

# Mining Train Delays

Boris Cule<sup>1</sup>, Bart Goethals<sup>1</sup>, Sven Tassenoy<sup>2</sup>, and Sabine Verboven<sup>2,1</sup>

<sup>1</sup>University of Antwerp, Department of Mathematics and Computer Science, Middelheimlaan 1, 2020 Antwerp, Belgium

<sup>2</sup>INFRABEL - Network, Department of Innovation, Barastraat 110, 1070 Brussels, Belgium

## Keywords

Pattern Mining, Data Analysis, Train Delays

## Abstract

The Belgian railway network has a high traffic density with Brussels as its gravity center. The star-shape of the network implies heavily loaded bifurcations in which knock-on delays are likely to occur. Knock-on delays should be minimized to improve the total punctuality in the network. Based on experience, the most critical junctions in the traffic flow are known, but others might be hidden. To reveal the hidden patterns of trains passing delays to each other, state-of-the-art data mining techniques are being studied and adapted to this specific problem.

## 1 Introduction

The Belgian railway network, as shown in Figure 1, is very complex because of the numerous bifurcations and stations at relatively short distances. It belongs to the group of the most dense railway networks in the world. Moreover, its star-shaped structure creates a huge bottleneck in its center, Brussels, as approximately 40% of the daily trains pass through the Brussels North-South junction.

During the past five years, the punctuality of the Belgian trains has gradually decreased towards a worrisome level. Meanwhile, the number of passengers, and therefore also the number of trains necessary to transport those passengers has increased. Even though the infrastructure capacity is also slightly increasing by doubling the number of tracks on the main lines around Brussels, the punctuality is still decreasing. To solve the decreasing punctuality problem, its main causes should be discovered, but because of the complexity of the network, it is hard to trace their true origin. It may happen that a structural delay in a particular part of the network seems to be caused by busy traffic, although in reality this might be caused by a traffic operator in a seemingly unrelated place, who makes a bad decision every day, unaware of the consequences of his decision.

We study the application of data mining techniques in order to discover related train delays in this data. Whereas Flier et al. [2] try to discover patterns underlying dependencies of the delays, this paper considers the patterns in the delays themselves. Mirabadi and Sharafian [6] use association mining to analyse the causes in accident data sets, whereas we consider the so called frequent pattern mining methods. Prototypical examples of these

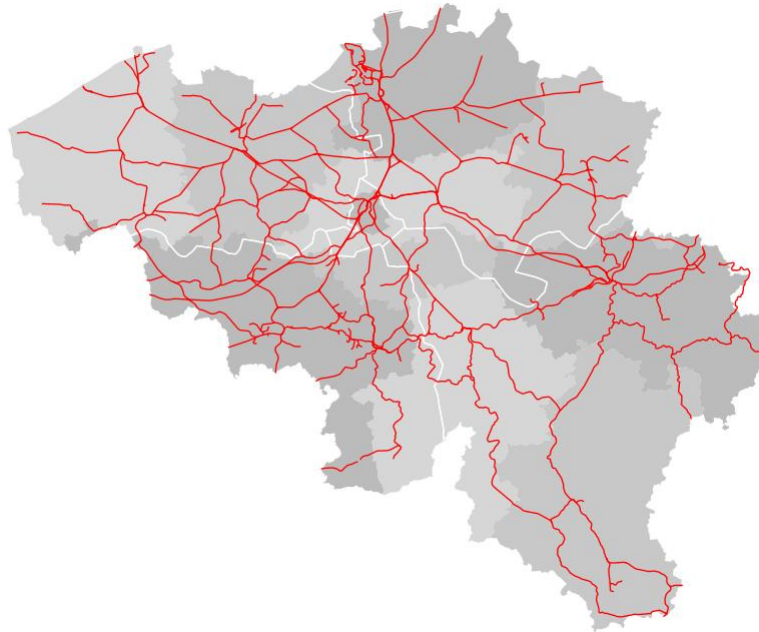


Figure 1: The Belgian Railway Network

methods can be found in a supermarket retail setting, where the marketer is interested in all sets of products being bought together by customers. A well known result being discovered by a retail store in the early days of frequent pattern mining, is that “70% of all customers who buy diapers also buy beer”. Such patterns could then be used for targeted advertising, product placement, or other cross-selling studies. In general, the identification of sets of items, products, symptoms, characteristics, and so forth, that often occur together in the given database, can be seen as one of the most basic tasks in Data Mining [7, 3].

