SUMMARY This paper presents a formal approach for generating train timetables in a mesoscopic level that is more concrete than the microscopic level, where each station is simply expressed in a black-box, and more abstract than the microscopic level, where the infrastructure in each station area is expressed in detail. The accuracy of generated timetable and the computational effort for the generation is a trade-off. In this paper, we design a formal mesoscopic modeling language by analyzing real railways, for example Tazawako-line as the first step of this work. Then, we define the constraint formulae for generating train timetables with the help of SMT-SAT solver that is an implementation of the constraint formulae. Finally, we demonstrate how RW-Solver with the help of SMT-Solver can be used for generating timetables in a case study of Tazawako-line.

key words: formal approach, train timetable, mesoscopic model, periodic timetable, SMT-Solver, railway capacity

1. Introduction

The train timetabling problem aims at determining the arrival time and departure time of each train at each station when constraints (e.g. running and dwelling times, headway times, station capacities, routes of trains) are given. Many approaches for generating timetables have been proposed and recent overviews about railway timetable generation are given, for example, in [1].

The approaches for the timetabling problem can be classified by the two perspectives: accuracy of railway models and strategy for solving constraints. About the accuracy of railway models, the following three levels are often discussed:

- In the microscopic level, models can express the details of the railway networks and the platforms in each station area as shown in Fig. 1(a). In this level, train timetables are often improved by simulations [2].
- In the macroscopic level, models usually represent each station as a black-box as shown in Fig. 1(c), and schedules in the models consist of departure and arrival times at the black-box [3]–[5].
- Mesoscopic levels are more concrete than the macroscopic level and more abstract than the microscopic level. For example, in a mesoscopic level (e.g. [6]), models can express available platforms for each direction in each station area as shown in Fig. 1(b) and it is still possible to automatically generate timetable even if it is difficult in the microscopic level.

About the strategy for solving constraints, there are the following two classes of methods:

- Exact solution method: Constraint solvers such as SAT-Solver are used for checking whether a timetable to satisfy the given set of constraints exists or not [7]. Such a problem is known to be NP-complete.
- Heuristic solution method: Some heuristic approaches such as genetic algorithm [8] and greedy procedure [9] have been applied for heuristically generating timetables to satisfy the given set of constraints in reasonable computational time.

Heuristic solution methods can be applied to real world railway networks even if exact solution methods cannot be applied because of the complexity. For example, a heuristic greedy algorithm is successfully applied to the mesoscopic model of the real rail-network of North-Eastern Italy in [6]. However, even if such heuristic solution methods cannot find a timetable to satisfy given constraints, it does not mean that the constraints are unsatisfiable because they often use stochastic approaches for finding global solutions by local searches, in other words, for avoiding local minimum solutions. For example, the solution algorithm presented in [6] uses randomness for iteratively varying priorities and departure times of trains. Such random variations are useful for obtaining better solutions, but the solution depends on the random variations.

On the other hand, if exact solution methods return “unsatisfiable”, it means the constrains are unsatisfiable. It is important to check unsatisfiability of constraints because it can avoid wasting time for scheduling under unsatisfiable constraints.

The performance of constraint solvers such as SAT-
Solver (e.g. MiniSat [10]) and SMT-Solver (e.g. Z3 [11], CVC4 [12]) has been significantly increasing in recent years. For example, SAT-Solver was successfully applied for generating periodic timetables from macroscopic models of public railway transport network in Germany by encoding a PESP (Periodic Event Scheduling Problem) to a satisfiability problem [7]. The PESP is widely used for expressing the periodic train timetabling problem in the macroscopic level and various solution techniques for PESP instances have been proposed. Here, it is noted that various requirements for periodic timetables can be modeled in the PESP, but some features, e.g. track allocations, cannot be modeled [5].

In this paper, we present an exact solution-method for automatically generating train timetables in a mesoscopic level, and explain our tool RW-Solver that is an implementation of the generation-method, and then demonstrate that RW-Solver with the help of SMT-Solver [13] can generate periodic timetables, that includes overtakings and crossings between trains, from a mesoscopic model of Tazawako-line. Here, a periodic timetable repeats the same pattern in a periodic time. Although timetables in the real world are not necessarily fully periodic, periodic timetables are often used as drafts of practical timetables.

