Hindawi
Journal of Advanced Transportation
Volume 2021, Article ID 8840516, 22 pages
https://doi.org/10.1155/2021/8840516

Research Article

Dynamic Automated Search of Shunting Routes within Mesoscopic Rail-Traffic Simulators

Antonin Kavička[1] **and Pavel Krýže**[2]

[1]*Faculty of Electrical Engineering and Informatics, University of Pardubice, Studentska 95, 532 10 Pardubice, Czech Republic*
[2]*Správa železnic, s. o. (Railway Infrastructure Administration, State Organization), Dlážděná 1003/7, 110 00 Prague, Czech Republic*

Correspondence should be addressed to Antonin Kavička; antonin.kavicka@gmail.com

Received 11 September 2020; Revised 16 October 2020; Accepted 18 March 2021; Published 5 April 2021

Academic Editor: Avinash Unnikrishnan

Software tools using computer simulations are frequently used in the research and optimization of railway transport systems. Such simulations serve to examine different railway traffic scenarios (which typically reflect different timetables and railway infrastructure configurations). During the simulation experiments, it is necessary, among other things, to solve tasks related to the determination of track routes along which individual trains or parts of train sets are moved. Many simulation tools require the basic and alternative permissible track routes to be manually specified before starting the simulations, which is a relatively tedious and time-consuming process. Classical graph algorithms cannot be applied to solve the problem of automatic calculation of the routes because they are unable to take into account the length of the object being moved or recognise changes in the direction of its movement. This article presents original innovative algorithms focused on automated dynamic search of track routes (applying an appropriate optimization criterion), which is performed during simulation experiments within simulators working at the mesoscopic level of detail. The algorithms are based on a mathematical model (represented by a specifically designed weighted digraph) that appropriately reflects the actual track infrastructure. The dynamic calculation of each specific track route for a train or a group of railway vehicles considers both the total train set length and the current railway infrastructure occupancy, including blocked parts of the infrastructure due to intervention of the interlocking system. In addition, the places where the train set movement direction is changed can be identified on each route found. Applications of the algorithms and of the mathematical model of the track layout are demonstrated on a model track infrastructure.

1. Introduction

The research method of computer simulation is frequently used to examine and optimise the operation of railway systems. In this context, software simulators, or simulation tools, specialized to simulate railway traffic are used with advantages. Such tools require decisions associated with various types of operational tasks/situations to be automatically taken during the simulation experiments. Examples include decision on the assignment of alternative platform tracks to arriving delayed trains, search for admissible train routes available for train set shunting in a currently occupied trackage, and train selection (from a set of more than one waiting train) for permission to enter a specific line track.

This article is mainly devoted to automated dynamic computations of track routes required to relocate rail vehicles on a rail infrastructure, intended for inclusion into traffic simulations. Addressing this problem was primarily motivated by efforts to improve the capabilities of current rail-traffic simulators, which often enable static specification of the above track routes to be made only manually (before starting the individual simulation experiments). The possibility of including dynamic computations of track routes for relocating objects (having a given length) into the run of a simulation experiment has a high potential to facilitate (and shorten) the process of simulation experiment parameterization. The dynamic track route computation procedures are based on original algorithms that clearly

represent new (as yet unused) solutions in the rail-traffic simulation domain.

For a simulator to be classified as a credible tool for practical uses, the traffic and decision processes used within the simulation experiments must be such that the simulation should approach the real railway system operation as best as possible. In other words, it is very important that the automated solutions implemented in the simulator should be based on suitable models and methods providing results applicable in real traffic situations.

The present article describes an innovative automated solution of one type of operational problems, namely, determination of the train route topology or shunting route topology for relocation of a train/train set within the railway infrastructure. Information for the specification of the relocation encompasses the relocation object length, starting and final positions (tracks), and current railway yard occupation. The solution has been developed for use in rail-traffic simulators working at the mesoscopic level of detail.

2. Literature Review

The operation of railway systems is examined and optimized by employing a number of different methods and techniques that are supported by various software tools. The main goal of such optimizations is to find such solutions which can be used to help ensure quality traffic. The term quality traffic can be interpreted (with some simplifications) as traffic that (i) basically (with minor deviations) follows the timetable and (ii) uses the service resources economically, i.e., means of transport (such as shunting locomotives), elements of infrastructure (on which rail vehicles are moved), and human resources (e.g., technical personnel working in the field of rail yards). A number of specific optimization tasks can be identified in the complex of provisions to ensure quality traffic, and each task may use suitable models of the infrastructure and models of the traffic and traffic management systems.

One of the important optimization tasks consists in solving the train routing problem. This extensive problem includes, for instance, the following partial tasks: rail line assignment to trains within the extensive areas of railway networks [1, 2]; coordinated assignment of track routes to more than one train in the railway station [3–6]; and identification of suitable track routes for rail vehicle shunting [7]. Among the relevant factors playing a role when solving such problems is the configuration of the specific infrastructure on which the traffic takes place. This implies that one should seek to select the best-fitting track infrastructure model and use the most suitable algorithms for each specific type of examination.

Mathematical structures of the graph type (and their specific implementations), which fall in the graph theory domain, are frequently used when describing a particular track infrastructure. Infrastructure models can be constructed both on directed graphs (digraphs) and on undirected graphs. Original track layout models built on the double vertex graph [8] and polar graph [7, 9] concepts make possible distinction between track segments of switches and station track segments.

Among typical tasks addressed by using graphs is the search for admissible (potentially shortest) track routes on which rail vehicles are shunted. The concept of the original Dijkstra's algorithm [10–12], which is primarily aimed at searching for the single-source shortest paths on a weighted directed graph, can be used with advantage for such purposes. Numerous modifications of this algorithm have also been used when examining railway systems [13–15].

One of the important methods used to investigate and optimise railway systems is computer simulation [16]. Different track layout models are used within software tools specialized to simulate rail traffic [7, 8, 14, 15]. Examples of relevant simulation tools in this domain include OpenTrack [17], RailSys [18], Villon [19], MesoRail [20], NEMO [21], and PULSim [22]. Such tools always apply the same level of detail—microscopic, mesoscopic, or macroscopic. On the contrary, different approaches can apply distinct levels of detail within different parts of a simulator—such simulations are referred to as multiscale simulations [23] or hybrid simulations [24, 25]. The existence of software platforms for combined traffic simulations, capable of examining interactions of different traffic modes (e.g., [26]), is also worth mentioning.

In order to work, the above software tools require data description of the rail infrastructure, which can be available in various formats of the configuration files. A standard exists for this purpose, viz. railML [27, 28] (open-source railway markup language). This standard defines a recommended format of the files for the exchange of data for railway applications. The RailTopoModel [29, 30] is also available: this is a logical object model designed for standardization of the representation of railway infrastructure-related data.

Continuation of that part of the research, the results of which were published in [7], was motivated by the need for specialized functions to be applied in the new simulation tool named MesoRail, serving rail-traffic simulations and working at the mesoscopic level of detail. Among such functions was automated computation of the topologies of the track routes on which relocation objects (such as trains and locomotives) will be moved within the track infrastructure model. For this, the appropriate original algorithms (which, however, have never been published so far) were redesigned and implemented. The algorithms were then tested with success within MesoRail for use in dynamic track route calculations. The track layout model used and the algorithms working on it make up original solutions that have never been used elsewhere (according to the available literature).

3. Models of the Rail Infrastructure

Automated dynamic search for train routes and shunting routes within the railway infrastructure uses a mathematical model, reflecting the track layout examined. The construction of such a model consists of 2 stages.

3.1. Primary Model. Stage 1 includes the creation of a primary model using an undirected edge-weighted graph as specified in Table 1. The edges in the graph represent the individual tracks (or their parts) as well as the track segments of the switches. The edges reflecting the tracks are referred to as destination edges, while the remaining edges are referred to as connecting edges.

This terminology mirrors the fact that tracks can be viewed as targets for rail vehicle relocation whereas switches and track crossings cannot. The graph vertices represent the contact points of the track segments reflected by the edges.

Where switches and crossings are modelled, their different types determining how they can be technically transited must be differentiated. Some examples of how different switches or crossings are represented in graph G_0 are illustrated in Figure 1. Admissible transits in Figure 1 are specified in Table 2.

Each edge in graph G_0 has a weight assigned, expressing the metric length of the track segment. An example of a primary model mirroring a demonstration railway yard (total track length: 3049 m) is shown in Figure 2.

When a specific rail vehicle relocation within the rail infrastructure is required, the starting and final positions are assumed to be associated with specific tracks (represented by appropriate edges in graph G_0). In order to make it possible to distinguish between the opposite track ends, either end of each edge in the graph can be assigned either the plus sign (+) or the minus sign (−). This enables us, for instance, to specify that end of the finish track through which the rail vehicle should reach the finish position. Such edge end labelling can also be interpreted as an alternative labelling of the vertex that is incident with the specific end of the specific edge. For example, the notation ${}^{-}e_5 = v_{12} = {}^{+}e_{20}$ can be used in Figure 2. The assignment of the signs to the opposite edge ends can be based on a rule (for instance, if the graph is placed in the two-dimensional coordinate system, then that track end having a lower value of coordinate x is assigned the plus sign, the opposite end, the minus sign).

The use of this marking was partly inspired by the polar graph concept [7, 9]. Each vertex in a graph of this type is composed of two poles—the plus pole and the minus pole—and each edge incident with a vertex can be classed in one of the following two categories: edges incident with the plus pole and edges incident with the minus pole of the vertex. This principle was loosely applied to the marking of the opposite ends of the undirected edges in graph G_0. Figure 2(b) also illustrates the graphical encapsulation of the groups of edges corresponding to the switch objects (designated with symbol S_i, $i = 1, \ldots, 6$).

3.2. Final Model. Stage 2 includes transformation of the primary model G_0 into the final model (Figure 3), which is represented by a specific weighted digraph G. The specification of this graph is presented in Table 3. If the transformation is used, an ordered pair of appropriate vertices ${}^{i}v_x \in V(G)$, $i = 1, 2$, is formed for each edge $e_x \in E(G_0)$ (that is, the bijective transformation function τ: $E(G_0) \longrightarrow \{[{}^{1}v_x, {}^{2}v_x] \mid {}^{1}v_x, {}^{2}v_x \in V(G), {}^{1}v_x \neq {}^{2}v_x, [{}^{1}v_x, {}^{2}v_x], [{}^{2}v_x, {}^{1}v_x] \notin E(G)\}$ is applied). The two vertices representing a track in the digraph G mirror the fact that the track can be entered/left in two opposite directions.

Furthermore, the vertices in digraph G can be categorised according to whether they can represent the relocation targets or not. Hence, we distinguish between destination vertices (represented by full circles in Figure 3) and connecting vertices (represented by empty circles).

Edges in the digraph G can be categorised as transit edges (shown as full lines in Figure 3) and reverse edges (shown as dashed lines). Transit edge expresses the possibility of transiting through the track modelled with a vertex (that represents a starting point of the transit edge) to adjacent tracks. Reverse edge expresses the fact that a train can change the direction of its motion (=reversal) on the track that is modelled by the starting vertex of the reverse edge.

Two transit edges representing the possibility of transiting between the tracks in 2 directions are constructed in graph G for each pair of adjacent edges (the edge construction methods are explained in Table 4). Reverse edges can start from destination vertices solely. Here, it is assumed that the whole relocation object stops on one track during the reversal operation. A situation where a train can occupy more than one track during reversal is not modelled in the digraph G. This simplification can be applied within simulators operating at the mesoscopic level of detail.

Applying the rule specified in Table 4, a reverse edge is constructed for each transit edge whose end vertex is a destination vertex. For example (referring to the graph in Figure 2), the reverse edge $[{}^{2}v_5, {}^{1}v_{18}]$ is formed to the transit edge $[{}^{2}v_{18}, {}^{2}v_5]$.

Digraph G is both vertex-weighted and edge-weighted, and the vacancies are preserved for the vertices. The following rules are applied to the weights and vacancies:

(i) The weights of the vertices in digraph G (expressed through function ω) represent the metric lengths of the track segments corresponding to the weights of the respective edges in graph G_0.

(ii) The weights of transit edges (expressed through function ε) in digraph G are identical with the weights of the vertices they proceed from. The weight of a specific transit edge represents the distance run by the relocation object by transiting the track that mirrors the starting vertex of that edge.

(iii) The weights of reverse edges (expressed through function ε) are identical for all those edges and do not attain fixed values: instead, they are set (to value L) always before running the computing algorithm searching for a given train or shunting route (for a relocation object whose length is L). This weight is interpreted so that if the relocation object stops on a certain track (corresponding to the starting vertex of a reverse edge) for a time and then continues by moving in the opposite direction, then the used part of the track is that part whose length is identical with that of the relocation object (i.e., L).

