TRAINRUNS.JL: AN OPEN-SOURCE TOOL FOR RUNNING TIME ESTIMATION

MARTIN SCHEIDT* & MAX KANNENBERG
TU Braunschweig, Institute of Railway Systems Engineering and Traffic Safety, Germany

ABSTRACT
Open science requires open-source tools that produce verifiable results. Nevertheless, a universal open-source tool in railway engineering is still missing. This paper presents an open-source tool for running time estimation to promote a change toward more open tools. Besides the tool's implementation, the need for simple data formats and data sources is discussed. The tool was tested against a commercial tool and revealed variations in the computation. Nonetheless, the tool can already deliver valuable results for simple investigations.
Keywords: driving dynamics, run time, railway, railroad, open science, driving mode, railml, railtoolkit, julia programming language.

1 INTRODUCTION

Computations for train runs in simulation, analytical assessments, or timetabling have become indispensable in railway engineering. However, the available standard software is closed source and, thus, a black box; therefore, it is difficult or even impossible to retrace exactly how the computations are performed. Furthermore, dynamic driving behaviors can only be investigated with standard simulation software at its given level. In addition, the high costs of software limit its uses in research and teaching.

The open-source tool TrainRuns.jl (TR) [1] presented in this paper has been developed to address this issue. Its functionality is based on state-of-the-art running time estimation equations. However, different approaches can be taken when converting the equations into software code. Therefore, the running time estimation implementation must be considered and compared against an already-established implementation. For such a comparison, train and path data must be loaded in a suitable format. For the use case of TR, RailML has proven to be impractical and too verbose, so a more straightforward custom schema based upon YAML was developed. For the test, routes and train data were chosen that push the limits in the computation.

In this paper, we present and discuss the following contributions:

- the importance of open sources for open science (Section 3),
- the tool TR (Sections 4 and 5),
- the YAML/railtoolkit schema (Section 6), and
- the testing of the tool against a commercial tool (Section 7).

2 RUNNING TIME ESTIMATION

Running time estimation is used to compute the position, speed, and time of a train run. This computed train run is used for many railway operation calculations; therefore, it is generally well known. In addition to scheduling, this estimation is used to optimize energy consumption, calculate capacity consumption, or run simulations. Therefore, the running time estimation is the basis for engineering and scientific investigation for running trains

* *ORCID: https://orcid.org/0000-0002-9384-8945*

(significant studies include Lehmann [2] and Wende [3]). In addition, the summary of calculating a train run is presented in Brűnger and Dahlhaus [4].

A computed train run rarely directly corresponds to an observed train run. Heppe et al. [5] closely examined a specific path section and searched for explanations to investigate the discrepancy between the computed and observed train runs. Other researchers, such as Hertel and Steckel [6] and Medeossi et al. [7], have tried to map distributions for the discrepancy. Kecman and Goverde [8] went a step further by not computing the train run but derivating future running times by parameterizing past train runs. The idea is tempting, but this idea may only be realized with proper data management, sufficiently busy sections, and no new vehicles. Wen et al. [9] provided an overview of the prediction of running time from the existing data.

Several input parameters for the train and track are required for the running time estimation. The train data determine the propulsion force and braking behavior. The opposing resistances are determined by the train and track data. The track parameters, which might change rapidly, and the complex acceleration behavior of the train make a continuous analytical calculation of the train run impractical. The iterations, number of parameters, and assumptions do not allow an exact calculation of the running time but only an estimation. Therefore, the acceleration and resistance computations are conducted using the algorithms implemented in software tools.

The general literature divides driving into four modes: accelerating, constant speed/cruising, coasting, and braking [10, p. 45] (see Fig. 1 ①–④). However, these four driving modes are simplistic and only cover cases under the assumption of a low constant

