Process mining approach for recovery of realized train paths and route conflict identification

TRAIL Research School, Delft, October 2012

Authors

Ir. Pavle Kecman, Dr. Rob M.P. Goverde

Faculty of Civil Engineering and Geosciences, Department of Transport and Planning, Delft University of Technology, The Netherlands

Contents

Abstract

Data records from train describer systems are a valuable source of information for analyzing railway operations performance and assessing railway timetable quality. This paper presents a tool based on process mining event data records from the Dutch train describer system TROTS. The underlying algorithms automatically identify route conflicts with conflicting trains, determine accurate arrival and departure times/delays at stations, and reconstruct the train paths on track section and blocking time level. Graphical user interface and visualizations of the time-distance diagrams and blocking time diagrams support and simplify the analysis of running times, dwell times, incidents, track obstructions, disruptions, and structural errors in the timetable design. The case study of a one day of traffic on a busy railway corridor in the Netherlands is presented, as well as the examples to describe the graphical user interface.

Keywords

Train describers, realisation data, process mining, route conflicts

1 Introduction

Improving the performance of railway infrastructure and train services is the core business of railway infrastructure managers and railway undertakings in Western Europe. Train delays decrease capacity, punctuality, reliability and safety, and should be prevented as much as possible.

Daamen et al. (2008) preseted a case study on a busy corridor in the Netherlands, which showed that 55% of arrival delays exceeding 3 minutes are caused by route conflicts. A route conflict occurs when a train (hindered train) movement is restricted by a stop signal because the block section protected by the signal is occupied by another train (hindering train). Registered delays at stations can not be with certainty attributed to route conflicts, therefore, it is difficult to identify and analyze them. Typically, train delays at stations are monitored and registered on-line using train detection, train describers, and timetable databases, but the accuracy is insufficient for process improvements. Railway operations thus require feedback of operations data to improve planning and control. Accurate data on the level of track sections and signal blocks are required to gain a better understanding of the realized train paths and conflicts between them.

Train describer records are a main source of infrastructure event data such as occupations and releases of track sections and aspect changes of signals. These infrastructure events can be matched to train number events that are also stored in these files to recover the realized train paths on track section level. Moreover, the realized blocking time diagrams can be derived by adding a process model of the signaling logic.

In an earlier work, Daamen et al. (2008) developed algorithms for automatic route conflict identification based on data records of the Dutch train describer system TNV, which were implemented in the tool TNV-conflict. Goverde & Meng (2012) developed the tool TNV-Statistics for a detailed statistical analysis of train realization data based on the output files of TNV-Conflict.

The TNV system was recently replaced by the new train describer system TROTS which contains an essential new approach to train number steps, and this came with a new format for the log files. In particular, train number steps are no longer given with respect to a route block to a next signal, but at section level. This means that a train number step no longer predicts to which signal the train is heading, as was customary with TNV, and therefore we cannot just look ahead at the signal aspect of the signal at the end of a block to identify a conflict. Therefore, the algorithms described in Daamen et al. (2008) had to be modified in a way described in the present paper.

Other approaches to train delay data mining include Conte (2007) and Flier et al. (2009) for determining systematic dependencies between delays in Germany and Switzerland, respectively, and Cule et al. (2011) for identifying frequent delay patterns in Belgium.

There are several reasons for increasing interest in traffic realization data analysis. First, infrastructure capacity is utilized extensively in Western European countries. In such conditions, when capacity consumption is close to the level of congestion and saturation (UIC, 2004), delays propagate easily through the network and it is therefore necessary to determine the optimal values and allocation of time reserves in order to increase robustness and resilience of the system. In that context, in the process of timetable construction, feedback in form of performance analysis is essential.

Second, adoption and implementation of EC Directive 2001/14/EC (EC, 2001), implies strictly regulated, transparent relations between all participants in the railway market. Punctuality norms and schedule violation penalties are imposed on infrastructure managers and train operating companies. Therefore, deriving accurate values of delays and partitioning them in primary and secondary delays is in interest of all parties.

The third reason has a more scientific importance. Namely, mathematical and simulation models of railway traffic use stochastic distribution of process times which reflect the variations caused by e.g. driving behavior, passenger volumes, weather conditions, etc. It is however an important feature of the models themselves to capture the interactions of trains and the resulting conflicts and knock-on delays. Consequently, partitioning realized process times data to hindered and unhindered trains is of great importance (Daamen et al., 2008).

In this paper, a process mining approach is implemented on the log files of the Dutch train describer system TROTS. The resulting railway operations performance analysis tool recovers and visualizes the realized train paths, blocking times, and route conflicts, and thus provides essential information for analyzing railway operations, that can be used for fine-tuning the railway timetable and operational processes. The tool supports both tabular output for statistical analysis, as discussed in Goverde & Meng (2012), and visualizations of the realized time-distance and blocking time diagrams with highlighted route conflicts. Moreover, several improvements have been implemented such as interpolating blocking times over non-logged signals so that route conflict identification is applicable over entire corridors, including 'dark territories' with aggregated track sections. The output of the new tool has the same format as that of TNV-Conflict by which TNV-Statistics is still applicable.