Table 1: Specification of the weighted undirected graph G_0—the primary model of the track infrastructure.

Symbols	Specifications
G_0	The weighted undirected graph (i) $G_0 = (V, E, \varphi, \varepsilon)$
$V(G_0)$	The set of vertices of the graph G_0 (i) $\lvert V(G_0)\rvert = n_0$
$E(G_0)$	The set of edges of the graph G_0 (i) $\lvert E(G_0)\rvert = m_0$ (ii) $E(G_0) = E_{\text{dest}}(G_0) \cup E_{\text{conn}}(G_0)$, $E_{\text{dest}}(G_0) \cap E_{\text{conn}}(G_0) = \varnothing$ (iii) The set $E_{\text{dest}}(G_0)$ contains destination edges (iv) The set $E_{\text{conn}}(G_0)$ contains connecting edges
φ	The incidence function related to the graph G_0 (i) φ: $E(G_0) \longrightarrow \{(v_x, v_y) \mid (v_x, v_y) \in V(G_0) \times V(G_0) \wedge v_x \neq v_y\}$
ε	The edge weight function related to the graph G_0 (i) ε: $E(G_0) \longrightarrow R^+$ (the set of positive real numbers)

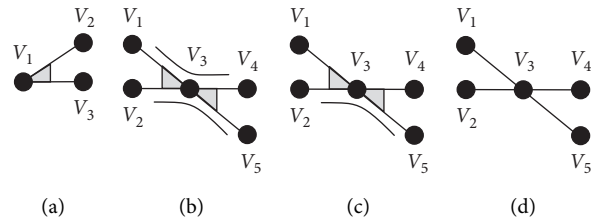

Figure 1: The models of switches and crossings applied within undirected graphs.

Table 2: Admissible transits through several kinds of switches and crossings.

The type of switch	Admissible transits
Switch (Figure 1(a))	$v_1 \longrightarrow v_2$, $v_2 \longrightarrow v_1$, $v_1 \longrightarrow v_3$, $v_3 \longrightarrow v_1$
Double slip switch (Figure 1(b))	$v_1 \longrightarrow v_5$, $v_5 \longrightarrow v_1$, $v_1 \longrightarrow v_4$, $v_4 \longrightarrow v_1$, $v_2 \longrightarrow v_4$, $v_4 \longrightarrow v_2$, $v_2 \longrightarrow v_5$, $v_5 \longrightarrow v_2$
Single slip switch (Figure 1(c))	$v_1 \longrightarrow v_5$, $v_5 \longrightarrow v_1$, $v_2 \longrightarrow v_4$, $v_4 \longrightarrow v_2$, $v_2 \longrightarrow v_5$, $v_5 \longrightarrow v_2$
Track crossing (Figure 1(d))	$v_1 \longrightarrow v_5$, $v_5 \longrightarrow v_1$, $v_2 \longrightarrow v_4$, $v_4 \longrightarrow v_2$

(iv) Furthermore, every vertex in digraph G has an available vacancy (expressed through function κ), expressing the metric free capacity of the respective track. Since each track in digraph G is modelled with a pair of vertices, the vacancies of the vertices express the metric lengths of the vacant parts of the track from the two opposite ends. Function κ can be used to express the rail infrastructure occupation by rail vehicles and also the blocking if its parts are unreachable due to the operation of the interlocking system (for instance, if a certain track x is blocked, then the equation $\kappa({}^1v_x) = \kappa({}^2v_x) = 0$ holds for the graph vertices ${}^1v_x, {}^2v_x \in V(G)$).

Examples of particular weight and vacancy values belonging to selected elements in digraph G are shown in the following (Example 1).

4. Algorithms Focused on Searching Train Routes

Now, after the final railway infrastructure model has been introduced, the algorithms for calculating train routes or shunting routes can be described. The search of the routes was based on the optimization principle consisting in minimisation of the distance run by the relocation object in question.

Dijkstra's algorithm [10, 11] for searching the shortest routes between the vertices on an edge-weighted graph was selected as the starting algorithm. However, it had to be appreciably modified for use in the calculation of the relocation trajectories for trains or train sets (having a defined length) on the railway infrastructure.

As mentioned earlier, the railway infrastructure was modelled by digraph G, specified in Table 3.

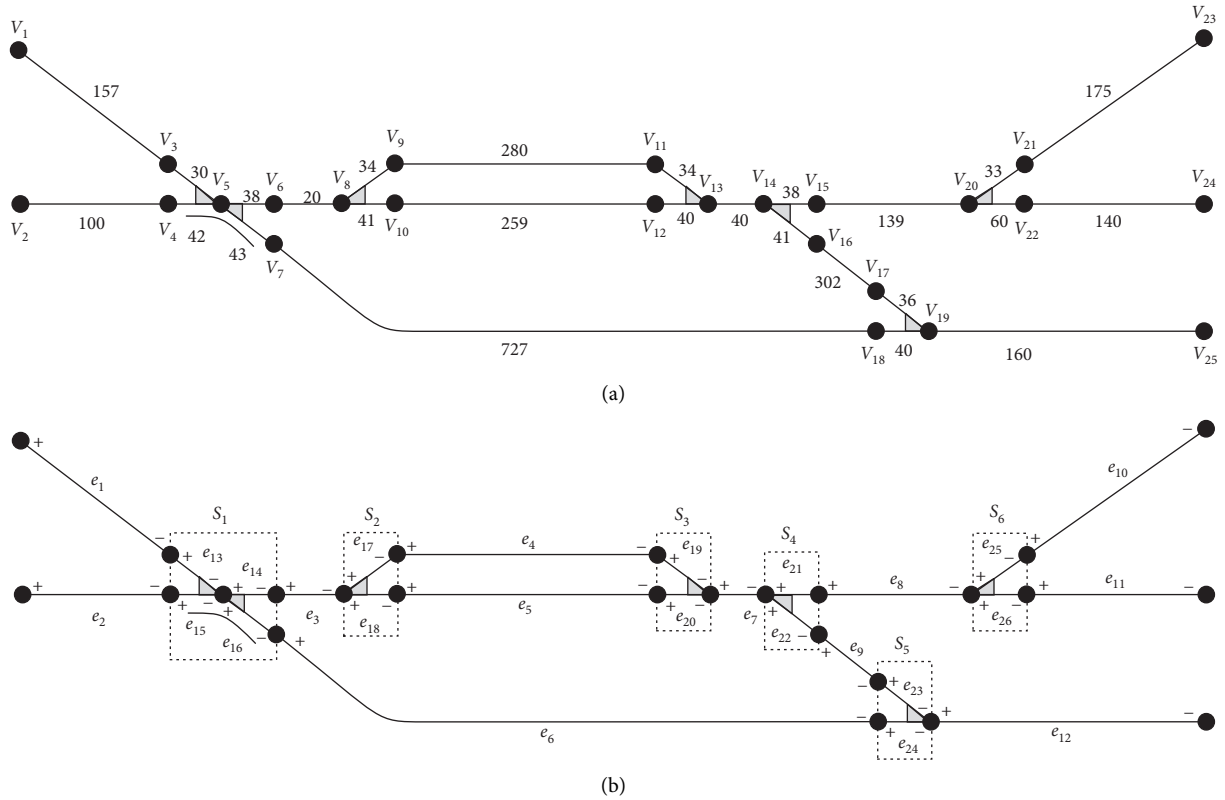

Figure 2: Prototype track infrastructure—a primary model based on an undirected graph G_0.

Now, additional auxiliary sets, row vectors, and subroutines/functions to be used in the algorithms must be specified in order to enable the algorithms to be formalized. The specifications are listed in Tables 5 and 6. The subroutines/functions utilise parameters applying the following convention: the symbol "↓" denotes an input parameter, symbol "↑" denotes an output parameter, and double symbol "↓↑" denotes an input-output parameter.

4.1. Basic Algorithm. The basic algorithm (formalized as Algorithm 1) searches for the shortest route in the track layout from a specific start vertex to a specific finish vertex. The start vertex represents the end of the start track transited by the train (whole length is L) when starting its relocation procedure, while the finish vertex represents the end of the finish track transited by the train when approaching the finish position.

The main function, *Shortest_Path*, calculates the shortest relocation route for an object (whose length is L), given the start vertex (from the set S_V) and finish vertex (from the set F_V). The topology of this route is available through the parameter *Topol*. For Algorithm 1, the sets S_V and F_V include one vertex each. Further modifications of this algorithm, however, will permit one or two vertices to be included in either set.

The following procedures are carried out within the subroutines/functions (called from the main function *Shortest_Path*).

First, the function *Start_Finish_Test* tests admissible combinations of the input parameter values for the route calculation. For the combination to be acceptable, the weight of the start vertex must not be lower than the relocation object length (L), and the finish vertex must have a sufficient vacancy. Furthermore, the Boolean expression $(x = y \wedge i \neq j)$ must not hold true for the start vertex ${}^{i}v_x$ and finish vertex ${}^{j}v_y$ (the above expression describes a situation where the start and finish vertices refer to the same track from which the relocation procedure is started and which is entered by transiting the same end).

Furthermore, the function *General_Init* is used for initialisation of the marks of distances and the marks of predecessors (through vectors D and P) for all vertices in the graph. The distance marks of all vertices in the digraph G are set at the value d^{∞} (which is larger than the longest distance between any vertices in the digraph G). The marks of predecessors are set at none for all vertices in the digraph G. The set T_V of temporarily marked vertices and the set U_V of ultimately marked finish vertices are initialised as empty sets.

Additional initialisation procedures associated with the start and finish vertices are subsequently run (through the function *Start_Finish_Init*). The start vertex is assigned a distance mark whose value is 0 (i.e., the length of the current shortest route is zero for the start vertex). Now, the set of forbidden vertices is constructed. The vertices from this set must not lie on the shortest route being sought. When the concept of forbidden vertices is applied, the relocation

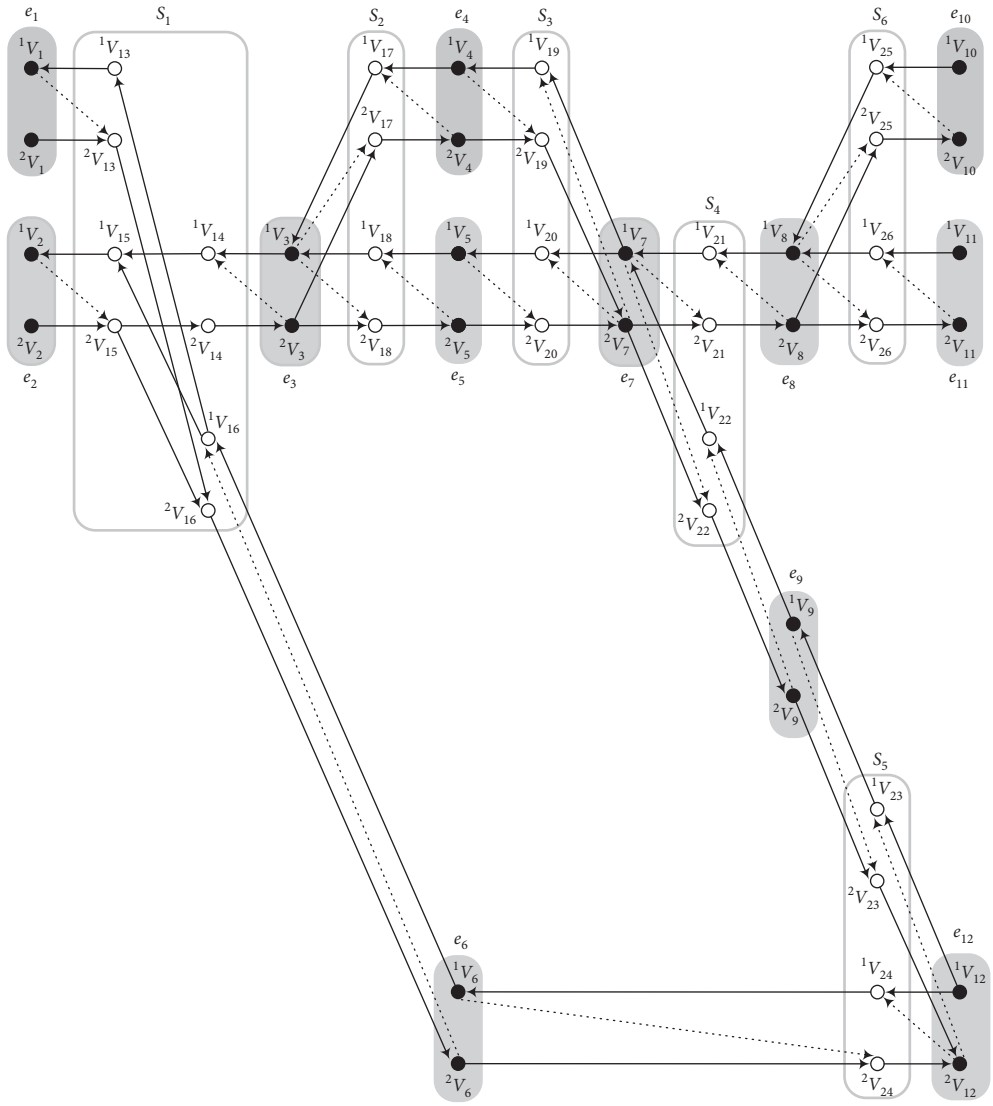

Figure 3: The final infrastructure model based on digraph G.

trajectory must not pass through vertices describing those ends of tracks over which it is inadmissible to reach the finish track (with respect to the input requirements for relocation).

