Research article
DOI: https://doi.org/10.18721/JCSTCS.17406
UDC 004.052.42

A METHOD FOR FINDING THE CORRESPONDENCE BETWEEN A RAILWAY STATION MODEL AND ITS VISUAL REPRESENTATION BASED ON GRAPHS

V.E. Ustinova ✉ , *A.S. Lutsenko*, *A.V. Shpak*, *G.V. Mironenkov*, *V.A. Ivlev*

Peter the Great St. Petersburg Polytechnic University, St. Petersburg, Russian Federation

✉ ustlera@list.ru

Abstract. The paper proposes a method for searching and comparing objects of the railway station model in the database format to the corresponding objects of the station visual representation in the SVG file format. The method is based on reducing the structure of a railway station to a directed asymmetric graph and using comparison algorithms to find correspondences between topological and actual characteristics of objects. The method includes comparing nodes and connections of the model graph with structural elements of the station visual representation. The paper also proposes a software tool that implements the proposed method. The software tool was tested in an experiment involving three employees, which revealed the average number of inconsistencies found, as well as the average time of the inconsistency search process before and after automation. The result of the research is that the method increases the accuracy of the process by two times and accelerates it by five times.

Keywords: graph, model representation, comparison of representations, railway station, visualization

Citation: Ustinova V.E., Lutsenko A.S., Shpak A.V., el al. A method for finding the correspondence between a railway station model and its visual representation based on graphs. Computing, Telecommunications and Control, 2024, Vol. 17, No. 4, Pp. 64–77. DOI: 10.18721/JCSTCS.17406

Научная статья
DOI: https://doi.org/10.18721/JCSTCS.17406
УДК 004.052.42

МЕТОД ПОИСКА СООТВЕТСТВИЯ МОДЕЛИ ЖЕЛЕЗНОДОРОЖНОЙ СТАНЦИИ ЕЕ ВИЗУАЛЬНОМУ ПРЕДСТАВЛЕНИЮ НА ОСНОВЕ ГРАФОВ

В.Е. Устинова ✉ , *А.С. Луценко*,
А.В. Шпак, *Г.В. Мироненков*, *В.А. Ивлев*

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

✉ ustlera@list.ru

Аннотация. В работе представлен метод поиска и сопоставления объектов модели железнодорожной станции в формате базы данных соответствующим объектам визуального представления станции в формате SVG-файла. Метод основан на приведении структуры железнодорожной станции к направленному несимметричному графу и использовании алгоритмов сравнения для поиска соответствий топологических и фактических характеристик объектов. Метод включает в себя сопоставление узлов и связей графа модели со структурными элементами визуального представления станции. Также в работе представлено программное средство, которое реализует предложенный метод. Программное средство было апробировано на эксперименте, в котором приняло участие три работника и который выявил среднее число найденных несоответствий, а также среднее время процесса поиска несоответствий до и после автоматизации. Результатом работы является то, что метод повышает точность процесса в 2 раза и ускоряет его в 5 раз.

Ключевые слова: граф, модельное представление, сравнение представлений, железнодорожная станция, визуализация

Для цитирования: Ustinova V.E., Lutsenko A.S., Shpak A.V., el al. A method for finding the correspondence between a railway station model and its visual representation based on graphs // Computing, Telecommunications and Control. 2024. Т. 17, № 4. С. 64–77. DOI: 10.18721/JCSTCS.17406

Introduction

Today, the Russian railway system is one of the most developed in the world and ranks third in terms of total length [1]. Due to such a large volume of the railway network, there are a large number of railway stations and, as a rule, several representations are created for them – a model and a visual [2, 3]. In the Russian Railways company, the model representation (electronic map) of the station exists in the form of an SQL file, while the objects are described in interconnected tables. The visual representation is demonstrated by an SVG file, while the objects are described using a set of tags [4, 5] (Fig. 1).

Discrepancies are often observed between the two representations of the same station: inconsistencies in the topology of individual sections of the electronic map and the graphical representation, names of objects, directions of direct passage and exit along the arrow, directions of traffic light adjustment. The presence of discrepancies means that it is impossible to design new stations until all discrepancies are eliminated. Manual updating of models and searching for inconsistencies is a time-consuming task, accompanied by a large number of human errors. Hence, the need to automate [6] this process and reduce the complexity of finding a match between the railway station model and the visual representation due to the algorithm implemented in the software for constructing and comparing graphs based on two representations of a railway station. This action can be presented as a goal, to achieve which it is necessary to perform a number of tasks, namely:

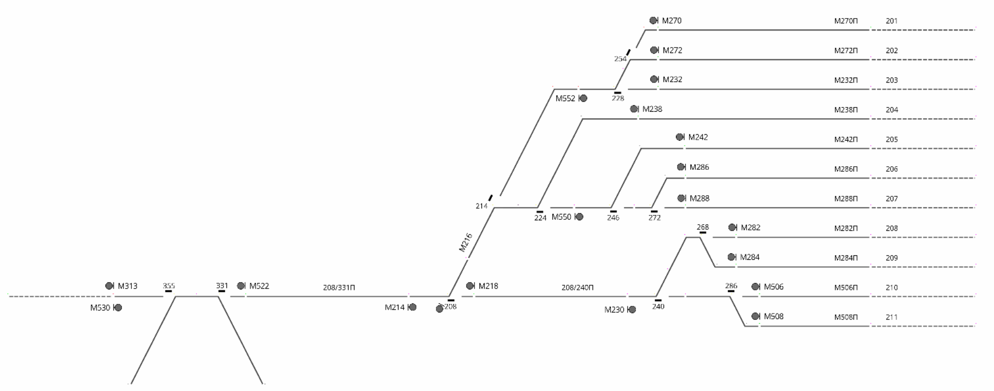

Fig. 1. Visual representation of the station

• analysis of existing solutions;

• development of the system architecture and its own method of automated comparison of the model and visual representations of the railway station plan for the Russian Railways company [7, 8];

• implementation of the method in the form of a software tool;

• conducting experiments and analysis of the obtained results to identify a reduction in the complexity of the process.