The remainder of the paper is structured as follows. Section 2 defines terminology and route conflicts in the context of blocking time theory, Section 3 explains the Dutch train describer system, whereas Section 4 formalizes the blocking time theory as a process model applied in the tool, and explains the process mining algorithm and subroutines. A case study with a description of the GUI, is given in Section 5. Section 6 gives a brief summary and presents further application of train describer data in the framework of an on-going research about model-predictive railway traffic management (Kecman et al., 2011).

2 Blocking time theory

Blocking time theory (Hansen & Pachl, 2008) is a concept of diagramming traffic, that captures all principles of train separation in fixed block sections (including interlocked station routes). Each block section may be occupied exclusively by one train at the time. The blocking time represents a time interval during which a block section is reserved for a specific train movement and therefore blocked for all other trains. In order to avoid breaking before the signal that protects the block, movement authority should be issued before the train reaches the breaking distance (previous block signal) in approach to the signal. Figure 1 shows the structure of blocking time of a train without a scheduled stop on a previous block (otherwise the blocking time does not include approach time).

The blocking time of a section consists of: the sight and reaction time before the approach signal (taken fixed as 12 s), the approach time (running time from approach

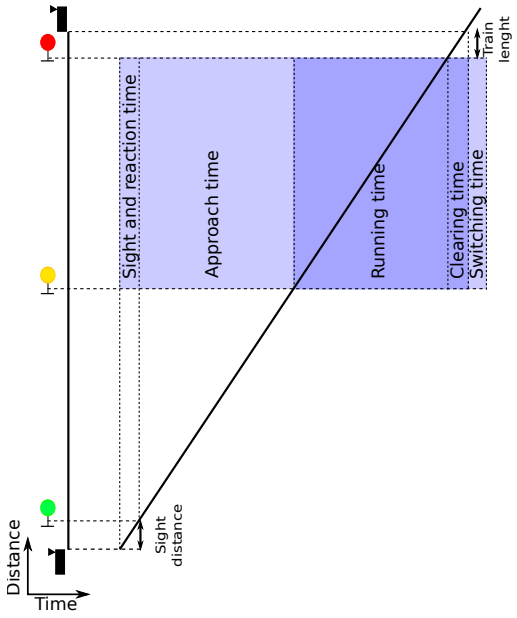

Figure 1: Blocking time of a running train

signal to the block signal), running time in the block, clearing time (time between moments when the first and the last axle of the train leave the block), switching time needed for signaling system to react (taken fixed as 2 s).

Conflict-free train run is ensured if the preceding train has cleared the block and switching time has passed by the time when the next train reaches the sight distance of the approach signal. In the context of blocking time theory, route conflicts can be defined as an overlap between blocking times (Figure 2).

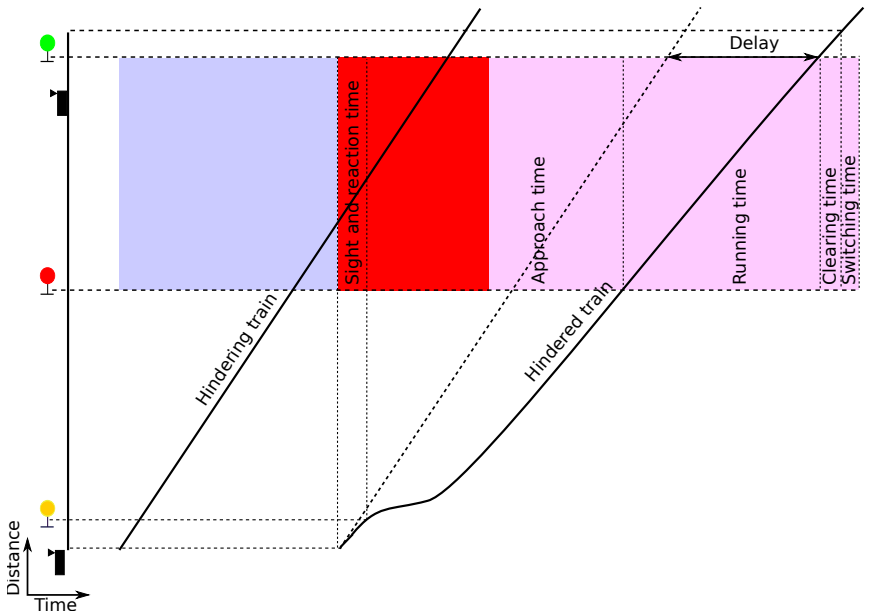

Figure 2: Route conflict

Figure 2 shows the blocking times and signal aspects that describe a route conflict that

occurs when the hindered train arrives at the sight distance of the approach signal at the moment when the hindering train is still running over the block. Conflict is depicted with an overlap in blocking times (indicated by red color). As a result of the route conflict, hindered train must reduce the speed after passing the approach signal, thus increasing the approach time.

3 Train describer systems

Train describer systems keep track of train positions in discrete steps over its route, based on train numbers and messages received from elements of the signaling and interlocking systems (sections, switches and signals) (Exer, 1995). One of the tasks of train describers is logging the generated train number messages and the incoming infrastructure element messages, resulting in chronologically ordered lists of infrastructure and train number messages.