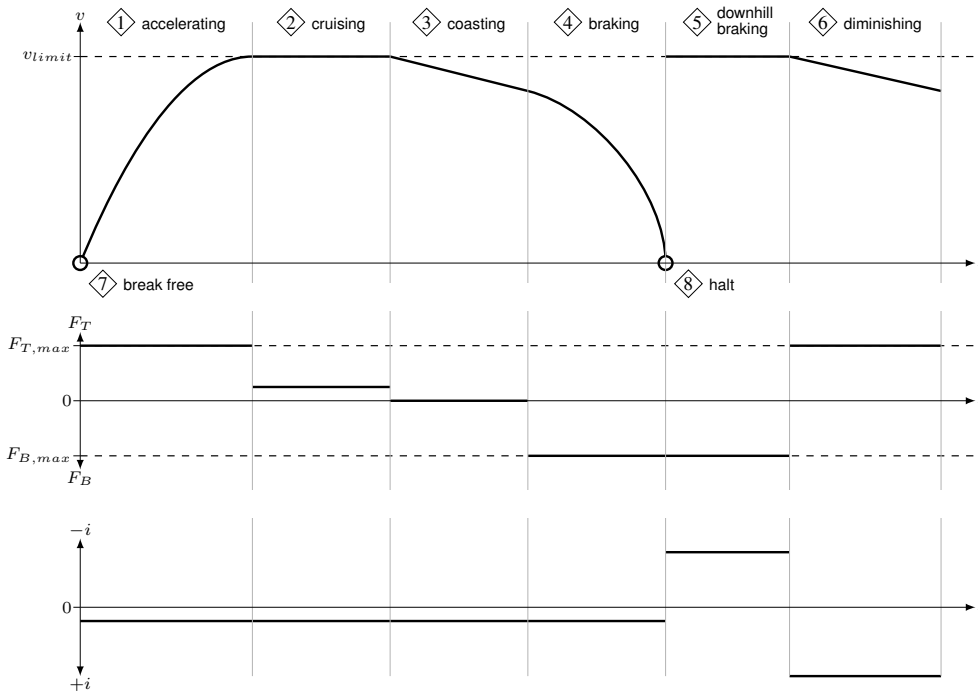

Figure 1: Eight driving modes with qualitative dimensions: Position (s), speed (v), force (traction F_T and braking F_B), and resistance (i).

resistance due to the track. Therefore, Monecke et al. [11] differentiated driving into six movement phases (see Fig. 1 ⟨5⟩, ⟨6⟩). The two additional movement phases are downhill braking and decreasing speed/diminishing. These additional modes account for cases when the train accelerates due to the track slope (downhill braking) or the train does not have enough power to hold its speed due to the slope (diminishing). Adding two more driving modes for the algorithmic computation is also helpful: halt and break free (see Fig. 1 ⟨7⟩, ⟨8⟩). Halt is used when the train is standing still. Break free represents the moment when the train wheels start moving and must overcome the momentum and resistance of the train as an initial state for the computation.

The algorithms for running time estimation are usually embedded in larger software packages. The primary purpose of these software packages is to construct a timetable or simulate operations for stability assessments. Many software packages implement the running time estimation. Some are not citable, and others are presented in a citable form at scientific conferences. A small overview of existing software is presented by Medeossi and de Fabris [12]. Currently, these software packages are used heavily in research and scientific studies. Even if it can be assumed that most of the studies were conducted conscientiously, the results are not independently verifiable because the raw data, parameters, and tools are not openly available.

3 STATEMENT OF NEED

Research results should be freely accessible to all online, particularly if the research has been funded by public money. Besides the results, research methods, or tools, the research process, raw data, and collaborative data processing should also be published. While the publication of raw data, parameters, and processes is a matter of domain culture, the accessibility of tools is dependent on whether the tools are open source. However, most mentioned software packages can only be used to a limited extent for transparent and reproducible research, as they are black boxes due to the unpublished source code.

Figure 2 illustrates the black-box problem for developing a software tool in a scientific context. First, the object of study is from the real world Ⓐ. In the example of running time estimation, the train behavior under different conditions. These conditions were identified using statistical methods ⟨1⟩. However, assumptions must also be made ⟨2⟩. For example, air resistance calculation assumes a constant headwind at 15 km/h [4, p. 70]. While air resistance is only a minor factor, the load and load distribution are essential. Assumptions must be made for the load most of the time. Nevertheless, these observations and assumptions form a model ⟨8⟩ that must be implemented with a programming language that introduces the need for adaptation and abstraction ⟨3⟩. In addition to the unintentional errors, this implementation can contain other simplifications and gimmicks that appear necessary from a computer science perspective. The implemented working model generates raw data ⟨4⟩, with which the model's plausibility can be checked ⟨7⟩. However, the raw data must be aggregated for further processing Ⓓ. The aggregation is performed by indicators that have proven helpful in the past. If the model does not sufficiently characterize the aggregation result ⟨5⟩ and the desired indicators, the model can be changed through fitting ⟨6⟩. The existing software packages are assumed to have been implemented conscientiously. However, in a scientific context, and in a context that evolves through constant verification of models and assumptions, it is problematic that the implementation cannot be verified. Furthermore, the black box does not allow the model to be changed independently when assumptions or indicators change.

Furthermore, the black-box property makes it challenging to understand how results were achieved. Many parameters are required in the running time estimation, usually computed over long distances, and the implementation can be conducted using different step procedures.

