RAILWAY STATION ROUTING ALGORITHM USING THE BACKTRACKING METHOD

Maria Catrina (GEACĂR)[1]

The problem of routing trains through railway stations can be addressed at multiple levels. This paper focuses on the problem of route determination and uses graph theory and a recursive algorithm to determine all possible routes between two track elements within the railway station.

The railway station topology is represented as a graph, with edges having associated the corresponding distances and/or track element status information. The method then uses the backtracking algorithm to determine all possible routing possibilities for trains within the station.

Keywords: route setting, graph, backtracking, ARS

1. Introduction

The network of elements of a railway station can easily be represented as a graph. Determining the routes for a given direction can be performed by "directing" the graph edges in order to ensure proper orientation and can be used in automatic route setting systems. The ARS systems release the signalman from unnecessary pre-setting of the elements required for establishing the route and gives time for other activities [1].

Intelligent Transport Systems for railways apply the automatic piloting of trains in order to maximize the use of track capacity, while reducing train delays [2].

Conclusions presented in the experiment from [3] show that automation provided by Automatic Route Setting Systems improve significantly the performance, as compared to manual condition, and performance is considerably more consistent when working with ARS.

An efficient train routing system can be obtained by setting a priority for each route. In many cases, automatic routing systems are based on a predefined list of routes, without a search in real time, according to the railway station conditions. In [4], the latest ARS systems use a route selection processor for reading the route from the train routing file according to the train description provided by the Automatic Train Tracking systems.

[1] Eng., Faculty of Transports, University POLITEHNICA of Bucharest, Romania, e-mail: maria.geacar@gmail.com

The paper presents a solution for determining routes in real time using a backtracking algorithm.

One of the key aspects of the backtracking algorithm relevant for this application is the ability to test the partial solutions and eliminate large branches of the search tree [5]. At the same time, the algorithm is searching until the last possible solution is given and it provides all the solutions.

The proposed method is based on associating each edge in the graph with the distance between the corresponding elements in the railway station topology. The problem of efficient route setting is therefore transformed into an optimal route problem between two graph vertices. In this case, the problem is solved using the backtracking algorithm to generate all possible routes. These routes are then sorted based on the total distance between the source and target vertices.

Some vertices in the graph can be "cancelled" or redesigned to meet certain requirements for the routing system and to better model the railway station topology.

2. Graphs. Graph theory

A graph [6] (directed or undirected) is an ordered pair of sets $G = (V, E)$. V is a non-empty and finite set of elements called *vertices*. E is a set of edges or lines, which are 2-element subsets of V (an edge is related with two vertices, and the relation is represented as an unordered/ordered pair of the vertices with respect to the particular edge).

The elements of E are unordered in the case of undirected graphs. The unordered pair of vertices x and y written as $[x, y]$.

In the case of directed graphs, the pairs in the E set are ordered. The pair of vertices x and y, written (x, y); x is called initial extremity of edge (x, y) and y is called final extremity of edge (x, y).

If an edge with extremities x and y exists, then vertices x and y are adjacent; each extremity of an edge is considered incident with the respective edge.

We will consider that the extremities of every edge are distinct (the graph does not contain any loops).

Between any given two vertices of the graph there can be at most only one edge.

The information associated with a graph can be as complex as required, but, in order to simplify the problem, we will consider that the vertices are labeled as numbers from 1 to n (where n is the number of vertices in the graph). This labeling is not a restriction (in the following chapters, the vertex number will

represent the position, within an array, of the information associated to the vertex).

A route (within a directed graph) is a sequence of vertices $(x_1, x_2, ..., x_n)$ in which for any pair of consecutive vertices x_i and x_{i+1} there is an edge (x_i, x_{i+1}).

The route is elementary if any vertex is encountered only once in the route. A route is simple if any edge is encountered only once in the route.

A circuit is a simple route in which the initial extremity is the same as the final extremity. A circuit is elementary if any vertex is encountered only once in the circuit (except for its extremities).

The circuit length represents the number of edges of the circuit.

3. Recursive algorithms. The backtracking method

The backtracking method [7], [8] is a programming technique that is applicable to algorithms offering multiple solutions and results in obtaining all solutions for a given problem. Each solution is stored in a stack-like data structure, in this case an array.

