Discovering During-Temporal Patterns (DTPs) in Large Temporal Databases *

Li Zhang[a], Guoqing Chen[a,†], Tom Brijs[b], Xing Zhang[a]

[a] *School of Economic and Management, Tsinghua University, Beijing, 100084, P.R.China*

[b] *Transportation Research Institute, Hasselt University, Diepenbeek, B3920, Belgium*

Abstract Large temporal Databases (TDBs) usually contain a wealth of data about temporal events. Aimed at discovering temporal patterns with *during* relationship (*during*-temporal patterns, DTPs), which is deemed common and potentially valuable in real-world applications, this paper presents an approach to finding such DTPs by investigating some of their properties and incorporating them as desirable pruning strategies into the corresponding algorithm, so as to optimize the mining process. Results from synthetic reveal that the algorithm is efficient and linearly scalable with regard to the number of temporal events. Finally, we apply the algorithm into the weather forecast field and obtain effective results.

Keywords: data mining; *during* relationship; temporal pattern

---

1 Introduction

In recent years, discovery of association rules [14] and sequential patterns [13] has been a major research issue in the area of data mining. While typical association rules usually reflect related events occurring at the same time, sequential patterns represent commonly occurring sequences that are in a time order. However, real-world businesses often generate a massive volume of data in daily operations and decision-making processes, which are of a richer temporal nature. For instance, a customer could buy a DVD machine after TV was bought; the duration of an ERP project partially overlapped the duration of a BPR project; and a patient suffered from cough during the period of fever. Apparently,

*The work was partly supported by the National Natural Science Foundation of China (70231010/70321001), Tsinghua University's Research Center for Contemporary Management, and the Bilateral Scientific and Technological Cooperation between China and the Flanders.

†Corresponding author.E-Mail: *chengq@em.tsinghua. edu.cn*

such temporal relationships (e.g., *after*, *overlap*, *during*, etc.) are kinds of real-world semantics that are, in many cases, considered meaningful and useful in practice. Usually, temporal relationships between events with different time stamps could be categorized into several types in forms of temporal comparison predicates such as *after*, *meet*, *overlap*, *during*, *start*, *finish*, and *equal* [10]. Though recent years have witnessed several efforts on discovering the after relationship [7,8,11,13], more in-depth investigations of the relationship are still badly needed, let along their explorations of other types of temporal relationships. Furthermore, results from the studies on the after relationship could hardly be simply extended to the case of some other relationships such as *during*, *overlap*, etc. This may be attributed to the fact that in the after relationship, events can generally be dealt with on a time point, whereas in other relationships, events are considered to be of a time interval nature.

On the other hand, both Rainsford [3] and Hoppner [6] have recently discussed the issues of finding temporal relationships between time-interval-based events using temporal comparison predicates [10], but with different mining approaches. Rainsford introduced temporal semantics into association rules, in forms of $X \Rightarrow Y \wedge P_1 \wedge P_2 \wedge ... \wedge P_n$ (n≥0), where X and Y are itemsets, and $X \cap Y = \emptyset$. $P_1 \wedge P_2 \wedge ... \wedge P_n$ is a conjunction of binary temporal predicates. While mining a database D_T, a rule is accepted when its confidence factor $0 \leq c \leq 1$ is equal to or larger than the given threshold. Similarly, each predicate P_i is measured with a temporal confidence factor $0 \leq tc_{P_i} \leq 1$. The algorithm firstly generates the traditional association rules without considering the temporal factors, and then finds all of the possible pairings of temporal items in each rule. Subsequently, these pairings are tested so that strong temporal relationships could be found. Obviously, the complexity of this sequentially executed algorithm rises rapidly as the number of typical rules grows. Differently, Hoppner proposed another technique for discovering temporal patterns in state sequences. He defined the supporting level of a pattern as the total time in which the pattern can be observed within a sliding window, which should be predetermined by the user. However, a major concern for this technique is how to decide a proper size for the sliding window, since the sliding window can affect the mining results. Furthermore, The changes of the sliding window will lead to a sub-patterns check. The check requires some backtracking mechanism, which is computationally expensive. Like many existing data mining algorithms, the algorithm needs to scan the database repeatedly, which would significantly lower its efficiency.

This paper will focus on a particular type of temporal relationships, namely *during*, which represents that one event starts and ends within the duration of another event. Notably, this *during* relationship could reflect the temporal semantics of during, start, finish and equal described in [10]. An approach will be proposed to discover the so-called *during*-temporal patterns (DTPs) in larger temporal databases, which are considered common and potentially valuable in real-world applications. One idea behind the approach is to design the corresponding algorithm so as to reduce the workload in scanning the database. In doing so, the database is partitioned into some disjoined datasets with two operations when calculating the support level of each pattern, so that scanning the whole database could be avoided. Furthermore, some properties of DTPs are investigated and then incorporated into the algorithm as pruning strategies to optimize the mining process for efficiency purposes.

The remainder of this paper is organized as follows. Section 2 formulates the problem and introduces related notions. In Section 3, the algorithmic details are provided, along with some of the related properties. The experiments on synthetic data and real weather data are discussed in Section 4, and Section 5 concludes the paper.

2 The problem formulation

Let $\mathcal{A}$=$\{a_1,a_2,...,a_m\}$ be a set of states, and $\mathcal{D_T}$ a temporal database as shown in Table 1. Given a database $\mathcal{D_T}$ with N records, each of which is in the form of $\{a,(st,et)\}$ with respect to event e, i.e., e=(a,t), where a is the state involved in the event, t=(st,et) is the time interval which indicates starting time (st) and ending time (et) of state a in the event. A specific event is denoted as e_l=(a_i,t_l) ($1 \le l \le N$ and $1 \le i \le m$) and $t_l = (st_l, et_l)$, i.e., $S(e_l)$=st_l and $E(e_l)$=et_l. For example, with a_1=rain, e_1=(a_1,(1,20)) in Table 1 means that it began to rain at 1:00h and ended at 20:00h.

Table 1: A Temporal Database

Event	**State**	**Starting Time**	**Ending Time**
e_1	a_1	1	20
e_2	a_3	1	4
e_3	a_4	5	7
e_4	a_1	22	28
e_5	a_2	2	8
e_6	a_3	10	13
e_7	a_5	25	35
e_8	a_3	23	28
e_9	a_4	25	27
e_{10}	a_6	25	26
e_{11}	a_1	30	40
e_{12}	a_3	30	38
e_{13}	a_4	34	38
e_{14}	a_6	37	37

Definition 1 Let e_l=(a_i,t_l) and e_k=(a_j,t_k) be two events in $\mathcal{D_T}$. We call e_l *during* e_k(or e_k *contains* e_l), denoted as $e_l <^d e_k$, if

$$S(e_l) \geq S(e_k) \text{ and } E(e_l) \leq E(e_k)$$

Generally, given a set of events $\{e_1, e_2, ..., e_k\}$, if e_{i+1} *during* e_i is satisfied for all i=1,2,...,k-1, we have $e_k <^d e_{k-1} <^d ... <^d e_2 <^d e_1$.

For any two states a_i and a_j, a_i is called to be *during* a_j, denoted as pattern $a_i \Rightarrow^d a_j$, if state a_i occurs during the period of another state a_j, which is a *during*-temporal pattern (DTP) with length 1. Generally, a DTP of length (k-1) ($k \geq 1$), namely DTP_{k-1}, is of the form:

$$a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1$$

When the length is 0 (i.e., DTP_0), the pattern is a single state actually. More generally, given two patterns α and β, the form $\alpha \Rightarrow^d \beta$ is also a DTP ($A_\alpha \cap A_\beta = \emptyset$, where A_α and A_β are the sets of states included in pattern α and β respectively). As a special case, it retrogresses to $a_i \Rightarrow^d a_j$ when the lengths of both patterns are 0.

Given a pattern $a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1$, $e_k <^d e_{k-1} <^d ... <^d e_2 <^d e_1$ supports this pattern if a_i is the state of e_i for all i=1,2,...,k. In the case, $e_k <^d e_{k-1} <^d ... <^d e_2 <^d e_1$ can be considered as an instance of this pattern. For example, in Table 1, $e_{10} <^d e_9 <^d e_8$ is an instance of the pattern $a_6 \Rightarrow^d a_4 \Rightarrow^d a_3$, and $e_{14} <^d e_{13} <^d e_{12}$ is another instance of this pattern. Further, a DTP α

$$a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_{j+1} \Rightarrow^d a_j \Rightarrow^d a_{j-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1$$

is characterized by the number of its instances: $e_k <^d e_{k-1} <^d ... <^d e_2 <^d e_1$, and a DTP $\beta \Rightarrow^d \gamma$, i.e.,

$$(a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_{j+1}) \Rightarrow^d (a_j \Rightarrow^d a_{j-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1)(1 \leq j \leq k-1)$$

is characterized by the number of its instances:

$$(e_k <^d e_{k-1} <^d ... <^d e_{j+1}) <^d (e_j <^d e_{j-1} <^d ... <^d e_2 <^d e_1)$$

which is $e_k <^d e_{k-1} <^d ... <^d e_{j+1} <^d e_j <^d e_{j-1} <^d ... <^d e_2 <^d e_1$ according to the associative law of events. Hence, $\beta \Rightarrow^d \gamma$ equivalently reads

$$a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_{j+1} \Rightarrow^d a_j \Rightarrow^d a_{j-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1 \ (1 \leq j \leq k-1)$$

Furthermore, for finding all instances of a DTP α, one may consider to scan the whole database. In fact, however, only a small part of the database, with respect to the set of states included in α, is useful. Thus, we can divide the database into m datasets (m is the number of the states in the database), each of which is the set of time intervals of a single state. Thus, when finding all instances of $a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_{j+1} \Rightarrow^d a_j \Rightarrow^d a_{j-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1$, only those datasets which include the sets of time intervals of $a_k, a_{k-1}, ..., a_2$, and a_1 are scanned. For this purpose, we define a time interval set $g(\alpha)$ and a state set $h(\alpha)$ to partition the database, and join the small datasets to count the support (which will be discussed in this and next sections).