Overview of existing solutions

At the first stage, a list of criteria for the automation system was formed, namely:

• open source;

• ease of use;

• affordability (less than 50000 rubles);

• user-friendly interface;

• ability to build a railway station model and its visualization in the form of a graph.

The authors assume that this list of criteria for the automation system is necessary to achieve this goal. Thus, based on these criteria, a number of the following software systems were selected.

1. *Topomatic Robur* software system

The system [9] contains tools for designing railway crossings and stations and forming models of transport infrastructure facilities, providing an opportunity to design new and reconstruct existing railways and stations.

2. *GeoniCS* software package

The complex [10, 11] is a CAD system for railway design. It is focused on assistance in the design of new tracks, restoration and overhaul of existing railways.

3. *Graphviz* tool

This tool [12] is an open source utility package for automatic graph visualization. It offers its own templates for graph visualization and provides the opportunity to create user's own version of the representation.

4. *Higres* tool

The system [13, 14] allows you to create models, which are hierarchical graphs with a given semantics set by the user in the editor, as well as external modules for processing graphs with a certain semantics.

5. *Gephi* software

This software [15] is a multiplatform open source software focused on the visual representation and exploration of graphs.

Below is a comparative table of the results of research on the functionality of these systems.

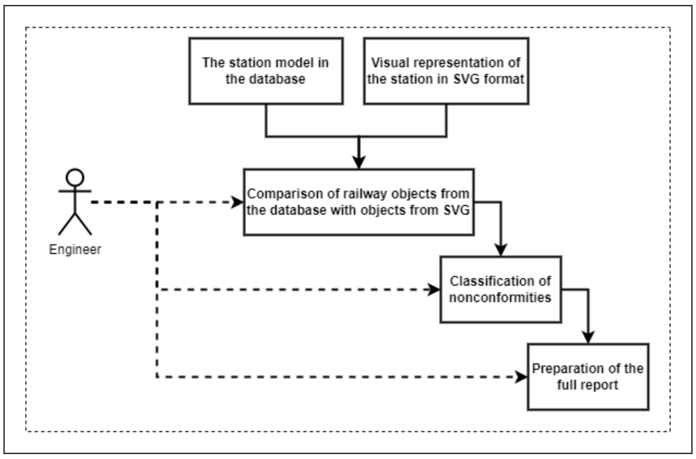
The station model in the database
Visual representation of the station in SVG format
Engineer
Comparison of railway objects from the database with objects from SVG
Classification of nonconformities
Preparation of the full report

Fig. 2. Use case diagram: before automation

Table 1

Overview of existing solutions

System	Open Source	Ease of use	Price, RUB	User-friendly interface	Building of a railway station model and its visualization in the form of a graph
Topomatic Robur	–	–	139900	+	–
GeoniCS	–	–	129600	+	–
Graphviz	+	+	–	–	–
Higres	–	+	–	+	–
Gephi	+	+	–	+	–

From the comparative data presented above, it follows that no system supports simultaneously the construction of a railway station model and its visualization in the form of a graph, which entails the need to implement our own method of automated comparison of the model and visual representations of a railway station plan.

Architecture and algorithm development

To study the architecture of the system, use case diagrams were designed [16]. The traditional method of comparing railway objects requires manual work from the tester (Fig. 2).

The proposed automation method [17, 18] for object matching is presented in Fig. 3.

Due to the absence of a human factor, the proposed method has a significant advantage – the probability of error during automated verification is much lower, and the verification speed is several times higher.

Then the flowchart of the algorithm was developed, which subsequently formed the basis of the software project (Fig. 4).

Next, it is necessary to consider the algorithm in more detail based on the above scheme.

Getting railway objects from the database

To construct a graph [19] describing the relations of neighboring railway facilities and their characteristics, it is necessary to obtain appropriate data. The graph elements include arrows, traffic lights, and track circuits.

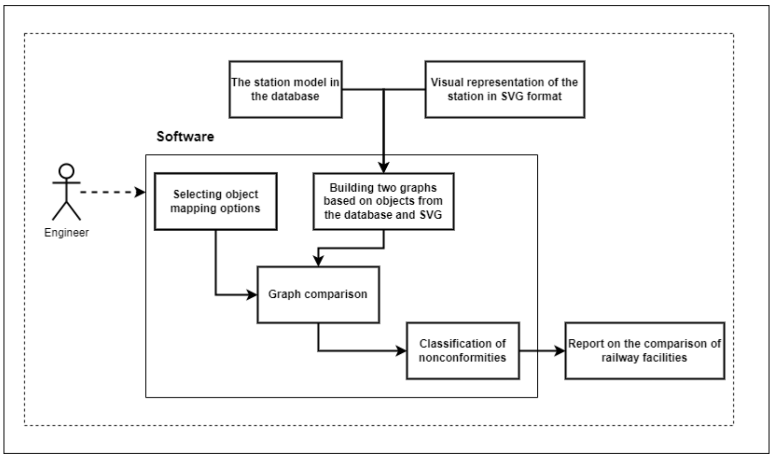
The station model in the database
Visual representation of the station in SVG format
Software
Engineer
Selecting object mapping options
Building two graphs based on objects from the database and SVG
Graph comparison
Classification of nonconformities
Report on the comparison of railway facilities