Figure 2 shows the overview of the generation of timetables by RW-Solver. RW-Solver generates constraint formulae from given mesoscopic railway models, and submits the constraint formulae to SMT-Solver, and takes the results from STM-Solver, and then generates graphical files (PDF) of the timetables.

The accuracy (the level of details) of mesoscopic railway models and the computational effort for the generation is a trade-off. Therefore, we carefully analyzed the essence needed for generating train timetables based on real railways, for example Tazawako-line as the first step of this work, where bullet trains (Shinkansen) and local trains are running on the same single tracks, and it means that a train often crosses and/or overtakes the other trains in multi-track stations. As a consequence, we define some attributes, such as requirements on available platform-numbers, in station-areas and inter-stations (lines between stations). This is the reason why our model is mesoscopic because macroscopic models cannot express platform allocations as explained in [5].

As related works, Fabris et al. [6] successfully generated timetables from mesoscopic models but they used a heuristic greedy procedure (i.e. not an exact solution-method). Großmann et al. [7] successfully applied SAT-Solver (i.e. an exact solution-method) but the target was PESP instances, in other words, macroscopic models (i.e. not mesoscopic).

This paper is organized as follows: First, we explain how to formalize the information of structure of railways and trains in a mesoscopic level in Sect. 2. Next, the constraint formulae for generating train timetables are defined in Sect. 3, and then are implemented in the tool RW-Solver in Sect. 4. Then, it is demonstrated how RW-Solver with the help of SMT-Solver can generate train timetables by a case study of Tazawako-line in Sect. 5. Finally, we conclude this paper in Sect. 6.

2. Formal Mesoscopic Modeling of Railway System

In this section, we define a formal modeling method of railway system necessary for generating timetables. A railway model consists of a structure model and a train model as shown in Fig. 3. The structure model shows properties of the infrastructure of a railway such as the available numbers of platforms and the headway times, and the train model shows properties of trains such as the routes and the running/dwelling times. The details of them are explained in Sect. 2.1 and Sect. 2.2.

2.1 Structure Model

The structure model of a railway network consists of a number of structure modules, where each module expresses the structure in a station-area or the structure of an inter-station (i.e. a connection-line between stations). In our modeling, station-areas and inter-stations are modeled in the same format.

The attributes of each structure module are given in Table 1. Each structure module is connected by global identifiers named Link-IDs that are specified in the attribute Links. The capacity of a structure module is the maximum number of trains that can exist in the module at the same time. Then, the set \{1, \ldots, c\} of integers named Track-IDs is assigned to the structure module, where \(c\) is the capacity. For
example, each Track-ID corresponds to a platform-number in a station-area. The attribute flag station is not used when solving constraints and is used only for generating graphical timetables.

Table 2 shows the attributes of each structure module in the simple example given in Fig. 3. In the modules M1 and M5, two trains in the opposite direction can cross but two trains in the same direction cannot overtake each other because the attribute fifos is specified, namely it means double-track. On the other hand, in the module M3, two trains can neither cross nor overtake because the attributes fifos and excs are specified, namely it means single-track.

The attribute att of the module m is often described in the form m.att in this paper. For example, M2.capacity = 3 in Table 2.

In Fig. 3, the integers of [1], [2], and [3] in the module M2 (a station) are Track-IDs and correspond to the platform numbers. The set [1, 2] in the module M1 (an inter-station) means that two trains can exist on the line between the Link-IDs 1 and 3 at the same time, for example, a train can follow the other train, where the order of the Track-IDs 1 and 2 has no meaning and the IDs are just used as two tokens acquired by two trains, at most.

2.2 Train Model

The train model consists of a number of train modules and each train is modeled by a train module, which specifies the route from the start Link-ID to the end Link-ID, the required time for running each structure-module, and so on. The attributes of each train module are given in Table 3. The route of each train is expressed by a sequence of Link-IDs that the train is passing through. The option req_tracks can specify tracks that each train can use in each module, for example, it can specify available platform-numbers that each train stops in station-areas.