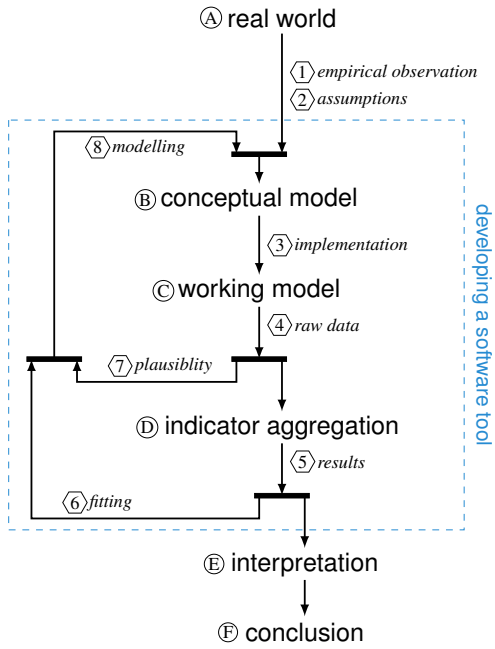

Figure 2: Observations are shaped into models that are used to generate indicators in the development of software tools.

These issues inevitably result in deviations. Because of the numerous parameters, two different implementations never produce the same result. The difference is relatively small, but if one takes the Japanese high-speed network as an example, where the timing is accurate to a few seconds, these deviations from the implementations are relevant.

Three software packages illustrate what is currently needed: OpenTrack [13], RailSys [14], and PULSim [15].

1. The "Open" in OpenTrack suggests that this is open-source software. Although OpenTrack is well suited for research [12], it has not yet been used to generate open science due to the unpublished source code. The "Open" in OpenTrack stands instead for the user-interface framework OpenStep, from which the Cocoa Framework for Mac OS X emerged over several further developments.
2. RailSys has a long history traced back to Völz [16]. However, the remarkable aspect of RailSys is a subprogram called Dynamis [17], [18]. Dynamis is integrated into RailSys and takes over the running time estimation. This example demonstrates that even a comprehensive software package contains smaller components that have a utility independent of the entire package and are worth marketing. It also reveals how much development time must be put into such software packages.
3. PULSim is currently the only open-source software for simulating railway operations. However, being open source is insufficient: the repository is currently unsuitable for potential users. There is no explanation of how the software can be used, modified, or further developed. With this current basis, no community can take care of the project, and the initiator only maintains it without considerable change. However, given the

long development time, having a single person taking care of the code is problematic, as perspectives can change quickly in an academic environment.

Open-source code is needed to overcome the black-box problem of tools in science. Besides the open-source code property, additional tool properties are needed for transparent and reproducible research. The code should be platform-independent so that the code can be run even after changes to user-interface frameworks or runtime environments. Furthermore, it should integrate into an environment for other research, such as computer-aided design, statistics, or simulation. The code should also be readable for advanced beginners in programming so that changes are efficient and transparently possible. A modular structure provides for this purpose, where individual modules can be exchanged, and the individual module already has sufficient utility. Moreover, it takes time and long-term development to meet the expectations of a robust tool.

4 WHAT CAN TRAINRUNS.JL DO OR CANNOT DO?

This paper should contribute to making open science in railway engineering possible. The running time estimation is the basis for these methods; thus, it is logical to start with this matter and create a tool to estimate running time. The tool must be interoperable with other tools or workflows. For the implementation to be as interoperable as possible, a package-oriented scripting language is needed that allows a combination of packages from different scientific domains. Furthermore, a suitable programming language for technical computation must be chosen. Likewise, an interactive language shell is also helpful for beginning programmers.

Currently, the programming languages Julia, MATLAB, Python, and R meet these criteria. All come with a package manager and are scriptable and used for scientific computation. However, MATLAB can be ruled out because of its commercial license and closed source, which are unsuitable for open science. The other three programming languages are a matter of taste. We decided on the newcomer, a fresh approach: the Julia programming language [19]. The naming convention for packages in Julia is to include the file extension *.jl* in the package repository name. The extension is omitted inside the Julia domain. In addition, TR is available as a package in Julia and can be installed via the package manager. The following minimal working example presents the usage.

```
using TrainRuns                          # load the module
train   = Train("freight.yaml")          # load train data from file
path    = Path("east_saxony.yaml")       # load moving section from file
runtime = trainrun(train, path)[end,:t] # returns run time in seconds
```

Three base variables are offered for the computation steps during the use of traction force (see Fig. 1 ◇1–◇3, ◇6): distance, time, and speed. For the braking computation (see Fig. 1 ◇4, ◇5), only the simplest method with constant deceleration is currently offered. The detail level of the computation and output can be set via optional settings.