3.1 The Dutch train describer TROTS

In the Dutch train describer system TROTS, the train steps are recorded on the level of track sections (a block section consists of one or more track sections), with both a message when a new track section is occupied by a train and when a track section is released by a train. Hence, train number step messages are coupled to track section messages.

The Dutch railway network is divided into multiple TROTS areas. Each area comprises one or more major station areas with complex topologies and 30 – 40 km of surrounding railway infrastructure. In order to reconstruct the train traffic over multiple TROTS areas it is necessary to merge the corresponding log files. TROTS log files are archived per day and area in large files of ASCII format of approximately 75 MB.

Infrastructure messages contain the following information: time stamp, event code, element type (section, signal, point), element name, and new state ('occupied'/'released', 'stop'/'go', 'left'/'right'). The train number step messages contain amongst others a time stamp, event code, train number, and a sequence of all occupied track sections. Each successive train number step message contains either a new occupied track section at the front or a released track section at the rear. The event code of a train number step corresponds to a section message with the same event code. This coding is used to match a message about a section occupation or release with a message of a train number step.

3.2 Shortcomings in TROTS log files

There are several issues in the TROTS log files that represent a potential source of inaccuracy and complicate performance analysis. The system architecture (ProRail, 2008) reveals that infrastructure messages and train number step messages are generated by different components of the system which sometimes results in a significant difference between the time stamps of the corresponding messages. Experiments show delays of up to 7 seconds of the train number step messages. In order to avoid possible inconsistencies, the developed tool does not use the time stamps of the train number step messages but only the ones of the corresponding infrastructure element messages.

Furthermore, infrastructure messages of a signal aspect change to stop cannot be coupled directly to any train number step or section occupation message. In order to overcome this, an additional input file is created in the form of a list of all signals together with the first section they protect by data mining the files in a preprocessing step. We use this input in the main algorithm to identify the train number that caused the signal aspect change via the corresponding section that got occupied.

Other sources of inaccuracy are the automatic block signals on the open track which are not logged. Without intermediate logged signals, an open track between two stations looks as one block from the exit signal at the station of departure to the home signal at the station of arrival, and headway conflicts can not be identified. Moreover, open tracks may contain aggregated track sections which are occupied and released as one. We therefore defined an additional input file containing a list of automatic block signals on the open tracks together with the corresponding (aggregated) sections listed by individual sections and their lengths. If a non-logged signal is at the boundary of two (aggregated) sections then the occupation of the following (aggregated) section is used as stop aspect event time. Otherwise, three-aspect two-block signaling logic is simulated to estimate aspect changes of non-logged signals on aggregated sections. In this case, the event time of the stop aspect change is estimated by the occupation time of the corresponding protected section, which is derived as a fraction of the running time of the train over the aggregated section proportional to the ratio of the distance to the signal and the length of the aggregated section.

4 Process mining algorithm

Process mining is a method of analyzing and extracting information about processes from event data logs using the process model (Van der Aalst, 2011). Blocking time theory provides the logic for building the process model from the log file. Signal passages are events that initiate processes such as blocking a part of the infrastructure and running over a block. Each complete train run can thus be represented as a graph built on-line by sweeping through the file. Moreover, route conflicts can be identified simultaneously by determining time difference between relevant events and verifying if the train separation principles are respected.

Due to large size of TROTS log files, it is necessary to build an algorithm that sweeps through the file and visits every line only once, thus avoiding long computation times and making the tool applicable in a real-time environment. An object-oriented approach is used to store the relevant data from the log files in infrastructure and train number objects which enables the algorithm to revisit the objects, and use and update the information therein (Daamen et al., 2008).

We denote by O the set of all objects, and by $S \in O$, $C \in O$ and $T \in O$ the sets of section, signal and train objects, respectively. An infrastructure message log line l from a TROTS file can be represented by ($type_l, code_l, t_l$,$name_l$,e_l) where $type_l \in$ {'section','signal'} indicating the type of infrastructure element, $code_l$ is the unique message code, t_l is a time stamp of the message, $name_l$ is the the name of an infrastructure element (section or signal) name and e_l is an event ('occupied'/'released' for section objects, 'stop'/'go' for signal objects). Every infrastructure element name has the structure σ_id, where σ is the name of the station area it is comprized by.

A train step message line is summarized by $(type_l, code_l, name_l)$ where $type_l =$ 'train', $code$ is the unique message code and $name_l$ is the train number. We define a mapping $\Omega : name_l \rightarrow o_k, o_k \in O$ that maps the signal, section and train name to their corresponding objects.

The unique message code of a train step message is identical to the code of an earlier message, which reports state change of the relevant section. As explained in Section 3.2, signal messages cannot be directly coupled to train messages. Input file `Infrastructure` contains the list of pairs $(signal, protected(signal))$ for each signal, associated with the first section of the protected block. By $signal(section)$ we denote the signal that protects the section in the direction of the running train. When a running train passes a signal, aspect is changed to 'stop'. Message reporting that event is followed by the message that reports occupation of the protected section by the running train. We use the section occupation message to identify the train that passed the signal causing aspect change to 'stop'.

Furthermore, we define an integrated message (t_k, r_k, e_k, n_l) that carries the complete information about the state change of infrastructure object (resource) $r_k \in \{S \cup C\}$ caused by the running train $n_l \in T$. The algorithm sweeps through the original TROTS messages and creates integrated messages before further processing.