The attributes of each train in Fig. 3 are given in Table 4. For example, the second set [2, 3] in the list in req_tracks of the local train T2 means that the train can use the platform No. 2 or 3 in the station M4, while the express train T3 must pass the platform No.2 (see the second set [2]). Here, the special symbol “_” in the list in req_tracks of the local train T2 means that every track number is permitted in the inter-station M3, in other words, it equals to the set of all the Track-IDs, thus [1, 2] in this case.

The attribute req_time (option) in train modules has higher priority than the attribute req_time in structure modules. In other words, even if the (default) time for running a module is given in the structure module, the time can be overwritten by req_times of each train. For example, the second pair (60, 120) in the list in req_times of the local train T2 means that the train must dwell for 60 seconds at least and for 120 seconds at most in the station M4, while the express train T3 must pass the station M4 for 10 ~ 20 seconds.

2.3 Global Constraints

Periodic timetable, which repeats the same pattern in a periodic time, has been already discussed in many papers (e.g. [7], [14]). We also give an option for generating periodic timetables by specifying the periodic time by the parameter period. The maximum time max_time is also an option and every train must reach the final structure module by the maximum time if it is specified. It is often larger than the periodic time.

3. Definition of Constraint Formulae

In our work, SMT (Satisfiability Modulo Theories)-Solver [13] is used for generating train timetables that satisfy constraints required by railway (i.e. structure and train) models defined in Sect. 2. SMT-Solver is an automatic theorem prover for checking the satisfiability of a first order formula.
with respect to background theories such as integer and real numbers.

In this section, we formalize the constraints as logical
formulæ such that a satisfiable solution of the logical formulæ
expresses the departure/arrival times and the track (platform)
allocation in a timetable of a given railway model. The
logical formulæ contain constants decided by the railway
models and global parameters, and time variables \( t_m(t, n) \)
for deciding departure and arrival times at each structure
module and track variables \( t_k(n, n) \) for deciding tracks (e.g.
platform numbers) used by each train, where all the numbers
used in this paper are integers. The time variables and the
track variables are explained as follows:

- \( t_m(t, n) \): The time when the train \( t \) passes the \( n \)’th link point,
  where the first \( n \) is 0.
- \( t_k(n, n) \): The Track-ID that the train \( t \) uses in the structure
  module just after passing the \( n \)’th link point.

For example, in Fig. 3, the time variable of the train
\( T_2 \) (Local 2) passing the Link-ID 5 is \( t_m(2, 3) \), and the track
variable of the train \( T_2 \) in the module \( M_2 \) is \( t_k(2, 3) \).

The constraint-formulæ for generating timetables are
defined in Fig. 4. Each constraint is explained as follows:

1. \( c_{\text{req_time}}(m) \): In the structure module \( m \), the running/dwelling
time must be between the lower bound \( lb \) and the upper bound \( ub \), and less than the period
tic time \( \text{period} \) for avoiding to collision with (periodic copies of) the self, where \( (lb, ub) \) is given by the
function \( \text{bounds}(m) \) that is also defined in Fig. 4, and
\( \text{req_times} \) in each train module has higher priority
than \( \text{req_time} \) in the structure module \( m \).

2. \( c_{\text{headway}}(m) \): For each link point of the structure
module \( m \), after a train passed the link, the next train
cannot pass the link until the headway time has elapsed,
where the function \( \text{trs}_{\text{link}}(m) \) returns the set of pairs
\( (t, n) \) such that, for each link point \( l \) of the module \( m \), the
train \( t \) passes the link point \( l \) as the \( n \)’th link point. The
basic function \( c_{\text{exclusion}}(s_0, e_0, s_1, e_1) \) is defined in
Fig. 5 and means that the time duration \([s_0, e_0] \) does not
overlap with \([s_1, e_1] \) even when periodic copies of trains
repeatedly exist, where \( \% \) is the remainder operator.

3. \( c_{\text{non_overtaking}}(m) \): In the module \( m \), trains cannot
overtake each others in the FIFO-tracks specified by the
option \( \text{fifos} \), where the function \( \text{trs}_{\text{fifos}}(m) \) returns
the set of pairs \( (t, n) \) such that, for each track
\( tk \) in the FIFO-track, the train \( t \) exists on the track \( tk \)
just after passing the \( n \)’th link point. The basic function
\( c_{\text{fifo}}(s_0, e_0, s_1, e_1) \) is defined in Fig. 5 and fundamen-
tially means that if \( s_0 \leq s_1 \) then \( e_0 \leq e_1 \), in other words,
if the train 0 enters this module before the train 1 then
the train 0 must exit from it before the train 1.