Definition 2 Let $\mathcal{A}$ and $\mathcal{T}$ be finite sets of states and time intervals respectively with respect to a temporal database $\mathcal{D}_\mathcal{T}$. For pattern α, i.e., $a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1$ ($k \geq 1$), we define the set of time intervals $g(\alpha)$ as the set of finest time intervals of all the instances of α. Formally, $g(\alpha)$ is the function mapping the set of patterns to the set of time intervals,

$$g(\alpha) = \{t|(a_i, t) \in \mathcal{D}_{\mathcal{T}}\}, \text{ if the length of } \alpha \text{ is 0, i.e., } \alpha \text{ is a single state } a_i.$$

$$g(\alpha) = \{t_k \in g(a_k)|\text{for all } i = 1, 2, ..., k-1, a_i \in A_\alpha, \exists t_i \in g(a_i), \text{such that } t_{i+1} \cap t_i = t_{i+1}\}, \quad (2\text{-}1)$$

if the length of α is larger than 0. In the definition, $t_l \cap t_k = (max\{st_l, st_k\}, min\{et_l, et_k\})$. Equivalently, we have

$$g(\alpha) = \{t_k|\text{for each instance of } \alpha : e_k(a_k, t_k) <^d e_{k-1}(a_{k-1}, t_{k-1}) <^d ... <^d e_1(a_1, t_1)\} \quad (2\text{-}2)$$

$g(a_i)$ includes all the intervals in which a_i occurred, so all instances of α can be found using $g(a_1)$, $g(a_2)$,...,$g(a_k)$. And $t_{i+1} \cap t_i = t_i$ means $e_{i+1} <^d e_i$, for i=1,2,...,k-1. That is, $t_{i+1} \cap t_i = t_i$ for i=1,2,...,k-1 means that $e_k(a_k, t_k) <^d e_{k-1}(a_{k-1}, t_{k-1}) <^d ... <^d e_2(a_2, t_2) <^d e_1(a_1, t_1)$. Hence, both (2-1) and (2-2) get the set of finest time intervals of all the instances of α.

Support	$g(a_1)$	$h(a_1)$
1	(1,20)	
2	(22,28)	a_2, a_3, a_4, a_6
3	(30,40)	

(a)

Support	$g(a_2)$	$h(a_2)$
1	(2,8)	a_4

(b)

Support	$g(a_3)$	$h(a_3)$
1	(1,4)	
2	(10,13)	a_4, a_6
3	(23,28)	
4	(30,38)	

(c)

Support	$g(a_4)$	$h(a_4)$
1	(5,7)	
2	(25,27)	a_6
3	(34,38)	

(d)

Support	$g(a_5)$	$h(a_5)$
1	(25,35)	a_4, a_6

(e)

Support	$g(a_6)$	$h(a_6)$
1	(25,26)	$\emptyset$
2	(37,37)	

(f)

Support	$g(a_3 \Rightarrow^d a_1)$	$h(\alpha)$
1	(1,4)	
	(10,13)	a_4, a_6
2	(23,28)	
3	(30,38)	

(g)

Support	$g(a_4 \Rightarrow^d a_1)$	$h(\alpha)$
1	(5,7)	
2	(25,27)	a_3, a_6
3	(34,38)	

(h)

Figure 1: The examples of the sets g and h

For example, given a temporal database $\mathcal{D}_{\mathcal{T}}$ as shown in Table 1. $g(a_1)$={(1,20), (22,28), (30,40)} and $g(a_3)$={(1,4), (10,13), (23,28), (30,38)}. According to $t_{i+1} \cap t_i = t_{i+1}$ in (2-1), we have (1,4)∩(1,20) =(1,4), (10,13)∩(1,20)=(10,13), (23,28)∩(22,28)=(23,28), and (30,38)∩(30,40)=(30,38), so $g(a_3 \Rightarrow^d a_1)$={(1,4), (10,13), (23,28), (30,38)}. According to (2-2), we need to find these intervals from original temporal database. The instances of $a_3 \Rightarrow^d a_1$ include $e_2 <^d e_1$, $e_6 <^d e_1$, $e_8 <^d e_4$ and $e_{12} <^d e_{11}$, which result in time intervals (1,4), (10,13), (23,28), and (30,38) respectively. That is, $g(a_3 \Rightarrow^d$

a_1)={(1,4), (10,13), (23,28), (30,38)}. More examples can be found in Figure 1. Next, we will define the support degree of a DTP pattern (Definition 3).

Definition 3 The support degree of a DTP α is the fraction of the support count of the pattern. That is,

$$support(\alpha) = \frac{|g(\alpha)|}{|g_0|}$$

where $|g(\alpha)|$ is the number of time intervals in $g(\alpha)$ without double-counting those intervals of the instances in that an event contains several events with the same state, and $|g_0|= max\{|g(a_i)|$, for i=1,2,...,$m\}$. Actually, $|g(\alpha)|$ is the number of instances supporting α without double-counting. α is said to be frequent if the support degree is not less than the given threshold (i.e., minsupport).

The support degree of pattern α in Definition 3 is the ratio of the number of time intervals included in all instances of α (without double-counting) over the maximum number of time intervals among $|g(a_i)|$ for all i. In other words, $support(\alpha)$ reflects the relative frequency of time intervals for α with respect to the number of time intervals for a most frequent state. In the first place, by 'without double-counting' we mean that the instances with an event containing several events having the same state will only be counted once, as they all support the same single pattern. In the second place, alternatively g_0 may be defined as $N = \sum_{i=1}^{m} |g(a_i)|$, for the same purpose. However, since $N = \sum_{i=1}^{m} |g(a_i)|$ is usually much larger than $|g(\alpha)|$, it will result in too small values for support degrees. Therefore, a scale-down measure is often considered desirable. As a matter of fact, other forms of g_0 could be possible, depending on the context and convenience. Notably, since g_0 is a fixed number, the choice of it is a technical treatment and does not affect the properties of DTPs. Take Table 1 as an example again. For a DTP pattern α: $a_3 \Rightarrow^d a_1$, we have $|g(\alpha)|$ =|{(1,4), (10,13), (23,28), (30,38)}|=3, and $|g_0|$=4. Thus, $support(\alpha)$=3/4=0.75. Here, we have an event that contains several events with the same state. That is, e_1=(a_1,(1,20)), e_2=(a_3,(1,4)), and e_6=(a_3,(10,13)), we have $e_2 <^d e_1$ and $e_6 <^d e_1$, which contribute to the same DTP $a_3 \Rightarrow^d a_1$. This counting is also similar to that described in [2].

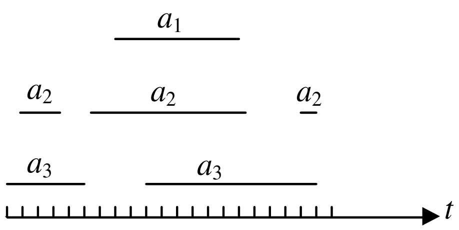

Figure 2: A counterexample for pattern transitivity

Note that the transitivity for *during* relationships between events exists. That is, if $e <^d e'$ and $e' <^d e''$, then we have $e <^d e''$. For instance, we have $e_{10} <^d e_9$ and $e_9 <^d e_8$ in Table 1, and $e_{10} <^d e_9 <^d e_8$. However, it is worth mentioning that the during relationship are not transitive between patterns in terms of support degree. Let us consider an example as follows. Given a temporal database $\mathcal{D}_{\mathcal{T}}$ ={(a_3,(1,5)), (a_2,(2,4), (a_2,(6,15)), (a_1,(8,15)), (a_3,(10,20)), (a_2,(20,20))} and minimal support count=1, with time intervals for the events being illustrated in Figure 2, we have $|g(a_1 \Rightarrow^d a_2)|$=1, $|g(a_2 \Rightarrow^d a_3)|$=2, but $|g(a_1 \Rightarrow^d a_3)|$=0. Thus, we cannot obtain longer patterns from short ones using transitivity. This gives rise to the effort to find other ways of generating longer patters. First, in examining whether DTPs are frequent, we may need to consider sub-patterns. Definition 4 introduces the notion.

Definition 4 For a DTP α with length *l*, we say that α is a sub-DTP of another DTP β with length *k*, denoted as $\alpha \preceq \beta$ if $l \leq k$ and there exists an order-preserving mapping φ: {1,2,...,*l*}→{1,2,...,*k*} such that

$$\alpha(1) \Rightarrow^d \alpha(2) \Rightarrow^d \ldots \Rightarrow^d \alpha(l) \text{ is the same to } \beta(\varphi(1)) \Rightarrow^d \beta(\varphi(2)) \Rightarrow^d \ldots \Rightarrow^d \beta(\varphi(l))$$

where $\alpha(i)$ is the *i*th state in the pattern α.

For instance, $a_6 \Rightarrow^d a_4 \Rightarrow^d a_1$ is one of sub-DTPs of the pattern $a_6 \Rightarrow^d a_4 \Rightarrow^d a_3 \Rightarrow^d a_1$. Moreover, we say $g(\alpha) \supseteq g(\beta)$ if for every $t_k \in g(\beta)$ there exists a time interval $t_l \in g(\alpha)$ such that $t_l \cap t_k = t_k$. In terms of events, $g(\alpha) \supseteq g(\beta)$ means that for every event e_k with t_k in $g(\beta)$, there exists an event e_l with t_l in $g(\alpha)$ such that $e_k <^d e_l$.

Note that $g(\alpha) \supseteq g(\beta)$ does not necessarily mean $|g(\alpha)| \geq |g(\beta)|$. For example, as shown in Figure 1, $g(a_1) \supseteq g(a_3)$ but $|g(a_1)| < |g(a_3)|$. Importantly, for two DTPs α and β with $\alpha \preceq \beta$, one could expect that a longer DTP in length will have a less chance of being supported than its sub-DTPs since the longer the length, the fewer its supporting instances in the database. Accordingly, the fewer the supporting instances for a longer DTP, the finer the set that contains the time intervals of the longer DTP. These statements are proved in Property 1.

Property 1 if $\alpha \preceq \beta$, then $g(\alpha) \supseteq g(\beta)$ and $|g(\alpha)| \geq |g(\beta)|$.

Proof: Suppose that $g(\alpha) \supseteq g(\beta)$ does not hold. Therefore, there exists at least a time interval $t_k \in g(\beta)$, such that $t_l \cap t_k \neq t_k$ for all $t_l \in g(\alpha)$. According to the definition about the set *g*, every state in A_β is active in t_k. However, not all the states in A_α are active in t_k since t_k cannot be totally contained by any interval in $g(\alpha)$. That is, some states in A_β are not active in t_k since $\alpha \preceq \beta$ and $A_\alpha \subseteq A_\beta$. This is a contradiction with $t_k \in g(A_\beta)$. That is, there must be $g(\alpha) \supseteq g(\beta)$ if $\alpha \preceq \beta$.

