Mining Train Delays

Boris Cule[1], Bart Goethals[1], Sven Tassenoy[2], and Sabine Verboven[2,1]

[1] University of Antwerp, Department of Mathematics and Computer Science, Middelheimlaan 1, 2020 Antwerp, Belgium

[2] INFRABEL - Network, Department of Innovation, Barastraat 110, 1070 Brussels, Belgium

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, we study, adapt and apply the state-of-the-art techniques for mining frequent episodes 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. In a related work, Flier et al. [2] try to discover dependencies in the underlying causes of the delays, but we search for patterns in the delays themselves. Mirabadi and Sharafian [6] use association mining to analyse the causes in accident data sets, while we consider frequent pattern mining methods. More specifically, we analyse a dataset consisting of a sequence of delayed trains using a recently developed frequent episode mining technique [8].

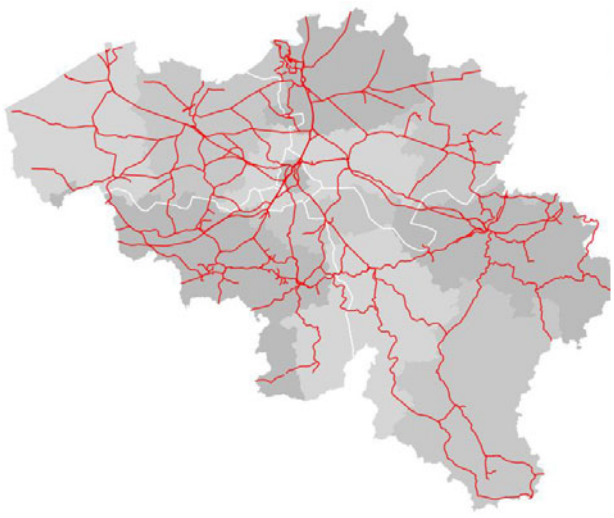

Fig. 1. The Belgian Railway Network

An episode in a sequence is usually considered to be a set of events that reoccurs in the sequence within a window of specified length [5]. The order in which the events occur is also considered important. The order restrictions of an episode are typically described by a directed acyclic graph.

We use a database provided by *Infrabel*, the Belgian railway maintenance company, containing the times of trains passing through characteristic points in the railway network. 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 precisely, we try to find all delays that frequently occur within a certain time window, counted over several days or months of data, and interdependencies among them. For example, we consider episodes 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 episodes, however, is intractable in practice as the number of such potentially interesting episodes grows exponentially with the number of trains [4]. Fortunately, efficient episode mining techniques have recently been developed, making the discovery of such episodes possible. A remaining challenge is still to distinguish the interesting episodes from the irrelevant ones. Typically, a frequent episode mining algorithm will find an enormous amount of episodes 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 give a detailed description of the dataset, and in Section 3, we discuss various patterns and describe the method we used. In Section 4, we discuss how we preprocessed the collected data, give a concrete case study at one particular geographical location, and report on preliminary experiments showing promising results. Finally, We conclude the paper with suggestions for future work in Section 5.

2 The Dataset

The Belgian railway network contains approximately 1800 characteristic geographic reference points — 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:

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

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

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

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.

Table 1 gives a small fictional example of the relevant part of the provided data (irrelevant columns were omitted). We now describe this dataset column per column. The first two columns are self-explanatory and contain the date and the Train ID respectively. The third column, PTCar, contains the geographical measurement points (the characteristic points referred to above), and is followed by arrival and departure times of the train in that point (computed using the approximation method described above). Finally, the sixth and seventh column contain the arrival and departure delay respectively, computed by comparing the actual and scheduled arrival and departure times. The actual dataset we worked with contained the complete information about all departures and arrivals of all trains in characteristic points in Belgium in January 2010.

3 Frequent Episodes

Before we present our chosen method of mining frequent episodes in the *Infrabel* dataset, we start off with a short discussion of why we opted not to use simpler patterns, such as frequent itemsets or sequences.

Table 1. An excerpt from a fictional train delay database

Date	Train ID	PTCar	Arrival Time	Departure Time	Arr. Delay	Dep. Delay
...						
15/02/2010	100	1255	-	06:07:23	-	23
15/02/2010	100	941	06:14:18	06:17:57	18	117
15/02/2010	100	169	06:37:25	06:38:30	205	210
...						
15/02/2010	100	445	07:28:03	-	183	-
15/02/2010	123	114	-	06:11:58	-	-2
15/02/2010	123	621	06:24:22	06:26:10	82	70
15/02/2010	123	169	06:31:51	06:33:49	231	229
...						
15/02/2010	123	541	07:10:37	-	97	-
...						