4. \( c_{\text{non_collision}}(m) \): In the structure module \( m \),
two trains cannot cross each other in the single-tracks
specified by the option \( \text{exclls} \), where the function
\( \text{trs}_{\text{exclls}}(m) \) returns the set of pairs \( (t, n) \) such that,
for each track \( tk \) in the single-tracks, the train \( t \) exists
on the track \( tk \) just after passing the \( n \)’th link point.
The basic function \( c_{\text{exclusion}} \) is used here again
for avoiding head-on collisions of the trains in the opposite
direction.

5. \( c_{\text{capacity}}(m) \): In the structure module \( m \), for each
pair \( (t, n) \) such that the train \( t \) exists in the module \( m \) just
after passing \( n \)’th link point, the Track-ID \( t_k(t, n) \) must
be between 1 and \( m. \text{capacity} \), where \( m. \text{capacity} \) is
the maximum number of trains existing in the module
at the same time.

6. \( c_{\text{track}}(m) \): In the structure module \( m \), for each pair
\( (t, n) \) such that the train \( t \) exists in the module \( m \) just
after passing the \( n \)’th link point, the Track-ID \( t_k(t, n) \)
must be one in the set \( t. \text{req_tracks}[n] \) of permitted
track numbers if it is specified (i.e. it is not \( _{\_\_} \)).

7. \( c_{\text{non_confliction}}(m) \): In the structure module \( m \),
two trains cannot exist in the same track at the same
time. The basic function \( c_{\text{exclusion}} \) is used here
again.

8. \( c_{\text{start_time}}(t) \): The train \( t \) must pass the first

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>route</td>
<td>int list</td>
</tr>
<tr>
<td>start_time</td>
<td>int x int</td>
</tr>
<tr>
<td>req_times</td>
<td>(int x int) list</td>
</tr>
<tr>
<td>req_tracks</td>
<td>(int set) list</td>
</tr>
<tr>
<td>total_time</td>
<td>int</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>( T_0 ) (outbound local)</th>
<th>( T_1 ) (outbound express)</th>
</tr>
</thead>
<tbody>
<tr>
<td>route</td>
<td>[1, 3, 5, 6, 7, 9]</td>
<td>[1, 3, 5, 6, 7, 9]</td>
</tr>
<tr>
<td>start_time</td>
<td>(0, 0)</td>
<td>(240, 720)</td>
</tr>
<tr>
<td>req_times</td>
<td>[[350, 360], (60, 120), (290, 300), (60, 120), (350, 360)]</td>
<td>[[240, 250], (10, 20), (240, 250), (10, 20), (240, 250)]</td>
</tr>
<tr>
<td>req_tracks</td>
<td>[[1, 2], [1, 2], [1, 2], [1, 2]]</td>
<td>[[1, 2], [2, 3, 2, 3], [1, 2], [1, 2]]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>( T_2 ) (inbound local)</th>
<th>( T_3 ) (inbound express)</th>
</tr>
</thead>
<tbody>
<tr>
<td>route</td>
<td>[10, 8, 6, 5, 4, 2]</td>
<td>[10, 8, 6, 5, 4, 2]</td>
</tr>
<tr>
<td>start_time</td>
<td>(0, 720)</td>
<td>(0, 720)</td>
</tr>
<tr>
<td>req_times</td>
<td>[[350, 360], (60, 120), (290, 300), (60, 120), (350, 360)]</td>
<td>[[240, 250], (10, 20), (240, 250), (10, 20), (240, 250)]</td>
</tr>
<tr>
<td>req_tracks</td>
<td>[[3, 4], [2, 3], [2, 3], [2, 3], [3, 4]]</td>
<td>[[3, 4], [2, 3], [2, 3], [3, 4]]</td>
</tr>
</tbody>
</table>
Tool Implementation: RW-Solver

We have implemented the constraint-formulae explained in Sect. 3 in our tool RW-Solver (RW is an abbreviation of Railway). RW-Solver solves the constraints with the help of SMT-Solver [13], where RW-Solver has the following two main functions:

- It generates the graphical timetable files (PDF) from the results of SMT-Solver.

