從資料串流中探勘頻繁情節

顏秀珍　　李御璽　　粘嘉菖
銘傳大學　　銘傳大學　　銘傳大學
資訊工程學系　　資訊工程學系　　資訊工程學系
sjyen@mail.mcu.edu.tw　leeys@mail.mcu.edu.tw　ruaddick@gmail.com

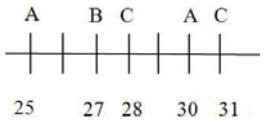
摘要

資料探勘技術在資料分析上面有很大的幫助，在時間序列上找尋頻繁的*情節*(episode)是其中一種。探勘*頻繁情節*能讓使用者根據目前發生的事件，預測未來發生的事件。傳統探勘頻繁情節都是利用階層式的概念，即先產生候選情節(candidate episodes)，再掃描序列資料，以決定其是否為頻繁情節。因而浪費了很多重覆掃描序列資料以及搜尋候選情節的時間。另外，時間序列上的資料會隨著時間不斷的增加，浪費很多重新探勘的時間，資料連續不斷無止境增加的環境稱為資料串流，在很多應用上，像是網路的錯誤偵測、網站搜尋的紀錄、氣象的監測等，使用者往往需要立即知道目前資料分析的結果，若是利用傳統探勘頻繁情節的方法，必須連同原始的序列資料重新探勘，因而無法提供即時的資訊。因此，在這篇論文中，我們提出一個方法，在資料串流的環境下，能夠只針對新增的資料做探勘，以更新既有的頻繁情節，而不需要再掃描原始的序列資料，也不需搜尋候選情節，就可立即找出新增資料後的頻繁情節。

關鍵字：資料探勘、頻繁情節、時間序列、資料串流

1. 前言

資料探勘是要從大量的資料中找尋有用的資訊。探勘*頻繁情節*(frequent episode)[1, 2, 3, 4, 5, 6]是從一連串事件發生的時間序列，找尋常常依序發生的事件，也就是發生某一事件後，哪一事件很有可能會接著發生。因此，當某一事件發生後，我們可以利用頻繁情節來預測將會發生的事件。例如網路錯誤偵測的應用中，我們可以利用頻繁情節來預測錯誤的發生，而在氣象的應用中，可以預測天氣的變化。一連串事件發生的序列，我們稱為*事件序列*(event sequence)，如圖一所示，英文字母代表事件的型態，而數字代表事件發生的時間點，我們可以將事件序列記錄成一序列的事件與其發生時間的組合，如圖二所示。因此，事件序列的表示方式為$\langle(a_1,t_1)(a_2,t_2)\ldots(a_n,t_n)\rangle$ $(n\geq 1)$，其中 $t_1<t_2<\ldots<t_n$ 且 $a_i\in E$ $(1\leq i\leq n)$，E 為所有事件的集合。



(A,25)(B,27)(C,28)(A,30)(C,31)

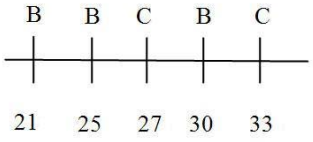
圖一 事件序列　　圖二 事件序列的記錄形式

(a) 有序情節　　(b) 不分序情節

圖三 情節的範例

事件發生的先後關係稱為*情節*(episode)[1, 2, 3, 4, 5, 6]。情節可以分成兩種類型，分別為*有序情節*(serial episode)和*不分序情節*(parallel episode)。有序情節的事件彼此之間是有時間先後關係的，例如圖三(a)，B 要發生在 A 之後，而且要發生在 C 之前。我們以〈ABC〉來表示，因此對於有序情節〈ABC〉和〈CAB〉是完全不相同的情節。而不分序情節的事件彼此之間沒有時間先後順序關係，表示這些事件都有發生即可。例如圖三(b)，也就是事件 ABC 是不分序情節，也就是 A、B 和 C 都有發生，但不考慮