Fig. 3. Use case diagram: after automation

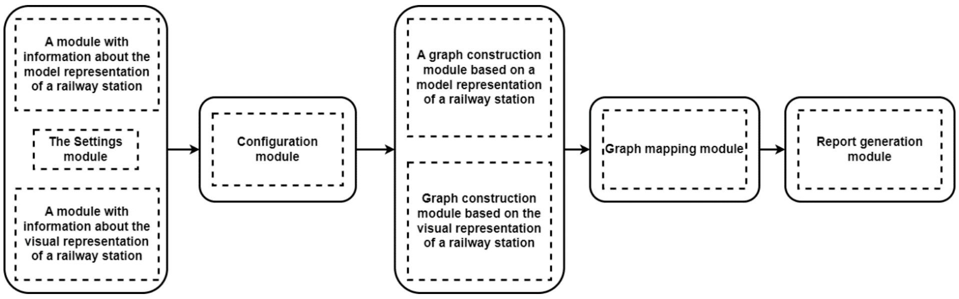
A module with information about the model representation of a railway station
The Settings module
A module with information about the visual representation of a railway station
Configuration module
A graph construction module based on a model representation of a railway station
Graph construction module based on the visual representation of a railway station
Graph mapping module
Report generation module

Fig. 4. The flowchart of the project

A database with railway objects is usually a model of a real railway station, part of the tracks, etc. Therefore, the database already contains all the necessary characteristics and values in order to determine which objects are connected. Thus, several SQL queries are enough to extract useful information from the database [20]. It is necessary to take into account that the graph can either coincide or diverge from the representation displayed in the database. The actual and reflected visual representation of one railway station is shown in Fig. 5.

Geolines are directional objects that make up track circuits and arrows. Each geoline has a "beginning" and "end" fields. To solve the orientation problem (related to the fact that the graph may differ from the database representation), it is necessary to add an option that reflects the order of objects from left to right. This includes reassigning the "beginning" and "end" of geolines, as well as reflecting the coordinates according to the rule $Lnew = Lmax + Lmin - L$, where L is the linear coordinate of the object, and $Lmax$ and $Lmin$ are the maximum and minimum coordinates among the list of all objects.

Getting railway objects from SVG file

The task is to read information about railway objects from SVG file and save the objects for further processing by the program. SVG file is a set of tags. Objects consist of lines and other elements and have a name, coordinates of key points, etc. in the description [21]. Operations that are performed on coordinates are also described. The task is to perform operations on all points to obtain the absolute coordinates of objects [22], and then save them along with the rest of the information about the objects.

First, the desired line is found and the coordinates of the object points are stored. Then all operations on the object are performed step by step. When all the points of the object are remembered and saved,

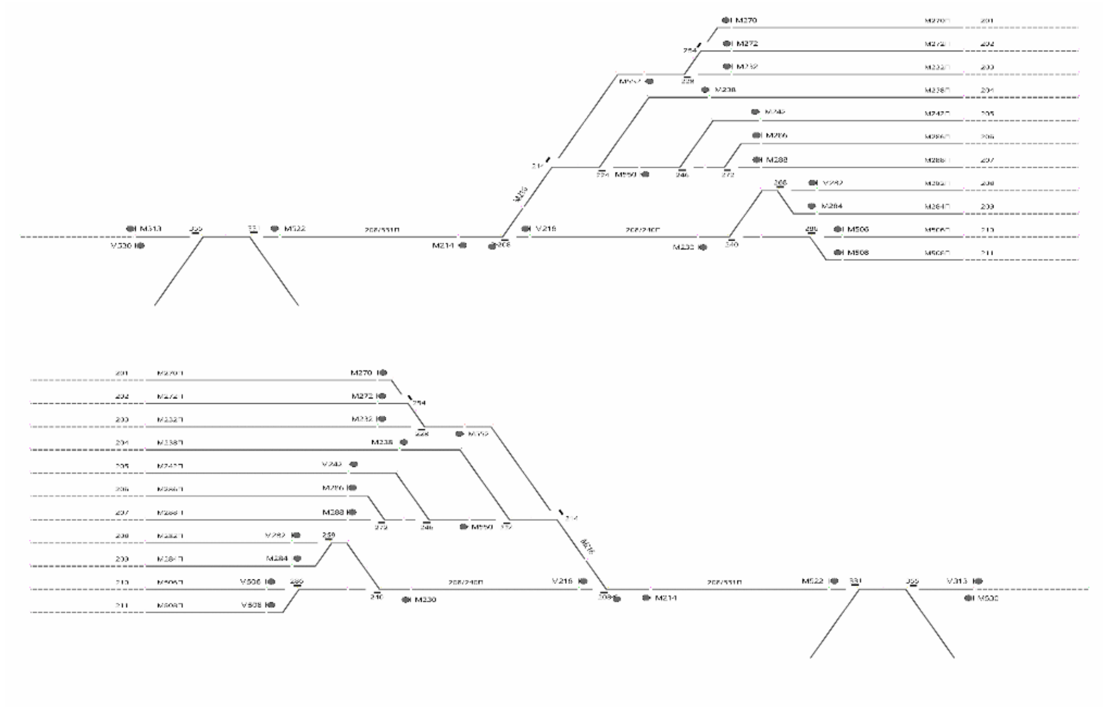

Fig. 5. Visual representations with different orientations

another one is created – *zero* (zero of the object), relative to which the rotation and reflection of the remaining points are carried out.

