DOI: 10.26565/2304-6201-2019-42-02

UDC 004.942+656.052.1

Algorithm of Intelligent Urban Traffic

D.G. Boguto, K.K. Kadomskiy, P.K. Nikolyuk, A.I. Pidgurska
Faculty of Physics and Engineering, Vasyl' Stus Donetsk National University, Vinnytsia, 21021, Ukraine
e-mail:p.nikolyuk@donnu.edu.ua

The aim of this work is solution of a specific issue – traffic problem in a big city. To solve this problem we examined the city roads intersection - the main source of the traffic congestion. Intersection is a basic element in the technology of urban traffic regulation. Therefore, first of all, it is necessary to implement the intellectual regulation of vehicles movement through a separate intersection. Such regulation is carried out with a help of a computer program that takes into account the vehicle road situation at the intersection and the corresponding adjustment of the traffic lights signal phases. At a second stage it is necessary to plan an optimal route for each vehicle using, for example, A*-algorithm and the spectrum of data received from an infrastructure of the urban network. As a result of an application of these two phases of urban traffic regulation, an optimal movement regime of all city mobile transport is achieved.

Keywords: *intersection, piezoelectric sensor, A*-algorithm; traffic light, Java.*

Перехрестя у місті-мегаполісі є ключовим елементом при регулюванні потоків транспортних засобів. В нашому дослідженні представлена програма, що дозволяє оптимізувати проїзд транспортних засобів (ТЗ) через кожне окреме перехрестя. Проте це перший етап роботи. На другому, і найголовнішому, етапі розглядається проблема прокладання маршруту кожному ТЗ від його початкової позиції до кінцевого пункту. Важливо те, що приведена програма прокладає маршрут, оптимальний по часу, що корінним чином відрізняє її від сучасних технологій GPS-навігації, які прокладають геометрично оптимальні маршрути. Технічно реалізація процесу прокладання маршруту здійснюється з допомогою використання п'єзоелектричних датчиків, що монтуються на кожному перехресті. Ці датчики обраховують число колісних автомобільних пар, які перетнули перехрестя. Згадані пристрої поділяються на два типи – вхідні та вихідні. Перші реєструють число ТЗ, що в'їхали на дорогу одного напрямку між сусідніми перехрестями. Другі – число ТЗ, що виїжджають із кожної окремої смуги руху. Відповідно число таких датчиків повинно рівнятись числу смуг дороги одного напрямку між сусідніми перехрестями. Відношення між числом автомобілів, що в'їхали на дану ділянку дороги, до числа автомобілів, що виїхали з неї за час горіння зеленої фази світлофора, є вагомим критерієм, який свідчить про динаміку руху ТЗ. Чим ближчим є це відношення до одиниці, тим динаміка руху краща. Тому використовуваний у дослідженні алгоритм пошуку оптимального маршруту вибиратиме ділянки дороги, що формують маршрут, саме із таких компонентів. Зауважимо, що представлений у нашій роботі алгоритм прокладає оптимальні по часу, а не по шляху, маршрути. Це досягається шляхом введення в якості ваг ребер графів, що імітують транспортну мережу міста, динамічних величин з особливими характеристиками. Введення таких ваг дозволяє в режимі поточного часу відслідковувати зміни у трафіку та миттєво передавати їх на виконання водіям ТЗ. Пропонована технологія дозволить синхронізувати потоки ТЗ, суттєво зменшити час проїзду кожного автомобіля по маршруту, більш ефективно використовувати транспортні артерії – міський трафік перейде на якісно новий рівень.

Ключові слова: *перехрестя, мегаполіс, п'єзоелектричний сенсор, A^*-алгоритм, світлофор, Java, оптимальний маршрут.*