Furthermore, each time interval $t_k \in g(\beta)$ is contained by the corresponding interval $t_l \in g(\alpha)$ since $g(\alpha) \supseteq g(\beta)$. Some t_l could contain several t_k, and some could not contain any t_k. Let $|g(\alpha)|=x$, $\{T_1, T_2, \ldots, T_x\}$ be the set of time interval sets, in which T_i(i=1,2,...,x) represents the set of time intervals contributing only once for the support count. Let the variable y_i be the flag which represents whether a new instance without double-counting is found in T_i. If a new instance without double-counting has been found, then y_i=1. Otherwise, y_i=0. Thus,

$$|g(\beta)| = \sum_{i=1}^{x} y_i \leq x \times 1 = x = |g(\alpha)|$$ ■

Property 2 All subsets of a frequent pattern are frequent.

Proof: Let β be a frequent pattern, and α is a sub-DTP of β. Thus, we have *support*(β)$\geq$ *minsupport* according to Definition 3, and

$$|g(\alpha)| \geq |g(\beta)|$$

according to Property 1. That is, *support*(α)$\geq$*support*(β)$\geq$minsupport, which means that α is frequent.

■

The above two properties are very important in the process of finding frequent patterns. Effort in scanning the database and examining longer DTPs could then be largely saved by only concentrating on those frequent (sub-)patterns. This is because any DTP containing a non-frequent sub-DTP will not be frequent.

A frequent pattern means that it occurs in forms of events in a sufficient level of frequency. Usually, one also needs to know how likely a pattern occurs given that the other pattern has already occurred. In other words, we are considering the notion of confidence. Concretely, given two DTPs β and γ, where β=$a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_{j+1}$ of length (k-j-1) and γ=$a_j \Rightarrow^d a_{j-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1$ of length (j-1)for j={1,2,,k-1}. Then β *during* γ is a composition of these two DTPs as $\beta \Rightarrow^d \gamma$ which forms a DTP α as follows:

$$\alpha: \ (a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_{j+1}) \Rightarrow^d (a_j \Rightarrow^d a_{j-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1)$$

$$= a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_{j+1} \Rightarrow^d a_j \Rightarrow^d a_{j-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1$$

Definition 5 The confidence degree of a DTP α: $\beta \Rightarrow^d \gamma$ is defined as the fraction of the support of pattern α over the support of the consequent pattern γ.

$$confidence(\alpha) = \frac{support(\alpha)}{support(\gamma)} = \frac{|g(\alpha)|}{|g(\gamma)|}$$

The DTP $\beta \Rightarrow^d \gamma$ is a valid DTP if the support and confidence degrees exceed the corresponding thresholds (minsupport and minconfidence).

Consider Figure 1 again as an example with β and γ being of length 0, i.e., β=a_3 and γ=a_1.

$$\textit{confidence}(a_3 \Rightarrow^d a_1) = \textit{support}(a_3 \Rightarrow^d a_1)/\textit{support}(\{a_1\})\text{=3/3=1.}$$

Finally, given a DTP α (i.e., $a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1 (k \geq 1)$), let us consider a set of states occurring during the period a_1 excluding those states already contained by A_α. The state in the set shall be during a_1 in a frequent manner. Concretely, such a set, $h(\alpha)$, is defined as:

$$h(\alpha) = \{a_j | a_j \notin A_\alpha, j = 1, 2, ..., k, \text{ and if } \alpha \text{ is of a lengh} \geq 1 \text{ then}$$

$$a_j \Rightarrow^d a_1, support(a_j \Rightarrow^d a_1) \geq minsuport\}$$

The set $h(\alpha)$ will be used conveniently in the process of generating candidate DTPs, which will be discussed in detail in the following sections. Merely for illustrative purposes, in Figure 1, we have, for example, $h(a_1)=\{a_2,a_3,a_4,a_6\}$. In comparison, we have $h(a_3 \Rightarrow^d a_1)=\{a_4,a_6\}$ where a_2 is not included since $a_2 \Rightarrow^d a_1$ is not frequent, and a_3 is not included either since a_3 is already in A_α.

3 The algorithm

Straightforwardly, the problem of mining DTPs can be solved by traversing the relationship trees through the paths representing all possible *during* relationships. First, by scanning the temporal database, all frequent DTPs of length 1 (DTP_1's) can be discovered. Each tree represents all the *during* relationships that have a public parent-state. Figure 3 shows an example, in which → represent *contain*.

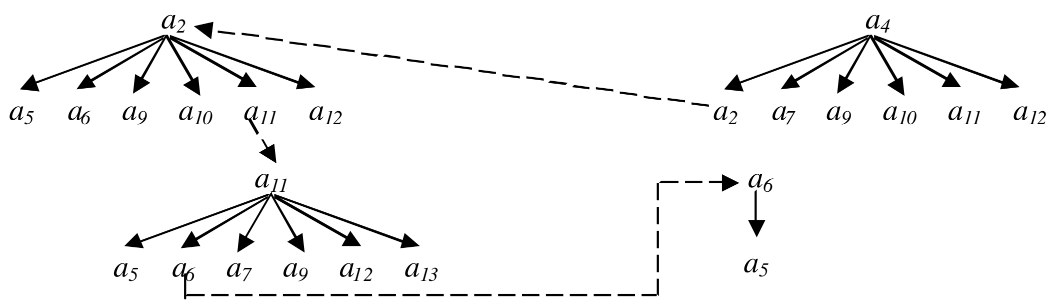

Figure 3: trees of frequent DTP_1

Next, we can traverse each tree using the following strategy:

(1) Let T_l be a tree of frequent DTP_1's. The root of the tree is $R(T_l)$. The set of leaf nodes of T_l is $L(T_l)=\{L_1,L_2,...,L_n\}$.

(2) Using the depth-first search, we completely solve one T_l (l =1,2,...,m) along one path before moving on to the next path. Search path P_j which represents the pattern α. The leaf node of P_j is L_j. Calculate the support degree *support*(α), and

If $L_j \in \{R(T_i) | i = 1,2,..,m, \text{ and } i \neq l\}$,

If *support*(α) < minsupp,

move on to the next path and repeat this step.

Else calculate the confidence degree and get the valid DTPs, and then turn to (3).

Else move on to the next path and turn to (2).

(3)Consider the leaf nodes of T_i which are child nodes of L_j in T_i, i.e., $L(T_l(L_j))$={$L(T_i\}$, the new tree T_l' is generated, $T_l'=T_l \oplus L(T_l(L_j))$, where $\oplus$ is an operation that treats all leaf nodes of T_i as the child nodes of L_j in and forms the new tree T_l'. Thus, path P_j is extended as a sub-tree, which represents several DTPs. Turn to (2). Stop it until no new tree is generated.

Take Figure 3 as an example. For the tree with the root being a_4, and P_j: $a_4 \rightarrow a_2$, we process the tree in the following order $a_4 \rightarrow a_2$, $a_4 \rightarrow a_2 \rightarrow a_5$, $a_4 \rightarrow a_2 \rightarrow a_6$,$a_4 \rightarrow a_2 \rightarrow a_6 \rightarrow a_5$..., $a_4 \rightarrow a_2 \rightarrow a_{11}$, $a_4 \rightarrow a_2 \rightarrow a_{11} \rightarrow a_5$, $a_4 \rightarrow a_2 \rightarrow a_{11} \rightarrow a_6$, $a_4 \rightarrow a_2 \rightarrow a_{11} \rightarrow a_6 \rightarrow a_5$, and so on.

Note that this algorithm firstly finds the DTPs by traversing the trees in the way of depth-first search such that Property 1 can be applied. However, it needs to enumerate all possible candidate DTPs from DTP1 and will consume much time. The method is referred to as Tree algorithm for distinguishing and comparing with the optimized approach that is to propose in the next subsections.

3.1 *DTP Algorithm*

To tackle the above problem, we propose a new algorithm, namely the DTP algorithm, which works in an iterative manner by alternating between the joining and pruning phases, after finding frequent DTP_0's. As the first step, support levels of all single states $DTP_0's$ are calculated and frequent single state patterns are found. Meanwhile, the whole database $\mathcal{D}_{\mathcal{T}}$ is divided into m' ($m' \leq m$) parts (m' frequent single states). Secondly, in the join phase of an iteration k, a collection $CDTP_k$ of new candidate patterns with length k is generated from the set of frequent $DTP_{k-1}'s(FDTP_{k-1})$. Thirdly, the collection $FDTP_k$ of new frequent $DTP_{k-1}'s$ are generated by computing the support degrees. Lastly, the patterns satisfying the confidence threshold are output after all iterations are terminated. More concretely, the algorithmic details are described as follows.

(1). Finding frequent DTP_0

By scanning the whole database $\mathcal{D}_{\mathcal{T}}$ only once and applying the function $g(a_i)$ to each event, $\mathcal{D}_{\mathcal{T}}$ is divided into m parts and the support of each single state can be computed easily. Let $m'=|FDTP_0|$ be the number of frequent single states. So, the number of valid datasets is m'. In the second scanning, $h(a_i)$ can be calculated for every frequent single state a_i. The procedure is shown in Figure 4.

$FDTP_0$**.Gen()**

```
1. FDTP_0=∅;
2. for all a_i∈A do
3.     if |g(a_i)|≥ minsupport then
4.         FDTP_0=FDTP_0∪{a_i};
5.     end if
6. end for
```

Figure 4: The procedure used for $FDTP_0$

(2). Join phase

The algorithm employs an iterative approach known as a level-wise search, where $FDTP_{k-1}$ is used to explore $FDTP_k$ ($k \geq 1$). First, the set of frequent DTP_0(namely $FDTP_0$) is found. $FDTP_0$ is used to find $FDTP_1$, which is used to find $FDTP_2$, and so forth, until no more frequent pattern can be found. The generation of each frequent pattern carries a search cost. To improve the efficiency of generating frequent patterns, two properties are presented as follows.

Let $p(\alpha,l)$ be a sub-pattern with the length (l-1) of the pattern α intercepted from left to right, and $p(\alpha,\text{-}l)$ be a sub-pattern with the length (l-1) of the pattern α from right to left. When l is 1, $p(\alpha,\pm 1)$ is a single state essentially; when l is 0, $p(\alpha,0)$ returns an empty set. For example, given the pattern α, i.e., $a_3 \Rightarrow^d a_2 \Rightarrow^d a_1$, we can get $p(\alpha,2)=a_3 \Rightarrow^d a_2$, $p(\alpha,\text{-}1)=a_1$, and $p(\alpha,0)=\emptyset$.

Property 3 Let β be a frequent pattern, i.e., $a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1$. The pattern γ: $a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_p \Rightarrow^d a_1$ is not frequent if $a_p \notin h(\beta)$.