4.1 Introduction to SMT-Solver

SMT (Satisfiability Modulo Theories) problem is a decision problem for logical first order formulae by adding theories of real numbers, integers, and so on. Thus, SMT-Solver is a tool for deciding the satisfiability of a given formula in the theories, and if the formula is satisfiable then SMT-Solver shows one of the solutions to satisfy the formula.

SMT-LIB [15] defines a standard format for describing SMT problems. For example, the following logical formula for integer-variables x and y is described in the SMT-LIB format as shown in Fig. 6.

\[(y - x < 2) \land (y + 2x < 4) \land (x > -1) \land (y > 0)\]
(declare-const x Int)
(declare-const y Int)
(assert (and (< (- y x) 2) (and (< (+ y (* 2 x)) 4)
(and (> x (- 1)) (> y 0))))))
(check-sat)(get-model)

Fig. 6  A code example in the standard SMT-LIB format.

// The structure module M1
struct {
  capacity = 4
  headway = 30
  links = [1,2,3,4]
  req_time = (240, 360)
  fifos = [(1,3),(4,2)]
}

// The train module T2
train {
  start_time = (0,720)
  route = [10,8,6,5,4,2]
  req_times = [(350,360),(60,120),(290,300),
  (60,120),(350,360)]
  req_tracks = [(3,4),(2,3),_,(2,3),(3,4)]
}

Fig. 7  A part of the RWM-script of the simple example in Fig. 3.

When SMT-Solver Z3 [11] takes the code in Fig. 6, it returns “sat” and a solution (x=0, y=1) of the formula.

4.2 RW-Solver: A Generator of Railway Timetables

RW-Solver is implemented in OCaml [16]. OCaml is a functional programming language that has a powerful type system (the types of programs are verified by the compiler before the execution). Therefore, the constraint formulae (functions) defined in Figs. 4 and 5 can be naturally and type-safely implemented in OCaml.

The input language to RW-Solver is called RWM, which is an abbreviation of Railway Model, and it is used for describing the attributes of railway models defined in Table 1 for structure modules and Table 3 for train modules, and global constraints explained in Sect. 2.3. For example, the structure module M1 in Table 2 and the train module T2 in Table 4 are described as shown in Fig. 7.

The OCaml files of source codes (about 2,700 lines), examples of RWM-files, and the manual of RW-Solver can be downloaded from the website [17]. The structure of the OCaml-files is shown in Fig. 8. The most important file is solver.ml, where the constraint formulae defined in Figs. 4 and 5 are implemented.

RW-Solver generates a PDF file of a timetable that satisfies the railway model described in a given RWM-file. For example, Fig. 9 is the timetable generated from the railway model in Fig. 3 by RW-Solver with the help of SMT-Solver Z3 [11], where two cycles of a periodic pattern are shown, and the period and the maximum time are specified as follows: period is 720 seconds and max_time is 1,470 seconds. If max_time is required to be 1,469 then RW-Solver returns “unsatisfiable”. In Fig. 9, the natural number at the end of each line in the graph is the arrival or departure time of the train. For example, Fig. 9 shows that the outbound local train (LCL0) dwells in the station M2 between 350 and 410 (second). In stations, the position of each horizontal line shows the platform-number (Track-ID) of the train. For example, LCL0 stops at the platform No.1 in the station M2. The labels on the lines in the inter-stations are the Track-IDs. For example, in Fig. 9, the integers 2 of LCL0 and 1 of EXP1 in the structure module M1 mean that the outbound trains LCL0 and EXP1 run on the upper track in M1.