Section objects $r_k \in S$, and signal objects $r_k \in C$, are attributed by a chronologically sorted list of activities. Each activity on a section is described by the train number, occupation time and release time. Signal object activities are defined by 'stop' aspect time and 'go' aspect time. Train objects $n_k \in T$ are attributed by the chronologically sorted lists of signals, sections and blocks on the train route. Keeping these lists as attributes for each object is useful for creating tabular and visual output for recovery of train paths. However, for route conflict identification we are interested in the state changes of infrastructure objects caused by a running train, and possibly its immediate predecessor. By time(r_k, i) we denote a moment when event i occurred on infrastructure object r_k. Table 1 defines the target events.

Table 1: Target events for infrastructure objects

Object	$i = 1$	$i = 2$	$i = 3$
Section	Release – preceding	Occupy – running train	Release – running train
Signal	'Go' – preceding train	'Stop' – running train	'Go' – running train

By keeping track of the sequence of signal and section messages for each train, we are able to fabricate the block objects $b_k \in B$, $B \subset O$ and thus fully describe every train run using blocking time theory, as a sequence of blocking times. For every two successive signal passages a new block object is created or the existing object is updated. Apart from the chronological list of activities (blocking times), every block object b_k is defined by a tuple $(c_1(b_k), c_2(b_k), Sections(b_k))$, that represents the signals on block boundaries and sections comprised by the block.

The process mining algorithm is able to give accurate estimates of actual arrival and departure times of all trains. Set `Platforms` contains all platform track sections in each station $\sigma \in \Sigma$ where Σ is the set of all stations. Each entry has the form $(\sigma, PlatformSections(\sigma))$.

Scheduled arrival and departure times, as well as the minimum dwell times for all trains are planned in the operational timetable given in the file `Timetable`. For every train n_k with a scheduled event time in station σ the file contains the array (σ, n_k) with entries $(a(\sigma, n_k),\ d(\sigma, n_k),\ dwell(\sigma, n_k))$, denoting the scheduled arrival, departure and the minimum dwell time, respectively.

Finally, by $\phi(r_k, n_l)$, we denote the object of the same type that precedes r_k on the route of the running train n_l and by $\tau(n_k, r_l)$, the immediate predecessor of train n_k on object r_l.

By applying the train separation principles of blocking time theory, the algorithm automatically identifies route conflicts. A conflict object is defined by a tuple ($signal$, $hindered$, $hindering$) representing the signal of conflict, hindered train and hindering train. A conflict object α is stored in set `OpenConflicts` until the hindered train passes the next signal after the signal of conflict. When the hindering train is identified α is stored in set `ClosedConflicts`.

The process mining algorithm is given in form of a pseudo code in Algorithm 1.

Input is the TROTS log file and previously defined sets. In lines 3–4 the necessary sets and variables are initialized. In the main loop (lines 6–29) the algorithm reads each line. Lines reporting a signal aspect change to 'stop' are stored in list `SignalLines` (lines 8–11). If l is a section message it is stored in the set `SectionLines` (line 14). For every message reporting occupation of the section protected by a signal, we find the corresponding signal message and assign to it the section message code in order to identify the running train (lines 15–19).

After every train step message the algorithm searches the sets `SignalLines` and `SectionLines` (in that order) and looks for the corresponding infrastructure message in the lists, based on the unique message code (lines 22–24). When the message is found, list of processes on objects is updated (line 25) and the integrated message is created (line 26). Finally, the Conflict identification algorithm is activated (line 28).

The main procedure for automatic route conflict identification is given in form of a pseudo code in Algorithm 2.

Lines 3-23 of Algorithm 1 refer to signal messages. The procedure is initiated only if the signal aspect is changed to 'stop'. We first delete conflict object at previous signal of the running train from the file `OpenConflicts` (lines 4–6). In lines 7-8, the previous and the current blocks of the running train are identified.

Conflict identification procedure for a train with a scheduled stop on the previous block is given in lines 10-19. The algorithm first derives the actual arrival and departure times from TROTS log files (lines 10-16). In lines 10-12 a list of times of all section occupations and releases in the platform block (including the passing time of exit signal) is created. Note that not all release times of platform sections are recorded by the time the train passes the exit signal, however that does not affect the method we propose. The period of standstill is determined as the longest time gap between two successive events (line 14). The time of the last section message before the standstill is set as the arrival time $arr(\sigma, n_l)$ (line 15) and the time of the first section message after the standstill is the departure time $dep(\sigma, n_l)$ (line 16). Note that the error of this arrival (departure) time estimate depends on the number of platform track sections and the