Because the backtracking algorithm determines all possible solutions, each final solution is built by updating the stack level by level, resulting in partial solutions. In order to be taken into account, both partial and final solutions must correspond to certain conditions, known as validation conditions. A solution that complies with such a condition is called a valid solution.

All stack configurations that represent final solutions consist of elements of the same well-defined set called the solution set. Each new partial solution is obtained by updating the previous partial solution with a new stack level. At each level, the algorithm uses untested values until obtaining a valid solution. Upon that the stack level is increased, the solution is updated and testing is restarted on the new level.

At a given moment, on a given level in the stack, all possible values in the solution set have been tested. In this case, the algorithm steps back to the previous stack level and resumes testing the values untested on this previous level.

Because testing on this level has been previously stopped when the algorithm encountered a value that generated a valid solution, it may contain other untested values. If no untested values generate a valid solution, then we step back another level in the stack. This stepping back gave the method the name backtracking.

By starting at the first level and repeating the algorithm until all values in the solution set have been tested on all stack levels, all final solutions can be obtained.

4. Simeria main railroad station. General information

The traffic signals in Simeria main railroad station are:
a) X end:
- Entry traffic signals: XO, XOF;XTO
- Exit traffic signals: Y1, Y2, Y3, YIV, YV, Y6, Y7, Y8;
- Route traffic signals: XOP, XOPF.

b) Y end:
- Entry traffic signals: YD, YDF, YP;
- Route traffic signals: X1, X2, X2, XIV, XV, X6;X7, X8;
- Exit traffic signals: YOP, YOPF.

c) to Triage:
- Exit traffic signals: YTJ, XP, XPF;
- Entry traffic signals: YC, YCF, YO.

5. Building the route list

The route list for Simeria railway station was built in two steps:
- the first step consists in building the directed graphs corresponding to the track equipment used for setting the routes (one graph for each direction);
- the second step is the application of the backtracking algorithm in order to determine all possible routes between two track elements.

The first step required the identification of track elements involved in setting the route, assigning a unique identifier for each of the elements (the label for the corresponding graph vertex) and the definition of the distance matrix.

The distance matrices (one for each graph), were defined according to the following rule:

$$M[x,y]=\begin{cases}0, \text{if there is no edge between vertices } x \text{ and } y \text{ (from } x \text{ to } y)\\ \text{the distance between } x \text{ and } y, \text{ if vertices } x \text{ and } y \text{ are}\\ \text{adjacent (from } x \text{ to } y)\end{cases} \quad (1)$$

The distance matrices for Simeria main railroad station are 69 by 69 matrices, corresponding to distances between the 69 determined track elements.

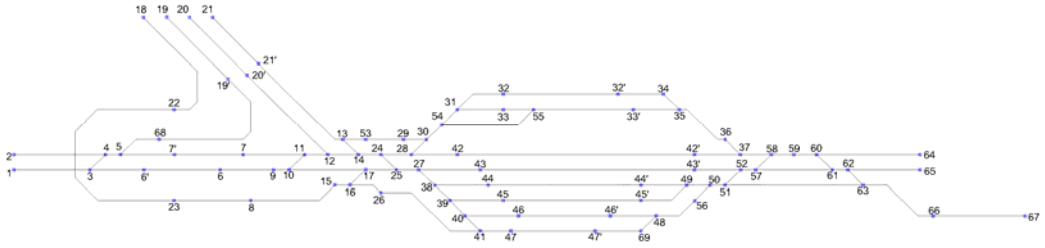

Fig. 1. Graph of Simeria railway station

The two matrices are different, reflecting the differences between the two directions, as well as the difference between the track elements taken into consideration (e.g. X1 for one direction and Y1 for the other).

Furthermore, the route list can be built taking into account the availability of track elements in the railroad station. For this reason, we can define the availability matrix as follows:

$$A[x,y]=\begin{cases}0, \text{if the track element between vertices } x \text{ and } y\\ (\text{from } x \text{ to } y) \text{ is not available (or does not exist)}\\ 1, \text{ if the track element between vertices } x \text{ and } y\\ (\text{from } x \text{ to } y) \text{ is available}\end{cases} \quad (2)$$

In the case that a railway traffic signal or a point machine does not validate the availability requirements, it cannot be used in the route searching procedure. In this case, the matrix elements (2) related to that particular traffic signal or point machine are cancelled (e.g. $a_{r,j}=0$ and $a_{i,r}=0$ for track element r, where $i,j=1,\ldots,n$).