其發生的先後關係，我們以(ABC)來表示，所以對於不分序情節，(ABC)和(CAB)是相同的情節。



圖四 事件序列

一個情節的*發生*(occurrence)[2, 3, 5, 6]為此情節的起始事件與終止事件發生的時間範圍。例如在圖一的事件序列中，情節⟨AC⟩有兩個發生[25,28]和[30,31]。對於某一情節的任兩個發生 $O_1=[t_1,s_1]$和 $O_2=[t_2,s_2]$，若 $t_1 \le t_2$ 且 $s_1 > s_2$ 或 $t_1 < t_2$ 且 $s_1 \ge s_2$，則對於此情節 O_1 包含 O_2。若情節 E 的發生 O_1 包含發生 O_2，表示 E 中的事件在 O_1 的關係比較遠，比較弱，而事件在 O_2 的關係比較近，比較強。因此，O_2 比 O_1 更能代表 E 中事件發生的時間範圍，若情節 E 的某一發生 MO 沒有包含 E 的任何發生，則 MO 稱為 E 的*最小發生*(minimal occurrence)，也是最有代表性的發生。例如圖四中，情節⟨BC⟩的發生[25,27]為最小發生，因為[25,27]沒有包含其他⟨BC⟩的發生。而[21,27]不是最小發生，因為它包含⟨BC⟩的另一個發生[25,27]。而⟨BC⟩共有 2 次最小發生，分別為[25,27]和[30,33]，若一個情節在事件序列上的最小發生次數有達到使用者所定義最小次數門檻值，則此情節稱為*頻繁情節*(frequent episode)[2, 3, 5]。例如在圖四中，若使用者所設定的最小次數門檻值為 2，則情節⟨BC⟩為頻繁情節。因此當事件 B 發生後，我們就可以預測事件 C 將會發生。

由於事件會不斷發生，事件序列的事件會隨著時間不斷增加，頻繁情節也會跟著改變，為了掌握最新的頻繁情節，當事件增加時，必須立即找出資料新增後的頻繁情節。然而，若利用以往探勘頻繁情節的演算法[1, 2, 3, 4, 5]，在事件新增後，必須連同原始的事件序列重新探勘，若資料不斷的新增，會浪費很多重新探勘的時間，造成探勘的效率上極為不佳。因此，我們考慮在資料串流中，如何能夠只針對新增的事件，更新原有的頻繁情節，而不需重新掃描原始事件序列。此篇論文提出一個新的演算法：Suffix frequent episode tree(SFET)演算法，在資料串流的環境下即時找出最新的頻繁情節，我們的方法是採用樹的結構，深度為 1 的節點儲存事件，其它節點則儲存頻繁情節。當事件新增之後，若頻繁情節的最後一個事件和新增的事件相同，才會進行更新的動作。因為只儲存頻繁情節，而且只針對新增的事件做探勘，因此不但加快了探勘的速度，而且在空間的使用上會減少很多。

本論文的組織如下：在第二章中，我們描述並討論與本研究相關的其它論文。第三章中，我們將介紹我們所提的資料串流的頻繁情節探勘演算法。

2. 相關技術

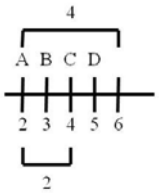
2.1 滑動式視窗探勘頻繁情節演算法

Mannila 等人[1]在 1995 年針對情節方面分成 3 大類，分別是*有序情節*(serial episode)、*不分序情節*(parallel episode)，第三種類型沒有用一個很明確的名稱來命名。此篇論文採用滑動式視窗探勘事件序列，並且在 Apriori 的架構上尋找頻繁情節。使用者必須先決定視窗寬度，以避免頻繁情節的相隔時間太久，接著產生長度為 1 的候選情節，再由滑動式視窗計算包含候選情節的視窗數，經由使用者定義的門檻值來決定是否為頻繁情節，再由長度為 1 的頻繁情節組合長度為 2 的候選情節，經過反覆的組合候選情節和計算，直到候選情節無法組合出來時，表示整個探勘過程已結束。然而，此篇論文並沒有詳細的論述找尋頻繁情節的演算法，在另外一篇有比較多的補充且更詳盡的論述。