Here, a database of *Infrabel* containing the times of trains passing through characteristic points in the railway network is being used. In order to discover hidden patterns of trains passing delays to each other, our first goal is to find frequently occurring sets of train delays. More specifically, we try to find all delays that frequently occur within a certain time window, counted over several days or months of data [5]. In this study, we take into account the train (departure) delays larger than or equal to 3 or 6 minutes. For example, we consider patterns such as: *Trains A, B, and C, with C departing before A and B, are often delayed at a specific location, approximately at the same time.*

Computing such patterns, however, is intractable in practice [4] as the number of such potentially interesting patterns grows exponentially with the number of trains. Fortunately, efficient pattern mining techniques have recently been developed, making the discovery of such patterns possible. A remaining challenge is still to distinguish the interesting patterns from the irrelevant ones. Typically a frequent pattern mining algorithm will find an enormous amount of patterns amongst which many can be ruled out by irrelevance. For example, two local trains which have no common characteristic points in their route could, however,

appear as a pattern if they are both frequently delayed, and their common occurrence can be explained already by statistical independence.

In the next Section, we explain the studied pattern mining techniques. In Section 3, we discuss the collected data at characteristic points and report on preliminary experiments, showing promising results, and we conclude the paper with suggestions for future work.

## 2 Pattern mining

### 2.1 Itemsets

The simplest possible pattern are itemsets [7]. Typically, we look for items (or events) that often occur together, where the user, by setting a frequency threshold, decides what is meant by ‘often’.

Formally, let  $\mathcal{I} = \{x_1, x_2, \dots, x_n\}$  be a set of items. A set  $X \subseteq \mathcal{I}$  is called an *itemset*. The database  $\mathcal{D}$  consists of a set of transactions, where each transaction is of the form  $\langle t, X \rangle$ , where  $t$  is a unique transaction identifier, and  $X$  is an itemset.

Given a database  $\mathcal{D}$ , the *support* of an itemset  $Y$  in  $\mathcal{D}$  is defined as the number of transactions in  $\mathcal{D}$  that contain  $Y$ , or

$$\text{sup}(Y, \mathcal{D}) = |\{t | \langle t, X \rangle \in \mathcal{D} \text{ and } Y \subseteq X\}|.$$

$Y$  is said to be *frequent* in  $\mathcal{D}$  if  $\text{sup}(Y, \mathcal{D}) \geq \text{minsup}$ , where *minsup* is a user defined minimum support threshold (often referred to as the frequency threshold). Support has a very interesting property — it is *downward-closed*. This means that the support of an itemset is always smaller than or equal to the support of any of its subsets. This observation is crucial for many frequent pattern algorithms, and can also be used to reduce the output size by leaving out everything that can be deduced from the patterns left in the output.

A typical transaction database can be seen in Table 1.

TID	Items Bought
1	{Bread, Butter, Milk}
2	{Bread, Butter, Cookies}
3	{Beer, Bread, Diapers}
4	{Milk, Diapers, Bread, Butter}

Table 1: An example of a transaction database.

As can be seen in Table 2 the *Infrabel* database does not consist of transactions of this type.

In order to mine frequent itemsets in the traditional way, the Infrabel data would need to be transformed. Therefore a transaction database can be created, in which each transaction consists of train IDs of trains that were late within a given period of time. Table 3 shows a part of the data from Table 2 transformed into a transaction database with each transaction consisting of trains delayed within a period of five minutes. Each transaction represents one such period. Mining frequent itemsets would then result in obtaining sets of train IDs that

Time stamp	Train ID
...	
06:05 15/02/2010	A
06:07 15/02/2010	B
06:09 15/02/2010	C
06:35 15/02/2010	D
...	