5. Case Study: Tazawako-Line

In this section, it is demonstrated how RW-Solver can be used for generating timetables by a realistic example. The example is Tazawako-line, where bullet trains (BLT) and local trains (LCL) are running on the same single tracks. Therefore, only in some multi-track stations, inbound trains and outbound trains can cross each others and bullet trains can overtake local trains.
5.1 The Railway Model of Tazawako-Line

The structure of Tazawako-line is shown in Fig. 10. It consists of 43 structure modules including 20 station-modules. The arrows on tracks show the available directions and no arrow means that both directions are available. For example, it is possible in Ookama-station that a train overtakes the other train, but it is impossible in Koiwai-station.

The train model consists of the four train-modules for \{inbound, outbound\} × \{local, bullet\} trains.

The attributes of Tazawako-line are shown in Table 5. Each column in Table 5 is explained as follows:

- **Module**: The name of each structure module that is a station or a inter-station. The name $M_i$ of inter-station is automatically assigned, where the natural number $i$ counts structure modules even though station names are overwritten by given names.
- **CP**: The capacity of each structure module.
- **Time.LCL** (or **Time.BLT**): The pair of the lower bound and the upper bound of the running or dwelling time of local trains (or bullet trains) in each module.
- **Trk.out** (or **Trk.in**): The set of track-numbers that outbound trains (or inbound trains) can use in each modules. The symbol "\_" means no requirements about track-numbers, thus all the tracks can be freely used.

For example, in Tazawako-station, an outbound bullet train dwells for 60 seconds at least and for 120 seconds at most, in the platform No.2 or 3, and an inbound local train dwells for 60 seconds at least and for 600 seconds at most, in the platform No.2 or 3.

The other attributes are given as follows:

\[
\text{headway} = \begin{cases} 
60 & \text{(in the inter-station-module $M_4$)} \\
30 & \text{(in the other structure-modules)} 
\end{cases}
\]

\[
\text{start\_time} = \begin{cases} 
(0, 3600) & \text{(in local train modules)} \\
(100, 700) & \text{(in bullet train modules)} 
\end{cases}
\]

\[
\text{total\_time} = \begin{cases} 
6000 & \text{(in local train modules)} \\
4200 & \text{(in bullet train modules)} 
\end{cases}
\]

\[
\text{period} = 2500 \quad \text{max\_time} = 6000
\]

where the time unit is "second".

5.2 The Generated Timetable for Tazawako-Line

RW-Solver can connect to any SMT-Solver such as Z3 [11]
The periodic timetable (2 cycles) of Tazawako-line generated by RW-Solver.

or CVC4 [12] in accordance with SMT-LIB standard [15]. Fig. 11 is the timetable automatically generated by RW-Solver with the help of SMT-Solver Z3 (version 4.5.0), from the railway model of Tazawako-line explained in the previous subsection. The generation took about 130 seconds on a laptop computer, where the OS is Debian GNU/Linux 8.6 in VirtualBox on Windows 10, the CPU is Intel Core i7-6567U (3.30 GHz), and the main memory is 8 GB. As a comparison, a similar timetable was able to be generated by RW-Solver with the help of SMT-Solver CVC4, but it took about 3,000 seconds. Currently, we have been mainly using Z3 because it was shown that Z3 has been faster than CVC4 in most of our tests in RW-Solver. RW-Solver can take the advantage of future improvements in the field of SMT-Solver.

In Kakunodate-station at 1,585 seconds in Fig. 11, an inbound bullet train (BLT.in) crosses an outbound local train (LCL.out) and overtakes an inbound local train (LCL.in) at the same time because Kakunodate has three platforms and one (No.3) of them can be used for both directions. Therefore, if the platform No.3 in Kakunodate is blocked then the timetable in Fig. 11 is unavailable.

Figure 12 is the timetable similarly generated by RW-Solver with Z3 in the case that the platform No.3 in Kakunodate is blocked, where the periodic time is modified to 2,700 seconds because this case is unsatisfiable without the modification of the period. In Fig. 12, the inbound local train (LCL.in) crosses the outbound local train (LCL.out) in Kakunodate similarly to them in Fig. 11. However, the station where inbound bullet train (BLT.in) crosses the outbound local train (LCL.out) is changed from Kakunodate to Jindai. It is difficult to manually change the stations of crossings and/or overtakings because a change affects the other crossings and/or the overtakings. RW-Solver can automatically generate timetables under various situations by changing the attributes.