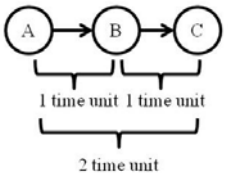
Mannila 等人[2]在 1997 提出了 1995 年那篇論文的擴充，把整個情節再詳細的介紹，並且有詳細的演算法，並且第三種類型命名為*非*

有序和非不分序情節 (non-serial and non-parallel episode)。此篇論文提出一個 WINEPI 演算法，因為有序情節和不分序情節性質不同，因此分別用不同概念去探勘頻繁情節。在有序情節中，因為有時間先後關係，利用了狀態表的方式去確認視窗內是否含有候選情節。候選情節一開始都有正在等待的事件，例如候選情節〈ABC〉，先等待事件為 A ，當視窗移動的時候，若新增的事件為 A 時，表示已經等到 A，此時繼續等待下一個事件為 B，當最後一個的事件 C 已經等到，則表示視窗包含〈ABC〉。另外，一個候選情節可能會有很多個狀態表在等待，例如候選情節〈ABC〉，當有一個狀態表正在等待 C，若新增的事件為 A 時，必須再產生另一個狀態表正在等待 B，因此必須使用大量的空間去儲存每個狀態表。而不分序情節因為沒有時間先後關係，因此利用候選情節的總事件數去計算，例如候選情節為〈AAC〉，當視窗移動的時候，只要達到 A 含有兩次，C 含有一次，即確定視窗包含〈AAC〉。相同的，候選情節可能有很多的表格去紀錄每個事件總數，也必須使用很多的空間去儲存。此篇論文的缺點是使用了大量空間儲存候選情節，再加上在 Apriori 的架構上探勘，因此必須常常掃描事件序列，若事件不斷增加的情形下，會因為不斷掃描事件序列使得效率不佳。另外，在探勘事件序列之前，必須先設定視窗寬度，當要改變視窗寬度時，就必須重新探勘事件序列，使得使用者當改變視窗寬度時，無法立即知道最新的探勘結果。

Casa-Garriga [4]在 2003 年提出了這篇論文，改進 Mannila 等人[2]因為必須先決定視窗寬度所產生的缺點。先前的方法在探勘的過程當中，視窗寬度永遠固定，不因組合出長度較長的候選情節而改變，當要找長度長的候選情節時，會因為使用者對視窗寬度設定的不同，導致長度比視窗寬度還要長的頻繁情節無法找到，例如圖五，若〈ABCD〉為頻繁情節，當視窗寬度為 4 個時間間隔單位，能找到此頻繁情節，但如果視窗寬度為 2 個時間間隔單位，因為視窗寬度無法包含整個頻繁情節，而無法找出此頻繁情節。因此，此篇論文提出一個*無限制情節*(unbounded episode)，每個彼此相鄰事件定義 1 個時間單位，時間單位由使用者決定，例如圖六為長度 3 的有序情節，情節有三個事件 A、B 和 C，A 和 B 為一個時間單位，B 和 C 也為一個時間單位，所以 A 到 C 總共為 2 個時間單位，視窗寬度即為 2 個時間間隔單位，若長度為 k 時，有 k 個事件，視窗寬度就為 k-1 個時間間隔單位。因此，隨著候選情節長度增加，探勘的視窗寬度也跟著增加，這樣不會因視窗寬度的限制，而使得一些可能為頻繁情節給遺漏掉。雖然此篇論文解決了視窗寬度的問題，方法上還是在 Apriori 的架構下，必須使用空間儲存大量的候選情節，而且事件若不斷新增，必須重新掃描事件序列，使得在效率上極為不佳。