Algorithm 1 Process mining TROTS log file

Input:TROTS log file, `Infrastructure`,`Platforms`,`Timetable`
Output: O, `ClosedConflicts`
`OpenConflicts` = $\emptyset$, `ClosedConflicts` = $\emptyset$, `SignalLines` = $\emptyset$, `SectionLines` = $\emptyset$, $l = 0$
`InfraLines` $\leftarrow$ {`SignalLines`,`SectionLines`}
while $l \leq$ number of lines in TROTS log file **do**
  $l = l + 1$
  Read line l
  if $type_l$='signal' **then**
    $r_l \leftarrow \Omega(name_l)$
    if e_l ='stop' **then**
      `SignalLines` $\leftarrow (code_l, t_l, r_l, e_l)$
  else if $type_l$='section' **then**
    $r_l \leftarrow \Omega(name_l)$
    `SectionLines` $\leftarrow (code_l, t_l, r_l, e_l)$
    if e_l ='occupy' **and** $signal(r_l) \neq \emptyset$ **then**
      for $k = 1$ to size (`SignalLines`) **do**
        if $protected(r_k) = r_l$ **then**
          $code_k \leftarrow code_l$
          break
  else if $type$ ='train' **then**
    $n_l \leftarrow \Omega(name_l)$
    for $j = 1$ to 2 **do**
      for $k = 1$ to size `InfraLines`$\{j\}$ **do**
        if $code_l = code_k$ **then**
          update processes on objects r_k, n_l
          integrated message (t_k, r_k, e_k, n_l)
          `InfraLines`$\{j\} \leftarrow$ `InfraLines`$\{j\} \setminus (code_k, t_k, r_k, e_k)$
          Run conflict identification algorithm (Algorithm 2)
          break

Algorithm 2 Conflict identification algorithm

Input:(t_l, r_l, e_l, n_l), O, `OpenConflicts`, `ClosedConflicts`, `Timetable`,`Platforms`
Output: O, `OpenConflicts`, `ClosedConflicts`
if $r_l \in C$ **and** e_l = 'stop' **then**
  $\alpha \leftarrow (\phi(r_l, n_l), n_l, \cdot)$
  if $\alpha \in$ `OpenConflicts` **then**
    `OpenConflict` $\leftarrow$ `OpenConflict` $\setminus \alpha$
  $b_{\text{prev}}(n_l) \leftarrow (\phi(r_l, n_l), r_l, Sections(b_{\text{prev}}(n_l)))$
  $b_{\text{curr}}(n_l) \leftarrow (r_l, \cdot, \cdot)$
  if $(Sections(b_{\text{prev}}(n_l)) \in PlatformSections(\sigma)$ **and** $(\sigma, n_l) \in$ `Timetable` **then**
    ψ=time$(o_l, 2)$
    for all $k : r_k \in Sections(b_{\text{prev}}(n_l))$ **do**
      $\psi \leftarrow \psi \cup \{$time$(r_k, 2)$,time$(r_k, 3)\}$
    sort vector ψ chronologically
    $k^* = \arg\max_k(\psi_{k+1} - \psi_k)$
    $arr(\sigma, n_l) = \psi_{k^*}$
    $dep(\sigma, n_l) = \psi_{k^*+1}$
    if time$(r_l, 1) > \max\left(d(\sigma, n_l), arr(\sigma, n_l) + dwell(\sigma, n_l)\right)$ **then**
      $\alpha \leftarrow (r_l, n_l, \cdot)$
      `OpenConflicts` $\leftarrow$ `OpenConflicts` $\cup \alpha$
  else
    if time$(\phi(r_l, n_l), 2) - 12 <$ time$(r_l, 1)$ **then**
      $\alpha \leftarrow (r_l, n_l, \cdot)$
      `OpenConflicts` $\leftarrow$ `OpenConflicts` $\cup \alpha$
else if $r_l \in S$ **and** e_l ='occupied' **then**
  $Sections(b_{\text{curr}}(n_l)) \leftarrow Sections(b_{\text{curr}}(n_l)) \cup r_l$
  $\alpha \leftarrow (c_1(b_k), n_l, \cdot)$
  if $\alpha \in$ `OpenConflict` **then**
    if time$(r_l, 1) >$ time$(\phi(c_1(b_{\text{curr}}(n_l)), n_l), 2)$ **then**
      $\alpha \leftarrow (c_1(b_{\text{curr}}(n_l)), n_l, \tau(n_l, r_l))$
      if $\alpha \notin$ `ClosedConflicts` **then**
        `ClosedConflict` $\leftarrow$ `ClosedConflict` $\cup \alpha$

distance between the stop location of the rear (front) of the train and the used section border.

After deriving the arrival and departure times, the algorithm checks whether the departing train was a victim in a route conflict (line 17). We assume here that the departing train was hindered if the exit signal was showing stop at the scheduled departure time (if the train had no arrival delay) or after the minimum dwell time has passed since the arrival time (if the train arrived with a delay). This subroutine lists all candidates for outbound route conflicts. Extended dwell times in stations can not directly be explained by route conflicts. In order to exclude the trains that waited for a feeder train to realize a connection, or the ones that had extended dwell time for some other reason, additional information from signalers and dispatchers is necessary.

If the train did not have a scheduled stop on the previous block (line 20), the algorithm checks if a route conflict exists, i.e. if the current signal showed 'stop' at the moment when the train passed the previous signal (line 21). If the train separation principle was violated, the set `OpenConflicts` is updated (line 12). Note that the time stamp of the approach signal message is modified with constant value 12 s, representing sight and reaction time, as described in Section 2.