Перекресток в городе-мегаполисе является ключевым элементом при регулировании потоков транспортных средств (ТС). В нашем исследовании представлена программа, позволяющая оптимизировать проезд ТС через каждый отдельный перекресток. Однако это первый этап работы. На втором, и главном, этапе рассматривается проблема прокладки маршрута каждому ТС от его начальной позиции до конечного пункта. Важно то, что приведенная программа прокладывает маршрут, оптимальный по времени, что коренным образом отличает ее от современных технологий GPS-навигации, которые прокладывают геометрически оптимальные маршруты. Технически реализация процесса прокладки маршрута осуществляется за счет использования пьезоэлектрических датчиков, монтируемых на каждом перекрестке. Эти датчики подсчитывают число колесных пар, пересекших перекресток. Упомянутые устройства делятся на два типа - входные и выходные. Первые регистрируют число ТС, въехавших на дорогу одного направления между соседними перекрестками. Вторые - число ТС, выезжающих с каждой отдельной полосы движения. Соответственно число таких датчиков должно равняться числу полос дороги одного направления между соседними перекрестками. Предлагаемая технология позволит синхронизировать потоки ТС и переведет городской трафик на качественно новый уровень.

Ключевые слова: *перекресток, мегаполис, пьезоэлектрический сенсор, А * -алгоритм, светофор, Java, оптимальный маршрут.*

1 Introduction

Traffic problems are extremely common for a modern metropolis but their resolution is to be achieved yet. First of all, such problems are due to traffic jam at the intersections and, as a consequence, the complexity of trip by each vehicle along chosen route. In order to prevent such phenomenon, the authors propose to apply a network of "smart" traffic lights at the first stage. Special sensors mounted at each intersection can significantly improve each individual crossroad capacity. But this is not enough. It is necessary to organize the optimal trip of each vehicle using mobile phones with a special applications or GPS-navigators coordinating their routes with city traffic management system (CTMS). Thus, it is possible to control and to make optimal routes for all moving vehicles (drivers of such vehicles will be called IR-drivers). In detail the situation regarding the route planning for each vehicle is described in [1, 2]. Therefore, the main problem that will be solved in this article consists in the algorithms' combination for regulating the vehicles movement both through a separate intersection and throughout whole city. With this approach congestion level in metropolis can be much less than it is nowadays, and traffic functioning will move about to a qualitatively new level.

The basic problem of traffic regulation in urban network is optimization problem for vehicle traffic through a separate intersection which is a main source of traffic jams. By organizing effective traffic through crossroads, we will achieve higher traffic efficiency throughout a city. For organizing traffic, each vehicle needs to be registered. And there are many methods to register moving objects [1].

The most effective way to solve this problem is a method based on piezoelectric sensors mounted into a road surface. International Road Dynamics Inc. has developed a very effective design of the piezoelectric road traffic sensor RoadTrax BL [3]. The device is easily mounted into a roadway and is quite sensitive. A vehicle wheel pair riding on the sensor zone, 250 mV output signal is created, which is quite enough for registration. Generally, road sensors are divided into types, shown in Fig. 1.