圖五 視窗寬度

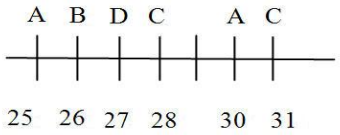


圖六 無限制情節

2.2 利用發生探勘頻繁情節演算法

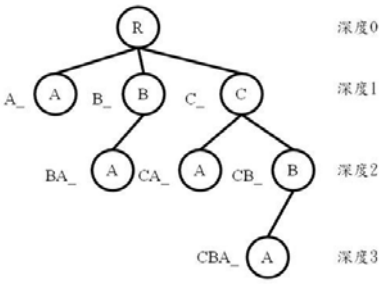
Mannila 等人在 1996 年[3]和 1997 年[2]提出 MINEPI 演算法，是最早利用*最小發生* (minimal occurrence)來探勘頻繁情節。MINEPI 也是在 Apriori 的架構上做探勘，與滑動式視窗不同的地方在於，滑動式視窗先決定視窗寬度，再計算事件序列上有幾個視窗包含候選情節，最後根據包含候選情節視窗數占總視窗數的百分比，再由門檻值去決定此候選情節是否為頻繁。最小發生是直接去計算此候選情節在事件序列上有幾次最小發生，因此每一次最小發生的開始時間和結束時間都會有所不同，當

找出全部的最小發生，若有達到使用者所定的次數門檻值，則此候選情節為頻繁情節。MINEPI 的優點在於不需要反覆掃描事件序列，因為可以利用已找出頻繁情節的結果來組合候選情節，例如一開始找到長度為 1 的頻繁情節為⟨A⟩:[2],[5]和⟨B⟩:[4],[7]，再利用此結果組合成⟨AB⟩為[2,4],[5,7]，並不需要再重新掃描事件序列。缺點為候選情節若在事件序列上找到太多最小發生，則必須使用大量的空間去儲存。另一個缺點是 MINEPI 所找到的最小發生並沒有一個門檻值去限制最大的持續時間，有可能會造成最小發生相隔時間太長，例如⟨AB⟩:[5,105]，表示第 5 秒的 A 發生後，經過了 100 秒之後才發生 B，彼此之間距離因就算太遠，若此發生為最小發生，還是會予以計算。



圖七 事件序列

Xi Ma 等人[5]在 2004 年提出了 Episode Prefix Tree(EPT)和 Position Pairs Set(PPS)演算法，要改善 MINEPI 必須儲存大量的候選情節，此兩個演算法過程中不會產生候選情節，而且是以字首(prefix)相同來探勘事件序列，例如用圖七的事件序列，以字首⟨A⟩來說，在事件序列上找到⟨AB⟩: [25,26]、⟨AD⟩: [25,27]、⟨AC⟩: [25,28], [25,31], [30,31]、⟨AA⟩: [25,30]，其中⟨AC⟩: [25,31]有包含⟨AC⟩: [25,28]，所以⟨AC⟩: [25,31]不為最小發生，再以字首⟨AD⟩為例，會找到⟨ADC⟩: [25,28], [25,31]、⟨ADA⟩: [25,30]，其中⟨ADC⟩: [25,31]有包含⟨ADC⟩: [25,28]，所以⟨ADC⟩: [25,31]不為最小發生。在 EPT 演算法中，以圖八為例，樹上每個節點會記錄事件和情節的最小發生次數，而且情節前綴樹有兩個特性，第一個特性為每一條路徑的最後一個節點的最小發生次數，若沒達到使用者定義的門檻值，表示此節點之後更長的路徑也不可能會繼續成長，因此必須刪除掉，例如門檻值設