Next, the admissible successors of the start vertices that can be reached through transit edges are remarked (through the function *Start_Mark*). These vertices-successors are assigned (i) distance marks with a value of 0 (through vector D) and (ii) marks of predecessor (in vector P) referring to the start vertex. The remarked vertices are also inserted into the set T_V.

The complex of initialisation procedures is followed by a computation cycle involving selection of a vertex (through the function *Vert_Select*) for processing in the current iteration step and the marking of all its admissible successors which is potentially changed (through the function *Succ_Mark*). This computation cycle is terminated if all the appropriate finish vertices have been ultimately marked or if T_V—the set of temporarily marked vertices—is empty.

The *Vert_Select* function always selects that vertex ${}^k v_z$ from the set T_V that has the lowest value of the distance mark ${}^k d_z$. The *Succ_Mark* function changes the marking of the accessible successors of the selected vertex ${}^k v_z$ if those successors (i) have an available adequate vacancy; (ii) they are not members of the set of forbidden vertices; and (iii) the current values of their distance marks are higher than the new values mirroring their reaching through another route (via vertex v_z). Moreover, the Boolean expression $(\kappa({}^k v_z) = \omega({}^k v_z))$ must hold true for the transit successors to be accessible from the vertex ${}^k v_z$. This expresses the condition that the track to which the vertex ${}^k v_z$ refers must be fully available for transiting. This means that the vacancy of the vertex ${}^k v_z$ must be equal to the weight of that vertex.

If the shortest route has been found, its topology is obtained through the function *Get_Path*. The topology is constructed stepwise from the finish vertex to the start vertex by using marks about the vertex predecessors (saved in the row vector P) on the shortest route.

Table 3: Specification of the weighted digraph G—the final model of the track infrastructure.

Symbols	Specifications
G	The weighted digraph (i) $G = (V, E, \varphi, \omega, \varepsilon, \kappa)$ (ii) The digraph G represents a result related to the transformation of the relevant undirected graph G_0
$V(G)$	The set of vertices of the digraph G (i) $V(G) = \{{}^k v_z \mid z = 1, \ldots, n\backslash 2,\ k = 1, 2\},\ \lvert V(G)\rvert = n$ (ii) $V(G) = V_{\text{dest}}(G) \cup V_{\text{conn}}(G),\ V_{\text{dest}}(G) \cap V_{\text{conn}}(G) = \varnothing$ (iii) The set $V_{\text{dest}}(G)$ contains destination vertices of the digraph G (iv) The set $V_{\text{conn}}(G)$ contains connecting vertices of the digraph G Note: symbol "\" denotes an integer division
$E(G)$	The set of directed edges of the digraph G (i) $\lvert E(G)\rvert = m$ (ii) $E(G) = E_{\text{trans}}(G) \cup E_{\text{rev}}(G),\ E_{\text{trans}}(G) \cap E_{\text{rev}}(G) = \varnothing$ (iii) The set $E_{\text{trans}}(G)$ contains transit edges (iv) The set $E_{\text{rev}}(G)$ contains reverse edges
φ	The incidence function related to the digraph G (i) φ: $E(G) \longrightarrow \{[{}^i v_x, {}^j v_y] \mid [{}^i v_x, {}^j v_y] \in V(G) \times V(G) \wedge {}^i v_x \neq {}^j v_y;\ x, y \in <1, \ldots, n\backslash 2>;\ i, j \in \{1, 2\}\}$
${}^{\text{out}}V({}^k v_z)$	The set of successors of the vertex ${}^k v_z$ (i) ${}^{\text{out}}V({}^k v_z) = \{{}^l v_s \mid [{}^k v_z, {}^l v_s] \in E(G)\},\ {}^{\text{out}}V({}^k v_z) \subset V(G)$ (ii) ${}^{\text{out}}V({}^k v_z) = {}^{\text{out}}V_{\text{trans}}({}^k v_z) \cup {}^{\text{out}}V_{\text{rev}}({}^k v_z),\ {}^{\text{out}}V_{\text{trans}}({}^k v_z) \cap {}^{\text{out}}V_{\text{rev}}({}^k v_z) = \varnothing$ (iii) The set of transit successors of the vertex ${}^k v_z$ ${}^{\text{out}}V_{\text{trans}}({}^k v_z) = \{{}^l v_s \mid [{}^k v_z, {}^l v_s] \in E_{\text{trans}}(G)\}$ (iv) The set of reverse successors of the vertex ${}^k v_z$ ${}^{\text{out}}V_{\text{rev}}({}^k v_z) = \{{}^l v_s \mid [{}^k v_z, {}^l v_s] \in E_{\text{rev}}(G)\}$
ω	The vertex weight function related to the digraph G (i) ω: $V(G) \longrightarrow R^+$ (ii) $\forall\ {}^1 v_z, {}^2 v_z \in V(G)$ $(z = 1,\ldots, n\backslash 2)$: $\omega({}^1 v_z) = \omega({}^2 v_z)$
ε	The edge weight function related to the digraph G (i) ε: $E(G) \longrightarrow R^+$ (ii) $\forall\ [{}^i v_x, {}^j v_y] \in E_{\text{trans}}$: $\varepsilon([{}^i v_x, {}^j v_y]) = \omega({}^i v_x)$ (iii) $\forall\ [{}^i v_x, {}^j v_y] \in E_{\text{rev}}$: $\varepsilon([{}^i v_x, {}^j v_y]) = L$, where L represents a parameter value reflecting the length of a relevant relocation object
κ	The vertex vacancy function (vacant capacity function) (i) κ: $V(G) \longrightarrow R_0^+$ (ii) If a track segment (reflected by a vertex ${}^k v_z \in V(G)$) is completely vacant, then $\kappa({}^k v_z) = \omega({}^k v_z)$

Table 4: Construction of edges belonging to the digraph G transformed from a related graph G_0.

Adjacent edges $\varphi(e_x) \cap \varphi(e_y) \neq \varnothing$ $e_x, e_y \in E(G_0),\ e_x \neq e_y$	Constructed transit edges $E_{\text{trans}}(G)$	Constructed reverse edges	
		$E_{\text{rev}}(G)$	Only if
${}^+e_x = {}^-e_y$	$[{}^1 v_x, {}^1 v_y],\ [{}^2 v_y, {}^2 v_x]$	$[{}^1 v_y, {}^2 v_x]$ $[{}^2 v_x, {}^1 v_y]$	${}^1 v_y \in V_{\text{dest}}(G)$ ${}^2 v_x \in V_{dest}(G)$
${}^-e_x = {}^+e_y$	$[{}^2 v_x, {}^2 v_y],\ [{}^1 v_y, {}^1 v_x]$	$[{}^2 v_y, {}^1 v_x]$ $[{}^1 v_x, {}^2 v_y]$	${}^2 v_y \in V_{\text{dest}}(G)$ ${}^1 v_x \in V_{\text{dest}}(G)$
${}^+e_x = {}^+e_y$	$[{}^1 v_x, {}^2 v_y],\ [{}^1 v_y, {}^2 v_x]$	$[{}^2 v_y, {}^2 v_x]$ $[{}^2 v_x, {}^2 v_y]$	${}^2 v_y \in V_{\text{dest}}(G)$ ${}^2 v_x \in V_{\text{dest}}(G)$
${}^+e_x = {}^-e_y$	$[{}^2 v_x, {}^1 v_y],\ [{}^2 v_y, {}^1 v_x]$	$[{}^1 v_y, {}^1 v_x]$ $[{}^1 v_x, {}^1 v_y]$	${}^1 v_y \in V_{\text{dest}}(G)$ ${}^1 v_x \in V_{\text{dest}}(G)$

At this point, the following comments should be added concerning the implementation of the above algorithm, transformation of its results, and an alternative view upon the start and finish points of the relocation trajectories:

(i) In view of the nature of the algorithm, a data structure called forward star [31] was used for implementation of the digraph G. The forward star structure consists of two basic arrays: a primary array that stores information on the graph vertices and a secondary array that stores information on the directed edges that start from the vertices (saved in the primary array). The secondary array is sorted by the starting edge vertices. This makes it possible for each vertex from the primary array to rapidly access (through its reference to the secondary array) the group of all its successors (which are always stored in the continuous part of the secondary array). For an effective work with the set T_V, the Fibonacci heap [11], as a very efficient realization of a priority queue, was chosen for its implementation. The

Table 5: Specifications of auxiliary sets and row vectors.

Symbols	Specifications
S_V	The set of start vertices (i) $S_V \subset V(G)$
F_V	The set of finish vertices (i) $F_V \subset V(G)$
X_V	The set of forbidden vertices (i) $X_V \subset V(G)$
T_V	The set of temporarily marked vertices (i) $T_V \subset V(G)$
U_V	The set of ultimately marked finish vertices (i) $U_V \subset V(G)$
D	The row vector of distances (i) $D = \|\|{}^k d_z\|\|$, ${}^k d_z \in R_0^+$, $z = 1, \ldots, n\backslash 2$, $k = 1, 2$ (ii) Each vertex ${}^k v_z \in V(G)$ is tagged by a mark ${}^k d_z$, which expresses the length of the currently detected shortest path from a vertex ${}^i v_x \in S_V$ to the vertex ${}^k v_z$ (iii) If the above path to the vertex ${}^k v_z \in V(G)$ does not exist, then ${}^k d_z = d^\infty$ ($d^\infty \in R_0^+$, and it is equal to the value that is greater than the length of the longest admissible path in the graph G) Note: symbol "\" denotes an integer division
P	The row vector of predecessors (i) $P = \|\|{}^k p_z\|\|$, ${}^k p_z \in V(G) \cup \{none\}$, $z = 1, \ldots, n\backslash 2$, $k = 1, 2$ (ii) Each vertex ${}^k v_z \in V(G)$ is tagged by a mark ${}^k p_z$, which corresponds to a predecessor of the vertex ${}^k v_z$ on the currently detected shortest path from a vertex ${}^i v_x \in S_V$ to the vertex ${}^k v_z$ (iii) If the above path to the vertex ${}^k v_z \in V(G)$ does not exist, then ${}^k p_z = none$ (the symbol *none* expresses the nonexistence of a relevant predecessor with regard to the vertex ${}^k v_z$)
Seq	The linearly ordered set of the shortest path topology (i) $Seq = \{[i, {}^k v_z] \mid i = 0, \ldots, q-1, {}^k v_z \in V(G), z \in \langle 1, \ldots, n\backslash 2\rangle, k \in \{1, 2\}, q = \|Seq\|\}$ (ii) The element $[0, {}^k v_z]$ represents a finish vertex (${}^k v_z \in F_V$) and the element $[q-1, {}^k v_z]$ a start vertex (${}^k v_z \in S_V$) of the shortest path

priorities of the elements (or vertices in the digraph G) expressed the values of the respective marks from the row vector D (the higher the priority of an element, the lower the value of its distance mark).

(ii) The configuration of the railway infrastructure can be specified for the needs of the MesoRail simulator [20], for example, within the TrackEd editor [24]. This editor stores infrastructure description in the XML format, which uses templates inspired by the railML standard [27, 28].

(iii) For the implementation approaches used (applying Dijkstra's algorithm concept), the asymptotical computational complexity of Algorithm 1 can be specified as $O(m + n \log n)$ [12], where $m = |E(G)|$ and $n = |V(G)|$. Nevertheless, for typical track routes that are not searched for long distances in real operational conditions, the relevant computations (within rail-traffic simulators) are concentrated on subgraphs (of the digraph G) that are usually not very extensive.