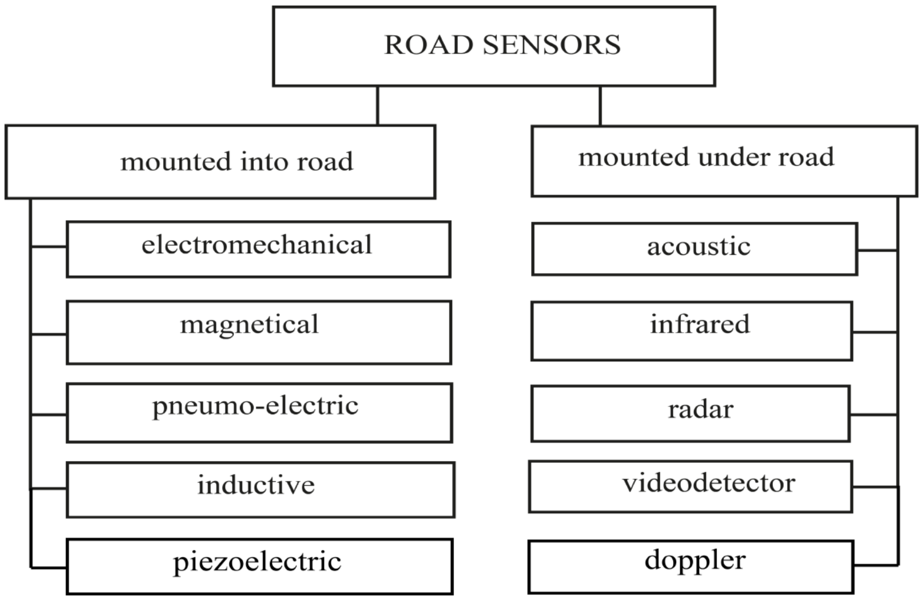

Fig.1 Classification of stationary sensors [4].The most suitable for the solution of the posed problem are sensors mounted into a roadway, namely, piezoelectric and inductive sensors. However, mounting inductive sensors into a road is more problematic. That is why the authors have chosen simple but very effective piezoelectric sensors. The study [5] is focused on the investigation of the piezoelectric sensor characteristics 12-DOF type. This is a new high-sensitivity sensor capable of recording across a wide range of values. The device operates at a frequency of 11 kHz with an error margin no more than 1%. Vehicles registration algorithm which determines their positions is considered in [6].

Theoretical aspects of traffic optimizing on a separate overloaded crossroad are investigated in [7]. So-called discrete-time model for a cross-shaped intersection was considered and optimized system of equations had been created. The similar work is presented in the paper [8].

A large-scale study of the combined information methods obtained from a variety of road sensors, such as detectors, video cameras and radars, is explored in [9]. The technology of connected vehicles is also being investigated here, which enables to collect and analyze communications such as Vehicle-to-Infrastructure (V2I) and Vehicle-To-Vehicle (V2V) and thus reduces the probability of traffic congestion, increases traffic safety and reduces fuel consumption.

From the analytical point of view, it is necessary to create an algorithm and a computer program that will ensure the effective switching of traffic lights in accordance with the traffic load. From the technical side vehicle registration is carried out using piezoelectric sensors that fix a number of wheel pairs, proportional to a number of vehicles, which either to drive into part of the road between adjacent intersections (so-called input sensors) or leave out this road segment – output sensors. In this case the input and output sensors at the neighboring intersections work consistently, transmitting data to CTMS. This allows controlling not only city crossings as autonomous objects, but all city routes as well.

2. Algorithms

The vehicle traffic regulation in a large city can be divided into two phases. The first phase includes a vehicle registration at separate intersections. The registration is organized as follows (Fig. 2). Here is shown a separate cross-shaped intersection equipped with piezoelectric sensors of RoadTrax BL mode [3], mounted into a road. Each sensor is a part of the signaling network connected to CTMS. When a vehicle wheel pair presses on such sensor, an electrical signal is being generated due to the piezoelectric effect and registered on CTMS.

The base of traffic regulation technology at the first phase is that an outgoing cell serves a separate regulated intersection. It is assumed that traffic lights cycle determined by a value that usually consists of the following components

$$\mathrm{T} = \mathrm{trh} + \mathrm{tgh} + \mathrm{tyh} + \mathrm{tp}\ , \qquad (1)$$

where trh – duration of the red light phase time n the horizontal direction;
tgh – duration of the green light phase in the horizontal direction;
tyh – duration of the yellow light phase in the horizontal direction;
tp – duration of the green light phase for pedestrians.