圖八 情節前綴樹

為 3 次，若⟨CBA_⟩的次數為 2 次，則此節點必須刪除掉，因為⟨CBA⟩已經不是頻繁情節，不可能出現比⟨CBA⟩長度更長的頻繁情節，這樣可以避免儲存不是頻繁情節而且不會去找長度更長的情節。第二個特性是設定*深度門檻值* (depth threshold)，若深度門檻值為 2，則樹在成長的過程中只會到深度 2，若有更長的情節可以成長也不會產生，通常設定深度門檻值是一序列$[k_1,k_2,\ldots,k_n]$，當第一次成長時，會從根節點開始成長到深度 k_1 的節點，若深度 k_1 的節點有達到次數的門檻值，表示這些節點可以繼續成長，則必須進行第二次成長，這時會從深度 k_1 的節點開始成長到深度的 k_1+k_2 節點，直到深度 k 的節點沒有達到次數門檻值才停止成長，此時情節前綴樹上的節點即為頻繁情節。此方法的優點在於不會產生候選情節，減少空間上的使用，但缺點是當節點可以繼續成長時，就必須要再反覆掃描事件序列。因此，此篇論文提出另一個方法 PPS 來解決反覆掃描的問題，PPS 先找出長度為 1 的情節，例如表一所示，表一為圖七事件序列上所找出的最小發生，一開始先找出⟨A⟩,⟨B⟩,⟨C⟩,⟨D⟩，因為為深度優先，所以必須把最先字首⟨A⟩整個找完，例如會先找⟨AA⟩,⟨AB⟩,⟨AC⟩,⟨AD⟩，再來找⟨ABC⟩,⟨ABD⟩…等，當組合不出來長度更長的情節時，如⟨ABC⟩，即可以找字首⟨B⟩，以此類推。因為在組合的過程中，若情節沒有達到頻繁的條件，則不會記錄，會比 MINEPI 更省空

間，而且此方法只需要一開始掃描事件序列，找出長度為 1 的頻繁情節，接下來只要針對已找出結果繼續組合，而不用再重新掃描事件序列，會比 EPT 更有效率。但由於必須先掃描一次事件序列，若事件序列不斷新增事件，也是會不斷的重新掃描事件序列，無法很有效率的探勘頻繁情節。另外，此篇論文有針對時間間隔太長的頻繁情節的缺點做改進，作者提 3 種限制，第一種為限制最小發生的事件之間最大相隔多少個事件，例如頻繁情節⟨AC⟩的最小發生，假設門檻值為 2，若事件序列上發現 A 到 C 之間有包含到 3 個事件以上，則此最小發生因為相隔太多事件，不認為是有效的最小發生。第二種與第一種差別在於限制最小發生的事件之間最小相隔多少個事件，此限制是用在相鄰事件並不希望太過於接近，至少要隔幾個事件才認為此最小發生是有效的。第三種為限制最小發生從開始到結束歷經的最大時間，例如假設門檻值為 5，則[5,8]是有效的，而[6,12]是無效的，這幾個限制讓一些最小發生相隔太久或是太過接近時，能有效避免掉。

表一 Prefix and Position-pair sets

Prefix	Position-pair sets	Prefix	Position-pair sets
⟨A⟩	(25),(30)	⟨D⟩	(27)
⟨B⟩	(26)	⟨AC⟩	(25,28),(30,31)
⟨C⟩	(28),(31)		

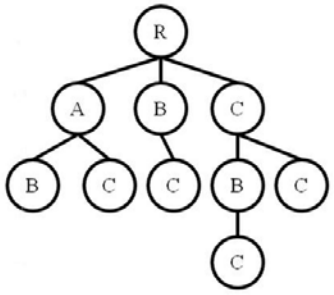
Mielikäinen 在 2004 年提出了在串流中找尋頻繁情節，此篇論文是首先提出在事件序列上事件不斷新增時探勘，以往的方法都必須反覆掃描事件序列，若事件序列不斷新增事件，會因為方法不適合，使得在探勘的效率上極為不佳。此篇論文所找的是*發生*(occurrence)，意即情節的發生若包含到另一個發生，都會予以計算，對某些應用上，像是網站的查詢，類似關鍵字有可能會重覆查詢，再換另一個關鍵字，例如一開始使用”資料探勘”關鍵字做查詢，若結果不理想，可能換成”資料探勘的應用”，這兩個關鍵字是很類似的，如果結果還是不理想，可能就會換成”資料採礦”或”資料挖掘”的關鍵字，所以”資料探勘”和”資料探勘的應用”對”資料採礦”是同等重要，因此，此篇論文並不討論此發生是否為最小發生。作者提出一個方法，當新增事件時能動態的更新目前所有情節的個數，但此方法是將所有可能不管任何長度的情節組合並儲存起來，例如已知情節有⟨A⟩,⟨C⟩,⟨AC⟩，當新增事件 B 時，會對目前已有的情節組合，例如會組成⟨B⟩,⟨AB⟩,⟨CB⟩,⟨ACB⟩，若事件不斷的新增，會儲存大量的情節，因此會造成空間上的浪費，而且每個情節都要去組合，在效率上也極為不佳。