5 IMPLEMENTATION

The package TR consists of a Julia module in several files and is roughly divided into the interfaces and core. Fig. 3 illustrates the package structure. The interfaces provide the means to transfer the necessary data to the core for computation. The settings can be used to select the model for calculating the acceleration. The stepping variable and step size can be selected for distance, time, and speed. Currently, both point masses and mass bands are supported. However, only constant deceleration is currently implemented for braking.

With the given path data, TR assumes that a train run to be computed starts with the first point and ends with the last point. Therefore, the path data must resemble a moving

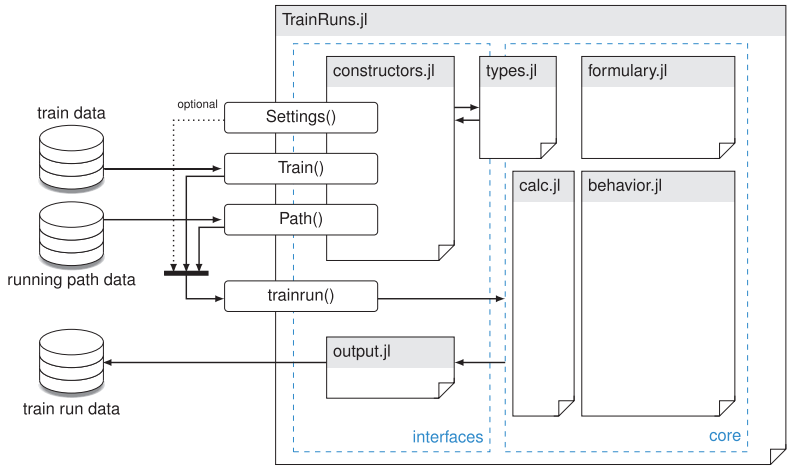

Figure 3: Architectural structure of the module.

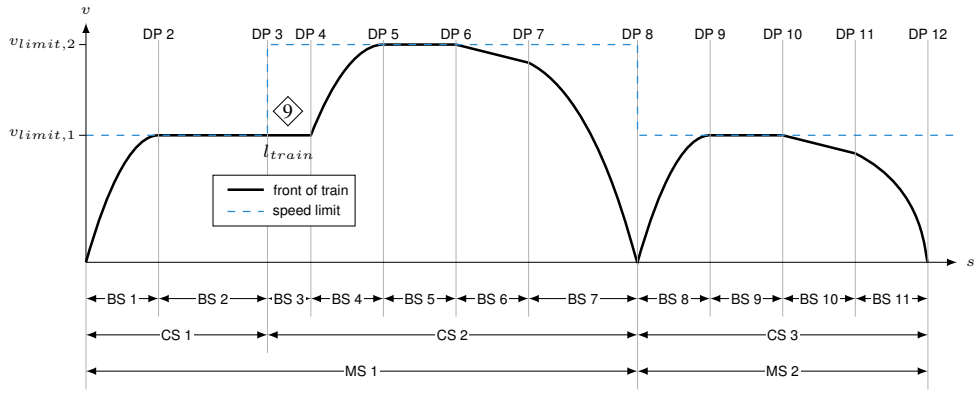

Figure 4: Division of a train run into different sections: Moving sections (MSs), characteristic sections (CSs), and behavioral sections (BSs) results in data points (DPs). The characteristic sections (CSs) are formed from the changes in the properties of the line. The BSs are the driving modes from Fig. 1 with the addition of clearing for the train length l_{train} (BS 3).

section (MS), as presented in Fig. 4. If computing one or more stops along the path data, a preprocessor must first divide the path data at the intended halts and pass the data individually to TR.

The core divides the given path data into characteristic sections (CSs) and behavioral sections (BSs) (cf. [20, p. 80 ff.]). The CSs are sections within the property, such as permitted speed or track resistances, of the infrastructure that do not change (see Fig. 4). A CS is then further divided into one or more BSs. The eight driving modes from Fig. 1 are used for the BS. Furthermore, an additional ninth mode in which the train must maintain a lower speed from a previous section until it has completely cleared the previous section is added (see Fig. 4, 9). The ninth mode is not a true BS but a simple implementation of clearing a speed restriction

with a train length. The forces for calculating the accelerations in the BS are computed using the classical equations from Wende [3] and Brünger and Dahlhaus [4]. These equations are assembled in the formulary.jl (see Fig. 3).

6 INPUT FORMAT

The train and path data are loaded via files. In addition, YAML with the railtoolkit schema [22] is supported as the file format. The popular XML format with the RailML schema is currently not supported. The YAML format is typically easier to read than XML. Furthermore, RailML data can only be manually created with significant effort. However, it must be possible to collect the train and path data using simple means to study them, especially for a scientific study. Nevertheless, besides readability and writability, other criteria exist where RailML is more reasonable than YAML/railtoolkit.