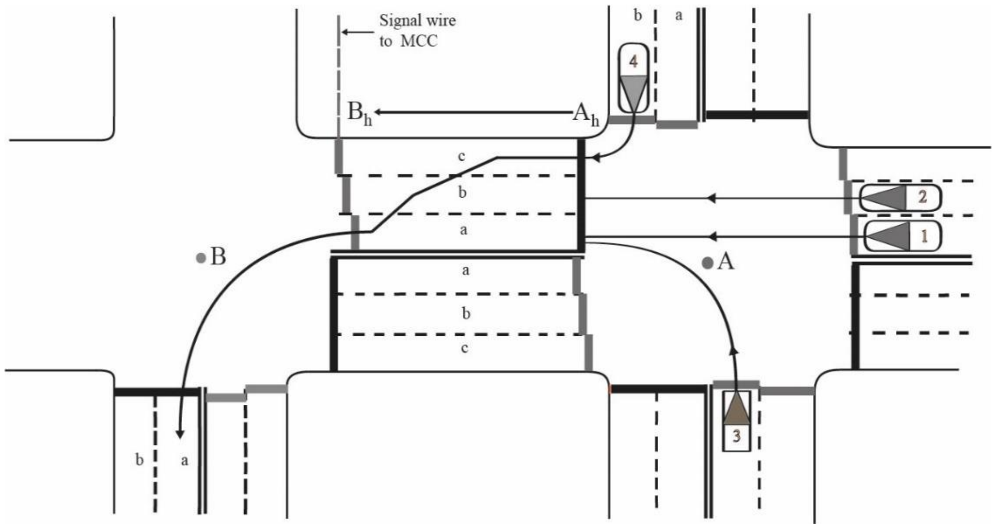

Fig.2. Crossroad A, connected with adjacent crossroad B [1]. The black rectangles depict the vehicles which move in the direction $A_h \to B_h$. Input sensors (black bars marked with letter A_h) and output sensors (three adjacent strips marked with letter B_h) are presented. Each input and output sensor is connected to the CTMS (only one signal wire is shown).

The corresponding notation is also available for the traffic lights cycle in the vertical direction: trv, $\mathrm{tgv}, \mathrm{tyv}$.

Using given variables, the traffic regulation program can be written as follows:

```
package IC;
import java.util.Random;
import static java.lang.StrictMath.abs;

interface Lights{
	int REDH =0;
	int YELLOWH =1;
	int GREENH =2;
	int REDV =3;
	int YELLOWV =4;
	int GREENV =5;
```

```
    int GREENP = 6;
    int ERROR = -1;
}
class T implements Lights {
    private int delay;
    private static int light = REDH;

    T(int sec) {
        delay = 1000 * sec;
    }

    public int shift() {
        int count = (light++) % 7;

        try {
            switch (count) {
                case REDH:
                    Thread.sleep(delay);
                    break;
                case YELLOWH:
                    Thread.sleep(delay / 3);
                    break;
                case GREENH:
                    Thread.sleep(delay / 2);
                    break;
                case REDV:
                    Thread.sleep(delay);
                    break;
                case YELLOWV:
                    Thread.sleep(delay / 3);
                    break;
                case GREENV:
                    Thread.sleep(delay / 2);
                    break;
                case GREENP:
                    Thread.sleep(delay );
                    break;
            }
        } catch (Exception e) {
            return ERROR;
        }
        return count;
    }
}
class TrafficRegulator {
    static int T = 96;
    private static IC.T t = new IC.T(1);
    static Random gn1 = new Random();
    static Random gn2 = new Random();
    static Random gn3 = new Random();
    static Random gn4 = new Random();

        public static void main(String[] args) {
        double k =abs ((gn1.nextDouble()  + gn2.nextDouble() ) /
                (gn3.nextDouble() + gn4.nextDouble()));
         double tg = 35;
    double tp = 23;
    double tyh = 2;

        double tgh = k * tg;
        double trh = (T - tyh - tgh- tp);
        double tgv = abs(2*tg -tgh);
        double trv = (T - trh);
        double tgp = 23;
        int tyv = 2;
        for (int j = 0; j < 7; j++)
            switch (t.shift()) {
                case Lights.REDH:
                    System.out.println("red horizontal!");
                    System.out.format("%.1f%n",trh );
                    break;
                case Lights.YELLOWH:
                    System.out.println("yellow horizontal!");
                    System.out.println(tyh);
                    break;
                case Lights.GREENH:
                    System.out.println("green horizontal!");
                    System.out.format("%.1f%n",tgh);
                    break;
                case Lights.REDV:
                    System.out.println("red vertical!");
                    System.out.format("%.1f%n",trv);
                    break;
                case Lights.YELLOWV:
                    System.out.println("yellow vertical!");
                    System.out.println(tyv);
                    break;
                case Lights.GREENV:
                    System.out.println("green vertical!");
                    System.out.format("%.1f%n",tgv);
                    break;
                case Lights.GREENP:
                    System.out.println("green pedestrian!");
                    System.out.println(tgp);
                    break;
                case Lights.ERROR:
                    System.out.println("Time error!");
                    break;
                default:
                   System.err.println("Unknown light.");
                    return;
            }

    }
}
```