Similarly, if one of the track sections is not available, the edge corresponding to its start and end vertices is cancelled (e.g. if the start and end vertices are p and q, $a_{p,q}=0$).

The availability matrix allows the generation of the route list based only on the availability of the track elements, disregarding the route distance.

If we wish to generate a route list based on the route distance, as well as the track element availability, based on rules (1) and (2) we can define another square matrix $T=(t_{i,j})$, where

$$t_{i,j}=m_{i,j}\cdot a_{i,j},\ i,j=1,\ldots,n \quad (3).$$

The route setting program was developed based on these matrices. The algorithm used for the program, using only the distance matrix (for simplification reasons) is presented next:

- the program starts on the row of the distance matrix corresponding to the starting track element of the route;

- the program searches for the first non-zero matrix element of this row (the first available edge);
- the column number of the non-zero element is stored in the solution vector;
- the procedure is repeated, recursively, on the row corresponding to the previously determined column number;
- when the program reaches the row number corresponding to the target vertex, a final solution has been determined and the program displays (or in this case stores in an output file) the determined route;
- if a final solution has been found, the program continues to search for another non-zero matrix element on the previous row;
- if there are no more non-zero elements on the current row, the program returns (backtracks) to the previous row and continues the search;
- the program ends when all possible edges beginning from the start vertex have been tested.

6. Implementation and results

This chapter presents a sample C++ program that determines the possible routes between entry signal XO from Orăştie and automatic block line exit toward Deva Bl. YDF.

```
/*Route determination program XO-Bl. YDF*/

#include <stdio.h>
#include <cstdio>
#include <iostream>
using namespace std;

int M[70][70], solution[70];
const int start = 1; /*start vertex*/
const int target = 65; /*target vertex*/
string name[70];
void readmatrix() /*reads distance matrix from matrixin.txt file*/
{
        int i, j;
                for (i = 1; i <= 69; i++)
                        for (j = 1; j <= 69; j++)
        cin>>M[i][j];
}
```

```
	void writesolution(int l, int dist) /*writes solution to XO-BlYDF.txt file*/
	{
		int i;
		cout<<name[start]<<" ";
		for (i = 1; i <= l; i++)
				cout<<name[solution[i]]<<" ";
			cout<<"Distance: "<<dist<<endl;
	}
	void back(int row, int solindex, int distance) /*recursive route determination
function*/
	{
		int ic,newdist;
		solution[solindex]=0;
		if (row == target) writesolution(solindex, distance);
		else for (ic = 1; ic <= 69;ic++)
		if (M[row][ic]>0)
		{
			solution[solindex] = ic;
			newdist = distance + M[row][ic];
			back(ic, solindex + 1, newdist);
		}
	}
	int main()
	{
	int i;
	  /*graph vertices names*/
	name[1]="XO";name[2]="XOF";name[3]="1";name[4]="3";name[5]="5";
	name[6]="XOP";name[7]="XOPF";name[8]="XTO";name[9]="11";name[10]="13";
	name[11]="15";name[12]="21";name[13]="19";name[14]="25";name[15]="27";
	name[16]="29";name[17]="35";name[18]="YB";name[19]="YO";
	name[20]="YCF";name[21]="YC";name[22]="BL1";name[23]="BL2";
	name[24]="31";name[25]="41";name[26]="39";name[27]="43";name[28]="37";
	name[29]="33";name[30]="45";name[31]="55";name[32]="X1";name[33]="X2";
	name[34]="38";name[35]="34";name[36]="32";name[37]="24";name[38]="57";
	name[39]="63";name[40]="65";name[41]="67";name[42]="XIII";
	name[43]="XIV";name[44]="X5";name[45]="X6";name[46]="X7";
	name[47]="X8";name[48]="42";name[49]="36";name[50]="30";name[51]="28";
	name[52]="22";name[53]="23";name[54]="51";name[55]="69";name[56]="40";
	name[57]="20";name[58]="18";name[59]="16";name[60]="14";name[61]="8";
	name[62]="6";name[63]="4";name[64]="Bl. YD";
	name[65]="Bl.YDF";name[66]="2";name[67]="Bl.YP";name[68]="7";
	name[69]="44";
	freopen("matrixin.txt","r",stdin);
	freopen("XO-BlYDF.txt","w",stdout);
		readmatrix(); /*read distance matrix*/
		for (i=1;i<=69;i++)
		solution[i]=0;
		back(start, 1, 0); /*initial call of recursive function. Paramaters: start vertex,
initial solution index, initial distance */
```

```
        return 0;
}
```