Proof: According to the notion of set h, $a_p \Rightarrow^d a_1$ is not frequent if $a_p \notin h(\beta)$. As we know, all sub-DTPs of a frequent DTP are frequent, and $a_p \Rightarrow^d a_1$ is one sub-DTP of γ. So, the pattern γ must not be frequent. ■

Thus, to find $FDTP_k$, a set of candidate DTPs with length k is generated by adding $a_p \in h(\beta)$ into the frequent pattern β. This set of candidates is denoted $CDTP_k$, which is a superset of $FDTP_k$. That is, its members may or may not be frequent, but all of the frequent DTP_k are included in $CDTP_k$. In order to obtain a minimal candidate set, we introduce the following property to check whether more sub-DTPs are frequent.

Property 4 Given a pattern $\beta \in FDTP_{k-1}$(k=1,2,...), for any $a_p \in h(\beta)$, if the two following conditions are satisfied:

(i) pattern α: $p(\beta, k) \Rightarrow^d a_p$, $\alpha \in FDTP_{k-1}$.

(ii)$p(\alpha,\text{-}k) \Rightarrow^d p(\beta,\text{-}1) \in FDTP_{k-1}$.

we can join α and β, and add a new candidate pattern, $\alpha \Rightarrow^d p(\beta,\text{-}1)$, into $CDTP_k$.

Proof: Let γ be $p(\alpha,\text{-}k) \Rightarrow^d p(\beta,\text{-}1)$. All of the three patterns α, β and γ are the sub-DTPs of the pattern $\alpha \Rightarrow^d p(\beta,\text{-}1)$. Once any of them is not frequent, the pattern $\alpha \Rightarrow^d p(\beta,\text{-}1)$ must not be frequent (Property 2). That is, we add the pattern $\alpha \Rightarrow^d p(\beta,\text{-}1)$ into the candidate set only when both α and γ are frequent. ■

That is, in the *kth* iteration, for each $a_p \in h(\beta)$, check if both patterns α and γ, i.e., $p(\beta,k) \Rightarrow^d a_p$ and $p(\alpha,\text{-}k) \Rightarrow^d p(\beta,\text{-}1)$ respectively, are frequent. Let us consider all the DTPs as a lattice as shown in Figure 5. And one sublattice is the set of the patterns with the same top element(e.g., a_1,a_3,a_4, and a_6).

For example, let β be $a_6{\Rightarrow}^d a_4{\Rightarrow}^d a_1$ in the sublattice a_1. For $a_3{\in}h(\beta)$, we can search the sublattices a_3 and a_1 to find the patterns satisfying Property 4, i.e., α $a_6{\Rightarrow}^d a_4{\Rightarrow}^d a_3$ and γ $a_4{\Rightarrow}^d a_3{\Rightarrow}^d a_1$. Hence, the new candidate pattern $a_6{\Rightarrow}^d a_4{\Rightarrow}^d a_3{\Rightarrow}^d a_1$ is obtained, which is added into sublattice a_1. The lattice of frequent patterns is shown in Figure 5.

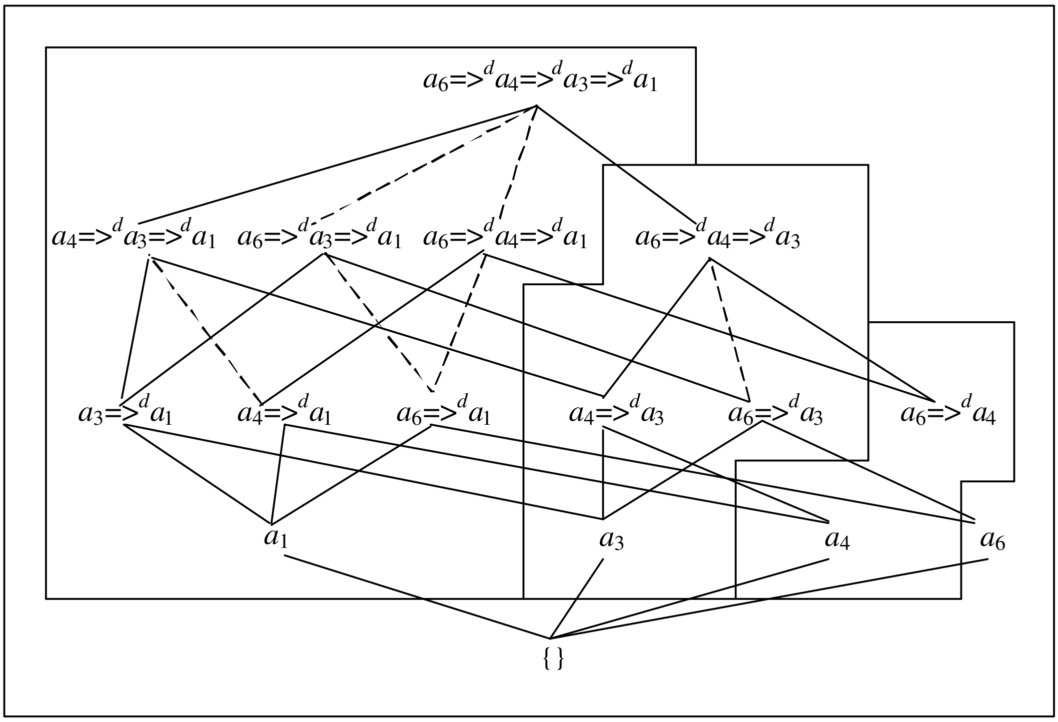

Figure 5: The lattice of frequent patterns

Next, the set $g(\alpha \Rightarrow^d p(\beta,-1))$ for new candidate pattern $\alpha{\Rightarrow}^d p(\beta$,-1) is calculated and the corresponding $h(\alpha \Rightarrow^d p(\beta,-1))$ are obtained. We define the operation $\cap^d$ of the sets $g(\alpha)$ and $g(\beta)$ as follows:

$$g(\alpha) \cap^d g(\beta) = \{t_l \in g(\alpha) | \exists t_k \in g(\beta), \text{ such that } t_l = t_l \cap t_k\}$$

That is, if time interval $t_l \in g(\alpha)$ is totally contained by a time interval $t_k \in g(\beta)$, then t_l is an element of the set $g(\alpha) \cap^d g(\beta)$.

Proposition 1 Let β be $a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_1$, α be $a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_p$ and γ be $a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_p \Rightarrow^d a_1$. The three patterns can generate a longer pattern σ: $a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_p \Rightarrow^d a_1$. The set g for the longer pattern σ is $g(\sigma) = g(\alpha) \cap^d g(\gamma) = g(\beta) \cap^d g(\gamma)$.

Proof: According to the definition of $g(\alpha)$, we have

$$g(\alpha) = g(a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_p)$$

$$= \{t_k \in g(a_k) | \exists t_p \in g(a_p), t_k \in g(a_k), t_i \in g(a_i), \text{such that}$$

$$t_{i+1} \cap t_i = t_{i+1}, t_2 \cap t_p = t_2, \text{for all } i = 2, 3, ..., k-1\}$$

$$g(\gamma) = g(a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_p \Rightarrow^d a_1)$$

$$= \{t'_{k-1} \in g(a_{k-1}) | \exists t'_p \in g(a_p), t'_{k-1} \in g(a_{k-1}), t'_1 \in g(a_1), t'_i \in g(a_i), \text{such that}$$

$$t'_{i+1} \cap t'_i = t'_{i+1}, t'_2 \cap t'_p = t'_2, t'_p \cap t'_1 = t'_p, \text{ for all } i = 2, 3, ..., k-2\}$$

$$g(\sigma) = g(a_k \Rightarrow^d a_{k-1} \Rightarrow^d ... \Rightarrow^d a_2 \Rightarrow^d a_p \Rightarrow^d a_1)$$

$$= \{t''_k \in g(a_k) | \exists t''_p \in g(a_p), t''_k \in g(a_k), t''_1 \in g(a_1), t''_i \in g(a_i), \text{such that}$$

$$t''_{i+1} \cap t''_i = t''_{i+1}, t''_2 \cap t''_p = t''_2, t''_p \cap t''_1 = t''_p, \text{ for all } i = 2, 3, ..., k-1\}$$

Firstly, we prove $g(\sigma) \subseteq g(\alpha) \cap^d g(\gamma)$. $\forall t''_k \in g(\sigma)$, from $g(\sigma)$ we have $t''_{i+1} \cap t''_i = t''_{i+1}$ and $t''_2 \cap t''_p = t''_2$ for $t''_i \in g(a_i)$, i=2,3,...,k-1. Let $t_k = t''_k$, t''_k meets the conditions in $g(\alpha)$. So $g(\sigma) \subseteq g(\alpha)$. And then, we need to prove for any $t''_k \in g(\sigma)$, there exists $t'_{k-1} \in g(\gamma)$, such that $t''_k \cap t''_{k-1} = t''_k$. $\forall t''_k \in g(\sigma)$, there is a group of time intervals t''_{k-1}, t''_{k-2},...,t''_2, t''_1, t''_p satisfying the conditions in the form of $g(\sigma)$. This group of time intervals can also meet the conditions in the form of $g(\gamma)$, so $t''_{k-1} \in g(\gamma)$. Take $t'_{k-1} = t''_{k-1}$, we have $t''_k \cap t'_{k-1} = t''_k$ since $t''_k \cap t''_{k-1} = t''_k$ according to the conditions in $g(\sigma)$. Hence, we get $g(\sigma) \subseteq g(\alpha) \cap^d g(\gamma)$.

Secondly, we prove $g(\alpha) \cap^d g(\gamma) \subseteq g(\sigma)$. $\forall t_k \in g(\alpha) \cap^d g(\gamma)$ means $\exists t_k \in g(\alpha)$, $t'_{k-1} \in g(\gamma)$, such that $t_k \cap t'_{k-1} = t_k$. $t'_{k-1} \in g(\gamma)$ means that there exist corresponding t'_{k-2},...,t'_2, t'_p, t'_1 satisfying $t'_{i+1} \cap t'_i = t'_{i+1}$, $t'_2 \cap t'_p = t'_2$, and $t'_p \cap t'_1 = t'_p$ for all i=2,3,...,k-2. Take $t''_k = t_k$, $t''_p = t'_p$, and $t''_i = t'_i$ for all i=1,2,...,k-1, and then we have

$t''_k \cap t''_{k-1} = t_k \cap t'_k = t_k$; $\quad t''_{i+1} \cap t''_i = t'_{i+1} \cap t'_i = t'_i = t''_{i+1}$ (i=2,3,...,k-2);

$t''_2 \cap t''_p = t'_2 \cap t'_p = t'_2 = t''_2$; $\quad t''_p \cap t''_1 = t'_p \cap t'_1 = t'_p = t''_p$

which means that $t_k \in g(\sigma)$.