Table 2: A simplified example of a train delay database.

are often late ‘together’. In the example given in Table 3, assuming a support threshold of 2, the frequent itemsets are  $\{A\}$ ,  $\{B\}$ ,  $\{C\}$ ,  $\{A, B\}$  and  $\{B, C\}$ .

TID	Delayed trains
...	
06:00 15/02/2010 - 06:04 15/02/2010	$\{\}$
06:01 15/02/2010 - 06:05 15/02/2010	$\{A\}$
06:02 15/02/2010 - 06:06 15/02/2010	$\{A\}$
06:03 15/02/2010 - 06:07 15/02/2010	$\{A,B\}$
06:04 15/02/2010 - 06:08 15/02/2010	$\{A,B\}$
06:05 15/02/2010 - 06:09 15/02/2010	$\{A,B,C\}$
06:06 15/02/2010 - 06:10 15/02/2010	$\{B,C\}$
06:07 15/02/2010 - 06:11 15/02/2010	$\{B,C\}$
06:08 15/02/2010 - 06:12 15/02/2010	$\{C\}$
06:09 15/02/2010 - 06:13 15/02/2010	$\{C\}$
06:10 15/02/2010 - 06:14 15/02/2010	$\{\}$
...	

Table 3: Transformed version of an extract of the data in Table 2.

Table 3 illustrates why the frequent itemset method is not the most intuitive to tackle our problem. First of all, it requires a lot of preprocessing work in order to transform the data into the necessary format. Second, the transformation results in a dataset full of redundant information, as there are many empty or identical transactions. This problem is further multiplied when we refine our time units to seconds instead of minutes. Finally, as will be shown later, itemsets are quite limited and other methods allow us to find much better patterns.

## 2.2 Sequences

Looking for frequent itemsets alone does not allow us to extract all the useful information from the *Infrabel* database. This database is, by its nature, sequential, and it would be natural to try to generate patterns taking the order of events into account, too. Typically, in these kinds of problems, the database consists of sequences, and the goal is to find frequent subsequences, i.e. sequences that can be found in many of the sequences in the database [9, 1].

Formally, a *sequence* is defined as an ordered list of symbols (coming from an alphabet  $\Sigma$ ), and is written as  $\mathbf{s} = s_1s_2 \cdots s_k$ , where  $s_i$  is the symbol at position  $i$ . A sequence  $\mathbf{s} = s_1s_2 \cdots s_k$  is said to be a *subsequence* of another sequence  $\mathbf{r} = r_1r_2 \cdots r_n$ , denoted as  $\mathbf{s} \subseteq \mathbf{r}$ , if for all  $s_i \in \mathbf{s}$ , there exists  $r_{j_i} \in \mathbf{r}$ , such that  $s_i = r_{j_i}$ , and the order of symbols is preserved, i.e.  $j_1 < j_2 < \cdots < j_k$ . In other words, sequence  $\mathbf{s}$  is embedded in sequence  $\mathbf{r}$ , though there may be gaps in  $\mathbf{r}$  between consecutive symbols of  $\mathbf{s}$ . Given a database of sequences  $\mathcal{D} = \{\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_n\}$ , the support of a sequence  $\mathbf{r}$  in  $\mathcal{D}$  is defined as the total number of sequences in  $\mathcal{D}$  that contain  $\mathbf{r}$ , or

$$\text{sup}(\mathbf{r}) = |\{\mathbf{s}_i \in \mathcal{D} | \mathbf{r} \subseteq \mathbf{s}_i\}|.$$

Note that, again, a subsequence of a frequent sequence must also be frequent.

A typical sequence dataset is given in Table 4.

Sequence ID	Sequence
1	ABCDE
2	BADCE
3	BDEAF

Table 4: An example of a sequences database.

Once again, as can be seen in Table 2, the *Infrabel* database differs from the typical database in this setting. However, we can approach the problem in a way similar to our approach in frequent itemset mining. Here, too, we can generate a database consisting of sequences, where each sequence corresponds to the train IDs that were late within a chosen period of time, only this time the train IDs in each sequence are ordered according to the time at which they were late (the actual, rather than the scheduled, time of arrival or departure). Now, instead of only finding which trains were late together, we can also identify the order in which these trains were late. However, as will be seen in section 2.3 better methods are available.