It is not impossible to solve not only departure/arrival-
times and Track-IDs but also a periodic time by SMT-Solver, but the computational effort will become very high. Currently, users give RW-Solver the periodic time. Although it is not easy to find the best periodic time, RW-Solver can help users to find it. It will be useful for finding a periodic time to firstly generate a non-periodic timetable (without specifying the period) by RW-Solver. We also found the periodic times of Figs. 11 and 12 in such a way.

6. Conclusion

We have presented a mesoscopic model enough concrete for expressing realistic railways such as Tazawako-line and enough abstract for the automatic generation of timetables with the help of SMT-Solvers. The model can be constructed by connecting structure-modules and the properties in the modules can be described as attributes. We have also defined the constraint formulae for generating timetables from the mesoscopic models and implemented them in our tool RW-Solver, and then demonstrated that it can generate timetables of Tazawako-line including crossings and overtakings.

RW-Solver can connect to any SMT-Solver in accordance with SMT-LIB standard [15]. It takes the advantage of future improvements in the field of SMT-Solver.

As related works, for generating more practical timetables, for example, partial periodicity [3] and flexible periodicity [4] were proposed because timetables are not necessarily entirely periodic. In RW-Solver, OCaml is used for expressing the constraints formulae, and it is an interesting future work to improve the constraints in RW-Solver for introducing such partial and/or flexible periodicity. Furthermore, it is also a future work to add more useful constraints such as connections of trains for passenger transfers, namely changeovers [5], in the reasonable computational effort. Such changeovers cannot be expressed in the current RW-Solver, but it will be possible to introduce constraints for changeovers, that make two time durations overlap, by a similar expression to “c_exclusion” in Fig. 5, that forbids two time durations overlapping.

Several criteria such as the total passenger waiting time exist to assess the quality of timetables. Such criteria can be expressed in an objective function and the solution is
expected to satisfy all the constraint and to minimize the objective function. Since the problem is NP-hard, various heuristic approaches of optimization of timetables have been proposed as overviewed in [1]. It will be possible to express our satisfaction problem extended with some criteria in a similar objective function to the cost function given in [6], and then solve it by heuristic approaches of optimization. In this case, RW-Solver will contribute for generating the first feasible timetable to be optimized. The combination of such works on optimization with our exact-solution method is also a future work.

References

Akimasa Okada received the B.E. degree in mathematical engineering and information physics and M.S. degree in information physics and computing from the University of Tokyo, Tokyo, Japan in 2004 and 2006, respectively. Since 2006, he has been an engineer with East Japan Railway Company in a field of signaling systems. He received Ph.D. degree at the University of Tokyo in 2016. His current research interests include safety analysis method, a sensing system, and two-dimensional communication. Dr. Okada is a member of IEICE, IEEE, and SICE.

Satoru Kitamura received the B.E. degree in department of computer science from Tokyo Institute of Technology, Tokyo, Japan in 1993. He joined East Japan Railway Company in 1993, and has been an engineer in a field of signalling systems. He has been serving as a chief researcher at R&D Center of JR East Group since 2016. His current research interests include safety of level crossings, optical fiber sensing technology application for railway, and safety analysis of signal system. He is a member of IEEJ.

Yamato Fukuta received B.E. degree in electronic engineering and M.S. degree in electronic engineering from the University of Tokyo, Tokyo, Japan in 1994 and 1996, respectively. Since 1996, he has been an engineer with East Japan Railway Company in a field of signaling systems. He received Ph.D. degree at the University of Tokyo in 2004. Currently he is engaged in management of the train control system of Tokyo Metropolitan area. He is a member of IEEJ and a fellow of IRSE.

Takashi Kunifuji is Principal Chief Researcher of Research and Development Centre of JR East group at East Japan Railway Company (JR East), Japan. He has engaged in research and development of railway signal control systems since 1998. He engaged in maintenance, design, and construction of signalling systems from 1992 to 1998. He has joined JR East since 1992. He has received Ph.D. degree in Engineering from the Tokyo Institute of Technology, Japan in 2008. He graduated from the Tsukuba University, Japan, in 1992 with a Master's degree in Electrical Engineering. He is a Fellow of IRSE and a member of IEEE, IEICE, IEEJ and IPSJ.