Like RailML, the YAML/railtoolkit schema is an interchange format with definitions for attributes and properties in the rail context. The YAML/railtoolkit schema is a reduced and simplified set compared to RailML and currently only captures rolling stock and a characteristic sequence of line resistances. Consider the following example of a running path in Fig. 5 to illustrate the readability between YAML/railtoolkit and RailML. The information in Fig. 5 is coded in the YAML/railtoolkit format as follows:

```yaml
%YAML 1.2
schema: "https://railtoolkit.org/schema/running-path.json"
schema_version: "2022.05"
paths:
  - name: "Example running path"
    id: example
    points_of_interest:
      # [ station in m,              label, measure front or rear ]
      - [       850.00,        view_point_1,                front ]
      - [      1000.00,    distant_signal_1,                front ]
      - [      2000.00,       main_signal_1,                front ]
      - [      9000.00,       main_signal_3,                front ]
      - [      9050.00,    clearing_point_1,                 rear ]
    characteristic_sections:
      # [ station in m, speed limit in km/h,  resistance in permil ]
      - [          0.0,                160,                 0.00 ]
      - [       5000.0,                160,                 5.00 ]
      - [       6000.0,                160,               -10.00 ]
      - [       7000.0,                 80,                15.00 ]
      - [       8000.0,                120,               -10.00 ]
```

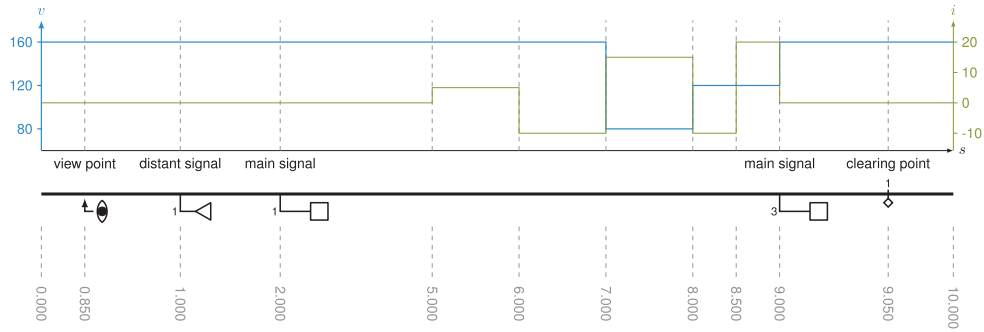

Figure 5: Example running path with symbology from Scheidt and Pachl [21], allowed speed v, and slope i.

```
- [       8500.0,                 120,                 20.00 ]
- [       9000.0,                 160,                  0.00 ]
- [      10000.0,                 160,                  0.00 ]
```

The same information from Fig. 5 encoded in RailML:

```
<?xml version="1.0" encoding="UTF-8"?>
<railml xmlns="https://www.railml.org/schemas/2021" version="2.5">
  <infrastructure id="example" name="Example running path">
    <tracks>
      <track id="01">
        <trackTopology>
          <trackBegin pos="0" id="01_tb">
            <openEnd id="01_start"/>
          </trackBegin>
          <trackEnd pos="10000" id="01_te">
            <openEnd id="01_end"/>
          </trackEnd>
        </trackTopology>
        <trackElements>
          <speedChanges>
            <speedChange id="01_sc01" pos="0"    vMax="160"/>
            <speedChange id="01_sc02" pos="7000" vMax="80"/>
            <speedChange id="01_sc03" pos="8000" vMax="120"/>
            <speedChange id="01_sc04" pos="9000" vMax="160"/>
          </speedChanges>
          <gradientChanges>
            <gradientChange id="01_gc01" pos="0"    slope="0.00"/>
            <gradientChange id="01_gc06" pos="5000" slope="5.00"/>
            <gradientChange id="01_gc07" pos="6000" slope="-10.00"/>
            <gradientChange id="01_gc08" pos="7000" slope="15.00"/>
            <gradientChange id="01_gc09" pos="8000" slope="-10.00"/>
            <gradientChange id="01_gc10" pos="8500" slope="20.00"/>
            <gradientChange id="01_gc11" pos="9000" slope="0.00"/>
          </gradientChanges>
        </trackElements>
        <ocsElements>
          <signals>
            <signal id="01_poi01" pos="850"  name="view_point_1"
                    virtual="true" switchable="false">
            </signal>
            <signal id="01_poi02" pos="1000" name="distant_signal_1"
                    virtual="false" switchable="true">
            </signal>
            <signal id="01_poi03" pos="2000" name="main_signal_1"
                    virtual="false" switchable="true">
            </signal>
            <signal id="01_poi04" pos="9000" name="main_signal_3"
                    virtual="false" switchable="true">
            </signal>
          </signals>
          <trainDetectionElements>
            <trainDetector id="01_poi05" pos="9050" name="clearing_point_1"/>
          </trainDetectionElements>
        </ocsElements>
      </track>
    </tracks>
  </infrastructure>
</railml>
```