Finally, lines 25-30 are visited when the message from a track section is received. In line 25 the current section is added to the set of sections on the current block. As the hindered train progresses along the block protected by the signal of conflict, the algorithm compares the previous release time of each section belonging to the block with the time when the hindered train passed the approach signal before the signal of conflict. The train that released the section for which condition in line 28 holds is the hindering train.

5 Case study

This section illustrates the application of the presented algorithm on one day of traffic (April 2nd 2010) in the TROTS areas The Hague and Rotterdam. The algorithm sweeps through the merged log files of the two areas and reconstructs the realized train paths of 2048 trains on the level of track sections. Moreover, all occupation times of 1396 track sections and all blocking times of 733 blocks are determined, as well as the aspect changes of 624 signals and the arrival and departure time estimates of all trains at 21 stations. Finally, 1011 route conflicts are identified.

5.1 Graphical user interface

In order to simplify the analysis of the output, a graphical user interface (GUI) has been created (Figure 3). The left part of the GUI contains tabbed panels for loading data (top left panel), visualization control (top right) and displaying results in tables (lower panel). The right part of the GUI is reserved for the visualization of traffic in either time-distance or blocking time diagrams.

The tabbed panel for loading data enables the user to either load the raw data and start the algorithm or load already processed data and display the results. In the lower tabbed panel the user can choose which results to display. In the tab *Trains* (Figure 4), a train

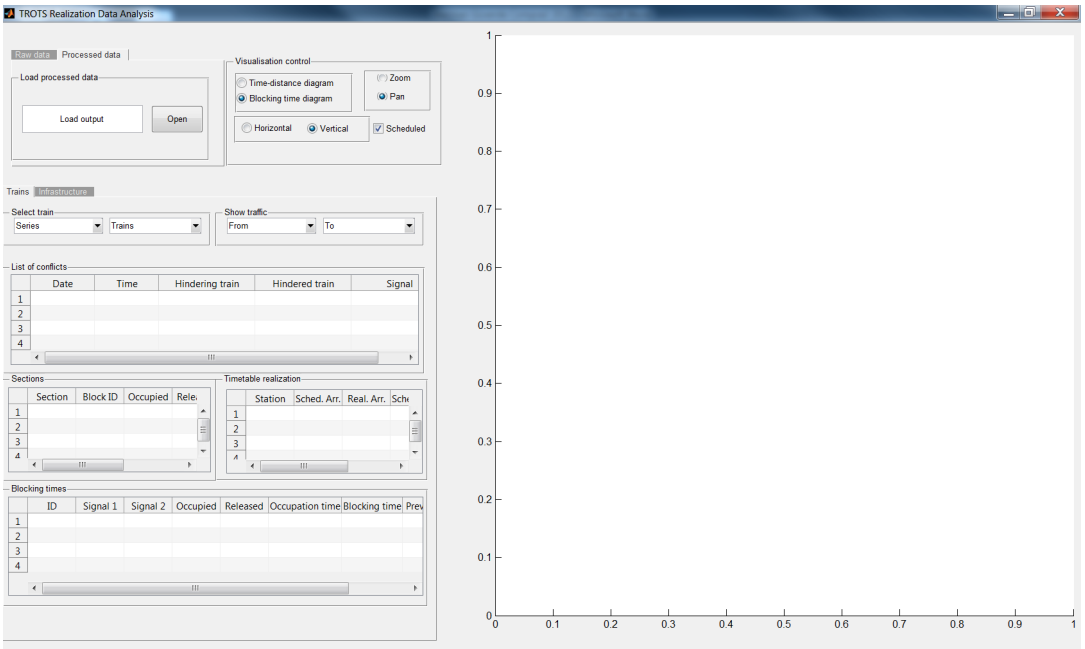

Figure 3: Graphical user interface

line can be selected from the pop-up menu which enables selecting a train from the chosen line. We can then select the whole train path or a part of it by selecting a start and end station. The results are then displayed in the tables on the left and the visualization panel on the right. The selected part of the train route is visualized together with all other trains that operated on the selected corridor 15 min before and after the selected train. The tables are the list of conflicts in which the selected train participated, the running times on all sections, the blocking times, and actual arrival and departure times at all stations.

The panel *Infrastructure* (Figure 5) enables the user to choose the corridor and the time interval and get the corresponding list of conflicts, list of sections, signals, blocks, and stations that were utilized by trains on the corridor within the selected time interval. Selection of the infrastructure element from the corresponding pop-up menu displays all the state changes of that element with the associated train number and time instants (in seconds from midnight).

The visualization control panel (upper right panel Figure 3) enables the user to switch between the blocking time diagram and time-distance diagram of traffic on the selected corridor and time interval. Also it is possible to turn on/off the zoom and pan tools and rotate the axis of the diagrams. Finally the selection of the check-box *Scheduled* also visualizes the scheduled train paths.

Figure 6 shows the time-distance diagram on the busy corridor between The Hague HS and Rotterdam in the Netherlands between 9:00 and 9:40 A.M. The number of tracks between the stations is indicated (the number of lines between station names abbreviations on the left side of the figure indicates the number of tracks) as well as the conflicts (red squares on the hindered train path at the location of the signal of conflict). Intercity trains are presented in blue color and local trains in magenta.