The resulting routes are shown in Table 1 below:

Table 1

Determined routes for XO-Bl. YDF

No.	XO – Bl. YDF possible routes (Traffic signals – point machines)	Length
1	XO 1 XOP 11 13 15 21 25 31 37 XIII 24 18 16 14 8 6 Bl. YDF	2664 m
2	XO 1 3 5 XOPF 15 21 25 31 37 XIII 24 18 16 14 8 6 Bl. YDF	2666 m
3	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 X6 36 30 28 22 20 8 6 Bl. YDF	2667 m
4	XO 1 XOP 11 13 15 21 25 31 41 43 57 X5 36 30 28 22 20 8 6 Bl. YDF	2667 m
5	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 X6 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2668 m
6	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 X6 36 30 28 22 20 8 6 Bl. YDF	2669 m
7	XO 1 XOP 11 13 15 21 25 31 41 43 57 X5 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2669 m
8	XO 1 3 5 XOPF 15 21 25 31 41 43 57 X5 36 30 28 22 20 8 6 Bl. YDF	2670 m
9	XO 1 XOP 11 13 15 21 25 31 37 45 51 69 X2 34 32 24 18 16 14 8 6 Bl. YDF	2670 m
10	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 X6 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2671 m
11	XO 1 XOP 11 13 15 21 25 31 37 45 51 55 X1 38 34 32 24 18 16 14 8 6 Bl. YDF	2671 m
12	XO 1 XOP 11 13 15 21 25 31 37 45 51 55 69 X2 34 32 24 18 16 14 8 6 Bl. YDF	2671 m
13	XO 1 3 5 XOPF 15 21 25 31 41 43 57 X5 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2672 m
14	XO 1 XOP 11 13 35 41 43 57 63 X6 36 30 28 22 20 8 6 Bl. YDF	2672 m
15	XO 1 3 5 XOPF 15 21 25 31 37 45 51 69 X2 34 32 24 18 16 14 8 6 Bl. YDF	2673 m
16	XO 1 XOP 11 13 35 41 43 57 X5 36 30 28 22 20 8 6 Bl. YDF	2673 m
17	XO 1 3 5 XOPF 15 21 25 31 37 45 51 55 X1 38 34 32 24 18 16 14 8 6 Bl. YDF	2674 m
18	XO 1 3 5 XOPF 15 21 25 31 37 45 51 55 69 X2 34 32 24 18 16 14 8 6 Bl. YDF	2674 m
19	XO 1 XOP 11 13 35 41 43 57 63 X6 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2674 m
20	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 65 X7 42 40 30 28 22 20 8 6 Bl. YDF	2675 m
21	XO 1 XOP 11 13 35 41 43 57 X5 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2675 m
22	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 65 67 X8 44 42 40 30 28 22 20 8 6 Bl. YDF	2677 m
23	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 65 X7 42 40 30 28 22 20 18 16 14 8 6	2677 m