Once train data and running path data are loaded, TR computes the run time for the train with the first location on the running path as the departure point and the last location on the running path as the destination. Together with supplementary information, the run time is provided in tabular form as a DataFrame. The tabular DataFrame output was chosen for easy further processing because Julia's DataFrames are suitable for statistical evaluations,

visualization, and exporting into comma-separated value files. The total run time, points of interest, data points (see Fig. 4), or all calculated support points from the step procedure can be returned depending on the settings.

7 TEST CASE AND COMPARISON

A test case was generated to compare the computation of TR with an existing tool. As test data, published path data from railML.org with the east Saxony line [24], generated by the software "Fahrplanbearbeitungssystem" (FBS), was utilized. The FBS software is primarily used to generate timetables [23] and provides a good reference. As test train data, two trains, a local and a freight train, were used to show the range of the computation differences. The standard local train for the line and a heavy freight train, which was heavy enough to handle the line at its power limit, was used. The train and path data are found in Scheidt and Kannenberg [25]. The standard local train is suitable to show the similarities, and the power limit of the freight train is suitable for clearly presenting the differences in computations.

Figure 6 depicts the test case for both trains. FBS has a continuous run time increase compared to TR, especially for the freight train. In addition, TR always reaches a slightly higher speed than FBS. When the speed limit is reached, the difference disappears for both trains, making it clear that the tools calculate different tractive forces. At the destination, compared to the beginning, it becomes apparent that TR is still missing a braking model that uses more than only a constant deceleration. Despite all the differences, with a few exceptions, TR follows the characteristic pattern of the speed of both trains by FBS.

In particular, the higher speed at kilometer 4 with a significant slope for the freight train affects the run time because the section clears much faster with TR. The relative difference is also most remarkable at that location as a result. The difference of 15 min is enormous, but the extreme example explains this because a 12-min difference is built up by kilometer 4. The remaining 3-min difference at about 100 km of distance is again acceptable.

The local train accelerates to its maximum with TR, while FBS is cut off beforehand (seen at kilometers 6 and 8). The FBS commuter train also seems to be more sensitive to slope, which could explain the slight dampening of the speed. However, there is hardly any difference between TR and FBS regarding the travel time required for the local train. Nevertheless, the difference raises questions about the implementation, which cannot be clarified lightly because of the closed source of FBS, making it clear that it is insufficient only to implement the equations from the literature.

8 CONCLUSION AND RESULTS

The discrepancy in Fig. 6 highlights the importance of an open-source implementation of the running time estimation. This open implementation is required for the reproducibility of results in open science. In addition to the open implementation, the accessibility of data for computation is another necessary aspect of reproducibility. The available data should follow the findable, accessible, interoperable, and reusable (FAIR) principles [26] for reproducible open science. In most cases, the infrastructure managers do not have FAIR infrastructure data. However, OpenStreetMap (OSM) can help. Considerable information on railway infrastructure can be found on OSM. However, data on the slope are currently lacking on OSM. Nevertheless, the OSM data are growing daily due to the many volunteers.

The situation is different for rolling stock data. Although considerable general data exist on both older and newer vehicles, many essential factors for calculating the run time can only be found as tables in books or, like for tractive effort, must first be determined and computed. As a remedy for a missing open collection, a database for TR was started in Scheidt [27].