Figure 7 displays the corresponding blocking time diagram for one direction that ap-

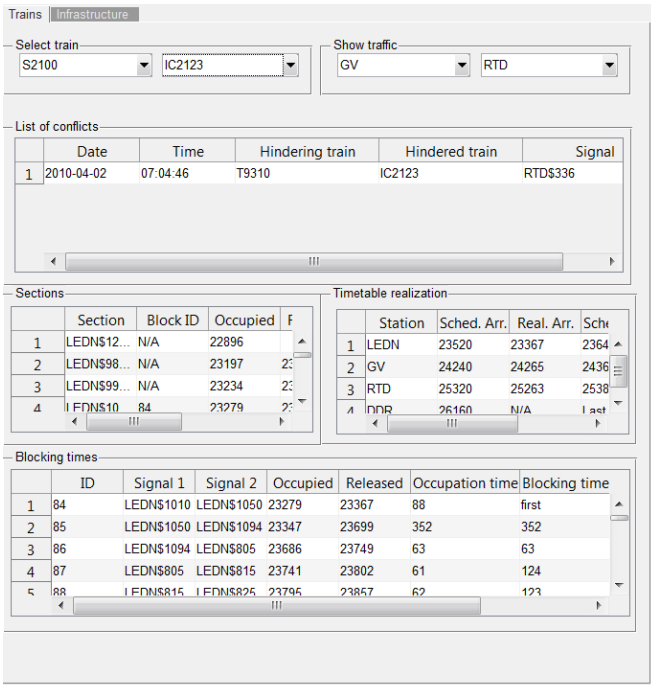

Figure 4: Train selection panel

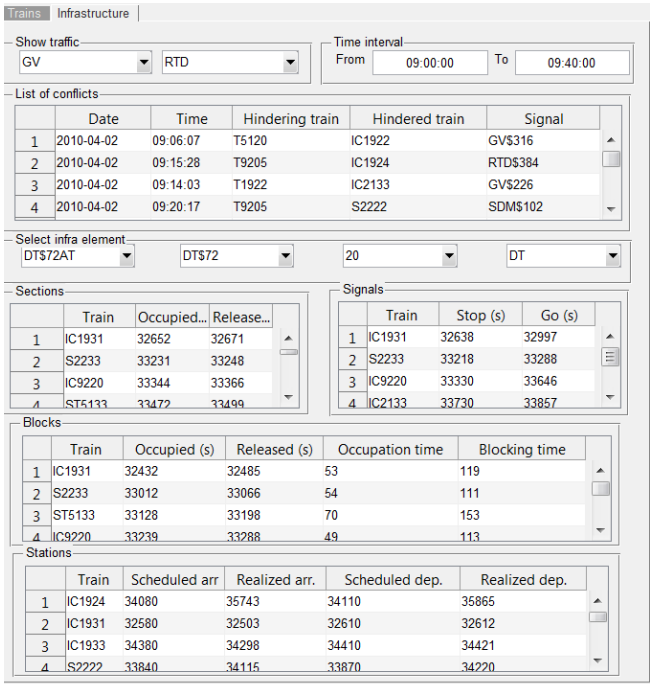

Figure 5: Infrastructure selection panel

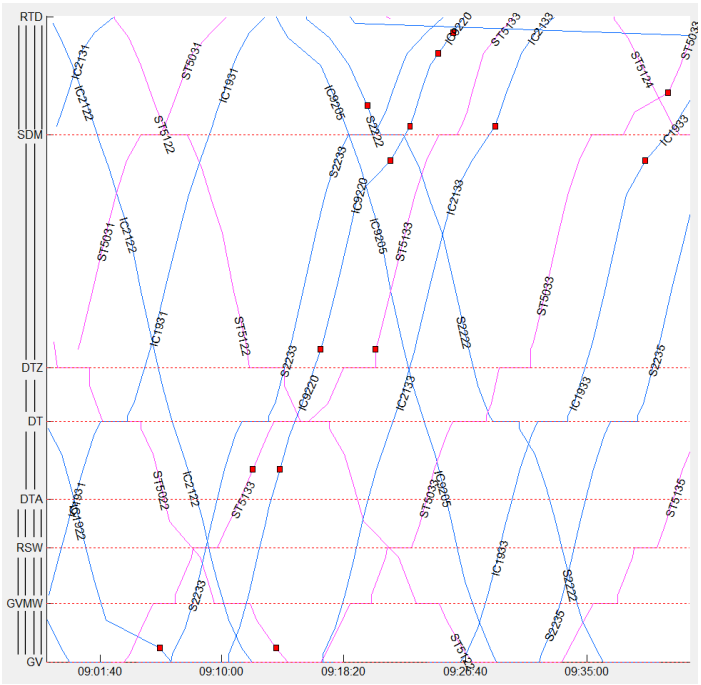

Figure 6: Time distance diagram

pears after selecting the appropriate radio button on the visualization control panel. Overlaps in blocking times indicating conflicts are denoted in red color. Note that trains on parallel tracks of four-track lines may overtake each other. Blocking times that appear to be overlapping but are not shown in red are non-conflicting parallel processes.