Thus, for any $t_k \in g(\alpha) \cap^d g(\gamma)$, $t_k \in g(\sigma)$. That is, $g(\alpha) \cap^d g(\gamma) \subseteq g(\sigma)$.

In the same way, we can get $g(\sigma) = g(\beta) \cap^d g(\gamma)$. ■

The proposition indicates that the set g for a new candidate can be computed by the frequent patterns $g(\alpha)$ and $g(\gamma)$, or $g(\beta)$ and $g(\gamma)$, while the set h for a new candidate pattern can only be obtained from $h(\gamma)$,

$$h(\alpha \Rightarrow^d p(\beta,-1)) = h(\alpha \Rightarrow^d p(\gamma,-1)) = h(\gamma) - \{a_k\}$$

Thus, we get the sets g and h for a new candidate pattern without scanning the original database and the sets g for the single states. The procedure of join phase is shown in Figure 6.

$CDTP_k$**.Gen()**

for all pattern $\beta \in FDTP_{k-1}$ **do**
  for all $a_j \in h(\beta)$ **do**
    for $\alpha \in FDTP_{k-1}, p(\alpha,\text{-}1)=a_j$ **do**
      if Property 4 is satisfied **then**
        $CDTP_k = CDTP_k \bigcup \{\alpha \Rightarrow^d p(\beta,\text{-}1)\}$
      end if
    end for
  end for
end for

Figure 6: The procedure used for $CDTP_k$

(3). Pruning phase

In this phase, if the support degree of a candidate pattern in $CDTP_k$ is smaller than the user-specified threshold, then prune it from $CDTP_k$. At last, $FDTP_k$ is obtained. Intercross Step 2 and Step 3 until Property 2 cannot be satisfied any longer. The procedure of pruning phase is shown in Figure 7.

$FDTP_k$**.Gen()**

$FDTP_0$ known, $FDTP_k=\emptyset$ (k=1,2,...)
for (k=1;$FDTP_{k-1}\neq\emptyset$;k++) **do**
  $CDTP_k = CDTP_k$.Gen($FDTP_{k-1}$);
  for all candidate patterns $\beta \in CDTP_k$ **do**
    $FDTP_k = \{\beta \in CDTP_k \,||\, g(\beta)| \geq \text{minsupport}\}$
  end for
end for
$FDTP = \bigcup_k FDTP_k$

Figure 7: The procedure used for $FDTP_k$

(4). Generating valid DTPs

Given a pattern α: $a_k \Rightarrow^d a_{k-1} \Rightarrow^d \ldots \Rightarrow^d a_2 \Rightarrow^d a_1$, it is necessary to know how frequent α is when a_1 has occurred, when $a_2 \Rightarrow^d a_1$ has occurred, when $a_3 \Rightarrow^d a_2 \Rightarrow^d a_1$ has occurred, and so on. Thus, we can get the pattern $\beta \Rightarrow^d \gamma$ as mentioned in Section 3,

$$(a_k \Rightarrow^d a_{k-1} \Rightarrow^d \ldots \Rightarrow^d a_{j+1}) \Rightarrow^d (a_j \Rightarrow^d a_{j-1} \Rightarrow^d \ldots \Rightarrow^d a_2 \Rightarrow^d a_1)\ (1 \leq j \leq k-1)$$

That is, a frequent pattern can generate (k-1) valid DTPs at most. Calculating the confidence degree starting from the longest consequent pattern, i.e., from the pattern with j=k-1, the patterns meeting the confidence threshold will be valid DTPs. The pattern α will be stopped to compute if confidence($\beta \Rightarrow^d \gamma$)

is less than the threshold. Otherwise, the confidence degree of a shorter consequent pattern, i.e., j=j-1, is computed next.

In discovering DTPs, a temporal database as shown in Tabel 1 is usually needed, which could be obtained either directly or by converting conventional databases. Since a DTP is acturally an event sequence in terms of time inclusion (i.e., during relationship), the records of a conventional database needs to be sorted by ascending start time primarily and descending end time secondarily. Consider the database in Table 1: $\mathcal{D}_{\mathcal{T}}$=$\{e_1,e_2,e_5,e_3,e_6,\ e_4,e_8,e_7,e_9,e_{10},e_{11},\ e_{12},e_{13},e_{14}\}$, and in a sorted form as $\mathcal{D}_{\mathcal{T}}$=$\{e'_1,e'_2,\ e'_3,...,e'_{12},e'_{13},e'_{14}\}$. Subsequently, the set of the resultant events $\{e'_i,e'_{i+1},...,e_{i+k}\}$ (k=1,2,...) is called a *during*-sequence if $e'_{i+j}<^d e'_{i+j-1}$ for all j=1,2,..,k and $e'_{i+k+1}\not<^d e'_{i+k}$. For example, in the sorted $\mathcal{D}_{\mathcal{T}}$, $\{e_1,e_2,e_5,e_3,e_6\}$ is a *during*-sequence since $e_6<^d e_3<^d e_5<^d e_2<^d e_1$ and e_6 is not during the next event e_4. $\{e_4,e_7,e_8,e_9,e_{10}\}$ is the next one in the example.

With data sorted in this way, the search space and the comparison of start time and end time can be reduced when the temporal database is scanned. Let $e_l(a_i,t_l),e_k(a_j,t_k)$, in which $t_l = (st_l, et_l)$ and $t_k = (st_k, et_k)$, be the *lth* and *kth* event about state a_i and a_j $(i \neq j)$ in the sorted $\mathcal{D}_{\mathcal{T}}$, respectively, and k is larger than l, e_l will not occur during the valid period of e_k unless $S(e_l)$=$S(e_k)$ and $E(e_l)$=$E(e_k)$.

Property 5 Assume that k is larger than l, and $e_k(a_j,t_k)\not<^d e_l(a_i,t_l)$. If there is another event $e_w(a_j,t_w)(k<w\leq N)$ with a_j, e_w must not occur during the period of e_l, i.e., $e_w\not<^d e_l$.

Proof : since $w>k$ and both e_k and e_w are the events with the same state a_j, we have $E(e_k)<S(e_w)$. And, $k>l$ and e_k is not during the period of e_l, so we have $E(e_k)>E(e_l)$.

Thus, we have $S(e_w)>E(e_l)$. That is, e_w must not occur during the period of e_l. ■

3.2 *An example*

Let us take an example to explain the DTP algorithm. We will execute the algorithm on the temporal database in Table 1 for minimal support count=2. From Figure 1 we know $FDTP_0$=$\{a_1,\ a_3,\ a_4,\ a_6\}$. Next, for each $a_j\in h(a_i)$, add $a_j\Rightarrow^d a_i$ into $CDTP_1$. Thus, we get $CDTP_1$=$\{a_3\Rightarrow^d a_1,\ a_4\Rightarrow^d a_1,\ a_6\Rightarrow^d a_1,\ a_4\Rightarrow^d a_3,\ a_6\Rightarrow^d a_3,\ a_6\Rightarrow^d a_4\}$, which corresponds to the sets shown in Figure 8.

Subsequently, Step 2 and Step 3 of the DTP algorithm are carried out iteratively. For each $a_j\in h(\beta)$, search the corresponding α and γ mentioned in Property 4. For example, $a_3\in h(a_4 \Rightarrow^d a_1)$ and the support degree of $a_3 \Rightarrow^d a_1$ is not less than the support threshold, so we can join $a_3 \Rightarrow^d a_1$ and $a_4 \Rightarrow^d a_3$ if both patterns are frequent, and obtain the new candidata pattern $a_4\Rightarrow^d a_3\Rightarrow^d a_1$ with the related sets $g(a_4\Rightarrow^d a_3\Rightarrow^d a_1)$ and $h(a_4\Rightarrow^d a_3\Rightarrow^d a_1)$ (as shown in Figure 9 (m)). Similarly, the sets $g(a_6\Rightarrow^d a_4\Rightarrow^d a_3\Rightarrow^d a_1)$ and $h(a_6\Rightarrow^d a_4\Rightarrow^d a_3\Rightarrow^d a_1)$ are obtained in Figure 10.

Support	$g(a_6 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,26)	a_3, a_4
2	(37,37)	

(i) generated by (a) and (f)

Support	$g(a_4 \Rightarrow^d a_3)$	$h(\alpha)$
1	(25,27)	a_6
2	(34,38)	

(j) generated by (c) and (d)

Support	$g(a_6 \Rightarrow^d a_3)$	$h(\alpha)$
1	(25,26)	a_4
2	(37,37)	

(k) generated by (c) and (f)

Support	$g(a_6 \Rightarrow^d a_4)$	$h(\alpha)$
1	(25,26)	$\emptyset$
2	(37,37)	

(l) generated by (d) and (f)

Figure 8: The sets of $CDTP_1$

Support	$g(a_4 \Rightarrow^d a_3 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,27)	a_6
2	(34,38)	

(m) generated by (g) and (j)

Support	$g(a_6 \Rightarrow^d a_3 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,26)	a_4
2	(37,37)	

(n) generated by (g) and (k)

Support	$g(a_6 \Rightarrow^d a_4 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,26)	a_3
2	(37,37)	

(0) generated by (h) and (l)

Support	$g(a_6 \Rightarrow^d a_4 \Rightarrow^d a_3)$	$h(\alpha)$
1	(25,26)	$\emptyset$
2	(37,37)	

(p) generated by (j) and (k)

Figure 9: The sets of $CDTP_2$

Support	$g(a_6 \Rightarrow^d a_4 \Rightarrow^d a_3 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,26)	$\emptyset$
2	(37,37)	

(q) generated by (m) and (p)

Figure 10: The sets of $CDTP_3$

Lastly, we calculate the confidence degree of the patterns from the bottom of every sublattice. In Figure 5, there are three sublittices.

4 Experiments

To assess the relative performance of these two algorithms and study their scale-up properties, we performed several experiments on a computer with 512 RAM and Pentium4 2.6GHz for some synthetic datasets and a real data set with weather information, which was stored on a local 20G disk.

4.1 *Generation of synthetic data*

To evaluate the performance of the algorithms over a large volume of data, we generated synthetic temporal events which mimic the events in the real word. We will show the experimental results from synthetic data so that the work relevant to data cleaning, which is in fact application dependent and also orthogonal to the incremental technique proposed, is hence omitted for simplicity. For obtaining reliable experimental results, the method to generate synthetic data we employed in this study is similar to the ones used in [12].