(iv) The result of Algorithm 1 computations for the shortest route found in the digraph G can be reversely transformed into the primary model, that is, into the undirected graph G_0. This means that the shortest route in the digraph G corresponds to the shortest walk in the graph G_0. The walks are outlined graphically in Figures 4–6, illustrating the solution of Examples 1–3 (see later).

(v) In standard parameterisation of the *Shortest_Path* function within Algorithm 1, the start and finish positions of the relocation procedure are assumed to be represented by vertices in the digraph G. This, in fact, means that the relocation object (whose length is L) is located "precisely" at the end of the respective track both at the beginning and at the end of the relocation operation. If it is required that the relocation operation starts and/or finishes on another part of the track, then Algorithm 1 must be slightly modified. The modification will encompass both the marking of the start vertex (and potentially its transit successors as well) and the ultimate marking of the finish vertex. The values of the distance marks assigned to the vertices within the *Start_Finish_Init* and *Start_Mark* functions will be set at a value different from 0 (denoted, e.g., $reloc_1$) describing the metric length of relocation to the postulated end of the start track. On the contrary, the value of the ultimate mark of the finish vertex of the relocation (denoted, for instance, *val*, and indicating the length of the shortest route found) will not describe the length of the relocation object trajectory precisely (denoted, for instance, *dist*). In fact, the length of the trajectory can be obtained as $dist = val + reloc_2$, where $reloc_2$ refers to the metric length of relocation from the postulated end of the finish track to the end position on that track.

Table 6: Specifications of subprograms/functions.

Subprograms/functions	Specifications
Shortest_Path($\downarrow S_V$, $\downarrow F_V$, $\downarrow L$, $\uparrow Topol$)	The calculation of the shortest path (the graph G) S_V (the set of start vertices) F_V (the set of finish vertices) L (the length of the relocation object) *Topol* (the topology of the shortest path)
Start_Finish_Test($\downarrow S_V$, $\downarrow F_V$, $\downarrow L$, $\uparrow\downarrow okay$)	The test of correctness of start and finish vertices S_V (the set of start vertices) F_V (the set of finish vertices) L (the length of the relocation object) *okay* (the test result)
General_Init()	The execution of general initialisation activities The initialisation of the sets T_V and U_V The initialisation of the row vectors P and D
Start_Finish_Init($\downarrow S_V$, $\downarrow F_V$)	The initialisation activities reflecting start vertices and finish vertices The initialisation of the set X_V The initialisation of the marks ${}^{i}d_x$ (${}^{i}d_x = 0$) related to the start vertices (${}^{i}v_x \in S_V$)
Start_Mark($\downarrow S_V$,$\downarrow L$)	The initialisation activities focused on markings of the successors of start vertices The update of the set T_V and the row vectors P and D
Vert_Select($\uparrow {}^{k}v_z$)	The selection of the vertex ${}^{k}v_z$ (with the current minimal distance mark) for the processing in the next step of the algorithm The vertex ${}^{k}v_z$ is selected/removed from the set T_V The set U_V is potentially updated
Succ_Mark($\downarrow {}^{k}v_z$, $\downarrow L$)	The potential execution of admissibly rewriting the marks belonging to all successors of the vertex ${}^{k}v_z$ The potential update of the set T_V and the row vectors P and D
Get_Path($\uparrow\downarrow Seq$)	The delivery of the topology related to the found shortest path *Seq* represents a sequence of vertices
Get_Indexes($\downarrow {}^{i}v_x$, $\uparrow i$, $\uparrow x$)	The auxiliary function delivering indexes of a vertex ${}^{i}v_x$
Min_Dist($\downarrow A$, $\uparrow {}^{k}v_z$)	The auxiliary function identifying a vertex ${}^{k}v_z$ (within a set A) associated with the minimum value of the relevant mark ${}^{k}d_z$
Try_Change_Mark($\downarrow {}^{k}v_z$, $\downarrow {}^{l}v_s$)	The auxiliary function potentially rewriting the marks belonging to the successor (the vertex ${}^{l}v_s$) of the vertex ${}^{k}v_z$ The potential update of the set T_V and the row vectors P and D
Predecessor($\downarrow {}^{k}v_z$, $\uparrow l$, $\uparrow t$)	The auxiliary function delivering indexes related to a predecessor (${}^{l}v_t$) of the vertex ${}^{k}v_z$ (on the relevant shortest path)

4.2. Examples of Using the Basic Algorithm. A few examples of the application of the basic algorithm (Algorithm 1) to the computation of various relocation actions are shown in the following. The first example concerns the transfer of a complete train between two station tracks.

Example 1. Computation of a shunting route topology for the train relocation ($L = 120$).

Figure 4 shows two models of a demonstration railway yard represented by graphs G and G_0. There is one train 120 m long (relocation object O_1) on track #5 (reflected by edge e_5 in graph G_0) of the yard. The train stands at the track end, referred to as ${}^{-}e_5$. The requirement is to find a route for transferring the object O_1 onto track #4 (represented by edge e_4 in graph G_0). The object O_1 should leave track #5 via the track end referred to as ${}^{-}e_5$ and enter track #4 via its end referred to as ${}^{-}e_4$. This setup reflects a situation where track #5 should be made available for other trains and the train in question should be transferred to track #4, so it can leave the yard in the direction to a rail line accessible via track #11.

Parameterization of the algorithm for the track route computation (shortest route in the digraph G) is as follows:

(i) $S_V = \{{}^{2}v_5\}$, $F_V = \{{}^{1}v_4\}$, and $L = 120$

The weights are given for selected vertices and edges of the relevant graphs, and current vacancies are quantified for selected vertices in the digraph G:

(i) $\omega\ ({}^{1}v_4) = \omega\ ({}^{2}v_4) = 280$, $\kappa\ ({}^{1}v_4) = \kappa\ ({}^{2}v_4) = 280$, ${}^{1}v_4$, ${}^{2}v_4 \in V(G)$, $\varepsilon\ (e_4) = 280$, $e_4 \in E(G_0)$

(ii) $\omega\ ({}^{1}v_5) = \omega\ ({}^{2}v_5) = 259$, $\kappa\ ({}^{1}v_5) = 0$, $\kappa\ ({}^{2}v_5) = 139$, ${}^{1}v_5, {}^{2}v_5 \in V(G)$, $\varepsilon\ (e_5) = 259$, $e_5 \in E(G_0)$

(iii) $\varepsilon\ ([{}^{1}v_4, {}^{1}v_{17}]) = \varepsilon\ ([{}^{2}v_4, {}^{2}v_{19}]) = 280$, ${}^{1}v_4, {}^{2}v_4$, ${}^{1}v_{17}$, ${}^{2}v_{19} \in V(G)$, $[{}^{1}v_4, {}^{1}v_{17}]$, $[{}^{2}v_4, {}^{2}v_{19}] \in E_{\text{trans}}\ (G)$

(iv) $\forall\ [{}^{i}v_x, {}^{j}v_y] \in E_{\text{rev}}\ (G)$: $\varepsilon\ ([{}^{i}v_x, {}^{j}v_y]) = 120$

The topology of the train route found (which corresponds to the shortest route in the digraph G and can be retransformed into the shortest walk in the graph G_0) is as follows:

$$e_4(350) \leftarrow e_{19}(316) \leftarrow e_7(276) \leftarrow e_{21}(238) \leftarrow e_8(118) \leftarrow e_{21}(80) \leftarrow e_7(40) \leftarrow e_{20}(0) \leftarrow e_5(0)$$

The numbers in parentheses (following each member of the walk) are the metric lengths of the relocation object's

```
(1) function Shortest_Path(↓S_V, ↓F_V, ↓L, ↑Topol)
(2)   Topol ← ∅
(3)   correct ← true
(4)   Start_Finish_Test(↓S_V, ↓F_V, ↓L, ↑↓correct) // admissibility of the start/finish vertices
(5)   if correct then
(6)     General_Init() // setting initial marks of all vertices
(7)     Start_Finish_Init(↓S_V, ↓F_V) // initialisation activities related to the start/finish vertices
(8)     Start_Mark(↓S_V, ↓L) // remarking of transit successors of the start/finish vertices
(9)     repeat
(10)      if T_V ≠ ∅ then
(11)        Vert_Select(↑^k v_z) // selection of a new current vertex
(12)        if U_V ≠ F_V then
(13)          Succ_Mark(↓^k v_z, ↓L) // remarking successors of the current vertex ^k v_z
(14)        end
(15)      end
(16)    until (T_V = ∅ or U_V = F_V) // algorithm termination testing
(17)    Get_Path(↑↓Topol) // getting a topology of the shortest path
(18)  end
(19) end
(20) function Start_Finish_Test(↓S_V, ↓F_V, ↓L, ↓↑okay)
(21)  if (S_V = ∅ or F_V = ∅) then
(22)    okay ← false
(23)    exit
(24)  end
(25)  for each ^i v_x ∈ S_V do
(26)    Get_Indexes(↓^i v_x, ↑i, ↑x)
(27)    for each ^j v_y ∈ F_V do
(28)      Get_Indexes(↓^j v_y, ↑j, ↑y)
(29)      if (x = y and i ≠ j) then
(30)        okay ← false // inadmissible combination of the start and finish vertices
(31)        exit
(32)      end
(33)      if (ω (^i v_x) < L or κ (^j v_y) < L) then
(34)        okay ← false // inadmissible weights/vacancies of the start/finish vertices
(35)        exit
(36)      end
(37)    end
(38)  end
(39) end
(40) function General_Init()
(41)  for z = 1 to n\2 do // symbol "\" denotes an integer division
(42)    for k = 1 to 2 do
(43)      ^k d_z ← d^∞ // initialisation of the row vector of distance marks
(44)      ^k p_z ← none // initialisation of the row vector of marks-predecessors
(45)    end
(46)  end
(47)  T_V ← ∅
(48)  U_V ← ∅
(49) end
(50) function Start_Finish_Init(↓S_V, ↓F_V)
(51)  for each ^i v_x ∈ S_V do
(52)    Get_Indexes(↓^i v_x, ↑i, ↑x)
(53)    for each ^j v_y ∈ F_V do
(54)      Get_Indexes(↓^j v_y, ↑j, ↑y)
(55)      a ← (3 − i)
(56)      b ← (3 − j)
(57)      if S_V = F_V then
(58)        X_V ← {^a v_x} // the forbidden vertex ^a v_x is a pair vertex to the start vertex ^i v_x
(59)      else
(60)        X_V ← {^a v_x, ^b v_y} // the forbidden vertex ^b v_y is a pair vertex to the finish vertex ^j v_y
(61)        ^i d_x ← 0 // initialisation of selected distance marks
```

Algorithm 1: Continued.

```
(62)          end
(63)       end
(64)    end
(65) end
(66) function Start_Mark(↓S_V, ↓L)
(67)    for each ^k v_z ∈ S_V do
(68)       for each ^l v_t ∈ ^out V_trans (^k v_z) do
(69)          if (^l v_t ∉ X_V and κ (^l v_t) ≥ L) then
(70)             T_V ← T_V ∪ {^l v_t} // insertion of admissible transit successors into the set T_V
(71)             ^l d_t ← 0 // initialisation of selected distance marks
(72)             ^l p_t ← ^k v_z // initialisation of selected marks-predecessors
(73)          end
(74)       end
(75)    end
(76) end
(77) function Vert_Select(↑^k v_z)
(78)    if T_V ≠ ∅ then
(79)       Min_Dist(↓T_V, ↑^k v_z) // selection of a vertex ^k v_z with the lowest distance mark ^k d_z
(80)       T_V ← T_V − {^k v_z}
(81)       if ^k v_z ∈ F_V then
(82)          U_V ← U_V ∪ {^k v_z}
(83)       end
(84)    end
(85) end
(86) function Try_Change_Mark(↓^k v_z, ↓^l v_s)
(87)    Get_Indexes(↓^k v_z, ↑k, ↑z)
(88)    Get_Indexes(↓^l v_s, ↑l, ↑s)
(89)    if ^l d_s > ^k d_z + ε ([^k v_z, ^l v_s]) then
(90)       ^l d_s ← ^k d_z + ε ([^k v_z, ^l v_s]) // remarking of the successor (^l v_s) of the vertex ^k v_z
(91)       ^l p_s ← ^k v_z
(92)       if ^l v_s ∉ T_V then
(93)          T_V ← T_V ∪ {^l v_s}
(94)       end
(95)    end
(96) end
(97) function Succ_Mark(↓^k v_z, ↓L)
(98)    for each ^l v_t ∈ ^out V_trans (^k v_z)do
(99)       if (^l v_t ∉ X_V and κ (^l v_t) ≥ L and κ (^k v_z) = ω (^k v_z)) then
(100)         Try_Change_Mark(↓^k v_z, ↓^l v_t) // potential remarking of the transit successor of ^k v_z
(101)      end
(102)   end
(103)   for each ^l v_r ∈ ^out V_rev(^k v_z) do
(104)      if (^l v_r ∉ X_V and κ (^l v_r) ≥ L) then
(105)         Try_Change_Mark(↓^k v_z, ↓^l v_r) // potential remarking of the reverse successor of ^k v_z
(106)      end
(107)   end
(108) end
(109) function Get_Path(↑↓Seq)
(110)   if U_V ≠ ∅ then
(111)      Min_Dist(↓U_V, ↑^j v_y)
(112)      Get_Indexes(↓^j v_y, ↑j, ↑y)
(113)      z ← y
(114)      k ← j
(115)      i ← 0
(116)      while (^k p_z ≠ none or ^k v_z ∈ S_V) do
(117)         Seq ← Seq ∪ {[i, ^k v_z]} // successive reconstruction of the shortest path topology
(118)         if ^k v_z ∈ S_V then
(119)            exit
(120)         end
(121)         Predecessor(↓^k v_z, ↑l, ↑t) // getting a predecessor of the vertex ^k v_z
(122)         z ← t
```