### 2.3 Episodes

One step further from both itemsets and sequences are episodes [5]. An episode is a temporal pattern that can be represented as a *directed acyclic graph*, or *DAG*. In such a graph, each node represents an event (an item, or a symbol), and each directed edge from event  $x$  to event  $y$  implies that  $x$  must take place before  $y$ . Clearly, if such a graph contained cycles, this would be contradictory, and could never occur in a database. Note that both itemsets and sequences can be represented as DAGs. An itemset is simply a DAG with no edges (events can then occur in any order), and a sequence is a DAG where the events are fully ordered (for example, a sequence  $s_1s_2 \cdots s_k$  corresponds to graph  $(s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_k)$ ). However, we can now find more general patterns, such as the one given in Figure 2. The pattern depicted here tells us that  $a$  always occurs before  $b$  and  $c$ , while  $b$  and  $c$  both occur before  $d$ , but the order in which  $b$  and  $c$  occur may vary.

Formally, an event is a couple  $(s_i, t_i)$  consisting of a symbol  $s$  from an *alphabet*  $\Sigma$  and a time stamp  $t$ , where  $t \in \mathbb{N}$ . A sequence  $\mathbf{s}$  is a set of  $n$  such events. We assume that the sequence is ordered, i.e. that  $t_i \leq t_j$  if  $i < j$ . An episode  $G$  is represented by an acyclic

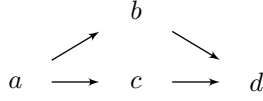


Figure 2: A general episode.

directed graph with labelled nodes, that is,  $G = (V, E, lab)$ , where  $V = v_1 \cdots v_K$  is the set of nodes,  $E$  is the set of directed edges, and  $lab$  is the labelling function  $lab : V \rightarrow \Sigma$ , mapping each node  $v_i$  to its label.

Given a sequence  $s$  and an episode  $G$  we say that  $G$  occurs in  $s$  if there exists an *injective* map  $f$  mapping each node  $v_i$  to a valid index such that the node  $v_i$  in  $G$  and the corresponding sequence element  $(s_{f(v_i)}, t_{f(v_i)})$  have the same label, i.e.  $s_{f(v_i)} = lab(v_i)$ , and that if there is an edge  $(v_i, v_j)$  in  $G$ , then we must have  $f(v_i) < f(v_j)$ . In other words, the parents of  $v_j$  must occur in  $s$  before  $v_j$ .

The database typically consists of one long sequence of events coupled with time stamps, and we want to judge how often an episode occurs within this sequence. We do this by sliding a time window (of chosen length  $t$ ) over the sequence and counting in how many windows the episode occurs. Note that each such window represents a sequence — a subsequence of the original long sequence  $s$ . Given two time stamps,  $t_i$  and  $t_j$ , with  $t_i < t_j$ , we denote  $s[t_i, t_j[$  the subsequence of  $s$  found in the window  $[t_i, t_j[$ , i.e. those events in  $s$  that occur in the time period  $[t_i, t_j[$ . The support of a given episode  $G$  is defined as the number of windows in which  $G$  occurs, or

$$sup(G) = |\{(t_i, t_j) | t_i \in [t_1 - t, t_n], t_j = t_i + t \text{ and } G \text{ occurs in } s[t_i, t_j[ \}|.$$

The *Infrabel* dataset corresponds exactly to this problem setting. For each late train, we have its train ID, and a time stamp at which the lateness was established. Therefore, if we convert the dataset to a sequence consisting of train IDs and time stamps, we can easily apply the above method.

## 2.4 Closed Episodes

Another problem that we have already touched upon is the size of the output. Often, much of the output can be left out, as a lot of patterns can be inferred from a certain smaller set of patterns. We have already mentioned that for each discovered frequent pattern (in our case, episode), we also know that all its subpatterns must be frequent. However, should we leave out all these subepisodes, the only thing we would know about them is that they are frequent, but we would be unable to tell how frequent. If we wish to rank episodes, and we do, we cannot remove any information about the frequency from the output.