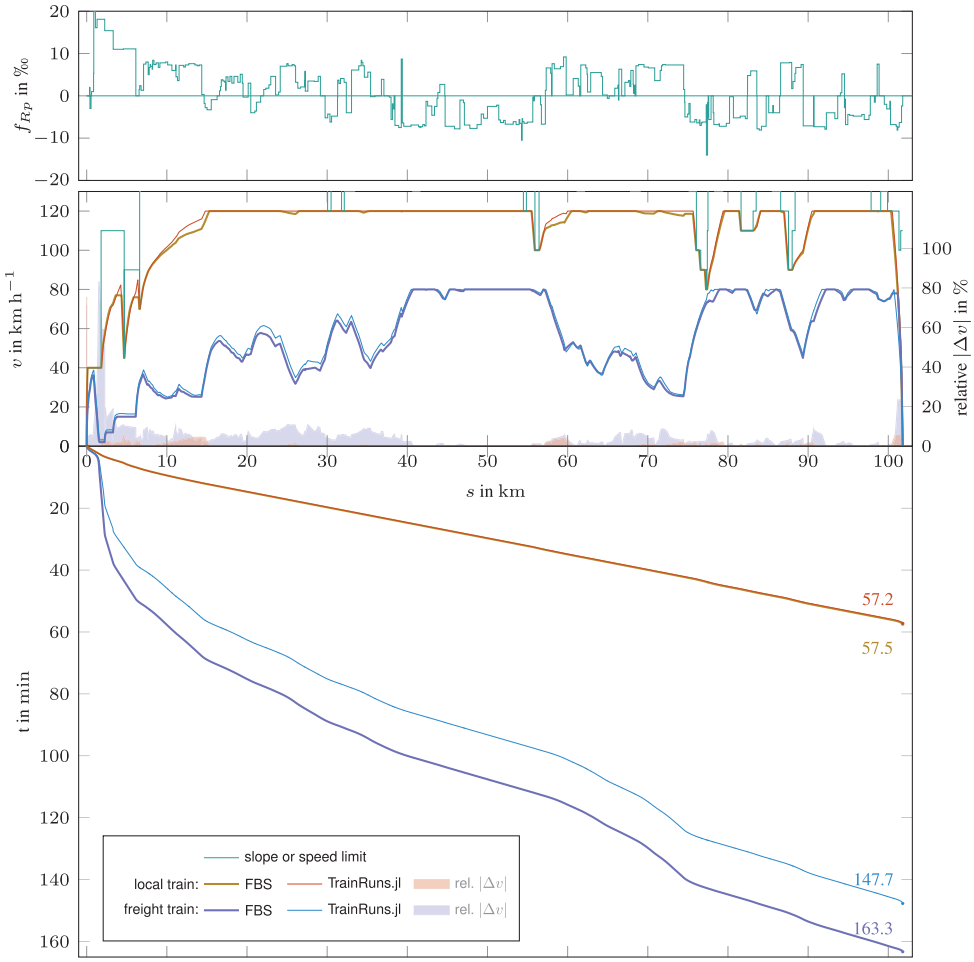

Figure 6: Comparison of the calculation between FBS [23] and TrainRuns.jl for a local train and a freight train on an east Saxony line [24] with a slope (f_{Rp}), showing time (t), speed (v), distance (s), and the relative difference between the two computed speeds for a tain with base to FBS (rel. $|\Delta v|$) are shown. The data are found in Scheidt and Kannenberg [25].

However, TR is not yet fully developed and will be continuously refined over its lifetime. Currently, the emphasis has been on the acceleration phase. Nevertheless, TR is a step toward open science and open data in railway engineering. This tool is suitable for qualitative calculations to compare different trains and is publicly available. We invite others to collaborate. The further development of TR is determined by whether it succeeds in forming a community around the tool and ensuring its usability for its purpose in research and teaching. The following is planned for the further development of TR: a braking model, energy-efficient driving, and visualization.

ACKNOWLEDGEMENTS

We acknowledge support by the German Research Foundation and the Open Access Publication Funds of the Technische Universität Braunschweig.