Algorithm 1: Continued.

```
(123)          k ← l
(124)          i ← i + 1
(125)        end
(126)      end
(127)  end
```

Algorithm 1: Computation of the shortest path from $S_V = \{{}^i v_x\}$ to $F_V = \{{}^j v_y\}$

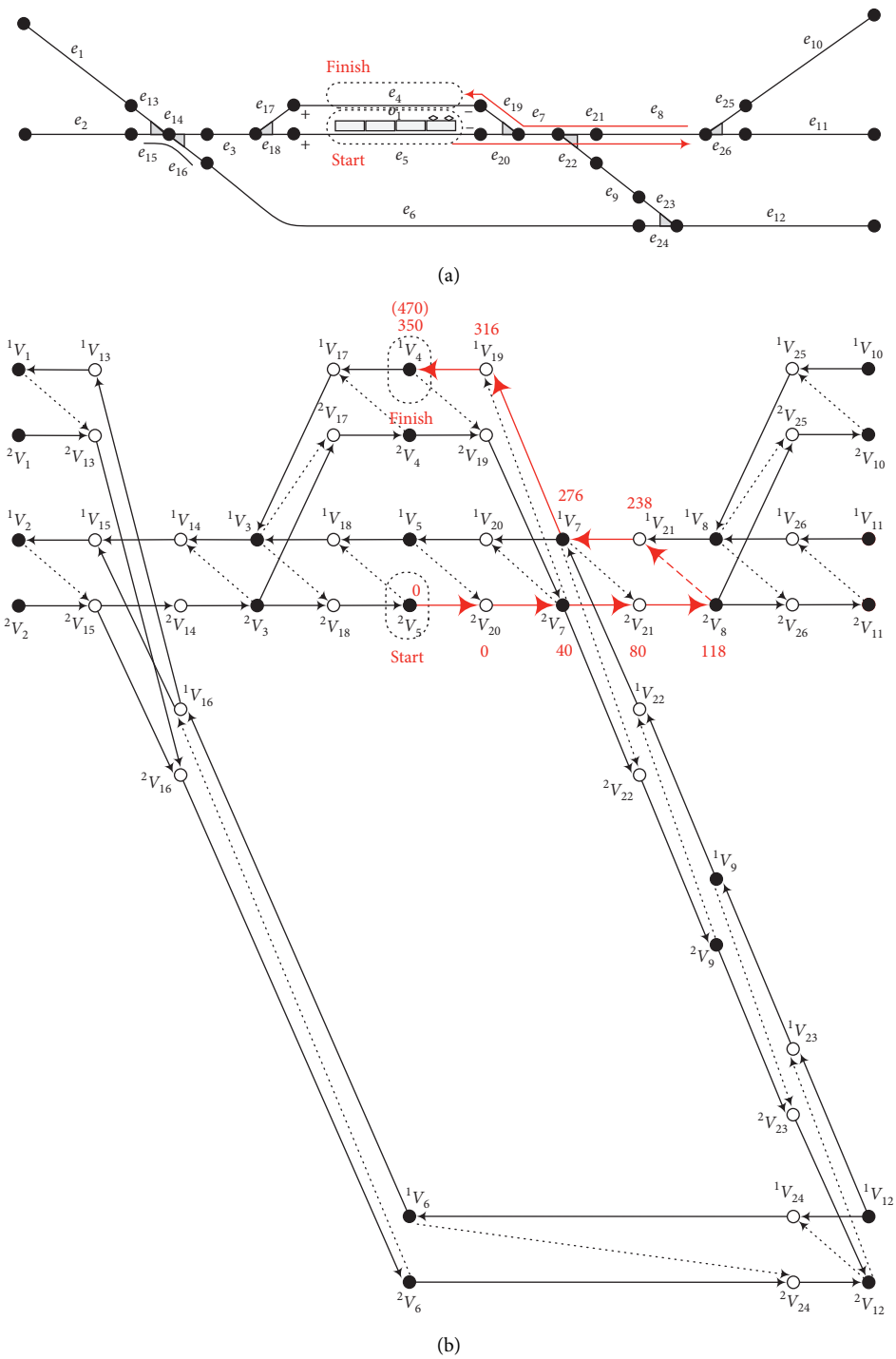

Figure 4: Train route related to Example 1.

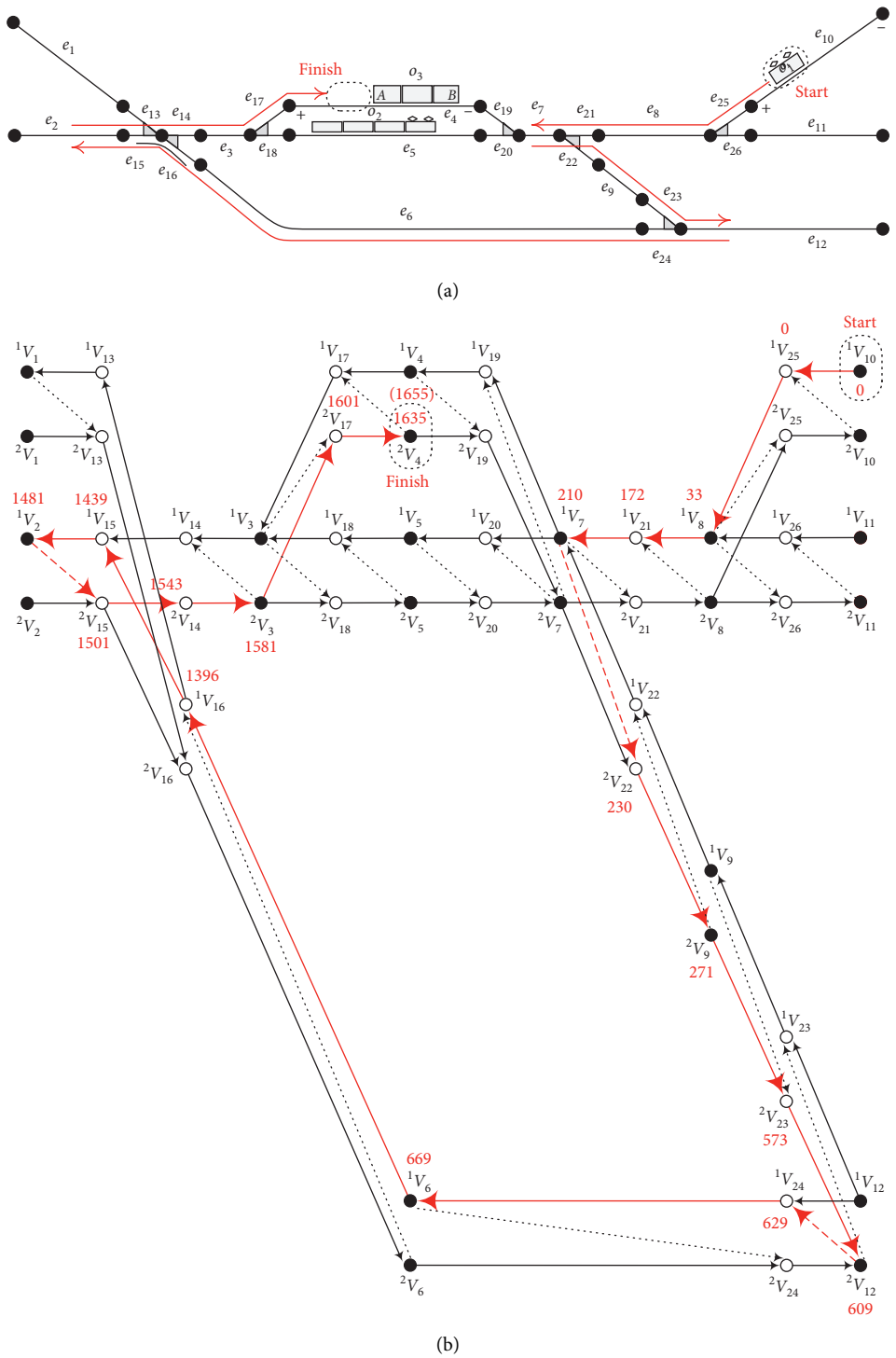

Figure 5: Train route related to Example 2.

trajectory to the points representing the input ends of the edges/tracks reached. The total length of the train's trajectory during the relocation is 470 m. The value for the last member of the walk is additionally increased by $L = 120$ due to the fact that the relocation operation must be finished by a partial transfer of the whole object to the finish track (edge). For the computed shortest route in the digraph G, the figure shows the ultimate values of the distance marks for the relevant vertices.

Moreover, the reduced distance matrix (RDM) (for $L = 120$) between all the admissible couples of the destination vertices in the digraph G is shown in Table 7 (beyond the scope of Example 1). The RDM contains the results of differently parameterized computations made by using Algorithm 1. The matrix reflects the fact that, apart from the relocation object, which is always present on a different start track, the railway yard is entirely empty. The RDM is

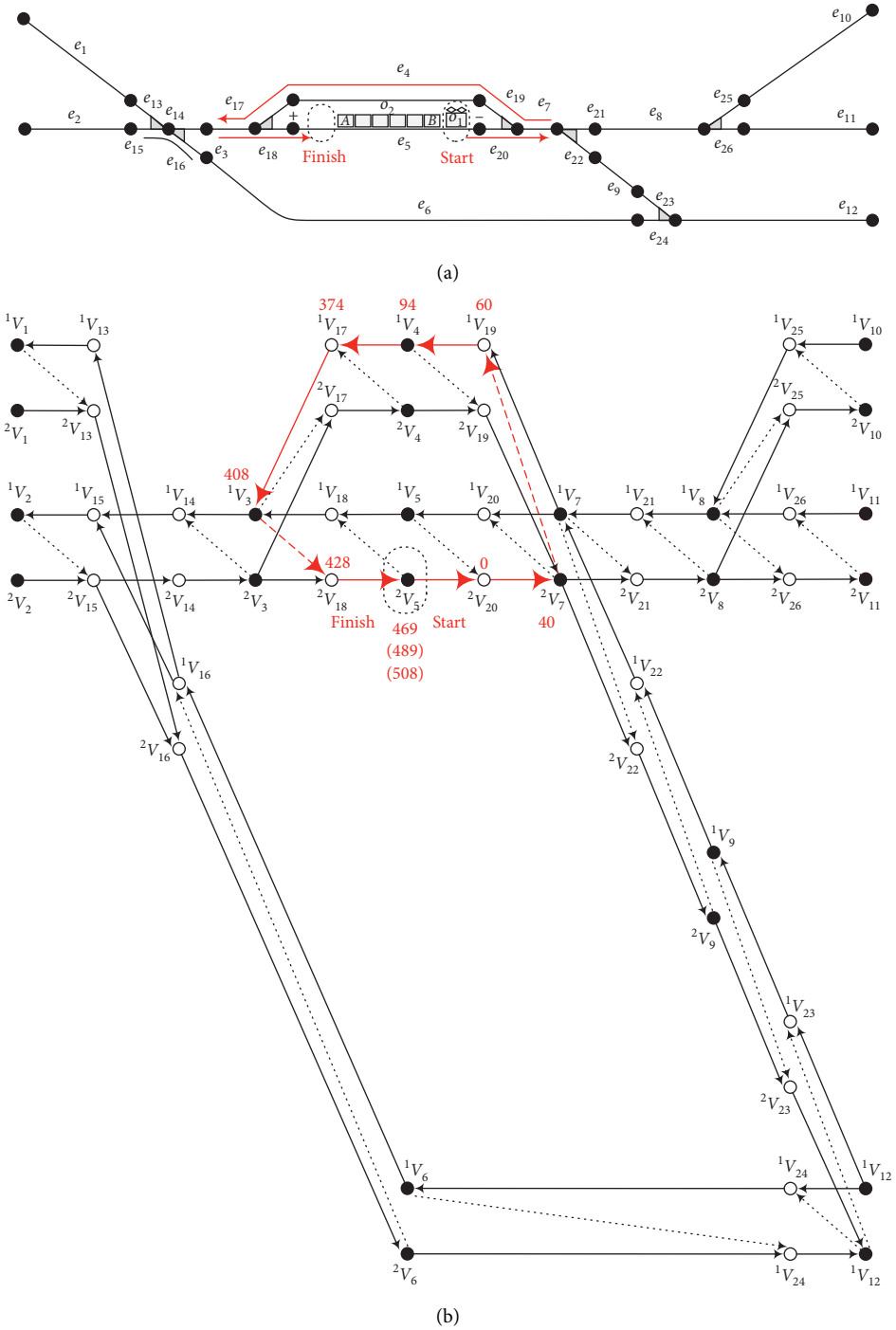

Figure 6: Train route related to Example 3.

obtained by simple transformation of the complete distance matrix (CDM). This transformation removes from the CDM those rows and columns whose all elements attain only the d^{∞} value (i.e., no shortest routes for the given parameter L exist between relevant vertices).