Another way to reduce output is to generate only closed patterns [7]. In general, a pattern is considered closed, if it has no superpattern with the same support. This holds for episodes, too.

Formally, we first have to define what we mean by a superepisode. We say that episode  $H$  is a *superepisode* of episode  $G$  if  $V(G) \subseteq V(H)$ ,  $E(G) \subseteq E(H)$  and  $lab_G(v) = lab_H(v)$  for all  $v \in G$ , where  $lab_G$  is the labelling function of  $G$  and  $lab_H$  is the labelling function of  $H$ . We say that  $G$  is a *subepisode* of  $H$ , and denote  $G \subseteq H$ . We say an episode is *closed* if there exists no episode  $H$ , such that  $G \subseteq H$  and  $sup(G) = sup(H)$ .

As an example, consider a sequence of delayed trains  $ABCXYZABC$ . Assume the time stamps to be consecutive. Given a sliding window of size 3 minutes, and a support threshold of 2, we find that the episode  $(A \rightarrow B \rightarrow C)$ , meaning that train A is delayed before B, and B before C, has frequency 2, but so do all its subepisodes of size 3, such as  $(A \rightarrow B, C)$ ,  $(A, B \rightarrow C)$  or  $(A, B, C)$ . These episodes can thus safely be left out of the output, without any loss of information.

Thus, if episode  $(A \rightarrow B)$  is in the output, and episode  $(A, B)$  is not, we can safely conclude that the support of episode  $(A, B)$  is equal to the support of episode  $(A \rightarrow B)$ . Furthermore, we can conclude that if these two trains are both late, then  $A$  will always depart/arrive first. If, however, episode  $(A, B)$  can be found in the output, and neither  $(A \rightarrow B)$  nor  $(B \rightarrow A)$  are frequent, we can conclude that these two trains are often late together, but not necessarily in any particular order. If both  $(A, B)$  and  $(A \rightarrow B)$  are found in the output, then the support of  $(A, B)$  must be higher than the support of  $(A \rightarrow B)$ , and we can conclude that the two trains are often late together, and  $A$  mostly arrives/departs earlier than  $B$ .

### 3 Experiments

In our experiments we have used the latest implementation of an algorithm, CloseEpi, for generating closed episodes, as described in [8].

#### 3.1 Data Description

The Belgian railway network contains approximately 1800 characteristic reference points which are stations, bifurcations, unmanned stops, and country borders. At each of these points, timestamps of passing trains are being recorded. As such, a train trajectory can be reconstructed using these measurements along its route. In practice, however, the true timestamps are not taken at the actual characteristic points, but at enclosing signals.

The timestamp  $t_{a,i}$  for arrival in characteristic point  $i$  is approximated using the times recorded at the origin  $S_{o,i}$ , and the destination  $S_{d,i}$  of the train passing  $i$  as follows:

$$t_{a,i} = t_{S_{o,i}} + \frac{d_{o,i}}{v_i} \quad (1)$$

where  $d_{o,i}$  is the distance from  $S_{o,i}$  to the characteristic point  $i$  and the velocity  $v_i$  is the maximal admitted speed at  $S_{d,i}$ . To calculate the timestamp  $t_{d,i}$  for departure in characteristic point  $i$  we use

$$t_{d,i} = t_{S_{d,i}} - \frac{d_{d,i}}{v_i}. \quad (2)$$

where  $d_{d,i}$  is the distance from  $S_{d,i}$  to the characteristic point  $i$ . Hence, based on these timestamps, delays can be computed by comparing  $t_{a,i}$  and  $t_{d,i}$  to the scheduled arrival and departure times.