3.1 Itemsets

The simplest possible pattern are itemsets [3]. 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'. In this setting, the data is usually organised in a transaction database, and we consider an event (or a set of events) frequent if the number of transactions in which it can be found (or its *support*) is greater than or equal to a user-defined support threshold.

In order to mine frequent itemsets in the traditional way, the *Infrabel* data would need to be transformed. In principle, a transaction database could be created, such that each transaction would consist of train IDs of trains that were late within a given period of time. Each transaction would represent one such period. Mining frequent itemsets would then result in obtaining sets of train IDs that are often late 'together'.

Clearly, though, the frequent itemset method is neither intuitive nor suitable for tackling our problem. It would require a lot of preprocessing work in order to transform the data into the necessary format, and the resulting database would contain many empty or identical transactions. Assume we look at time windows of five minutes. As our time stamps are expressed in seconds, each second would represent a starting point of such a window. It is easy to see that many consecutive windows would contain exactly the same train IDs. Most importantly, though, frequent itemsets are simply too limited to extract all the useful information from the *Infrabel* database, as they contain no temporal information whatsoever.

3.2 Sequences

The *Infrabel* 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, when searching for sequential patterns, the database consists of a set 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].

Again, in order to mine frequent sequences in the traditional way, the *Infrabel* data would need to be transformed. We could approach this in a way similar to the one described above for itemsets — namely, we could create a sequence for each time window of a certain length. Each such sequence would then contain the train IDs of trains that were late within that window, only this time the train IDs in each sequence would be 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.

This method, though clearly superior to the frequent itemset technique, still suffers from the same problems in terms of preprocessing and redundancy of the resulting dataset, and still does not allow us to generate patterns that we wish to find. In this setting, we are only capable of discovering patterns with a total order. If, for example, we have a situation in which trains A and B, when both delayed, cause train C to also be delayed, but the order in which A and B come into the station does not matter, this method will fail to discover this as a pattern (assuming that the supports of sequences ABC and BAC are below the desired threshold).

3.3 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. 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 $s_1 s_2 \cdots s_k$ corresponds to graph $(s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_k)$). However, episodes allow us to find more general patterns, such as the one given in Figure 2. The pattern depicted here tells us that a 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* Σ and a time stamp t, where $t \in \mathbb{N}$. A sequence $\mathbf{s}$ is an ordered set of events, i.e., $t_i \leq t_j$ if $i < j$. An episode G is represented by a directed acyclic 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.

$$\begin{tikzcd}
 & b \arrow[dr] & \\
a \arrow[ur] \arrow[r] & c \arrow[r] & d
\end{tikzcd}$$

Fig. 2. A general episode

Given a sequence $\mathbf{s}$ and an episode G we say that G occurs in $\mathbf{s}$ 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 $\mathbf{s}$ 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 $\mathbf{s}$. Given two time stamps, t_i and t_j, with $t_i < t_j$, we denote $\mathbf{s}[t_i, t_j[$ the subsequence of $\mathbf{s}$ found in the window $[t_i, t_j[$, i.e., those events in $\mathbf{s}$ 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

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

The *Infrabel* dataset corresponds almost 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.

3.4 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. It is inherent in the nature of the support measure that for each discovered frequent pattern, 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 the 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 $ABCXYZABC$. For simplicity, assume the time stamps to be consecutive minutes. Given a sliding window size of 3 minutes, and a support threshold of 2, we find that 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 of 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, and $(B \rightarrow A)$ is not, 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.

In our experiments, we have used the latest implementation of the ClosEpi algorithm for generating closed episodes [8]. In this work, the authors actually mine only strict episodes, whereby they insist that an episode containing two nodes with the same label must have an edge between them, but this restriction is not relevant here, as we never find such episodes. For an episode to contain two nodes with the same label, it would need to contain the same train ID twice, and, in our dataset, no train ID is used twice on the same day.

4 Experiments

In this section we first describe the preprocessing steps that we had to take in order to transform the provided database into a valid input for the ClosEpi algorithm. We then present a case study consisting of a detailed analysis of patterns found in one particular train station in Belgium.