There are three types of operations used for SVG in this project. **Translate** moves an object in two-dimensional space; **rotate** rotates it by some angle; **scale** reflects the object along one or both axes (parameters – 1 or –1). These are affine transformations; they can be expressed in terms of affine transformation matrices [23, 24], however, frequent matrix multiplication takes longer than an algorithm based on the features of affine transformations [25]. During the writing and testing process, it was recorded:

1. When an object is rotated/reflected, and then moved along any of the axes, the axes are also rotated/reflected, and movement is carried out along the rotated/reflected axis, and not along the absolute one.
2. After rotation, the object is reflected along the rotated axes.
3. After reflection along one of the axes, the direction of rotation changes to the opposite.

All the observations obtained were incorporated into the algorithm. The operation implementation is based on the ***refindCoordWithCorner*** function. It allows you to find new coordinates of a point when the axes are rotated by a certain angle. It takes as parameters the old coordinates and the angle by which the point should be rotated. Let *xTranslate* be the old coordinate of the point on the x axis, *yTranslate* be the old coordinate of the point on the y axis, *curRotate* be the current angle between the vector drawn to the point from the origin and the vector of the positive direction of the x axis. If *xTranslate* = 0, this angle is 90 or –90. Otherwise, arctg(*yTranslate*/*xTranslate*) is taken (with an addition of 180 – in case of a negative *xTranslate*). Then, the angle to which the axes should be rotated, is added to the calculated angle. The new coordinates are calculated as follows:

1. The distance between the origin and the point is measured using the formula of the hypotenuse of a right triangle:

$$dist = \sqrt{xTranslate^2 + yTranslate^2}.$$

2. The coordinates are determined by multiplying the distance by the sine and cosine of the new angle: x = *dist* * cos(*curRotate*), y = –*dist* * sin(*curRotate*).

To implement the operations, we introduce a number of variables:

1. *sumRotation* is the sum of all the angles that the point was rotated by during the operations;
2. *rotationDirection* is the current direction of rotation of the object, equal to 1 or –1 (counterclockwise or clockwise, depending on previous reflection operations; if the object is reflected along one of the axes, its direction of rotation is opposite to the standard one);
3. *xDirection*, *yDirection* are equal to 1 or –1; show whether the object is currently reflected along two-dimensional axes or not;
4. *zero* is the zero of the object relative to which the reflection and rotation operations are performed.

Implementation of **rotation**: the angle to which the object should be rotated is indicated in parentheses. The angle is multiplied by the *rotationDirection* and added to the *sumRotation*. The coordinates of the current point relative to zero are determined; they are passed to the *refindCoordWithCorner* function along with the angle. The resulting values are added to zero.

Reflection implementation (**scale**): *xDirection* and *yDirection* are the values of the point reflection. The initial values of the current point's coordinate relative to zero are determined; they are multiplied by *xDirection* and *yDirection* and passed to *refindCoordWithCorner* along with the *sumRotation* angle. The resulting values are added to zero. These are the coordinates of the reflected point. Then the *rotationDirection*, *xDirection*, and *yDirection* are updated.

Translation implementation: *xTranslate*, *yTranslate* are the values of the movement. They are multiplied by *xDirection* and *yDirection* and passed to the *refindCoordWithCorner* function along with the *sumRotation*. The resulting values are added to the current point.

After calculating the absolute coordinates, the objects are filled in.

Plotting graphs based on data from a database and an SVG file

After all the necessary objects have been obtained from each representation of the railway station, you can start building the graph. It was decided to divide the construction into two stages: building a graph of track circuits and arrows and attaching traffic lights to the graph built in the previous stage.

This solution was chosen because the traffic light has a single coordinate, which is the "junction" between the track circuits and the arrows.

Thus, first, a graph is constructed, determining the neighborhood of track circuits and arrows, and then for each node of the graph it is determined which traffic lights belong to it.

Ultimately, the graph is represented as a structure with fields; the *structure* field is a dictionary in which the key is the node of the graph, and the value is a list of neighbor nodes. Below is an example in the form of five entries from this dictionary.

```
203 Circuit: none; 355 Switcher; none;
355 Switcher: 203 Circuit; 331 Switcher; none;
331 Switcher: 355 Switcher; 208/331П Circuit; none;
208/331П Circuit: 331 Switcher; 208 Switcher; none;
208 Switcher: 208/331П Circuit; 260/240П Circuit; 214 Switcher;
```

Listing 1 – *Structure* dictionary entries

Consider object 203:

1. The value "Circuit" indicates that this object is a track circuit.
2. The first element has the value "none" – the object in question is the leftmost in its group and has no neighbors on the left.
3. The second element of the list has the value "355 Switcher" – to the right of our object there is an object 355, which is an arrow.

4. The third element of the list has the value “none”, which is typical for objects of the track circuit type, because the third element is designed to show which element is adjacent to an object of the arrow type along the turn line (for example, for arrow 208 along the turn line, arrow 214 will be the neighbor).

