference on Very Large Data Bases, pp. 407-419, September, 1995.

[22] M. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering Frequent Episodes in Sequences," In Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD'95), pp. 210-215, August, 1995.

[23] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," In Proceedings of International Conference on Extending Database Technology, Volume 1057, Springer-Verlag, pp. 3-17, 1996.

[24] E. H. Han, V. Kumar, S. Shekhar, M. Ganesh, and J. Srivastava,

"Search Framework for Mining Classification Decision Trees," Technical Report TR-96-023, Department of Computer Science, University of Minnesota, 1996.

[25] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: A Fast Scalable Classifier for Data Mining," In Proceedings of the 5th International Conference on Extending Database Technology, pp. 18-32, March, 1996.

[26] P. Cheeseman and J. Stutz, "Baysian Classification (Autoclass): Theory and Results," Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, pp. 153-180, 1996.

[27] J. Han, Y. Cai, and N. Cercone, "Data-Driven Discovery of Quantitative Rules in Relational Databases," IEEE Transactions on Knowledge and Data Mining, AAAI/MIT Press, pp. 1-34, 1996.

[28] L. Kaufman and P. J. Rousseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, John Wiley & Sons, 1990.

[29] D. Fisher, "Optimization and Simplification of Hierarchical Clusterings," In Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining (KDD'95), pp. 118-123, August, 1995.

[30] H. Toivonen, "Sampling Large Databases for Association Rules," In Proceedings of the 22nd International Conference on Very Large Data Bases, pp. 134-145, September, 1996.

[31] J. S. Park, M-S. Chen, and P. S. Yu, "An Effective Hash-based Algorithm for Mining Association Rules," In Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 175-186, May, 1995.

[32] J. Han and Y. Fu. "Discovery of multiple-level association rules from

large databases," In Proceedings of the 21st International Conference on Very Large Data Bases, pp. 420-431, September, 1995.

[33] R. Agrawal and R. Srikant, "Mining Sequential Patterns," In Proceedings of the 11th International Conference on Data Engineering, pp. 3-14, March, 1995.

[34] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal and M. C. Hsu, "PrefixSpan: Mining Sequential Patterns Efficiently by PrefixProjected Pattern Growth," In Proceedings of 2001 International Conference on Data Engineering (ICDE'01), pp. 215-224, April, 2001.

[35] M. J. Zaki, "SPADE: An Efficient Algorithm for Mining Frequent Sequences," Journal of Machine Learning, Springer Netherlands, Volume 42, No. 1-2, pp. 31-60, January, 2001.

[36] M. Y. Lin and S. Y. Lee, "Fast Discovery of Sequential Patterns through Memory Indexing and Database Partitioning," Journal of Information Science and Engineering, Volume 21, No. 1, pp. 109-128, 2005.

[37] M. Garofalakis, R. Rastogi, and K. Shim, "Spirit: Sequential Pattern Mining with Regular Expression Constraints," In Proceedings of International Conference on the Very Large Data Bases, pp. 223-234, September, 1999.

[38] C. H. Cai, A. W. C. Fu, C. H. Cheng, and W. W. Kwong, "Mining Association Rules with Weighted Items", In Proceedings of International Database Engineering and Applications Symposium (IDEAS), pp. 68-77, July, 1998.

[39] W. Wang, J. Yang, and P. S. Yu, "Efficient Mining of Weighted Association Rules (WAR)", In Proceedings of the 6th ACM SIGKDD

International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD 2000), pp. 270-274, August, 2000.

[40] F. Tao, F. Murtagh, and M. Farid, "Weighted Association Rule Mining using Weighted Support and Significance Framework", In Proc. of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD 2003), pp. 661-666, August, 2003.

[41] U. Yun and J. J. Leggett, "WLPMiner: Weighted Frequent Pattern Mining with Length-Decreasing Support Constraints", In Proceedings of Pacific-Asia International Conference on Knowledge Discovery and Data Mining (PAKDD), pp. 555-567, May, 2005.

[42] U. Yun and J. J. Leggett, "WIP: mining Weighted Interesting Patterns with a strong weight and/or support affinity," In Proceedings of the 6th SIAM International Conference on Data Mining (SDM 2006), pp. 623-627, April, 2006.

[43] H. Yao, H. J. Hamilton, and C. J. Butz, "A Foundational Approach to Mining Itemset Utilities from Databases," In Proc. of the 5th SIAM International Conference on Data Mining (SIAM 2004), pp. 482-486, April, 2004.

[44] H. Yao and H. J. Hamilton, "Mining itemset utilities from transaction databases," International Journal of Data & Knowledge Engineering, Volume 59, Issue 3, pp. 603-626, December, 2006.

[45] M. S. Chen, J. S. Park, and P. S. Yu, "Data Mining for Path Traversal Patterns in a Web Environment," In Proceedings of 16th International Conference on Distributed Computing Systems, pp. 385-392, May, 1996.

[46] B. Mobasher, N. Jain, E. Han, and J. Srivastava. Web Mining: Pattern Discovery from World Wide Web transactions. Technical Report 96-050, Department of Computer Science, University of Minnesota, 1996.

[47] C. I. Ezeife and Y.Lu, "Mining Web Log Sequential Patterns with Position Coded Pre-Order Linked WAP-Tree," International Journal of Data Mining and Knowledge Discovery, Kluwer Academic Publishers, Volume 10, Issue 1, pp. 5-38, June, 2005.

[48] J. Borges and M. Levene, "Mining Association Rules in Hypertext Databases," In Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD'98), pp. 149-153, August, 1998.

[49] S. Brin, R. Motwani, and C. Silverstein, "Beyond Market Baskets: Generalizing Association Rules to Correlations," In Proceedings of ACM SIGMOD International Conference on Management of Data, Volume 26, Issue 2, pp. 265-276, June, 1997.

[50] S. D. Lee and H. C. Park, "Mining Frequent Patterns from Weighted Traversals on Graph using Confidence Interval and Pattern Priority," International Journal of Computer Science and Network Security, Volume 6, No. 5A, pp. 136-141, May, 2006.

[51] S. D. Lee and H. C. Park, "Discovery of Frequent Patterns from Weighted Traversals and Performance Enhancement by Traversal Split," Journal of Korean Institute of Maritime Information & Communication Sciences, Volume 11, No. 5, pp. 940-948, May, 2007.

[52] S. D. Lee and H. C. Park, "Mining Weighted Frequent Patterns from Path Traversals on Weighted Graph," International Journal of

Computer Science and Network Security, Volume 7, No. 4, pp. 140-148, April, 2007.