Table 2 summarizes the meaning of the parameters used in the experiments. The number of input-events in the temporal database relies on $|Q|$ and $|T|$. That is, the average number of events is $|D_T|$=$|Q|$*$|T|$. The starting time and ending time of each event in a *during*-sequence are generated randomly based on the *during* relationship. We generated datasets by setting $|L|$=5, N=50 and P=25. Table 3 summarizes the dataset parameter settings.

Table 2: Parameters

$\|Q\|$	Number of *during*-sequences
$\|T\|$	Average number of events per *during*-sequences
$\|L\|$	Average length of maximal potentially large patterns
N	Number of states
P	Number of maximal potentially large patterns

Table 3: Parameters settings (Synthetic datasets)

Name	$\|Q\|$	$\|T\|$	$\|D\|$	size(MB)
Data1-Q10-T5	10000	5	50,000	0.83
Data2-Q10-T10	10000	10	100,000	1.79
Data3-Q20-T5	20000	5	100,000	1.82
Data4-Q20-T10	20000	10	200,000	3.72
Data5-Q30-T10	30000	10	300,000	5.65
Data6-Q40-T10	40000	10	400,000	7.58
Data7-Q50-T10	50000	10	500,000	9.51

4.2 *The relative performance with synthetic datasets*

Figure 11 shows the execution times for the first four synthetic datasets given in Table 3 for decreasing values of minimum support. We did not plot the execution times of the Tree algorithm for some lower support values since they are too large compared to the execution times of DTP algorithm. As the support threshold decreases, the execution times of both the algorithms increase because of the increase in the total number of candidate and large patterns. When the support threshold is higher, there are only a limited number of frequent patterns with length 2 produced. So both algorithms consume less time. However, as the support threshold decreases, the performance difference becomes prominent in that DTP algorithm significantly outperforms the Tree algorithm.

4.3 *The reduction of candidate patterns*

As explained previously, the DTP algorithm substantially reduces the number of candidate patterns generated. The experimental results in Table 4 and Table 5 show the number of candidate patterns of both algorithms for different supports on the two datasets. As shown in Table 5, the DTP algorithm

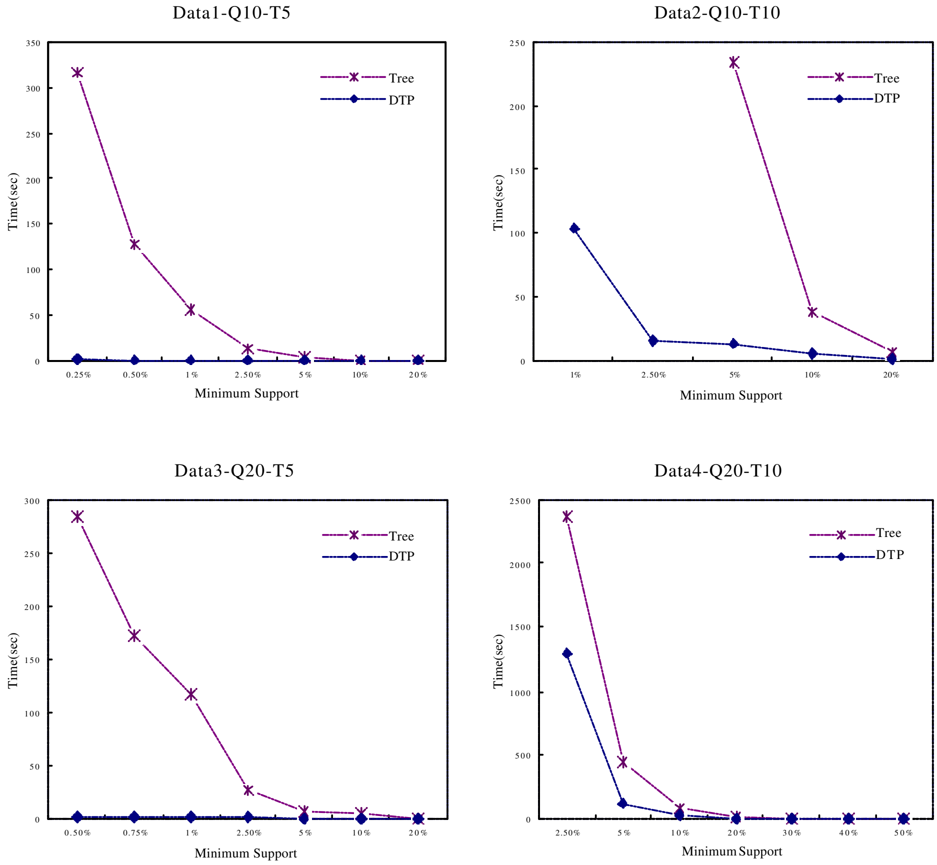

Figure 11: Execution times

leads to a 55%-75% candidate reduction rate compared to the Tree algorithm when Data3-Q20-T5 is used. In another dataset Data4-Q20-T10, the DTP algorithm can achieve higher reduction rate in generating candidate DTPs, such as 200%, as shown in Table 4. Similar phenomena were observed when other datasets were used. This feature of the DTP algorithm could help efficiently reduce the execution time (as mentioned in Section 5.1).

Table 4: Reduction on candidate patterns with Data4-Q20-T10

minsupport	candidate patterns		frequent patterns
	DTP	Tree	
40%	34	112	27
30%	167	364	133
20%	552	1202	460
10%	2483	7074	1920
5%	10105	34046	8160
2.50%	35281	142009	29712

Table 5: Reduction on candidate patterns with Data3-Q20-T5

minsupport	candidate patterns		frequent patterns
	DTP	Tree	
20%	16	53	11
10%	118	351	68
5%	454	1029	251
2.50%	1223	2827	699
1%	3166	9334	2237
0.75%	4901	12845	3022
0.50%	7202	19090	4482
0.25%	13662	39555	8691
0.10%	30964	113144	21500

4.4 *Scale-up*

Figure 12 shows how DTP algorithm scales up as the number of input-events is increased from 100,000 to 500,000 when the datasets Data2, Data4, Data5, Data6 and Data7 in Table 3 are used. The minimum support level was set to 5%. As shown in Figure 12, the algorithm is approximately linear scalable over the number of input events.

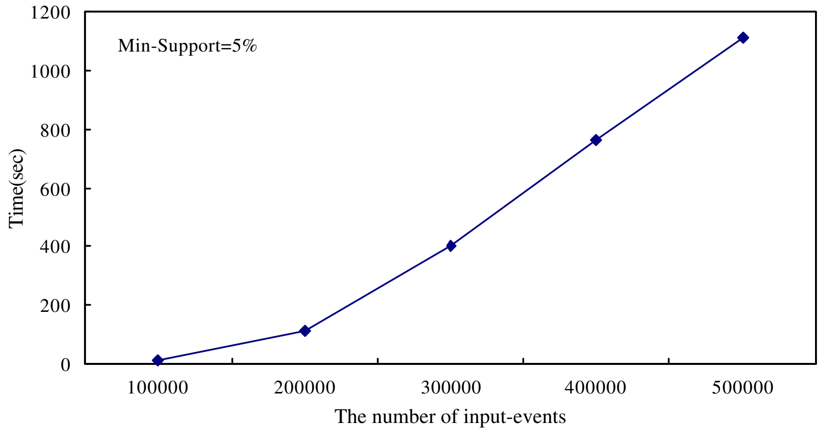

Figure 12: Scale-up

4.5 *Experimtents with weather dataset*

The DTP algorithm has been applied to a weather dataset. The data set was obtained from a weather station in The Netherlands in 2002, which contains weather records with 17 attributes for each hour of the year. The attributes include wind direction, average wind speed, maximum wind gust, average hourly temperature, percentage relative humidity, global hourly radiation, hourly sunshine duration, hourly precipitation duration, hourly precipitation amount,horizontal visibility, fog, snow, etc. Most of the values are continuous. We discretized the data and then converted the database into a temporal one, in a form similar to Table 1.

Figure 13 compares the execution time of the two algorithms on the real dataset, with different minimum support degrees. From this figure, we can see that DTP algorithm is more efficient than *Tree* algorithm, especially at the lower support level.

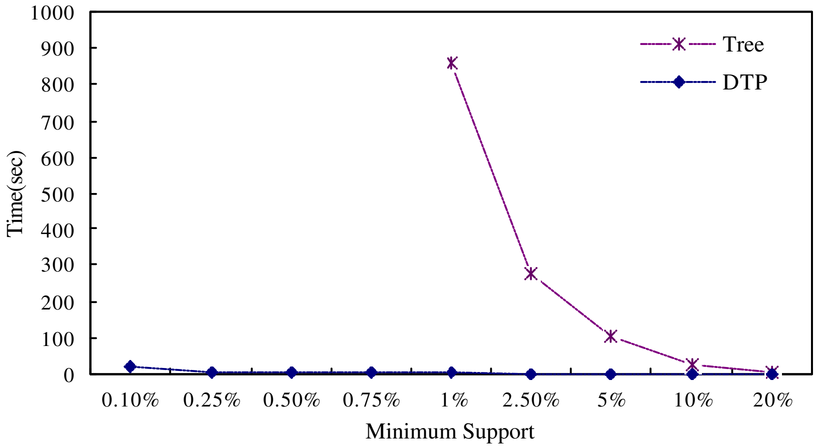

Figure 13: Execution times on the weather dataset

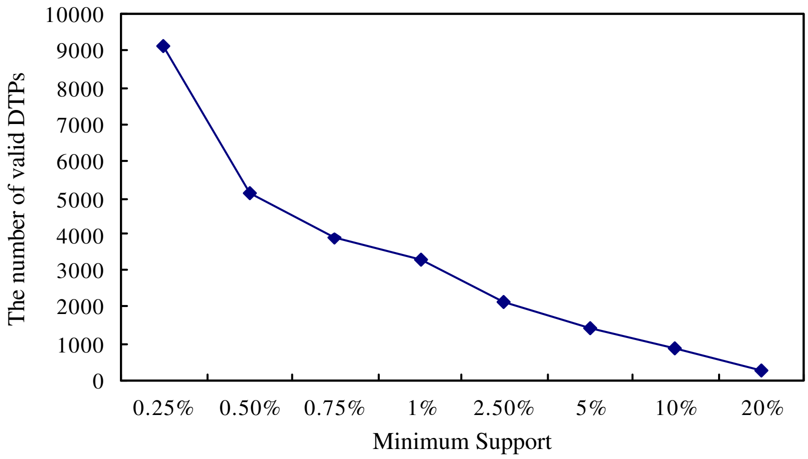

Figure 14: The number of valid DTPs

Figure 14 shows the number of valid DTPs with various support degrees. Some interesting results were generated by applying the algorithm. Firstly, we discovered that there were significant temporal relationships between wind and rain, humidity and radiation, and temperature, radiation and sunshine duration. For example, in terms of wind and rain, strong wind often came prior to rain and lasted until or beyond the end of the rain (*rain during strong wind*). Another example is that the horizontal visibility was usually weak during the time that fog was heavy or it snowed (*weak horizontal visibility during heavy fog or snowing*). Secondly, some more complex rules were also discovered. For instance, when the sunshine duration was shortened, the global hourly radiation would decrease and the horizontal visibility would become worse. In fact, before sunshine duration reached zero and the night fell, the temperature would have begun to decrease. This rule can be expressed as *worse horizontal sight* *during no global hourly radiation during no sunshine during very lower temperature.*

5 Conclusions and future work

In this paper, we have studied the problem of discovering *during*-temporal patterns between events and proposed the DTP algorithm. By analyzing the properties of the *during* relationship, we have developed an optimization technique with pruning strategies that enabled us to retrieve the patterns with minimal database scan. The experimental results have illustrated the effectiveness and efficiency of the algorithm. An ongoing effort centers on extending this algorithm to discovering the dynamic temporal database, since in the light of the fact that the content of a TDB always keeps growing and the discovered patterns need to be maintained periodically over time. While there are some studies on mining of dynamic database and maintenance of association rules [1,2,4,5,15], our focus is on the maintenance of temporal patterns, so as to reduce the overhead of rediscovering patterns in the presence of data updates.