The *trafficlights* field is a dictionary in which the key is the node of the graph, and the value is a list of traffic lights located at the “ends” of this object. Below is an example in the form of five entries from this dictionary.

```
203 Circuit: M530, M313
355 Switcher: M530, M313;
331 Switcher: M522;
208/331П Circuit: M522, M214;
208 Switcher: M214, M218, M216;
```

Listing 2 – *Trafficlights* dictionary entries

It can be seen that the M530 and M313 traffic lights are located on the track circuit 203, as well as on the arrow 355, which allows us to conclude that these traffic lights are located between these two objects.

The *ends* field is an auxiliary dictionary for determining the neighbor when adding another track circuit or arrow. In this dictionary, the key is a point that is the “end” of a geoline belonging to a track circuit or an arrow. Therefore, the track circuit has only two such points, because it consists of one geoline, and the arrow has four such points, because the arrow consists of three geolines converging at one point.

The point at which the arrow's geolines converge must also be considered as the “end point”, because there are situations in which two arrows have common geolines.

Because the track circuits and arrows differ from each other, a structure was introduced that transforms each object into a node of the graph.

After converting all objects into nodes, it is necessary to sort them by coordinates, after which each of the nodes joins the graph in turn.

The algorithm for attaching a node X to a graph:

1. Check if there are points in the auxiliary dictionary of the *ends* graph that coincide with the ends of X located in the end field.

2. If none of the points match, then X does not have any neighbors yet, so X is added to the *structure* dictionary as a key, and the value for this key is “none, none, none”. We also need to add all the endpoints of X to the dictionary of endpoints of the *ends* graph so that the next nodes that are being added can join X.

3. If at least one point coincides with an already existing point in the dictionary of graph endpoints, then further steps are taken depending on what type of object X has.

4. If X is a track circuit, then you need to consider two situations of joining a neighbor Y:

a. Y is the neighbor on the right.

If **Y is a track circuit**, then we change the value of the key Y in the *structure* dictionary to the same list, only X will be the first element.

If **Y is an arrow**, then we look at which of the endpoints X coincides with Y, and depending on this we decide whether to put X as the neighbor on the left (the first element of the list) or the neighbor along the rotation line (the third element of the list).

After that, an entry for the key X is added to the *structure*, and the value for this key is “none, Y, none”. Next, the endpoint is removed from the auxiliary dictionary, which is now the junction point of objects X and Y, and the remaining point is added, which was not used when joining X.

b. Y is the neighbor on the left.

Similar to the previous situation, only the first and second elements are interchanged for each object.

5. If X is an arrow, then we need to consider the situations when adding to 1 neighbor and to 2 neighbors, but all actions will be similar to attaching a track circuit:

a. update the data in the nodes that the new node is joining;

b. take into account the possible positions of the "turn line" for objects of the arrow type, checking which end of the arrow the new node is attached to;

c. clear the dictionary containing the ends of the graph from the already occupied ends in order not to check once again those to which it should not be physically possible to attach the object. The exception is the center points of arrow type objects.

Once all the track circuits and arrows have been added to the graph, we can proceed to the next stage.

This stage is divided into 2 parts:

1. Filling in the *tl_to_attach* dictionary, the key of which is the id of the geo point (for objects from the database) or XY coordinates (for objects from SVG), and the value is a list of traffic lights attached to this geo point/coordinate.

2. For each node of graph X (track circuit or arrow), a list of traffic lights belonging to one of the endpoints of X. This list is placed in the *trafficlights* dictionary by the key X.

At this stage, the construction of the graph of railway objects can be considered fully completed.

Graph comparison

Here we compare asymmetric graphs whose nodes are different objects [26]: track circuits, arrows, traffic lights. A node structure was invented to compare these objects, which brought them to a single view. In addition, the algorithm must identify different types of errors and mark nodes with different colours [27]: green – no errors, yellow – name error, red – location error or node is missing.

The comparison algorithm consists of the following sequential steps:

1. Checking all nodes for presence in another graph.
2. Check for duplicate nodes, which are track circuits and arrows.
3. Check for non-repeating nodes, which are track circuits and arrows.
4. Checking the nodes that are traffic lights.

Since the graphs are asymmetric, most of the steps are duplicated for both graphs. Next, let us look at each of the steps of the algorithm.

1. Checking all nodes for presence in another graph.

At this step of the algorithm, the program sequentially passes through all nodes in both graphs and searches for a matching pair in the other graph for each node. If a match is found, the node is considered paired. Nodes are only considered paired if they have the same name and type.

2. Check for duplicate nodes, which are track circuit and arrow objects.

At this stage of the algorithm, we check the neighbours of each node, looking for duplicates. A node can have at most three neighbours: the top, front, and turn neighbours. Only arrow type objects can have three neighbours; the rest always have at most two neighbours. This is because of physical factors, as this algorithm was designed for railway station layouts, not abstract graphs.

The check begins with the first paired node in the graph. Its neighbours also have a paired node in another graph. If the neighbouring nodes match, the node is marked as green; if not, it's marked red. Then the same process is repeated for all other nodes.

3. Check for non-repeating nodes, which are track circuits and arrows.

At this stage of the algorithm, the neighbours of the nodes are checked, which are not repeated. The check starts from the first unpaired node in the graph. All its neighbours that have a pair are found, as well as its position relative to the top node: the front neighbour or the neighbour on the turn. And then it is checked whether their pairs in another graph have a neighbour with the same location and type. If the same node is located from each paired neighbour, then both nodes are marked yellow, if not, they are marked red.