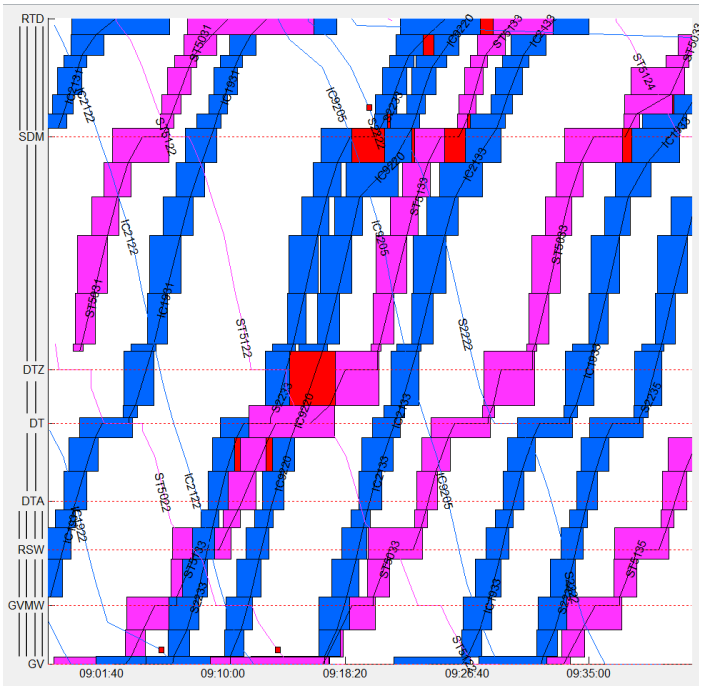

Figure 7: Blocking time diagram

6 Conclusions

In this paper we presented a tool for reconstruction of train paths and automatic conflict identification based on process mining train describer data. Usefulness for identifying systematic delay dependencies and analyzing delays during incidents and severe disruptions . The tool is compatible with the Dutch train describer system TROTS. Straightforward applicability of the tool for other train describer systems strongly depends on their data structure. However, using the principles of blocking time theory as a process model in mining the event log files, is a generic method for analysis of running times and dwell times, and identification of route conflicts for fixed block signaling systems.

The tool provides flexibility in analyzing particular train paths and traffic on the corridor. Visual and tabular output simplify analysis and highlight severe disruptions as well as minor disturbances as a result of variability of process times.

Further developments are mainly directed towards automatic analysis by providing useful statistical indicators for structural flaws in the timetable, as well as detecting severe disruptions and identifying primary delays, see also Goverde & Meng (2012).

Another stream of research within mining and analysis of train realization data, focuses on deriving accurate predictions of process times within the monitoring and short-term prediction component of a model-predictive controller for railway traffic management (Kecman et al., 2011). We aim at exploiting advanced statistical and machine learning methods to capture complex dependencies between process times in heavily utilized railway networks. The developed tool presented in this paper is the basis for this ongoing work.

Acknowledgements

This paper is a result of the research project funded by the Dutch Technology Foundation STW: Model-Predictive Railway Traffic Management (project no. 11025).

References

Conte, C. (2007) *Identifying Dependencies Among Delays*, Phd thesis, University of Göttingen.

Cule, B., B. Goethals, S. Tassenoy, S. Verboven (2011) Mining train delays, in: *Proceedings of the 4th International Seminar on Railway Operations Modelling and Analysis (RailRome 2011).*

Daamen, W., R. M. P. Goverde, I. A. Hansen (2008) Non-Discriminatory Automatic Registration of Knock-On Train Delays, *Networks and Spatial Economics*, 9(1), pp. 47–61.

EC (2001) European commission directive 2001/14/ec of the european parliament and of the council of 26 february 2001 on the allocation of railway infrastructure capacity and the levying of charges for the use of railway infrastructure and safety certification, *Off. J. Eur. Communities L75*, pp. 29–46.

Exer, A. (1995) Rail traffic management, in: Bailey, C., ed., *European Railway Signalling*, IRSE, A&C Black, London.

Flier, H., R. Gelashvili, T. Graffagnino, M. Nunkesser (2009) Mining Railway Delay Dependencies in Large-Scale Real-World Delay Data, in: Ahuja, R. K., R. H. Möhring, C. D. Zaroliagis, eds., *Robust and Online Large-Scale Optimization Models and Techniques for Transportation Systems*, vol. 5868 of *Lecture Notes in Computer Science*, Springer, Berlin, pp. 354–368.

Goverde, R. M. P., L. Meng (2012) Advanced monitoring and management information of railway operations, *Journal of Rail Transport Planning & Management*, 1(2), pp. 69–79.

Hansen, I. A., J. Pachl, eds. (2008) *Railway timetable & traffic - analysis, modelling, simulation*, EUrail press, Hamburg.

Kecman, P., R. M. P. Goverde, T. J. J. Van den Boom (2011) A model-predictive control framework for railway traffic management, in: *Proceedings of the 4th International Seminar on Railway Operations Modelling and Analysis (RailRome 2011)*.

ProRail (2008) Trots protocol interface design description (in dutch).

UIC (2004) Capacity, leaflet code 406 r, Union International des Chemins de Fer (UIC).

Van der Aalst, W. M. P. (2011) *Process mining*, Springer, Heidelberg.