References

[1] A. Veloso, W. Meira Jr., M.B.De Carvalho, S.Parthasarathy, and M.Zaki. (2003). Parallel, Incremental and Interactive Mining for Frequent Itemsets in Evolving Databases. In Proceedings of the Sixth SIAM Workshop on High Performance Data Mining, May 2003.

[2] C. Jin, W. Qian, C. Sha, J. Yu, and A. Zhou. (2003). Dynamically Maintaining Frequent Items over A Data Stream. In Proceedings of the 12th ACM CIKM International Conference on Information and Knowledge Management, pages 287–294, 2003.

[3] Chris P.Rainsford, and John F.Roddic. (1999). Adding Temporal Semantics to Association Rules. In *Proc. of the 3rd European conf. on principles and practice of knowledge discovery in databases*, pages 504-509, 1999.

[4] D.W.Cheung, J.Han, V.T.Ng, and C.Y.Wong. (1996). Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique. Proc, Int'l Conf. Data Eng., (ICDE'96)S.Y.W.Su,ed., pp.106-114,1996.

[5] D.W.Cheung, S.D.Lee, and B.Kao. (1997). A General Incremental Technique for Maintaining Discovered Association Rules. In Proc. Fifth Int'l Conf. Database Systems for Advanced Applications, 1997.

[6] F.Hoppner. (2001). Discovery of Temporal Patterns—Learning Rules about the Qualitative Behaviour of Time Series. In *PKDD'01*, number 2168 of LNAI, Freiburg, Germany, pages 192-203, 2001.

[7] Guoqing Chen, Jiang Ai, Wei Yu. (2002) Discovering Temporal Association Rules for Time-Lag Data. *Proceedings of International Conference on E-Business* (ICEB2002), Beijing, May 2002.

[8] G.Das, K.I.Lin, H. Mannila. (1998). Rule discovery from time series. In *Proceedings of the international conference on KDD and Data Mining*, 1998.

[9] J. Chang and W. Lee. (2003). Finding Recent Frequent Itemsets Adaptively over Online Data Streams. In Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 487–492, 2003.

[10] James F.Allen. (1983). Maintaining Knowledge about Temporal Intervals. *Communication of the ACM*, Volume 26, Number 11, P832-843, November 1983.

[11] Mohammed.J.ZAKI. (2000). SPADE: An Efficient Algorithm for Mining Frequent Sequences. *Machine Learning*, 0, 1-31, 2000.

[12] R.Agrawal, and R.Srikant. (1994). Fast algorithm for mining association rules. In *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, September, 1994.

[13] R.Agrawal, and R.Srikant. (1995). Mining Sequential Patterns. *International Conference on Data Engineering*(ICDE), March, Taiwan, 1995.

[14] R.Agrawal, T.Imielinski, and A.Swami. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207-216, Washington DC, May 26-28, 1993. 10.

[15] Y.Chi, H.J.Wang, Philip S.Yu, and Richard R. Muntz. (2004). Catch the Moment Maintaining Closed Frequent Itemsets. http://citeseer.ist.psu.edu.

Table 1: A Temporal Database

Event	**State**	**Starting Time**	**Ending Time**
e_1	a_1	1	20
e_2	a_3	1	4
e_3	a_4	5	7
e_4	a_1	22	28
e_5	a_2	2	8
e_6	a_3	10	13
e_7	a_5	25	35
e_8	a_3	23	28
e_9	a_4	25	27
e_{10}	a_6	25	26
e_{11}	a_1	30	40
e_{12}	a_3	30	38
e_{13}	a_4	34	38
e_{14}	a_6	37	37

Table 2: Parameters

$\|Q\|$	Number of *during*-sequences
$\|T\|$	Average number of events per *during*-sequences
$\|L\|$	Average length of maximal potentially large patterns
N	Number of states
P	Number of maximal potentially large patterns

Table 3: Parameters settings (Synthetic datasets)

Name	$\|Q\|$	$\|T\|$	$\|D\|$	size(MB)
Data1-Q10-T5	10000	5	50,000	0.83
Data2-Q10-T10	10000	10	100,000	1.79
Data3-Q20-T5	20000	5	100,000	1.82
Data4-Q20-T10	20000	10	200,000	3.72
Data5-Q30-T10	30000	10	300,000	5.65
Data6-Q40-T10	40000	10	400,000	7.58
Data7-Q50-T10	50000	10	500,000	9.51

Table 4: Reduction on candidate patterns with Data4-Q20-T10

minsupport	candidate patterns		frequent patterns
	DTP	Tree	
40%	34	112	27
30%	167	364	133
20%	552	1202	460
10%	2483	7074	1920
5%	10105	34046	8160
2.50%	35281	142009	29712

Table 5: Reduction on candidate patterns with Data3-Q20-T5

minsupport	candidate patterns		frequent patterns
	DTP	Tree	
20%	16	53	11
10%	118	351	68
5%	454	1029	251
2.50%	1223	2827	699
1%	3166	9334	2237
0.75%	4901	12845	3022
0.50%	7202	19090	4482
0.25%	13662	39555	8691
0.10%	30964	113144	21500

Support	$g(a_1)$	$h(a_1)$
1	(1,20)	
2	(22,28)	a_2, a_3, a_4, a_6
3	(30,40)	

(a)

Support	$g(a_2)$	$h(a_2)$
1	(2,8)	a_4

(b)

Support	$g(a_3)$	$h(a_3)$
1	(1,4)	
2	(10,13)	a_4, a_6
3	(23,28)	
4	(30,38)	

(c)

Support	$g(a_4)$	$h(a_4)$
1	(5,7)	
2	(25,27)	a_6
3	(34,38)	

(d)

Support	$g(a_5)$	$h(a_5)$
1	(25,35)	a_4, a_6

(e)

Support	$g(a_6)$	$h(a_6)$
1	(25,26)	$\emptyset$
2	(37,37)	

(f)

Support	$g(a_3 \Rightarrow^d a_1)$	$h(\alpha)$
1	(1,4)	
	(10,13)	a_4, a_6
2	(23,28)	
3	(30,38)	

(g)

Support	$g(a_4 \Rightarrow^d a_1)$	$h(\alpha)$
1	(5,7)	
2	(25,27)	a_3, a_6
3	(34,38)	

(h)

Figure 1: The examples of the sets g and h

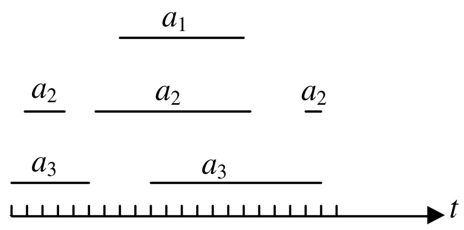

Figure 2: A counterexample for pattern transitivity

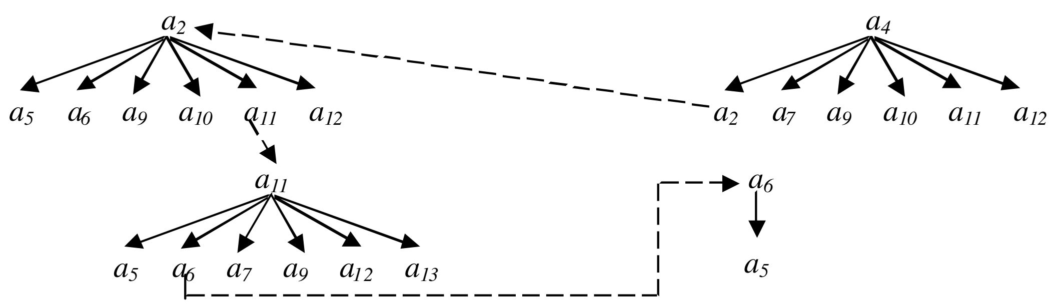

Figure 3: trees of frequent DTP_1

$FDTP_0$**.Gen()**

```
FDTP_0 = ∅;
for all a_i ∈ 𝒜 do
    if |g(a_i)| ≥ minsupport then
        FDTP_0 = FDTP_0 ∪ {a_i};
    end if
end for
```