4. Checking the nodes that are traffic lights.

The traffic light is located at the junction of two track circuits, so it will always have two neighbours. When comparing traffic lights, their name and the nearest neighbours are compared. Since checking traffic lights is the final stage of the comparison algorithm, when comparing traffic lights, the colour of their neighbours' markings is considered.

If the traffic light has a pair in another column, then it can only be marked in red or green. If the traffic light already has a pair in another graph, it makes no sense to check the name match again, so at this step the neighbours of the traffic lights, which are marked green, play the main role. If the nearest neighbour of the traffic light turns out to be marked red, the nearest green neighbours are taken. If they match, then the traffic lights are marked green, if not, they are marked red.

If the traffic light does not have a pair in another column, then it can only be marked in red or yellow. This decision is made based on its neighbours and the neighbours of the traffic light with which the comparison takes place. The nearest neighbour with a green mark is located, based on it, a traffic light without a pair is searched in another graph, and if the other neighbours match, the traffic lights turn yellow, otherwise red.

This comparison algorithm works correctly in most cases. Errors occur only in some atypical connections. Most of the atypical connections are associated with arrows and the problem of unpredictable rail connections to them.

Software implementation

Based on the created method of searching for the correspondence of the railway station model to its visual representation, software was developed.

Requirements have been formulated for the product – the user, by uploading two files containing a model and a visual representation of the railway station, should receive an output including a log file describing the inconsistencies found, as well as SVG file on which:

1. the objects for which a complete match has been found in the electronic map are displayed in green;
2. displayed in yellow are those objects for which a topological correspondence has been found in the electronic map, but there is some discrepancy in characteristics;
3. displayed in red are those objects for which no match has been found in the electronic map.

Non-functional requirements have been formulated for the product:

1. programming language – Python 3.11;
2. the interface language is Russian;
3. platform – Windows 10;
4. minimalistic application design.

The requirements were met by implementing a simple graphical application using the PySide6 library [28]. The application contains buttons for uploading files, selecting the type of data reflection and starting file comparison – all the necessary functionality.

The Git version control system was used during development [29].

Results of the system operation

In order to demonstrate the reduction of labor intensity and increase the accuracy of the process of searching for compliance with the railway station model using a visual representation, a comparison of the time spent on work and the number of inconsistencies found with and without using the program was carried out.

Three employees of the same qualifications participated in the measurements, who independently compared two representations of the same station.

It is demonstrated in Table 2 how long each stage of the work took, from which it is possible to obtain an average value. The average number of inconsistencies found is also determined.

Then the employees compared the representations of the railway station (already different, but similar in complexity, structure and number of inconsistencies of the first station).

Table 2

Assessment of the complexity of the non-automated process

	Worker 1	**Worker 2**	**Worker 3**	**Average**
SVG and DB compliance analysis, min.	16	12	20	16
SVG adjustment, min.	8	10	15	11
Search and cataloging of DB inconsistencies, min.	27	24	24	25
DB adjustment, min.	13	13	16	14
Re-validation of SVG and DB compliance, min.	8	4	4	5
The whole process, min.	72	63	79	**71**
The number of inconsistencies found	3	7	5	**5**

In this experiment, automation of the process was used (Table 3).

The calculations are similar to the previous table.

Table 3

Assessment of the complexity of the automated process

	Worker 1	**Worker 2**	**Worker 3**	**Average**
File selection, program launch, min.	1	1	1	1
SVG correction based on the results of automated analysis, min.	4	5	3	4
Database correction based on the results of automated analysis, min.	6	8	10	8
Re-validation of SVG and DB compliance, min.	2	2	2	2
The whole process, min.	12	15	15	**14**
The number of inconsistencies found	10	10	10	**10**

The assessment of labor intensity is carried out according to a formula combining a simple summation of the resources spent on individual components and division by the number of these components [30]. The average time value in each of the sections is determined by the following formula:

$$S = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} Tij}{n},$$

where Tij is the time spent on stage j by employee i, n is the number of employees, $n = 3$; m is the number of stages; for the first section (“Before automating the process”) $m = 5$, for the second section (“After automating the process”) $m = 4$.

Time was measured in seconds, rounded up to minutes; rounding up occurred if the number of seconds that made up the remainder of the minutes exceeded 30. Thus, we observe that the automation of the process has significantly reduced the average time required to complete the work – from 71 minutes to 14 minutes, that is, by about 5 times. To demonstrate the results of reducing labor intensity, a histogram was constructed (Fig. 6) as one of the most common and effective ways to visualize statistical data [31].

The experiment was conducted for a small station (up to 30 graphic elements). During the further operation of the software, it was revealed that the time gain of the automated process over the non-automated one increases with the growth of the station size (average values are presented):

1. small station (up to 30 graphic elements) – 5 times;
2. medium-sized station (from 31 to 100 graphic elements) – 5.5 times;
3. large station (from 101 graphic elements) – 6.5 times.