3. 資料串流的頻繁情節探勘演算法

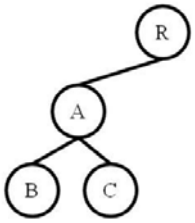
從文獻探討中可以發現，滑動式視窗探勘演算法和利用發生探勘演算法，在做法上大多是用 Apriori 演算法，找出頻繁情節。當事件新增時，則必須重新探勘事件序列，另外像是利用情節前綴樹演算法也是遇到必須反覆探勘事件序列，在效率上極為不佳，而唯一在資料串流上找尋頻繁情節，因為儲存大量並非是頻繁情節的資訊，造成空間上的浪費。因此本論文提出兩個新的演算法，來改進上述缺點。在做法上，我們會建立一個類似頻繁情節樹的結構，避免儲存大量不必要的資訊，而且會只針對新增的事件，去更新事件序列上相關的頻繁情節，避免整個重新探勘。

3.2 Suffix frequent episode tree(SFET)演算法

在圖九中，表示整個的樹狀結構，以根節點當為深度 0，深度 1 的節點紀錄事件，其餘的節點必須為頻繁情節，並且每個節點上會記錄此頻繁情節的所有最小發生的開始和結束時間，以及最小發生的次數。而每個節點代表字尾相同的情節，例如圖十，節點 B 和 C 在節點 A 之下，則頻繁情節分別是⟨BA⟩和⟨CA⟩。



圖九 Suffix frequent episode tree



圖十 頻繁情節⟨BA⟩和⟨CA⟩

演算法的流程如下：

當事件 e 新增時

1. 情節 e 節點若已存在，更新情節 e 節點的次數，若不存在，則建立情節 e 的節點。
2. 情節 e 節點若無達到次數門檻值，則結束此次事件的新增。

   情節 e 節點若剛達到次數門檻值，則與其他深度為 1 的頻繁節點作組合。
   - 組合後若次數有達到次數門檻值，則建立情節 e 節點的子節點，例如節點 A 剛為頻繁，與頻繁節點 B 作組合，若有達到門檻值，則表示⟨BA⟩為頻繁情節，所以節點 A 必須建立一個子節點 B。
   - 組合後若次數沒有達到門檻值，則不做任何建立子節點的動作。

   情節 e 節點若已達到次數門檻值，則與其他深度為 1 的頻繁節點作組合，在組合之前必須先找出組合後節點的最後一次最小發生的開始時間和結束時間。例如 A 的最小發生為[3],[6],[7]，B 的最小發生為[2],[4],[8]，而⟨BA⟩已經存在的最小發生為[2,3],[4,6]，若新增的事件 A 為第 9 秒，因為⟨BA⟩最後一次的最小發生開始時間為 4，表示組合時 B 從[8]開始組合，而不需再從頭。同樣的，因為⟨BA⟩最後一次的最小發生結束時間為 6，表示組合時 A 從[7]開始組合，這樣一來可以避免重頭開始組合。
3. 瀏覽情節 e 節點的所有子節點，找到從子節點到節點 e 的路徑(不包含 e)，並且根據路徑找出相同深度的節點作組合。例如⟨BA⟩，其路徑為 B，則找出所有在 B 下的子節點作組合，即⟨_B⟩和⟨BA⟩作組合，如⟨CB⟩和⟨BA⟩作組合。另外像⟨CBA⟩其路徑為 CB，則找出所有在 CB 下的子節點，即⟨_CB⟩和⟨CBA⟩做組合，如⟨ACB⟩和⟨CBA⟩或者是⟨BCB⟩和⟨CBA⟩。
4. 若是子節點剛達到次數門檻值，即可直接建立組合後的節點。