The program results are following:
red horizontal – 29.6; yellow horizontal! – 2.0; green horizontal! - 41.4; red vertical! – 66.4; yellow vertical! – 2; green vertical! – 28.6; green pedestrian! – 23.0. Process finished with exit code 0.
Let us analyze briefly this program regulating vehicle journeys through cross-shaped crossroads. Constant values of a type REDH are chosen – red light in horizontal direction. Basically, the program analyzes a traffic load of the crossroads: the more direction, horizontal or vertical is loaded the longer is

the green phase in the corresponding direction, within a traffic cycle, which is equal to 96 seconds in the chosen case. In order to simulate a real crossroad situation, the four generators of random variables are included in the program to simulate the loading. Actually, we are interested in ratio of the number of vehicles on the horizontal and vertical directions. In the program this ratio is denoted by coefficient k. The variation of this value simulates the traffic load changes at the intersection. Therefore, a range of numbers at the program output changes randomly each time. In particular, the value that specifies the green phase time (as well as the red one) in the horizontal and vertical directions changes from one traffic lights cycle to another.

Consequently, the presented program significantly improves the intersections capacity due to the "intellectual" mode of correlating the various traffic lights phases with the traffic loads on different directions at the crossroads. The program will produce different range of values at each run, but they will be correlated with the traffic load at the intersection: the more the direction is loaded the longer green phase in this direction will be and, accordingly, for the less loaded direction green light phase will be decreased proportionally. Generators of random variables Random gn = new Random () are used as simulation values determining the loading of each direction. The generators are set randomly, but within a real number of vehicles that load the intersection in each traffic lights cycle. Thus, the traffic lights become "intelligent".

We now proceed to the second phase of a traffic regulation. Let us note that "intelligent" regulation is carried out not only at a separate intersection, but also between adjacent intersections. Fig. 2 represents the scheme of such a regulation, where the sensors belonging to a separate intersection are shown, as well as the sensors that regulate traffic on $A \to B$ lane, between adjacent intersections. This section of the road is controlled by the input and output sensors working as a unit for the purpose of a route planning (see below). In other words, each intersection does not work autonomously, but in close cooperation with adjacent intersections. The interaction is carried out due to the fact that input and output sensors (in Fig. 2 sensors A and B) work as a unit. Therefore it is possible to follow up the vehicle dynamics along the $A \to B$ road lane. On qualitative level, the movement dynamics is the ratio between the vehicles which have arrived to $A \to B$ lane during certain time (for example, during the traffic lights cycle) and vehicles that have left the lane. Technically, the process is organized as follows: the sensor A (input sensor) registers vehicles that enter the $A \to B$ lane from all possible directions of the intersection A. In turn, the complex of output sensors at the intersection B registers vehicles that have left the road lane. Thus, the entire transport network of the city is controlled by an intelligent traffic management system. The most important traffic problem is calculating for each IR-driver the requested route allowing to minimum possible travel time. For this purpose, it is necessary to calculate such a route for a given moment of time and for a given travel congestion.