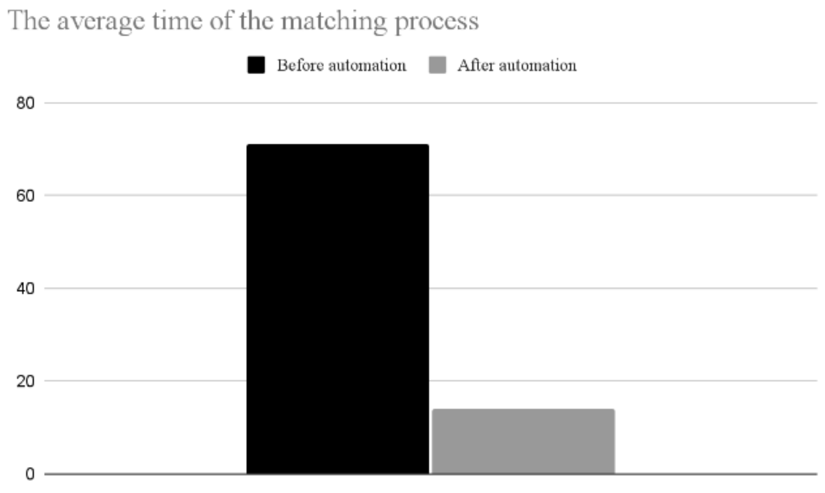

Fig. 6. The average time of the matching process

In addition, automation improves accuracy: instead of five inconsistencies found, 10 were found – and these are all the inconsistencies that were between the stations initially. Thus, automation made it possible to increase accuracy by $N = S2/S1 = 2$ times, where $S1$ is the number of inconsistencies found before automating the process, $S2$ is the number of inconsistencies found after automating the process.

Conclusion

This article presents an overview and comprehensive comparative analysis of existing approaches to comparing models to a real object. An innovative method for searching and comparing objects of the railway station model to the corresponding objects of the visual representation of the station is proposed. The proposed algorithm is implemented in the model matching software. The results of reducing labor intensity by 5 times and increasing accuracy by 2 times are demonstrated.

The work was carried out under the supervision of the Scientific Research and Design Institute of Informatization, Automation and Communications in Railway Transport.

REFERENCES

1. **Okladnikov S.M. et al.** Transport v Rossii. 2020: Statisticheskii sbornik [Transport in Russia. 2020: Statistical Digest]. Moscow: Federal'naia sluzhba gosudarstvennoi statistiki, 2020, 108 p.

2. **Kanashin N.V.** Creation of digital models of railway stations by terrestrial laser scanning. Transport of the Russian Federation, 2010, Vol. 28, No. 3, Pp. 66–67.

3. **Umansky V.I., Dolganuk S.I.** Digital models of station track development. World of Transport and Transportation, 2014, Vol. 12, No. 1(50), Pp. 126–133.

4. **Telegin A.I., Timofeev D.N., Chitalov D.I., Pudovkina S.G.** SVG-marking of two-dimensional graphics: The experience of SVG-using in creating of two-dimensional graphics. Miass: South Ural State University (National Research University), 2015, 73 p.

5. **Mikheev P.N.** SVG – novyi standart vektornoi grafiki v Web [SVG – the new standard for vector graphics on the Web]. Journal of Radio Electronics, 2001, Vol. 9, Pp. 7.

6. **Ivlev V.A., Nikiforov I.V., Yusupova O.A.** Automation method for configuring IT infrastructure for IT projects. Proceedings SPIE 12637, International Conference on Digital Transformation: Informatics, Economics, and Education (DTIEE2023), 2023. DOI: 10.1117/12.2680779

7. **Lutsenko A.S., Ustinova V.E., Ivlev V.A., Mironenkov G.V.** Realizatsiia programmnogo obespechenie dlia verifikatsii sootvetstviia model'nogo i vizual'nogo predstavleniia plana zheleznodorozhnoi stantsii [Implementation of software for verification of conformity between model and visual representation of a railway station plan]. Sovremennye tekhnologii v teorii i praktike programmirovaniia [Modern technologies in the theory and practice of programming], 2024, Pp. 59–61.

8. **Shalin A.P., Batrakov V.N.** Verification and validation in conformity assessment. Production Quality Control, 2022, Vol. 4, Pp. 47–50.

9. **Smogunov V.V., Mitrokhina N.Yu.** Sistemnyi analiz metodov proektirovaniia avtomobil'nykh dorog [System analysis of road design methods]. University proceedings. Volga region. Technical sciences, 2011, Vol. 20, No. 4, Pp. 116–127.

10. **Buchkin V.A., Ryzhik E.A., Lenchenkova E.P.** Comparative Analysis of Software Packages. World of Transport and Transportation, 2013, Vol. 11, No. 2(46), Pp. 112–121.

11. **Soboleva E.L., Archipova O.B.** Research of geoengineering systems for computer-aided design. Geo-Siberia, 2010, Vol. 1, No. 2, Pp. 42–45.

12. **Kozonogova E.V., Kurushin D.S., Dubrovskaya J.V.** Computer visualization of the identify industrial clusters task using GVMap. Scientific Visualization, 2019, Vol. 11, No. 5, Pp. 126–141. DOI: 10.26583/sv.11.5.11

13. **Gordeev D.S.** A survey of visualization techniques of algorithms on graphs. Scientific Visualization, 2018, Vol. 10, No. 1, Pp. 18–48. DOI: 10.26583/sv.10.1.02

14. **Kasyanov V.N., Zolotuhin T.A.** Visual graph – a system for visualization of big size complex structural information on the base of graph models. Scientific Visualization, 2015, Vol. 7, No. 4, Pp. 44–59.

15. **Ulizko M.S., Antonov E.V., Artamonov A.A., Tukumbetova R.R.** Visualization of Graph-based representations for analyzing related multidimensional objects. Scientific Visualization, 2020, Vol. 12, No. 4, Pp. 133–142. DOI: 10.26583/sv.12.4.12

16. **Janushko V.V., Erkin S.N.** Construction of the process automation product design based on UML (use case) diagrams. Izvestiya SFedU. Engineering Sciences, 2009, Vol. 101, No. 12, Pp. 64–71.