No.	XO – Bl. YDF possible routes (Traffic signals – point machines)	Length
	Bl. YDF	
24	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 65 X7 42 40 30 28 22 20 8 6 Bl. YDF	2678 m
25	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 65 67 X8 44 42 40 30 28 22 20 18 16 14 8 6 Bl. YDF	2679 m
26	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 65 67 X8 44 42 40 30 28 22 20 8 6 Bl. YDF	2680 m
27	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 65 X7 42 40 30 28 22 20 18 16 14 8 6 Bl. YDF	2680 m
28	XO 1 XOP 11 13 35 41 43 57 63 65 X7 42 40 30 28 22 20 8 6 Bl. YDF	2681 m
29	XO 1 XOP 11 13 35 41 43 57 63 65 67 X8 44 42 40 30 28 22 20 8 6 Bl. YDF	2683 m
30	XO 1 XOP 11 13 35 41 43 57 63 65 X7 42 40 30 28 22 20 18 16 14 8 6 Bl. YDF	2683 m
31	XO 1 XOP 11 13 35 41 43 57 63 65 67 X8 44 42 40 30 28 22 20 18 16 14 8 6 Bl. YDF	2685 m
32	XO 1 XOP 11 13 15 21 25 31 41 43 XIV 22 20 8 6 Bl. YDF	2706 m
33	XO 1 XOP 11 13 15 21 25 31 41 43 XIV 22 20 18 16 14 8 6 Bl. YDF	2708 m
34	XO 1 3 5 XOPF 15 21 25 31 41 43 XIV 22 20 8 6 Bl. YDF	2709 m
35	XO 1 3 5 XOPF 15 21 25 31 41 43 XIV 22 20 18 16 14 8 6 Bl. YDF	2711 m
36	XO 1 XOP 11 13 35 41 43 XIV 22 20 8 6 Bl. YDF	2712 m
37	XO 1 XOP 11 13 35 41 43 XIV 22 20 18 16 14 8 6 Bl. YDF	2714 m

In order to evaluate the impact of the track element availability, we simulate a fault on point machine no. 35. When running the program using the updated matrix values (3) (for distance and availability), the result shows a total number of 24 routes (compared to a total of 37 for the distance-only case), as seen in Table 2:

Table 2

Determined routes for XO-Bl. YDF (point machine 35 unavailable)

No.	XO – Bl. YDF routes (Traffic signals – point machines)	Length
1.	XO 1 3 5 XOPF 15 21 25 31 37 XIII 24 18 16 14 8 6 Bl. YDF	2666 m
2.	YDFXO 1 XOP 11 13 15 21 25 31 41 43 57 63 X6 36 30 28 22 20 8 6 Bl. YDF	2667 m
3.	XO 1 XOP 11 13 15 21 25 31 41 43 57 X5 36 30 28 22 20 8 6 Bl. YDF	2667 m

No.	XO – Bl. YDF routes (Traffic signals – point machines)	Length
4.	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 X6 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2668 m
5.	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 X6 36 30 28 22 20 8 6 Bl. YDF	2669 m
6.	XO 1 XOP 11 13 15 21 25 31 41 43 57 X5 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2669 m
7.	XO 1 3 5 XOPF 15 21 25 31 41 43 57 X5 36 30 28 22 20 8 6 Bl. YDF	2670 m
8.	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 X6 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2671 m
9.	XO 1 3 5 XOPF 15 21 25 31 41 43 57 X5 36 30 28 22 20 18 16 14 8 6 Bl. YDF	2672 m
10.	XO 1 3 5 XOPF 15 21 25 31 37 45 51 69 X2 34 32 24 18 16 14 8 6 Bl. YDF	2673 m
11.	XO 1 3 5 XOPF 15 21 25 31 37 45 51 55 X1 38 34 32 24 18 16 14 8 6 Bl. YDF	2674 m
12.	XO 1 3 5 XOPF 15 21 25 31 37 45 51 55 69 X2 34 32 24 18 16 14 8 6 Bl. YDF	2674 m
13.	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 65 X7 42 40 30 28 22 20 8 6 Bl. YDF	2675 m
14.	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 65 67 X8 44 42 40 30 28 22 20 8 6 Bl. YDF	2677 m
15.	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 65 X7 42 40 30 28 22 20 18 16 14 8 6 Bl. YDF	2677 m
16.	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 65 X7 42 40 30 28 22 20 8 6 Bl. YDF	2678 m
17.	XO 1 XOP 11 13 15 21 25 31 41 43 57 63 65 67 X8 44 42 40 30 28 22 20 18 16 14 8 6 Bl. YDF	2679 m
18.	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 65 67 X8 44 42 40 30 28 22 20 8 6 Bl. YDF	2680 m
19.	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 65 X7 42 40 30 28 22 20 18 16 14 8 6 Bl. YDF	2680 m
20.	XO 1 3 5 XOPF 15 21 25 31 41 43 57 63 65 67 X8 44 42 40 30 28 22 20 18 16 14 8 6 Bl. YDF	2682 m
21.	XO 1 XOP 11 13 15 21 25 31 41 43 XIV 22 20 8 6 Bl. YDF	2706 m
22.	XO 1 XOP 11 13 15 21 25 31 41 43 XIV 22 20 18 16 14 8 6 Bl. YDF	2708 m
23.	XO 1 3 5 XOPF 15 21 25 31 41 43 XIV 22 20 8 6 Bl. YDF	2709 m
24.	XO 1 3 5 XOPF 15 21 25 31 41 43 XIV 22 20 18 16 14 8 6 Bl. YDF	2711 m