The next example concerns the motion of a shunting engine along a rather complex route for attachment to a specified end of a train set.

Example 2. Computation of a shunting route for the locomotive relocation ($L = 20$).

Figure 5 shows diagrams of graphs G_0 and G, which represent infrastructure models analogous to those in Figure 4. A 120 m-long train (object O_2) stands on track #5 of the yard; a 100 m-long train set (object O_3) stands on track #4; and a 20 m-long shunting locomotive (object O_1) stands on track #10. This locomotive is to be moved to end A of

Table 7: Reduced distance matrix (RDM) for the parameter $L = 120$ and vacant track layout.

From	To											
	${}^{-}e_1$ ${}^{1}v_1$	${}^{-}e_4$ ${}^{1}v_4$	${}^{-}e_5$ ${}^{1}v_5$	${}^{-}e_6$ ${}^{1}v_6$	${}^{+}e_6$ ${}^{2}v_6$	${}^{-}e_8$ ${}^{1}v_8$	${}^{+}e_8$ ${}^{2}v_8$	${}^{-}e_9$ ${}^{1}v_9$	${}^{+}e_9$ ${}^{2}v_9$	${}^{+}e_{10}$ ${}^{2}v_{10}$	${}^{+}e_{11}$ ${}^{2}v_{11}$	${}^{+}e_{12}$ ${}^{2}v_{12}$
${}^{-}e_1$ ${}^{2}v_1$	—	1533	1539	—	193	—	1765	1116	—	1937	1964	960
${}^{-}e_4$ ${}^{2}v_4$	1533	—	470	733	—	—	232	—	235	404	431	573
${}^{-}e_5$ ${}^{2}v_5$	1539	470	—	739	—	—	238	—	241	410	437	579
${}^{+}e_6$ ${}^{1}v_6$	193	—	—	—	—	—	—	—	—	—	—	—
${}^{-}e_6$ ${}^{2}v_6$	—	733	739	—	—	—	965	316	—	1137	1164	160
${}^{+}e_8$ ${}^{1}v_8$	1765	232	238	965	—	—	—	—	467	—	—	805
${}^{-}e_8$ ${}^{2}v_8$	—	—	—	—	—	—	—	—	—	153	180	—
${}^{+}e_9$ ${}^{1}v_9$	—	235	241	—	—	—	467	—	—	639	666	—
${}^{-}e_9$ ${}^{2}v_9$	1116	—	—	316	—	—	—	—	—	—	—	156
${}^{+}e_{10}$ ${}^{1}v_{10}$	1937	404	410	1137	—	153	—	—	639	—	333	977
${}^{+}e_{11}$ ${}^{1}v_{11}$	1964	431	437	1164	—	180	—	—	666	333	—	1004
${}^{+}e_{12}$ ${}^{1}v_{12}$	960	573	579	160	—	—	805	156	—	977	1004	—

train set O_3; in other words, it must enter track #4 by transiting its end ${}^{+}e_4$.

The track route computation algorithm (identifying the shortest route in the digraph G) is parameterized as follows:

(i) $S_V = \{{}^{1}v_{10}\}$, $F_V = \{{}^{2}v_4\}$, and $L = 20$

The weights are given for selected vertices and edges in the graphs, and current vacancies are quantified for selected vertices in the digraph G:

(i) $\omega\ ({}^{1}v_4) = \omega\ ({}^{2}v_4) = 280$, $\kappa\ ({}^{1}v_4) = 160$, $\kappa\ ({}^{2}v_4) = 20$, ${}^{1}v_4, {}^{2}v_4 \in V(G)$, $\varepsilon\ (e_4) = 280$, $e_4 \in E(G_0)$

(ii) $\omega\ ({}^{1}v_5) = \omega\ ({}^{2}v_5) = 259$, $\kappa\ ({}^{1}v_5) = 139$, $\kappa\ ({}^{2}v_5) = 0$, ${}^{1}v_5, {}^{2}v_5 \in V(G)$, $\varepsilon\ (e_5) = 259$, $e_5 \in E(G_0)$

(iii) $\omega\ ({}^{1}v_{10}) = \omega\ ({}^{2}v_{10}) = 175$, $\kappa\ ({}^{1}v_{10}) = 155$, $\kappa\ ({}^{2}v_{10}) = 0$, ${}^{1}v_{10}, {}^{2}v_{10} \in V(G)$, $\varepsilon\ (e_{10}) = 175$, $e_{10} \in E(G_0)$

(iv) $\forall\ [{}^{i}v_x, {}^{j}v_y] \in E_{\text{rev}}\ (G)$: $\varepsilon\ ([{}^{i}v_x, {}^{j}v_y]) = 20$

The topology of the train route found is as follows:

$e_4(1635) \leftarrow e_{17}(1601) \leftarrow e_3(1581) \leftarrow e_{14}(1543) \leftarrow e_{15}(1501) \leftarrow e_2(1481) \leftarrow e_{15}(1439) \leftarrow e_{16}(1396) \leftarrow e_6(669) \leftarrow e_{24}(629) \leftarrow e_{12}(609) \leftarrow e_{23}(573) \leftarrow e_9(271) \leftarrow e_{22}(230) \leftarrow e_7(210) \leftarrow e_{21}(172) \leftarrow e_8(33) \leftarrow e_{25}(0) \leftarrow e_{10}(0)$

The total length of the locomotive's relocation trajectory is 1655 m. The value given for the last member of the walk is increased by $L = 20$.

In the last example, shown in the following, a shunting locomotive should be moved from one end of a train set to the other end of that train set.

Example 3. Computation of a shunting route for reapproaching the same train set ($L = 20$).

A group of wagons, 200 m total length (object O_2), stands on track #5 of the yard whose models are shown in Figure 6. A 20 m-long shunting locomotive (object O_1) was disconnected from wagon group end B in order to attach it to end A of that group because this is needed for the planned operations. This means that the locomotive should reapproach the wagon group by transiting the end ${}^{+}e_5$ of track #5.

The track route computation algorithm (identifying the shortest route in the digraph G) is parameterized as follows:

$S_V = \{{}^{2}v_5\}$, $F_V = \{{}^{2}v_5\}$, and $L = 20$

The weights are given for selected vertices and edges in the graphs, and current vacancies are quantified for selected vertices in the digraph G:

(i) $\omega\ ({}^{1}v_5) = \omega\ ({}^{2}v_5) = 259$, $\kappa\ ({}^{1}v_5) = 0$, $\kappa\ ({}^{2}v_5) = 39$, ${}^{1}v_5, {}^{2}v_5 \in V(G)$, $\varepsilon\ (e_5) = 259$, $e_5 \in E(G_0)$

(ii) $\omega\ ({}^{1}v_{10}) = \omega\ ({}^{2}v_{10}) = 175$, $\kappa\ ({}^{1}v_{10}) = 155$, $\kappa\ ({}^{2}v_{10}) = 0$, ${}^{1}v_{10}, {}^{2}v_{10} \in V(G)$, $\varepsilon\ (e_{10}) = 175$, $e_{10} \in E(G_0)$

(iii) $\forall\ [{}^{i}v_x, {}^{j}v_y] \in E_{\text{rev}}\ (G)$: $\varepsilon\ ([{}^{i}v_x, {}^{j}v_y]) = 20$

The topology of the train route found is as follows:

$e_5(469) \leftarrow e_{18}(428) \leftarrow e_3(408) \leftarrow e_{17}(374) \leftarrow e_4(94) \leftarrow e_{19}(60) \leftarrow e_7(40) \leftarrow e_{20}(0) \leftarrow e_5(0)$

The trajectory run by locomotive O_1 from the starting position (${}^{-}e_5$) to the opposite end (${}^{+}e_5$) of the same track (track #5) is 469 m long. If the complete length of the

locomotive enters track #5, then it has run a trajectory of 489 m long. And it requires another 19 m for attachment to train set O_2, whereby the total trajectory length is 508 m.

For demonstration reasons (beyond the scope of Example 3), the RDM (for $L = 20$) for the destination vertices in the digraph G, containing results of the calculations for differently parameterized Algorithm 1, is included in Table 8. This matrix matches the situation where, apart from the relocation object (which is always present on a different start track), the railway yard is entirely empty. From this matrix, one can read that the trajectory of relocation between track #10 (its end ${}^{+}e_{10}$) and track #4 (its end ${}^{+}e_{4}$) is 664 m long. The shortest track route found passes through track #5. The route for the relocation (between the same tracks) is different (and differently long) from that in Example 2 because track #5 was occupied in that example.

4.3. Modifications of the Basic Algorithm. Conditions for the starting and destination positions different from those associated with Algorithm 1 may be used in the parameterization of the dynamic search of routes within a railway yard. The algorithm uses a uniquely defined track end from which the sought-for-train/shunting route should start, and this also applies to the end of the finish track. From the point of view of the final mathematical model (digraph G), it is exactly determined by one start vertex and exactly one finish vertex for the t route sought (thus, the algorithm searches for a single-source single-destination shortest path). In other words, the relation $|S_V| = |F_V| = 1$ holds for the set of start vertices S_V and the set of finish vertices F_V.

However, if the whole track x must be specified as the starting element (hence, leaving it through either of its ends is permissible), then the appropriate parameter of the *Shortest_Path* function is defined as $S_V = \{{}^{1}v_x, {}^{2}v_x\}$. The members of the set S_V correspond to the opposite ends of the track x in the digraph G. Analogously, the set F_V can be constructed as $F_V = \{{}^{1}v_y, {}^{2}v_y\}$ if the entire track is regarded as the destination element and entering it via either end is permissible. So, three different modifications of Algorithm 1, for different variants of construction of the sets S_V and F_V, are feasible. A summary overview of the *Shortest_Path* function parameterization options is presented in Table 9.

Algorithms 2–4 differ from the basic Algorithm 1 only due to changes in two subroutines, *Start_Finish_Test* and *Start_Finish_Init*, while the remaining parts are identical. The differences in the subroutines include different procedures of testing the admissibility of combinations of the input parameter values and different ways of performing initialisation actions associated with the start and finish vertices.

The first alternative to the basic algorithm is Algorithm 2, which seeks the shortest admissible route between two two-member sets of vertices (S_V and F_V) in the digraph G (two-sources two-destinations shortest path). One run of this algorithm provides the required route (through the function *Get_Path*), and the start vertex (from the set S_V) and finish vertex (from the set F_V) of that route are uniquely identified. The vertices define the direction in which the start track is left and the direction from which the finish track is entered. The *Start_Finish_Init* function potentially eliminates (among other things) from the set of finish vertices that vertex whose vacancy is inadequate for accommodating the relocation object whose length is L.

The next modification of the basic algorithm is Algorithm 3, computing the routes from two start vertices to one finish vertex in the digraph G (two-sources single-destination shortest path). When the process is over, the shorter of the two potentially computed routes is identified. The vertex ${}^{b}v_y$, which is the pair vertex to the vertex ${}^{j}v_y \in F_V$, is included into the set of forbidden vertices through the *Start_Finish_Init* function.

The last modification of the basic algorithm is Algorithm 4, seeking the shorter of two routes proceeding from one start vertex to two different finish vertices in the digraph G (single-source two-destinations shortest path). That vertex whose vacancy is inadequate with respect to parameter L is potentially eliminated from the set of finish vertices by the *Start_Finish_Init* function. In addition, that function augments the set of forbidden vertices with the vertex ${}^{a}v_x$—the pair vertex to the vertex ${}^{i}v_x \in S_V$.