17. **Petrov A.A., Nikiforov I.V., Ustinov S.M.** Algorithm of ESXi cluster migration between different vCenter servers with the ability to rollback. Informatsionno-upravliaiushchie sistemy [Information and Control Systems], 2022, Vol. 2, Pp. 20–31. DOI: 10.31799/1684-8853-2022-2-20-31

18. **Kovalev A.D., Nikiforov I.V., Drobintsev P.D.** Automated approach to semantic search through software documentation based on Doc2Vec algorithm. Informatsionno-upravliaiushchie sistemy [Information and Control Systems], 2021, Vol. 1, Pp. 17–27. DOI: 10.31799/1684-8853-2021-1-17-27

19. **Magomedov A.M.** Postroenie i vizual'noe redaktirovanie grafa s sokrashchennymi spiskami smezhnosti vershin [Construction and visual editing of a graph with reduced adjacency lists of vertices]. State registration of a computer program RU 2017617670, 2017.

20. **Berezutskaia L.A., Troshkina G.N., Iudintsev A.Iu.** SQLite main commands and their comparison to SQL for an Android app. LinguaNet, 2019, Pp. 289–293.

21. **Sedykh D.V., Zuyev D.V., Gordon M.A.** Branch format of technical documentation on devices of railway automation and remote control. Part 4. Presentation of elements. Transport Automation Research, 2017, Vol. 3, No. 4, Pp. 563–577.

22. **Ivlev V.A., Nikiforov I.V., Leont'eva T.V.** Obrabotka dannykh v geoinformatsionnykh sistemakh dlia vybora mestopolozheniia reklamy [Processing data in geographic information systems to select advertising locations]. Sovremennye tekhnologii v teorii i praktike programmirovaniia [Modern technologies in the theory and practice of programming], 2019, Pp. 27–30.

23. **Lisyak V.V., Lisyak M.V.** The resulting matrix of interactive geometrical transformations composition search method in CAD. Izvestiya SFedU. Engineering Sciences, 2008, Vol. 81, No. 4, Pp. 73–78.

24. **Kudrina M.A., Murzin A.V.** Affinnye preobrazovaniia ob"ektov v komp'iuternoi grafike [Affine transformations of objects in computer graphics]. Trudy mezhdunarodnogo simpoziuma "Nadezhnost' i kachestvo" [Proceedings of the International Symposium "Reliability and Quality"], 2014, Vol. 1, Pp. 307–310.

25. **Ustinova V.E., SHpak A.V., Ivlev V.A., Mironenkov G.V.** Sistema preobrazovaniia koordinat ob"ektov v svg-faile dlia poiska sootvetstviia modeli zheleznodorozhnoi stantsii ee vizual'nomu predstavleniiu [System of transformation of coordinates of objects in svg-file for search of correspondence of model of railway station to its visual representation]. Sovremennye tekhnologii v teorii i praktike programmirovaniia [Modern technologies in the theory and practice of programming], 2024, Pp. 132–134.

26. **Sysoev V.V.** A Framework Similarity Estimation of Unweighted Undirected Graphs. Modern Information Technologies and IT-Education, 2022, Vol. 18, No. 3, Pp. 655–665. DOI: 10.25559/SITITO.18.202203.655-665

27. **Khonina O.I., Zabrodin A.V.** Graph Coloring Problem in the Context of Schedule Optimization: Software Solution. Intellectual Technologies on Transport, 2023, Vol. 35, No. 3, Pp. 32–37. DOI: 10.24412/2413-2527-2023-335-32-37

28. **Fataliyev A.** Most important TPPs for performing graphical user interface and scientific calculations in Python programming environment. The Scientific Heritage, 2023, Vol. 124, Pp. 37–41. DOI: 10.5281/zenodo.10077616

29. **Voinov N., Rodriguez Garzon K., Nikiforov I., Drobintsev P.** Big Data Processing System for Analysis of GitHub Events. 2019 XXII International Conference on Soft Computing and Measurements (SCM), 2019, Pp. 187–190. DOI: 10.1109/SCM.2019.8903782

30. **Zharinov I., Zharinov O., Shek-Iovsepyantz R., Suslov V.** Robustness drop estimation of design documentation preparation by CALS-technologies in instrument making. Scientific and Technical Journal of Information Technologies, Mechanics and Optics, 2012, Vol. 80, No. 4, Pp. 151–153.

31. **Egorova N.E., Arbuzova A.A.** On the formation of the skill of processing and visualization of statistical data. Pozharnaia i avariinaia bezopasnost' [Fire and emergency safety], 2019, Vol. 13, No. 2, Pp. 38–45.

INFORMATION ABOUT AUTHORS / СВЕДЕНИЯ ОБ АВТОРАХ

Ustinova Valeria E.
Устинова Валерия Евгеньевна
E-mail: ustlera@list.ru

Lutsenko Anton S.
Луценко Антон Степанович
E-mail: anton.lutsenko.03@mail.ru

Shpak Adelina V.
Шпак Аделина Владимировна
E-mail: adelina.shpak@yandex.ru

Mironenkov Grigorii V.
Мироненков Григорий Васильевич
E-mail: mironenkov97@gmail.com

Ivlev Vladislav A.
Ивлев Владислав Александрович
E-mail: nevidd@yandex.ru

Submitted: 24.06.2024; Approved: 05.11.2024; Accepted: 18.11.2024.
Поступила: 24.06.2024; Одобрена: 05.11.2024; Принята: 18.11.2024.