4.1 Data Preprocessing

The preprocessing of the *Infrabel* dataset consisted of two main steps. First, we noted that 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 avoid this, we split the database into smaller datasets, each of which contained only information about delays in one particular spatial reference point. Further, to avoid mixing apples and pears, we split each spatial point into two datasets — one containing departure information, and the other arrival data. In this way, we could find episodes containing trains that are late at, for example, arrival, at approximately the same time, *in the same place.*

Second, we needed to eliminate unnecessary columns and get the data into the desired input format for the ClosEpi algorithm. In other words, our dataset needed to consist only of a time stamp and a train ID. Clearly, we first needed to discard all rows from the database that contained trains that were not delayed at all (depending on the application, these were the trains that were delayed less than either 3 or 6 minutes). Once this was done, the actual delay became irrelevant, and could also be discarded. Obviously, in the arrival dataset, all information about departures could also be removed, and vice versa, and as the spatial point was clear from the dataset we used, we could also remove it from the actual content of the dataset. Finally, we merged the date and time columns to create one single time stamp, and ordered each dataset on this new merged column. Starting from the fictional example given in Table 1, assuming we consider trains with a delay of 3 or more minutes at arrival as delayed, for characteristic spatial point 169 we would obtain the input dataset given in Table 2.

Table 2. The data from Table 1 after preprocessing for arrivals at spatial point 169

Time stamp	Train ID
...	
06:31:51 15/02/2010	123
...	
06:37:25 15/02/2010	100
...	

4.2 Example

The Belgian railways recognise two types of delays, those of 3 or more minutes (the blocking time) and those of 6 or more minutes (the official threshold for a train to be considered delayed), so we ran experiments for both of these parameters. We have chosen a window size of 30 minutes (or 1800 seconds) — if two trains are delayed more than half an hour apart, they can hardly form a pattern.

We tested the algorithm on 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 in Figure 3. The total number of trains leaving Zottegem in the month of January 2010 was 4412. There were 696 trains with a departure delay at Zottegem of 3 minutes or more, of which 285 had a delay at departure larger than or equal to 6 minutes. The delays are mainly situated during peak hours. As the number of delayed trains passing through Zottegem is relatively small, the output can be manually evaluated. As can be seen in Figure 3, the two railway lines intersecting at Zottegem are line 89, connecting Denderleeuw with Kortrijk, and line 122, connecting Ghent-Sint-Pieters and Geraardsbergen. Line 89 is situated horizontally on the scheme and line 122 goes diagonally from the 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. For example, Figure 4 shows the occupation of the tracks in the station in the peak period between 17:35 and 17:50 on a typical weekday. We will analyse some of the patterns discovered in this period later in this section.

The trains passing through Zottegem are categorized as local trains (numbered as the 16-series and the 18-series), cityrail (22-series) going to and coming from Brussels, intercity connections (23-series) with fewer stops than a cityrail or a local train, and the peak hour trains (89-series).

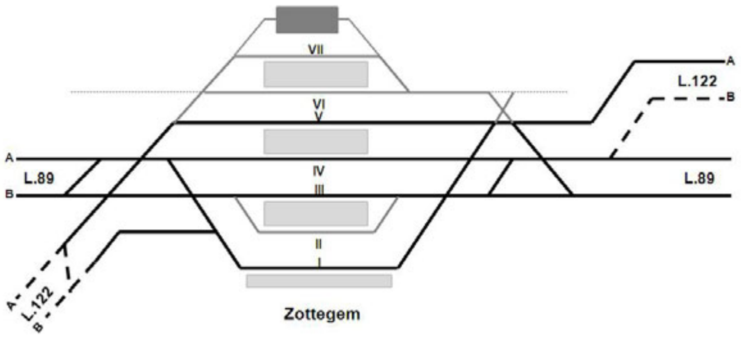

Fig. 3. The schematic station layout of Zottegem

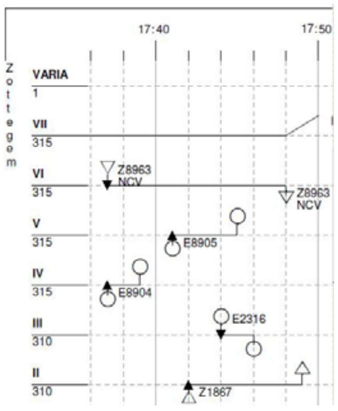

Fig. 4. Occupation of the tracks during evening peak hour at Zottegem

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. All experiments lasted less than a minute. In Tables 3–6 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.