#### 3.2 Data Pre-processing

If we look at all data in the *Infrabel* database as one long sequence of late trains coupled with time stamps, we will find patterns consisting of trains that never even cross paths. To

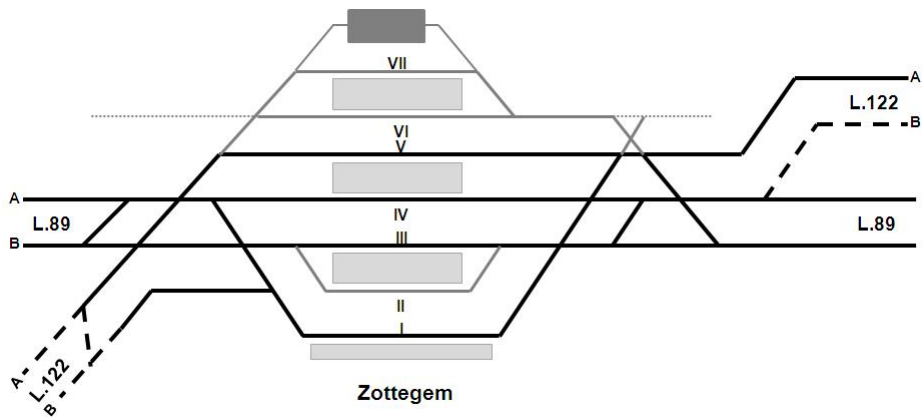


Figure 3: The schematic station layout of Zottegem

avoid this, we generate one long sequence for each spatial reference point. In this way, we find trains that are late at approximately the same time, *in the same place*.

### 3.3 Example

To test the CloseEpi algorithm the decision was made to focus on delays at departure of 3 or more minutes and 6 or more minutes. These choices relate respectively to the blocking time and the official threshold for a train in delay.

For our experiments, we have chosen a window size of 30 minutes (or 1800 seconds). Although the support of a pattern does not immediately translate into the number of days in which it occurs, this can be easily estimated or even simply counted in the original dataset. More specifically, the lower bound of the number of days the pattern occurs is the support of the pattern divided by 1800, rounding the number upwards, and the upper bound is given by the minimum of the upper bounds of all its sub-patterns.

We tested the algorithm on the data collected in Zottegem, a medium-sized station in the south-west of Belgium. Zottegem was chosen as it has an intelligible infrastructure, as shown on Figure 3. The total number of trains leaving Zottegem in the month of January 2010 is 4412. There are 696 trains with a departure delay at Zottegem of more than 3 minutes and 285 trains have a delay at departure which is equal or larger than 6 minutes. The delays are mainly situated during peak hours. Because the number of trains with a delay passing through Zottegem is relatively small, the output can be manually evaluated. The two lines intersecting at Zottegem are: line 89 connecting Denderleeuw with Kortrijk and line 122 connecting Ghent-Sint-Pieters and Geraardsbergen. On the station lay-out of Zottegem (Figure 3) line 89 is situated horizontally on the scheme and line 122 goes diagonally from upper right corner to the lower left corner. This intersection creates potential conflict situations which adds to the station's complexity. Moreover, the station must also handle many connections, which can also cause the transmission of delays.

The trains passing through Zottegem are categorized as local trains (numbered as the 16 series and the 18 series), a cityrail (22 series) going to and coming from Brussels, an



intercity connection (23 series) with fewer stops than a cityrail or local train, and the peak hour trains (89 series).

The output of the *ClosEpi* algorithm is a rough text file of closed episodes with a support larger than the predefined threshold. An episode is represented by a graph of size  $(n, k)$  where  $n$  is the number of nodes and  $k$  the number of edges. Note that a graph of size  $(n,0)$  is an itemset. We aimed to discover the top 20 episodes of size 1 and 2, and the top 5 episodes of size 3 and 4, so we varied the support threshold accordingly. In Tables 5–8 some of the episodes which were detected in the top 20 most frequently appearing patterns are listed. For example, the local train no. 1867 from Zottegem to Kortrijk is discovered as being 3 or more minutes late at departure on 15 days, and 6 or more minutes on 8 days in the month of January 2010.