Figure 4: The procedure used for $FDTP_0$

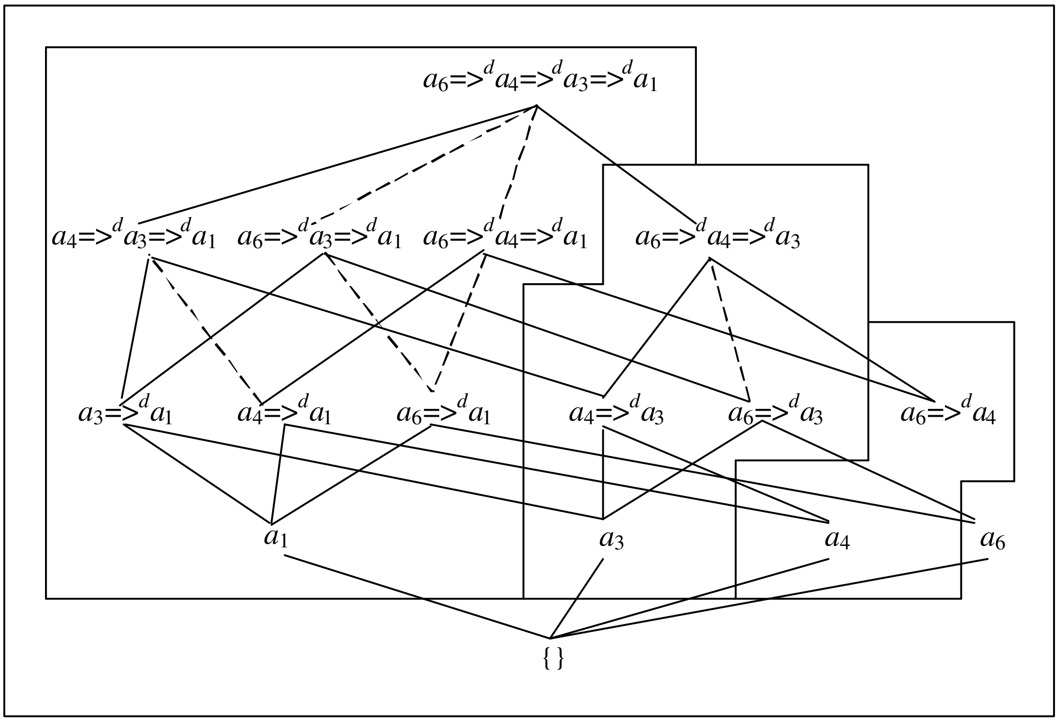

Figure 5: The lattice of frequent patterns

$CDTP_k$.**Gen()**

for all pattern $\beta \in FDTP_{k-1}$ **do**
    for all $a_j \in h(\beta)$ **do**
        for $\alpha \in FDTP_{k-1}, p(\alpha,\text{-}1) = a_j$ **do**
          if Property 4 is satisfied **then**
              $CDTP_k = CDTP_k \bigcup \{\alpha \Rightarrow^d p(\beta,\text{-}1)\}$
          end if
        end for
    end for
end for

Figure 6: The procedure used for $CDTP_k$

$FDTP_k$**.Gen()**

```
FDTP_0 known, FDTP_k=∅ (k=1,2,...)
for (k=1;FDTP_{k-1}≠∅;k++) do
     CDTP_k=CDTP_k.Gen(FDTP_{k-1});
     for all candidate patterns β∈CDTP_k do
          FDTP_k={β∈CDTP_k||g(β)|≥minsupport}
     end for
end for
FDTP=⋃_k FDTP_k
```

Figure 7: The procedure used for $FDTP_k$

Support	$g(a_6 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,26)	a_3, a_4
2	(37,37)	

(i) generated by (a) and (f)

Support	$g(a_4 \Rightarrow^d a_3)$	$h(\alpha)$
1	(25,27)	a_6
2	(34,38)	

(j) generated by (c) and (d)

Support	$g(a_6 \Rightarrow^d a_3)$	$h(\alpha)$
1	(25,26)	a_4
2	(37,37)	

(k) generated by (c) and (f)

Support	$g(a_6 \Rightarrow^d a_4)$	$h(\alpha)$
1	(25,26)	$\emptyset$
2	(37,37)	

(l) generated by (d) and (f)

Figure 8: The sets of $CDTP_1$

Support	$g(a_4 \Rightarrow^d a_3 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,27)	a_6
2	(34,38)	

(m) generated by (g) and (j)

Support	$g(a_6 \Rightarrow^d a_3 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,26)	a_4
2	(37,37)	

(n) generated by (g) and (k)

Support	$g(a_6 \Rightarrow^d a_4 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,26)	a_3
2	(37,37)	

(0) generated by (h) and (l)

Support	$g(a_6 \Rightarrow^d a_4 \Rightarrow^d a_3)$	$h(\alpha)$
1	(25,26)	$\emptyset$
2	(37,37)	

(p) generated by (j) and (k)

Figure 9: The sets of $CDTP_2$

Support	$g(a_6 \Rightarrow^d a_4 \Rightarrow^d a_3 \Rightarrow^d a_1)$	$h(\alpha)$
1	(25,26)	$\emptyset$
2	(37,37)	

(q) generated by (m) and (p)

Figure 10: The sets of $CDTP_3$

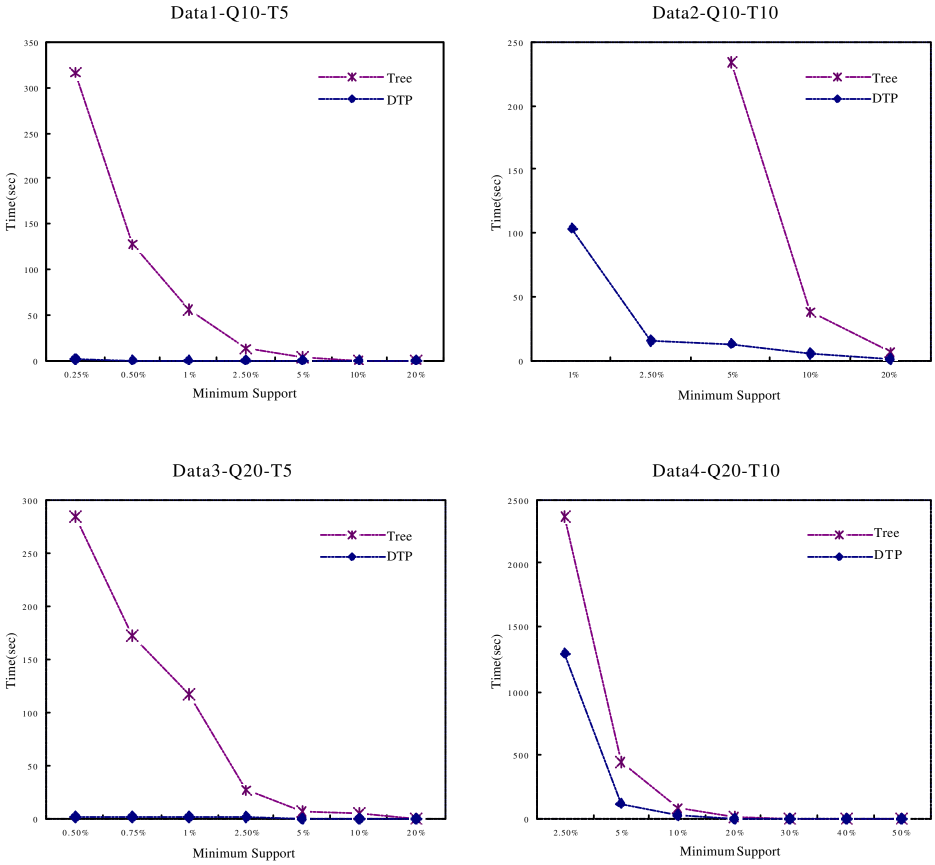

Figure 11: Execution times

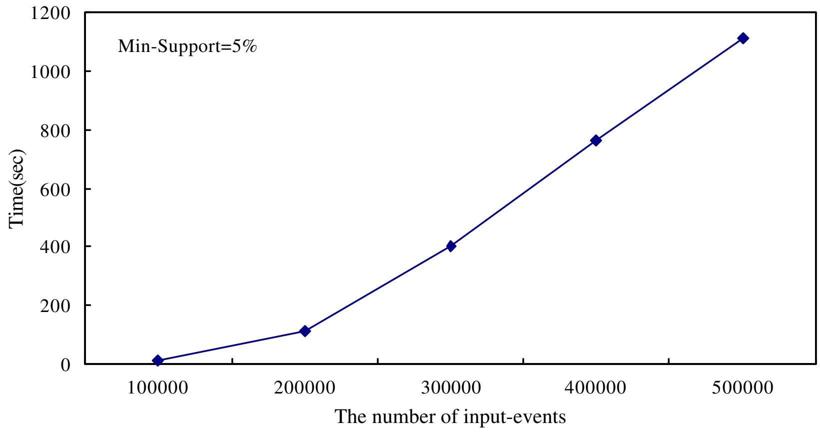

Figure 12: Scale-up

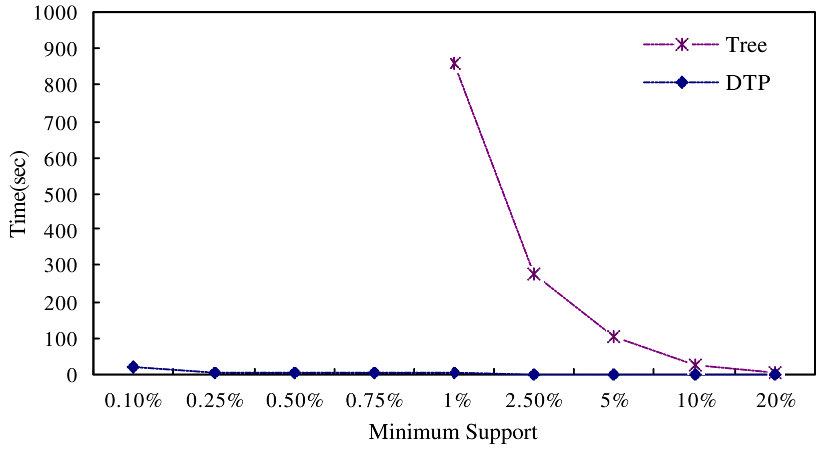

Figure 13: Execution times on the weather dataset

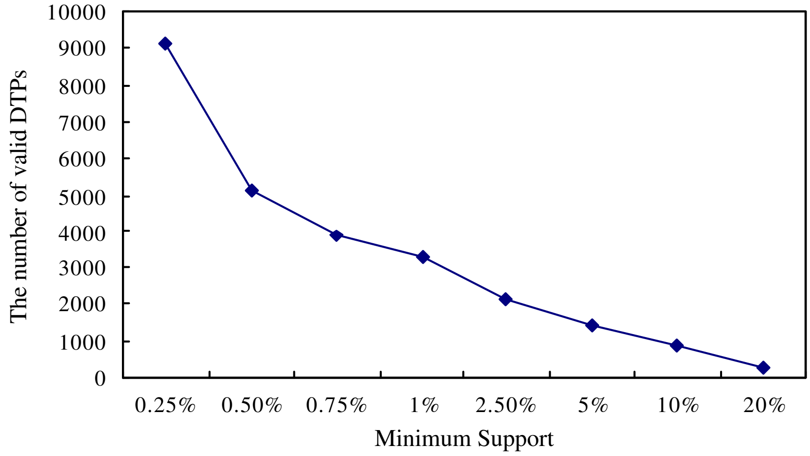

Figure 14: The number of valid DTPs