4.4. Verification and Validation. The life cycle of simulation studies/projects consists of a number of partial phases, as described in detail in [32, 33]. Discussed in the following is a part of the life cycle (consisting of several sequentially linked phases), with focus on the construction, verification, and validation of models that are typically used in simulation studies.

(a) Phase of designing and forming a conceptual model
(b) Conceptual model validation phase
(c) Phase of designing and building up a computerized model
(d) Computerized model verification phase
(e) Operational computerized model validation phase

The models can be briefly characterized as follows:

(i) The conceptual model reflects (with an appropriate degree of abstraction) the object of investigation
(ii) The computerized model represents an implementation of the conceptual model on a computer

In view of the scope of this article, attention (with respect to verification and validation) will not be paid to the whole target rail-traffic simulator (as implemented within the MesoRail tool): instead, only a part of it will be targeted, viz. that part that mirrors the rail infrastructure of the object of investigation and functions determined for computing track routes along which the rail vehicles in question can move on the rail infrastructure.

The conceptual model, which reflects the rail infrastructure of the object of investigation (representing the railway system in question) and the functions calculating track routes, uses the following:

Table 8: The RDM for the parameter $L = 20$ and vacant track layout.

From \ To	$^{-}e_1\ {}^{1}v_1$	$^{-}e_2\ {}^{1}v_2$	$^{-}e_3\ {}^{1}v_3$	$^{+}e_3\ {}^{2}v_3$	$^{-}e_4\ {}^{1}v_4$	$^{+}e_4\ {}^{2}v_4$	$^{-}e_5\ {}^{1}v_5$	$^{+}e_5\ {}^{2}v_5$	$^{-}e_6\ {}^{1}v_6$	$^{+}e_6\ {}^{2}v_6$	$^{-}e_7\ {}^{1}v_7$	$^{+}e_7\ {}^{2}v_7$	$^{-}e_8\ {}^{1}v_8$	$^{+}e_8\ {}^{2}v_8$	$^{-}e_9\ {}^{1}v_9$	$^{+}e_9\ {}^{2}v_9$	$^{+}e_{10}\ {}^{2}v_{10}$	$^{+}e_{11}\ {}^{2}v_{11}$	$^{+}e_{12}\ {}^{2}v_{12}$
$^{-}e_1\ {}^{2}v_1$	—	198	1639	298	712	352	726	359	—	93	1259	658	—	736	916	739	908	935	860
$^{-}e_2\ {}^{2}v_2$	198	—	1651	100	514	154	528	161	939	105	1271	460	—	538	928	541	710	737	872
$^{+}e_3\ {}^{1}v_3$	298	100	1751	—	1445	—	1451	—	—	205	1371	—	—	1429	1028	—	1601	1628	972
$^{-}e_3\ {}^{2}v_3$	1639	1651	—	1751	414	54	428	61	839	—	—	360	—	438	—	441	610	637	779
$^{+}e_4\ {}^{1}v_4$	352	154	54	—	468	—	1505	115	893	259	1425	414	—	492	1082	495	664	691	833
$^{-}e_4\ {}^{2}v_4$	712	514	414	1445	—	468	114	1506	533	619	—	54	—	132	1442	135	304	331	473
$^{+}e_5\ {}^{1}v_5$	359	161	61	—	1506	115	489	—	908	266	1432	429	—	507	1089	510	679	706	848
$^{-}e_5\ {}^{2}v_5$	726	528	428	1451	114	1505	—	489	539	633	—	60	—	138	1456	141	310	337	479
$^{+}e_6\ {}^{1}v_6$	93	105	—	205	619	259	633	266	1044	—	—	565	—	643	—	646	815	842	984
$^{-}e_6\ {}^{2}v_6$	—	939	839	—	533	893	539	908	—	1044	459	—	—	517	116	—	689	716	60
$^{+}e_7\ {}^{1}v_7$	658	460	360	—	54	414	60	429	—	565	1731	—	—	—	1388	—	—	—	1332
$^{-}e_7\ {}^{2}v_7$	1259	1271	—	1371	—	1425	—	1432	459	—	—	1731	—	58	—	61	230	257	399
$^{+}e_8\ {}^{1}v_8$	736	538	438	1429	132	492	138	507	517	643	58	—	—	—	1466	119	—	—	457
$^{-}e_8\ {}^{2}v_8$	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	53	80	—
$^{+}e_9\ {}^{1}v_9$	739	541	441	—	135	495	141	510	—	646	61	—	—	119	1469	—	291	318	1413
$^{-}e_9\ {}^{2}v_9$	916	928	—	1028	1442	1082	1456	1089	116	—	—	1388	—	1466	—	1469	1638	1665	56
$^{+}e_{10}\ {}^{1}v_{10}$	908	710	610	1601	304	664	310	679	689	815	230	—	53	—	1638	291	—	133	629
$^{+}e_{11}\ {}^{1}v_{11}$	935	737	637	1628	331	691	337	706	716	842	257	—	80	—	1665	318	133	—	656
$^{+}e_{12}\ {}^{1}v_{12}$	860	872	779	972	473	833	479	848	60	984	399	1332	—	457	56	1413	629	656	—

Table 9: Variants of algorithms for calculating the shortest paths.

	Start vertices $S_V \subset V(G)$	Finish vertices $F_V \subset V(G)$	Search type
Algorithm 1	$S_V = \{^{i}v_x\}$	$F_V = \{^{j}v_y\}$	Single-source single-destination
Algorithm 2	$S_V = \{^{1}v_x, {}^{2}v_x\}$	$F_V = \{^{1}v_y, {}^{2}v_y\}$	Two-sources two-destinations
Algorithm 3	$S_V = \{^{1}v_x, {}^{2}v_x\}$	$F_V = \{^{j}v_y\}$	Two-sources single-destination
Algorithm 4	$S_V = \{^{i}v_x\}$	$F_V = \{^{1}v_y, {}^{2}v_y\}$	Single-source two-destinations

(i) A mathematical model weighted digraph, as described in the Final Model section. The digraph is a result of transformation of the primary mathematical model—a weighted undirected graph. The primary model reflects rather intuitively the topological and metric situation of the rail infrastructure associated with the object of investigation.

(ii) The concept of Dijkstra's algorithm for searching for the shortest paths in graphs. The algorithm had to be modified appreciably for use in problems of determining track routes on the final infrastructure model.

Validation of the conceptual model included, in particular, assessment of suitability of both mathematical models for description of the relevant part of the object of investigation and of the algorithms reflecting the selected operations on the object of investigation. This phase used the IV&V (independent verification and validation) approach [33], applied after completing the development of the

```
(1) function Start_Finish_Test(↓S_V, ↓F_V, ↓L, ↓↑okay)
(2)    if (S_V = ∅ or F_V = ∅ or S_V = F_V) then
(3)       okay ← false
(4)       exit
(5)    end
(6)    for each ^i v_x ∈ S_V do
(7)       if (ω (^i v_x) < L or (κ (^1 v_y) < L and κ (^2 v_y) < L)) then
(8)          okay ← false
(9)          exit
(10)      end
(11)   end
(12) end
(13) function Start_Finish_Init(↓S_V, ↓F_V)
(14)   for each ^j v_y ∈ F_V do
(15)      if κ (^j v_y) < L then
(16)         F_V = F_V − {^j v_y} // exclusions of finish vertices with insufficient vacant capacities
(17)      end
(18)   end
(19)   for each ^i v_x ∈ S_V do
(20)      Get_Indexes(↓^i v_x, ↑i, ↑x)
(21)      ^i d_x ← 0
(22)   end
(23)   X_V ← ∅
(24) end
```

Algorithm 2: Modification of Algorithm 1 for $S_V = \{^1v_x, ^2v_x\}$ and $F_V = \{^1v_y, ^2v_y\}$.

```
(1) function Start_Finish_Test(↓S_V, ↓F_V, ↓L, ↓↑okay)
(2)    if (S_V = ∅ or F_V = ∅ or F_V ⊂ S_V) then
(3)       okay ← false
(4)       exit
(5)    end
(6)    for each ^i v_x ∈ S_V do
(7)       for each ^j v_y ∈ F_V do
(8)          if (ω (^i v_x) < L or κ (^j v_y) < L) then
(9)             okay ← false
(10)            exit
(11)         end
(12)      end
(13)   end
(14) end
(15) function Start_Finish_Init(↓S_V, ↓F_V)
(16)   for each ^i v_x ∈ S_V do
(17)      Get_Indexes(↓^i v_x, ↑i, ↑x)
(18)      ^i d_x ← 0
(19)   end
(20)   Get_Indexes(↓^j v_y, ↑j, ↑y)
(21)   b ← (3 − j)
(22)   X_V ← {^b v_y} // a forbidden vertex ^b v_y represents a pair vertex to the finish vertex ^j v_y
(23) end
```

Algorithm 3: Modification of Algorithm 1 for $S_V = \{^1v_x, ^2v_x\}$ and $F_V = \{^jv_y\}$.

conceptual model. This implied in practice that the validation was made by an independent third-party professional (from the Czech Railway Infrastructure Administration, State Organization), who was a renowned expert in the railway traffic domain and in the application of computer simulations for the needs of railway traffic optimizations (and was the second author of this paper). The face validation (expert validation) [33] method was used, requiring

```
(1) function Start_Finish_Test(↓S_V, ↓F_V, ↓L, ↓↑okay)
(2)    if (S_V = ∅ or F_V = ∅ or S_V ⊂ F_V) then
(3)       okay ← false
(4)       exit
(5)    end
(6)    for each ^i v_x ∈ S_V do
(7)       if (ω (^i v_x) < L or (κ (^1 v_y) < L and κ (^2 v_y) < L)) then
(8)          okay ← false
(9)          exit
(10)      end
(11)   end
(12) end
(13) function Start_Finish_Init(↓S_V, ↓F_V)
(14)   for each ^j v_y ∈ F_V do
(15)      if κ (^j v_y) < L then
(16)         F_V = F_V − {^j v_y} // exclusions of finish vertices with insufficient vacant capacities
(17)      end
(18)   end
(19)   Get_Indexes(↓^i v_x, ↑i, ↑x)
(20)   ^i d_x ← 0
(21)   a ← (3 − i)
(22)   X_V ← {^a v_x} // a forbidden vertex ^a v_x represents a pair vertex to the start vertex ^i v_x
(23) end
```

Algorithm 4: Modification of Algorithm 1 for $S_V = \{^i v_x\}$ and $F_V = \{^1 v_y, {}^2 v_y\}$.

Table 10: Mean calculation times of Algorithm 1 related to searching a shortest path.

	$\lvert V(G)\rvert$ (—)	$\lvert E(G)\rvert$ (—)	L (—)	Calculated paths (—)	$^{\text{mean}}t$ (ms)
Demonstration railway yard Total length of tracks: 3.05 km	52	78	0	2,652	0.02
			20	2,652	0.02
			50	462	0.03
			100	380	0.03
			200	56	< 0.01
			300	12	< 0.01
Real railway yard Total length of tracks: 75.96 km	1178	1905	0	1,386,506	5.40
			20	184,470	5.30
			50	71,556	5.00
			100	38,220	4.80
			200	23,562	4.80
			300	17,292	4.80
			400	12,882	4.70
			500	11,990	4.60

that the professional performing the validation has deep knowledge of the relevant application domain (railway traffic in this case). The conclusions from the conceptual model validation process were as follows: (i) the designed mathematical models of the rail infrastructure adequately reflect the characteristics of the real railway infrastructure; and (ii) the algorithms for track route computations relating to rail vehicle shunting and train runs have been correctly logically designed so as to be able to provide outputs (i.e., specific track routes) usable in actual railway systems.

When developing the computerized model, data structures (mentioned in the conclusion of the Basic Algorithm section) were selected so as to be applicable to the implementation of the weighted digraph (reflecting the rail infrastructure) and to enable effective computations of the implemented track route searching algorithms.

The computerized model verification phase included testing of the model's logical correctness. This means that the track routes found for the specific objects of relocation were tested especially with respect to the following conditions: (i) the length of the relocation object is respected; (ii) those infrastructure elements that are currently occupied by rail vehicles or blocked by the interlocking system are not used; and (iii) reversals are performed on admissible track segments. The testing was made on different data instances (describing 3 different rail yards) applying different relocation object lengths and differently occupied track layouts. Verification was made by the person who had developed the

computerized model (and is the first author of this paper). The performance of the algorithms was found to be logically correct (after eliminating a few minor errors).