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 in 15079 windows in January 2010. The pattern *trains no. 8904 and 1867 have a delay at departure of 3 or*

Table 3. Episodes of size (1,0) representing the delay at departure in station Zottegem during evening peak hour (16h – 19h) for January 2010

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

more minutes, and train no. 8904 leaves before 1867 appears in 13557 such windows. Among the top 20 patterns with pairs of trains (Table 4), it can be noticed that the pattern 1867 → 8963 was only discovered in the search for 6 or more minutes delay at departure. The pattern also appeared while searching for delays of 3 or more minutes, but its support was not high enough to appear in the top 20.

Table 4. Episodes of size (2,k) representing the delay at departure in station Zottegem during evening peak hour (16h – 19h) for January 2010

Episode			Support	
Train ID	Relation	Train ID	Delay ≥ 3'	Delay ≥ 6'
1867		8904	15079	-
1867	←	8904	13557	-
1867		8905	18341	-
1867	←	8905	12995	-
1867		8963	18828	8888
1867	→	8963	-	5327
8904	→	8905	18608	9506
8904		8963	18410	10391
8904	→	8963	16838	8819
8905		8963	20580	10608
8905	→	8963	13325	5078
8905	←	8963	-	5530

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 5 and 6. Some episodes shown here occurred quite often, but to discover the desired number of 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 these really are interesting patterns.

Let us now return to the peak trains shown in Figure 4. 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 3). In real-time the peak-hour train no. 8905 follows train no. 8904 on the

Table 5. Episodes of size (3,k) representing the delay at departure in station Zottegem during evening peak hour (16h – 19h) for January 2010

Episode			Support	
Train ID	**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 6. Episode of size (4,k) representing the delay at departure in station Zottegem during evening peak hour (16h – 19h) for January 2010

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

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. It is therefore not surprising that this episode can be found in the output given in Table 4. Trains no. 1867 and no. 8963 both offer connections to trains no. 8904 and no. 8905. So, if train 8904 has a delay, it will be transmitted to trains 1867 and 8963. This is also stated in Table 6, 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.

5 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 an *Infrabel* database 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 in the *Infrabel* data. There are still many opportunities for improvement, however. For example, a good visualization of the discovered episodes would certainly help in identifying the most interesting patterns in the data more easily. 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. Interestingness measures other than support, or other data preprocessing techniques, could also 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., Srikant, R.: Mining sequential patterns. In: Proc. of the 11th International Conference on Data Engineering, pp. 3–14 (1995)
2. Flier, H., Gelashvili, R., Graffagnino, T., Nunkesser, M.: Mining Railway Delay Dependencies in Large-Scale Real-World Delay Data. In: Ahuja, R.K., Möhring, R.H., Zaroliagis, C.D. (eds.) Robust and Online Large-Scale Optimization. LNCS, vol. 5868, pp. 354–368. Springer, Heidelberg (2009)
3. Goethals, B.: Frequent Set Mining. In: The Data Mining and Knowledge Discovery Handbook, ch. 17, pp. 377–397. Springer, Heidelberg (2005)
4. Gunopulis, D., Khardon, R., Labbuka, H., Saluja, S., Toivonen, H., Sharma, R.S.: Discovering all most specific sentences. ACM Transactions on Database Systems 28(2), 140–174 (2003)
5. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery 1, 259–298 (1997)
6. Mirabadi, A., Sharifian, S.: Application of Association rules in Iranian Railways (RAI) accident data analysis. Safety Science 48, 1427–1435 (2010)
7. Tan, P.-N., Steinbach, M., Kumar, V.: Introduction to Data Mining. Pearson Addison Wesley (2006)
8. Tatti, N., Cule, B.: Mining Closed Strict Episodes. In: Proc. of the IEEE International Conference on Data Mining, pp. 501–510 (2010)
9. Wang, J.T.-L., Chirn, G.-W., Marr, T.G., Shapiro, B., Shasha, D., Zhang, K.: Combinatorial pattern discovery for scientific data: some preliminary results. ACM SIGMOD Record 23, 115–125 (1994)