Train ID	Route	Support	
		Delay $\geq 3'$	Delay $\geq 6'$
1867	Zottegem – Kortrijk	27000	14400
8904	Schaarbeek – Oudenaarde	28800	18000
8905	Schaarbeek – Kortrijk	27000	14400
8963	Ghent-Sint-Pieters – Geraardsbergen	25200	12600

Table 5: Episodes of size  $(1,0)$  representing the delay at departure in station Zottegem during evening peak hour (16h – 19h) for January 2010

A paired pattern can be a graph of size  $(2,0)$ , meaning the trains appear together but without a specific order, or of size  $(2,1)$ , where there is an order of appearance for the two trains. For example, train no. 1867 and train no. 8904 appear together as being 3 or more minutes late on at least 9 days and at most on 15 days in January 2010. The pattern *trains no. 8904 and 1867 have a delay at departure of 3 or more minutes, and train no. 8904 leaves before 1867* appears on at least 8 days and at most 15 days in January 2010.

Among the top 20 patterns with pairs of trains (Table 6), it can be noticed that the pattern  $1867 \rightarrow 8963$  was only discovered in the search for 6 or more minutes delay at departure. This means that the pattern will also appear while searching for 3 or more minutes of delay at departure but the support of this pattern is not high enough to appear in the top 20 output.

The patterns which include lots of information are to be found in the output of episodes of size 3 and up, as can be seen in Tables 7 and 8. But to discover the episodes of sizes  $(3,k)$  and  $(4,k)$  the threshold had to be lowered to 5500 which corresponds to a minimal appearance of the pattern on 4 days. The question remains if this really is an interesting pattern.

In the example the peak-hour train no. 8904 often departs from the station with a delay of 3 minutes with a support of 28800 and a support of 18000 for a delay of 6 minutes (see Table 5). In real-time the peak-hour train no. 8905 follows train no. 8904 on the same trajectory, 4 minutes later. This can also be detected by looking at the occupation of the tracks in Figure 4. It is, therefore, obvious that whenever no. 8904 has a delay, the 8905 will also have a delay. Trains nos. 1867 and 8963 both offer a connection to nos. 8904 and 8905. So, if train 8904 has a delay, it will be transmitted to trains 1867 and 8963. This is also stated in Table 8, which shows an episode of size four, found by the *ClosEpi* algorithm, where trains no. 8904, 1867, 8905, and 8963 are all late at departure, and 8904 departs before the other three trains.

Train id	Episode		Support	
	Relation	Train id	Delay $\geq 3'$	Delay $\geq 6'$
1867		8904	15079	-
1867	$\leftarrow$	8904	13557	-
1867		8905	18341	-
1867	$\leftarrow$	8905	12995	-
1867		8963	18828	8888
1867	$\rightarrow$	8963	-	5327
8904	$\rightarrow$	8905	18608	9506
8904		8963	18410	10391
8904	$\rightarrow$	8963	16838	8819
8905		8963	20580	10608
8905	$\rightarrow$	8963	13325	5078
8905	$\leftarrow$	8963	-	5530

Table 6: Episodes of size  $(2,k)$  representing the delay at departure in station Zottegem during evening peak hour (16h – 19h) for January 2010

Train id	Episode		Support	
	Relation	Train id	Delay $\geq 3'$	Delay $\geq 6'$
8904	$\rightarrow$	8905	14358	7510
	$\searrow$	8963		
8904		8905	11069	-
		$\downarrow$ 8963		
8904	$\rightarrow$	8905	15804	8956
		8963		

Table 7: Episodes of size  $(3,k)$  representing the delay at departure in station Zottegem during evening peak hour (16h – 19h) for January 2010

Looking at the data for February 2010 (not included here) the pattern described in Table 8 is discovered for 3 or more minutes of delay with a support of 12126. In the cases of 6 or more minutes delay the pattern is discovered under the stronger form  $8904 \rightarrow 8905 \rightarrow 8963 \rightarrow 1867$  with a support of 4604, meaning that if these trains have a delay at departure of 6 or more minutes, peak hour train no. 8904 departs before no. 8905, which leaves before no. 8963, which in turn leaves before the local train no. 1867.