The IV&V approach and the face validation method were also applied (by the same expert as during the conceptual model validation phase) during the operational validation of the computerized model. The expert assessment of results of the track route computations (using the same track layout variant as in the computerized model verification phase) included, in particular, assessment of the operational and technical usability of the track routes identified (for different relocation object lengths) for the track layouts as currently occupied. Following analysis of all the cases of track routes examined (as obtained by using the original algorithms for the computations of the shortest paths on the weighted digraph), the expert concluded that the algorithms performed correctly from both operational and technical aspects. Based on that, this partial computerized model was embedded in the target simulator within the MesoRail software tool.

4.5. Technical Notes for Practical Use of Algorithms within Rail-Traffic Simulators. A few technical comments should be added regarding practical use of the algorithms within software tools serving to simulate railway traffic (at the mesoscopic level of detail):

(a) A combined approach can be applied in the simulators when searching for the optimal train routes and shunting routes. This may include selecting from a precalculated static set of essential train routes (constructed before starting the simulation experiments) and application of dynamic computation during the simulation when seeking for the optimum shunting routes. In this manner, the simulation will not be burdened by dynamic search for selected (very frequently repeating) train routes.

(b) It is typical of railway traffic that dedicated track routes are not set on the infrastructure between extremely distant start and finish tracks. Instead, the approach of stepwise construction of several shorter routes is preferred. Taking this into account, the algorithms discussed can be augmented with a limitation regarding the maximum admissible length of the track route being sought. This limitation can be used to put restrictions to the spread of the computation within the appropriate graph. This approach can eliminate undesirable instances of very complex and long shunting routes (found, e.g., in a densely occupied part of the track infrastructure). This means in practice that the algorithm may include identification of the lowest value ($d^{\min}$) among all distance marks for the vertices currently included in the set T_V. The $d^{\min}$ value can be indicated simply when withdrawing a vertex from the set T_V through the function *Vert_Select.* If this value exceeds an expertly defined limiting value $d^{\max}$, the computation is terminated with the result that no route was found. For the relocation object, this means that, given the current traffic situation, it will have to stay where it is for some time until the traffic situation changes (e.g., when the relevant tracks become available for the operation), and the search for the track route can be repeated.

(c) Given the degree of abstraction applied, the algorithms presented work with a simplification against reality, mainly concerning reversals within the motion of a relocation object. A reversal is defined as a situation where the relocation object stops at a certain position and then continues its motion in the direction opposite to the initial direction. As mentioned above, the infrastructure model presented and the appropriate algorithms always consider a reversal as an operation occurring on one track. In reality, however, more than one track (and switch) can be engaged in the reversal of a relocation object. The simplification is appropriate for simulators working at the mesoscopic level of detail (intended, e.g., for examining the capacity of the infrastructure) because shunting operations, in particular, are not supposed to be examined in great detail. However, when examining railway traffic at the microscopic level, such simplification might provide an unsatisfactory solution, unusable for application.

(d) For practical use of the algorithms in simulation experiments, one must have an idea of how much real time the computations take—such information is important for the assessment of their usability in the simulations because they should not be very time consuming. By way of illustration of the time demands, computation experiments were performed for Algorithm 1 (with the highest potential for use in traffic simulations) in a model of a basic railway yard (shown in Figure 3) and in a more extensive model of a larger real railway marshalling yard in Teplička nad Váhom, Slovakia. The latter model (represented by a digraph) contained 1178 vertices (reflecting 589 track segments) and 1905 edges (the total length of tracks in the infrastructure was 75,958 m). Different values of parameter L (reflecting the relocation object lengths) were applied, viz. $L = 0$, 20, 50, 100, 200, 300, 400, and 500. The shortest routes were computed for all admissible vertex couples, which satisfied the condition that their weight (or vacancy) was not lower than the applied value of parameter L (both models reflected empty railway yards). This, however, also means that—particularly for the more extensive model—such complex routes were computed as would never be actually computed within the simulators. Such routes were included in the experiments for testing purposes only. And in addition, connecting vertices were also considered as the start vertices and finish vertices (except for destination vertices) in order to increase the number of the routes computed.

The results of the calculations providing mean durations ($^{\text{mean}}t$) of searches for the shortest route are listed in Table 10. Generally, the mean time of computation of one shortest route within a more extensive demonstration model

does not exceed 6 milliseconds, which is a very good figure with respect to the needs of railway traffic simulators. The computations were made on a common PC equipped with an Intel i7-8550U CPU@1.80 GHz processor and 16.0 GB RAM.

The results also reflect computations that did not find the shortest route because none exists. The search for the shortest route between the vertices ${}^{2}v_5$ and ${}^{1}v_{10}$ for $L = 100$ in the model shown in Figure 3 is an example: although both vertices have a sufficient vacancy, the vertex ${}^{1}v_{10}$ is inaccessible because it mirrors that end of track #10 that constitutes the boundary of the model.

5. Conclusions

In conclusion, it has been demonstrated that the presented algorithms for automated dynamic calculations of the shortest track route topologies can be used with advantage within mesoscopic rail-traffic simulators to solve current traffic problems. The routes can then be used for shunting postulated relocation objects within the railway infrastructure in the actual occupancy situation. The data to be input when searching for the route include the start and finish positions and length of the object to be relocated.

So, the algorithms presented offer a good potential for extending and improving the scope of mesoscopic railway traffic simulators as regards automated identification of track routes especially for rail vehicle shunting.

The data specification of the rail infrastructure configuration, which currently uses the XML format, can be transformed in the future into a data description that will be fully compliant with the railML standard.

From the managerial aspect, it is noteworthy that it is convenient for the simulation projects in the railway traffic domain to acquire and use such simulation tools as support rapid constructions of the simulation models and fast parameterizations of the simulation experiments. Hence, if the simulation tool selected is equipped with functionalities that automatically compute track routes for specific relocation objects during the run of the simulation program, then such a tool has a comparative advantage over other tools lacking such functionalities. This is so because in this case, the user is freed from the requirement to tediously predefine many track routes, and hence, the time of the development of the target simulation models reflecting the operation of the railway system examined is appreciably shortened.

Data Availability

There is no supplementary information attached to the research article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The research presented in this article was supported by the ERDF/ESF Project: Cooperation in Applied Research between the University of Pardubice and companies, in the Field of Positioning, Detection and Simulation Technology for Transport Systems (PosiTrans) (no. CZ.02.1.01/0.0/0.0/17_049/0008394).

References

[1] J.-F. Cordeau, P. Toth, and D. Vigo, "A survey of optimization models for train routing and scheduling," *Transportation Science*, vol. 32, no. 4, pp. 380–404, 1998.

[2] A. Almech, E. Roanes-Lozano, C. Solano-Macías, and A. Hernando, "A new approach to shortest route finding in a railway network with two track gauges and gauge change-overs," *Mathematical Problems in Engineering*, vol. 2019, Article ID 8146150, 16 pages, 2019.

[3] L. G. Kroon, H. Edwin Romeijn, and P. J. Zwaneveld, "Routing trains through railway stations: complexity issues," *European Journal of Operational Research*, vol. 98, no. 3, pp. 485–498, 1997.

[4] R. Freling, R. M. Lentink, L. G. Kroon, and D. Huisman, "Shunting of passenger train units in a railway station," *Transportation Science*, vol. 39, no. 2, pp. 261–272, 2005.

[5] J.-A. Adlbrecht, B. Hüttler, N. Ilo, and M. Gronalt, "Train routing in shunting yards using Answer Set Programming," *Expert Systems with Applications*, vol. 42, no. 21, pp. 7292–7302, 2015.

[6] J. Riezebos and W. Van Wezel, "K-Shortest routing of trains on shunting yards," *OR Spectrum*, vol. 31, no. 4, pp. 745–758, 2009.

[7] A. Kavicka and L. Janosikova, "Trackage modelling and algorithms for finding the shortest train route," *Communications—Scientific Letters of the University of Zilina*, vol. 1, no. 2, pp. 9–21, 1999.

[8] M. Montigel, *Modellierung und Gewährleistung von Abhängigkeiten in Eisenbahnsicherungsanlagen*, ETH Zurich, Zürich, Switzerland, 1994.

[9] B. Zelinka, "Polar graphs and railway traffic," *Applications of Mathematics*, vol. 19, no. 3, pp. 169–176, 1974.

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[11] T. Cormen, *Introduction to Algorithms*, MIT Press, Cambridge, MA, USA, 2nd edition, 2001.

[12] M. Barbehenn, "A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices," *IEEE Transactions on Computers*, vol. 47, no. 2, 1998.

[13] F. Schulz, D. Wagner, and K. Weihe, "Dijkstra's algorithm online," *ACM Journal of Experimental Algorithmics*, vol. 5, p. 12, 2000.

[14] F. Schulz, D. Wagner, C. Zaroliagis, C. Stein, and D. M. Mount, "Using multi-level graphs for timetable information in railway systems," *Algorithm Engineering and Experiments (Lecture Notes in Computer Science)*, Springer, Berlin, Germany, pp. 7–12, 2002.

[15] D. Wang, X. Chen, and H. Huang, "A graph theory-based approach to route location in railway interlocking," *Computers & Industrial Engineering*, vol. 66, no. 4, pp. 791–799, 2013.

[16] G. Medeossi and S. De Fabris, "Simulation of rail operations," in *Handbook of Optimization in the Railway Industry*, R. Borndörfer, T. Klug, L. Lamorgese et al., Eds., Springer International Publishing, Cham, Switzerland, pp. 1–24, 2018.

[17] A. Nash and D. Huerlimann, "Railroad simulation using OpenTrack," *WIT Transactions on the Built Environment*, vol. 74, 2004.

[18] A. Radtke and J.-P. Bendfelt, *Handling of Railway Operation Problems with RailSys*, Institute of Transport, University of Hanover, Hanover, Germany, 2011.
[19] N. Adamko and V. Klima, "Optimisation of railway terminal design and operations using Villon generic simulation model," *Transport*, vol. 23, no. 4, pp. 335–340, 2008.
[20] R. Divis and A. Kavicka, "Design and development of a mesoscopic simulator specialized in investigating capacities of railway nodes," in *Proceedings of the 27th European Modeling and Simulation Symposium, EMSS 2015*, pp. 52–57, Bergeggi, Italy, September 2015.
[21] B. Sewcyk and M. Kettner, *Network Evaluation Model NEMO*, Institute of Transport. Germany: University of Hanover, Hanover, Germany, 2001.
[22] Y. Cui, U. Martin, and J. Liang, "PULSim: user-based adaptable simulation tool for railway planning and operations," *Journal of Advanced Transportation*, vol. 2018, Article ID 7284815, 11 pages, 2018.
[23] Y. Cui and U. Martin, "Multi-scale simulation in railway planning and operation," *PROMET—Traffic&Transportation*, vol. 23, no. 6, pp. 511–517, 2011.
[24] R. Novotny and K. Antonin, "Unitary hybrid model of railway traffic," in *Proceedings of the 29th European Modeling and Simulation Symposium, EMSS 2017, International Multidisciplinary Modeling and Simulation Multiconference, I3M 2017*, pp. 181–186, Barcelona, Spain, September 2017.
[25] W. Burghout, H. N. Koutsopoulos, and I. Andreasson, "A discrete-event mesoscopic traffic simulation model for hybrid traffic simulation," in *Proceedings of the IEEE Conference on Intelligent Transportation Systems, ITSC 2006*, pp. 1102–1107, Maui, HI, USA, November 2006.
[26] X. Chen, S. He, T. Li, and Y. Li, "A simulation platform for combined rail/road transport in multiyards intermodal terminals," *Journal of Advanced Transportation*, vol. 2018, Article ID 5812939, 19 pages, 2018.
[27] https://www.railml.org/en/.
[28] T. Ciszewski and W. Nowakowski, "RailML as a tool for description of data model in railway traffic control systems," in *Proceedings of the International Conference-Transport Means*, pp. 935–940, Kaunas University of Technology, Trakai, Lithuania, October 2018.
[29] http://www.railtopomodel.org/en/.
[30] A. Hlubuček, "RailTopoModel and railML 3 in overall context," *Acta Polytechnica CTU Proceedings*, vol. 11, pp. 16–21, 2017.
[31] J. Ebert, "A versatile data structure for edge-oriented graph algorithms," *Communications of the ACM*, vol. 30, no. 6, pp. 513–519, 1987.
[32] J. Banks, *Handbook of Simulation [Online]*, John Wiley & Sons, Hoboken, NJ, USA, 1998.
[33] R. G. Sargent, "Verification and validation of simulation models," in *Proceedings of the 2010 Winter Simulation Conference*, pp. 166–183, IEEE, Baltimore, MD, USA, December 2010.