7. Conclusions

The results for the entire railway station contain several hundred routes, for more than one hundred route start-end combinations (for each one of the travel directions), showing the amount of time (and workload) required if the route lists are determined manually.

In order to determine the optimal route for given direction of the train and railway conditions, other algorithms could be used. The backtracking algorithm was chosen because it provides all possible routing solutions. By adding a graphical user interface, this method could be successfully used as a design tool (for the interlocking system) for any given railroad station.

If for a small railway station, the problem of routing trains can be easier to solve, for a main railroad station or a railway national node (like Simeria), determining routes for a specific direction can be difficult. As seen in Table 1, only for the passing route on the first track line between the corresponding automatic block lines from Orăştie to Deva, 37 alternative routes were found.

Also, by using both the distance and availability matrices, this method could be integrated into an automatic route setting system, determining the routes both by the minimum route length and by the track equipment condition. In this case, the information regarding the availability of the track elements (obtained from the interlocking) can be used in the ARS system in order to have a real-time picture of the railway station status, enabling a more efficient and safer route determination process.

Observing the tables, it can be seen that lengths of the routes have small differences between them, so precise measurements are a request for the proposed method in determining the optimal route. Considering the time for command and operating the point machines in the requested position for the chosen/given route, it can be useful to sort the list of alternative routes (same start-end of the route) by number of elements contained (traffic signals and point machines). Also, sorting by the number of contained elements can be done after sorting by length in case of identical lengths.

Regarding the operator occupation time, using an automatic route setting system reduces the signaller actions considerably, maintaining his attention on important aspects. For a large area with multiple directions, and therefore many traffic signals and point machines, the quantity of the alternatives for one single route command requires analysis time (see Table 1). In this case, choosing the best route for consecutive trains, arriving in the railway station at small time intervals requires experience on the topology station. The proposed method offers a solution of determining efficiently routes by algorithmic principles.

Acknowledgment

The work has been funded by the Sectorial Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family and Social Protection through the Financial Agreement POSDRU/88/1.5/S/61178.

R E F E R E N C E S

[1] *D. Lutovac, T. Lutovac*, Towards an universal computer interlocking system, Facta Universitatis (NI_S), Series: Electronics and Energetics vol. 11, No.1 (1998), 25-49

[2] *I. A. Hansen,* Improving railway punctuality by automatic piloting, Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE, 792 – 797

[3] *N. Balfe, J. R. Wilson, S. Sharples, T. Clarke*, Effects of Levels of Signalling Automation and workload and Performance, Rail Human Factors around the World: Impacts on and of People for Successful Rail Operations, 2012, 404-411

[4] *J. Pachl*, Railway Operation and Control, ISBN 0-9719915-1-0, Library of Congress Control No.: 2002108972, 2002

[5] *S. W. Golomb, L. D. Baumert*, Backtrack programming, J.ACM, 12(4) 516-524 (1965)

[6] *S. Cataranciuc*, Teoria Grafurilor în Probleme □i Aplica□ii, Universitatea de Stat din Moldova, Facultatea de Matematică □i Informatică, 2004 (Graph Theory in Problems and Applications, Moldova State University, Faculty of Mathematics and Informatics, 2004)

[7] *F. Rossi, P. van Beek, T. Walsh*, Handbook of Constraint Programming, cap. 4, Backtracking Search Algorithms, 2006

[8] *M. Craus*, Proiectarea algoritmilor, Universitatea Tehnică "Ghe. Asachi" Iaşi (Algorithm design, "Ghe. Asachi" Technical University Iaşi)