## 4 Conclusion and Outlook

We have studied the possibility of applying state-of-the-art pattern mining techniques to discover knock-on train delays in the Belgian railway network using a database of *Infrabel* containing the times of trains passing through characteristic points in the network. Our experiments show that the ClosEpi algorithm is useful for detecting interesting patterns

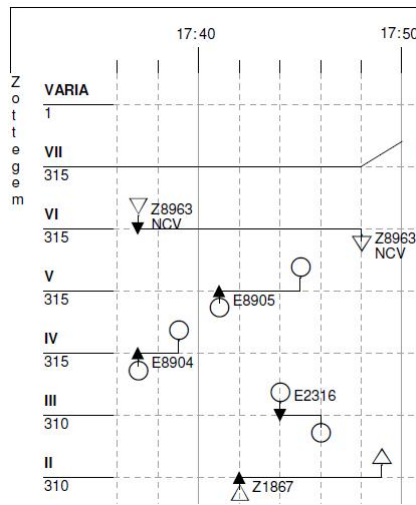


Figure 4: Occupation of the tracks during evening peak hour at Zottegem

Train id	Episode Relation	Train id	Support	
			Delay $\geq 3'$	Delay $\geq 6'$
8904	$\nearrow$	1867	10024	6104
	$\rightarrow$	8905		
	$\searrow$	8963		

Table 8: Episode of size  $(4,k)$  representing the delay at departure in station Zottegem during evening peak hour (16h – 19h) for January 2010

in the *Infrabel* data. There are still many opportunities for improvement, however. For example, a good visualization of the discovered patterns would certainly help in identifying the most interesting patterns in the data more easily. Also, next to the the support measure, other interestingness measures could also be considered. Selecting patterns solely based on the support measure still hides a lot of potentially interesting patterns, which could be found using other criteria.

In order to avoid finding too many patterns consisting of trains that never even cross paths, we only considered trains passing in a single spatial reference point. As a result, we can not discover knock-on delays over the whole network. In order to tackle this problem, the notion of a pattern needs to be redefined, but also the interestingness measures, or other data pre-processing techniques need to be investigated.

## Acknowledgments

The authors would like to thank Nikolaj Tatti for providing us with his implementation of the ClosEpi algorithm.

## References

- [1] Agrawal, R., and Srikant, R., “Mining sequential patterns”, *Proc. of the 11th International Conference on Data Engineering*, vol. 0, 3–14, 1995.
- [2] Flier, H., Gelashivili, R., Graffagnino, T., and Nunkesser, M., “Mining Railway Delay Dependencies in Large-Scale Real-World Delay Data”, *Robust and Online Large-Scale Optimization, Lecture Notes in Computer Science*, vol. 5868, 354–36, 2009 .
- [3] Goethals, B., “Frequent Set Mining”, *The Data Mining and Knowledge Discovery Handbook*, chap. 17, 377–397, Springer, 2005.
- [4] Gunopulis, D., Khardon, R., Labbuka, H., Saluja, S., Toivonen, H., and Sharma, R.S., “Discovering all most specific sentences”, *ACM Transactions on Database Systems*, vol. 28(2), pp. 140–174, 2003.
- [5] Mannila, H., Toivonen, H., and Verkamo, A.I., “Discovery of Frequent Episodes in Event Sequences”, *Data Mining and Knowledge Discovery*, vol. 1, 259–298, 1997.
- [6] Mirabadi, A. and Sharifian, S., “Application of Association rules in Iranian Railways (RAI) accident data analysis”, *Safety Science*, vol. 48, 1427–1435, 2010.
- [7] Tan, P.-N., Steinbach, M., and Kumar, V., *Introduction to Data Mining*, Pearson Addison Wesley, 2006.
- [8] Tatti, N., and Cule, B., “Mining Closed Strict Episodes”, *Proc. of the IEEE International Conference on Data Mining*, 2010.
- [9] Wang, J. T.-L., Chirn, G.-W., Marr, T.G., Shapiro, B., Shasha, D., and Zhang, K., “Combinatorial pattern discovery for scientific data: some preliminary results”, *ACM SIGMOD Record*, vol. 23, 115–125, 1994.