   若是已達到門檻值，必須找出組合後的節點，找出最後一次最小發生的開始時間和結束時間來避免重新組合。

   若是未達到門檻值，則不建立組合後的節點。
5. 當所有節點 e 的子節點都瀏覽過，則結束此次事件的新增。

由上面流程中可以發現，我們的演算法不會儲存非必要的資訊，例如非頻繁情節，減少空間上的使用，而且只針對新增的事件做更新頻繁情節，使得在效率上有所提升。但因為找到路徑之後，必須找出另一個頻繁情節作組合，當要往下一層尋找節點時，必須花費時間

搜尋此節點，一旦路徑太長而且子節點太多時，搜尋時間會過久。

太多，就必須每一個都要確認是否可以組合，會在確認上花點時間。

4. 結論

對於滑動式視窗和利用發生探勘事件序列，大部分演算法基於在關聯性規則架構上探勘，因此在過程中會產生大量的候選情節，造成使用大量的空間。另外，有些方法上如 EPT 和 PPS，雖然並沒有產生候選情節，但是在探勘的過程中必須反覆掃描事件序列，對於資料串流上，事件不斷的新增，造成這些方法必須重新探勘事件序列，無法利用之前所探勘的結果，使得在效率上即為不佳。為了解決這些問題，我們提出了 SFET 演算法，在探勘的過程中樹上節點只儲存頻繁情節，並不會記錄候選情節，在空間的使用上會較少。此外，我們的方法只會針對新增的資料來做探勘，並不會將整棵樹都瀏覽過，因此在效率上更為快速。我們的方法會與另一篇在資料串流上探勘和 PPS 來比較。我們所提出的方法都各互有缺點，我們也會比較這兩個方法在效率上哪個較佳。

我們的方法上目前只能針對有序情節做探勘，因此未來我們也希望想出一個方法能夠探勘不分序情節。另外，利用最小發生去找出頻繁情節都是由使用者定義一個門檻值次數，決定是否情節為頻繁，但設定次數並沒有一個很清楚界限，使用者很難決定該設為多少。因此我們希望提出一個用百分比的方式，去決定情節是否為頻繁，例如，若事件序列長度為 4，長度為 2 的有序情節在此事件序列上最多就只有 3 次，知道最多次數就可以用百分比來表示，使用百分比也讓使用者容易去決定門檻值該設為多少。最後，在許多應用上在同一個時間有可能會發生很多事件，例如氣象，有可能因為溫度和濕度等因素同時進行而造成下雨，因此未來我們也希望能夠在此類型的資料做探勘。

參考文獻

1. H. Mannila, H. Toivonen and A. I. Verkamo, Discovering frequent episodes in sequences, In Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95), pages 210-215, 1995.
2. H. Mannila, H. Toivonen and A. I. Verkamo, Discovering frequent episodes in sequences, Data Mining and Knowledge Discovery(DMKD) 1(3): 259-289, November 1997.
3. H. Mannila and H. Toivonen, Discovering generalized episodes using minimal occurrences, In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96), pages 146-151, 1996.
4. G. Casas-Garriga, Discovering unbounded episodes in sequential data, Knowledge Discovery in Databases:PKDD 2003, volume 2838 of Lecture Notes in Artificial Intelligence, pages 83–94. Springer-Verlag, 2003.
5. X. Ma, H. PANG, K. L. TAN, Finding constrained frequent episodes using minimal occurrences, Proceedings of the Fourth IEEE International Conference on Data Mining, 2004, pages 471-474.
6. T. Mielikäinen, Discovery of serial episodes from streams of events, Proceedings of the 16th International Conference on Scientific and Statistical Database Management (SSBDM), pages 447-448.