REFERENCES

[1] Kannenberg, M. & Scheidt, M., Trainruns.jl, Version: v1.0.1, 2022. DOI: 10.5281/zenodo.6448563.
[2] Lehmann, H., *Fahrdynamik der Zugfahrt Theorie und Anwendung*, Shaker: Aachen, 2005.
[3] Wende, D., *Fahrdynamik des Schienenverkehrs*, Springer-Verlag: Wiesbaden, 2003.
[4] Brünger, O. & Dahlhaus, E., Running time estimation. *Railway Timetabling and Operations*, eds I. A. Hansen & J. Pachl, Eurailpress DVV Media Group, pp. 65–89, 2014.
[5] Heppe, A., Fengler, W. & Mersiovsky, J., Untersuchungen der fahrweise von zügen anhand von betriebsdaten und fahrtenschrieben. *ETR Eisenbahntechnische Rundschau*, **2012**(10), pp. 52–57, 2012.
[6] Hertel, G. & Steckel, J., Eine neue Philosophie der Fahrzeitberechnung für Zugfahrten. *Wissenschaftliche Zeitschrift Hochschule für Verkehrswesen Friedrich List*, pp. 104–111, 1992.
[7] Medeossi, G., Longo, G. & de Fabris, S., A method for using stochastic blocking times to improve timetable planning. *Journal of Rail Transport Planning and Management*, **1**(1), pp. 1–13, 2011.
[8] Kecman, P. & Goverde, R.M.P., Predictive modelling of running and dwell times in railway traffic. *Public Transport*, **7**(3), pp. 295–319, 2015. DOI: 10.1007/s12469-015-0106-7.
[9] Wen, C., Mou, W., Huang, P. & Li, Z., A predictive model of train delays on a railway line. *Journal of Forecasting*, **39**(3), pp. 470–488, 2019. DOI: 10.1002/for.2639.
[10] Pachl, J., *Railway Operation Control*, 4th ed., VTD Rail Publishing: Mountlake Terrace (USA), 2018.
[11] Monecke, L., Müller, L. & Anthes, R., Dynamisonline-train run simulation via the internet. *World Congress Railway Research, 4th*, UIC: Köln, Germany, 2001.
[12] Medeossi, G. & de Fabris, S., Simulation of rail operations. *Handbook of Optimization in the Railway Industry*, Springer International Publishing, pp. 1–24, 2018. DOI: 10.1007/978-3-319-72153-8_1.
[13] Nash, A. & Huerlimann, D., Railroad simulation using OpenTrack. *WIT Transactions on The Built Environment, Computers in Railways IX*, vol. 74, WIT Press: Southampton and Boston, 2004. DOI: 10.2495/CR040641.
[14] Bendfeldt, J.P., Mohr, U. & Müller, L., RailSys, A system to plan future railway needs. *WIT Transactions on The Built Environment, Computers in Railways VII*, vol. 50, WIT Press: Southampton and Boston, 2000. DOI: 10.2495/CR000241.
[15] Cui, Y., Martin, U. & Liang, J., PULSim: User-based adaptable simulation tool for railway planning and operations. *Journal of Advanced Transportation*, **2018**, pp. 1–11, 2018. DOI: 10.1155/2018/7284815.
[16] Völz, W.D., Ermittlung der Leistungsfähigkeit von Knotenpunkten spurgeführter Verkehrssysteme mittels Graphentheorie, PhD thesis, Hannover: Lehrstuhl und Institut für Verkehrswesen, Eisenbahnbau und-betrieb; Technische Universität Hannover, 1976.
[17] Habich, G. & Eickmann, F., Dynamis – ein simulationsmodell zur bearbeitung fahrdynamischer fragestellungen. *ETR Eisenbahntechnische Rundschau*, **39**, pp. 83–87, 1990.

[18] Radtke, A., Müller, L. & Schumacher, A., Dynamis: a model for the calculation of running times for an efficient time-table construction. *WIT Transactions on The Built Environment, Computers in Railways VI*, WIT Press, vol. 37, WIT Press: Southampton and Boston, 1998. DOI: 10.2495/CR980301.
[19] Bezanson, J., Karpinski, S., Shah, V.B. & Edelman, A., *Julia: A Fast Dynamic Language for Technical Computing*, 2012. DOI: 10.48550/ARXIV.1209.5145.
[20] Cui, Y., *User-Based Adaptable High Performance Simulation Modelling and Design for Railway Planning and Operations*, Habilitation, Stuttgart: Technische Universität Stuttgart, 2017.
[21] Scheidt, M. & Pachl, J., TikZ-trackschematic library: A symbology towards a universal graphical description for operational scenarios in railway research. *RailBeijing 2021 – the 9th International Conference on Railway Operations Modelling and Analysis (ICROMA)*, 2021. DOI: 10.24355/dbbs.084-202204140847-0.
[22] Scheidt, M., railtoolkit/schema, Version: 2022.05, 2022. DOI: 10.5281/zenodo.6522824.
[23] iRFP – Institut für Regional- und Fernverkehrsplanung, Das fahrplanbearbeitungssystem FBS, Version: 1.7.9. www.irfp.de/das-fahrplanbearbeitungssystem-fbs.html. Accessed on: 29 May 2022.
[24] railML.org e.V., East Saxony railway network by FBS (railml 2.0, 2.2; timetable, infrastructure, rolling stock), available after login, 2016. www.railml.org/en/user/exampledata.html. Accessed on: 29 May 2022.
[25] Scheidt, M. & Kannenberg, M., *Supplement Data – Trainruns.jl: An Open-Source Tool for Running Time Estimation*, Zenodo, 2022. DOI: 10.5281/zenodo.6842604.
[26] Wilkinson, M.D. et al., The FAIR Guiding Principles for Scientific Data Management and Stewardship. *Scientific Data*, **3**(160018), 2016. DOI: 10.1038/sdata.2016.18.
[27] Scheidt, M., Rolling-stock-data, Version: 2022.06, Zenodo, 2022. DOI: 10.5281/zenodo.6467449.