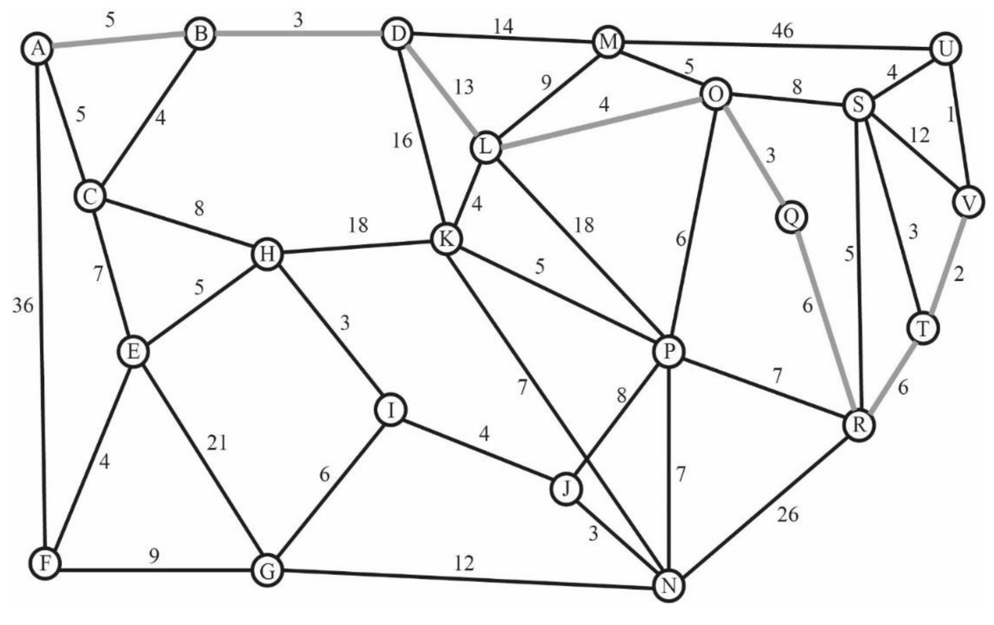

Fig. 3 The optimum route (gray thick line) between two vertices of weighted graph, planning by means of A-algorithm.*

The city transport network can be represented as a weighted graph (Fig. 3). This is the first key message. Using a theory of graphs, it is possible to plan optimal routes by means of appropriate algorithms, namely, Dijkstra', Floyd-Warshall', or A* [10]. For our purposes, the most suitable option will be the A*-algorithm. This algorithm allows planning optimal route between two given vertices of the weighted graph (for example, A and V on fig.3).The second message is that weight of the graph edges rapidly change, so they need to be registered on-line, constantly updating the database where these values are stored. Input and output sensors, working as a pair, produce a spectrum of quantities $N_{A_hB_h}$ and $n_{A_hB_h}$. The first value is the number of vehicles that drive into the lane between adjacent intersections (on Fig. 2 it is $A \to B$ lane) during traffic light cycle, and the second – the number of vehicles that left the same road on the neighboring intersection B during the same time. The closer is the ratio $N_{A_hB_h}/n_{A_hB_h}$ to one; the better is the movement dynamics for the selected lane. On the contrary, the ratio less than one indicate either the blocked state of the intersection B (Fig. 2) or a nearly blocked state. Correspondingly, weight of an edge (in other words, the vehicle impedance of the lane $\to$ B) assumes a bigger value, i.e. $\frac{N_{A_hB_h}}{n_{A_hB_h}} \gg 1$. In this situation the program looking for the optimal route in the graph dismisses such a weighed lane. Therefore, for optimal route planning on the graph (or in urban network), it is necessary to minimize the following value:

$$\sum_{h=1}^{f}(N_{A_hB_h}/n_{A_hB_h})l_{A_hB_h} \to \min. \qquad (2)$$

Here h is an index denoting lane numbers along the route (in other words, the edges of the graph), which are interconnected and form an inseparable trajectory (a simple chain path in the terms of graph theory), connecting the origin A and the destination V positions of IR-driver path (Fig. 3).The value $l_{A_hB_h}$represents geometrical distance between adjacent intersections. The symbol f indicates a number of lanes of type A_hB_h. For each declared pair (A_i, B_i) the program forms an inseparable chain linking points A_i and V_i from (2). Thus, all city roads are under control of CMTS. The values on the left side of the expression (2) are weights of the graph edges (Fig. 3). These values should be updated every 10 seconds and, accordingly, the intelligent traffic control system will plan routes for all IR-drivers taking into account traffic situation for a given time period. In other words, the system works in "on-line" mode, which radically distinguishes it from the existing similar systems, Google-maps, for example. Transmission of data received by CMTS from city network sensors to each user (IR-driver) is used. Urban traffic situation is very dynamic and changeable. That is why CMTS database is updated every 10 seconds, and, accordingly, the data received by IR-drivers have an online mode too.

For planning a route for each IR-driver, A*-algorithm is used. The program [10] corresponding to the graph in Fig. 3 is following:

```
package IC-1;
import java.util.PriorityQueue;
import java.util.HashSet;
import java.util.Set;
import java.util.List;
import java.util.Comparator;
import java.util.ArrayList;
import java.util.Collections;

public class AstarSearchAlgo {
   //h scores is the stright-line distance from A to V
   public static void main(String[] args) {
 //initialize the graph map
Node A = new Node("A", 34);
Node B = new Node("B", 31);
Node C = new Node("C", 30);
Node D = new Node("D", 27);
Node E = new Node("E", 29);
Node F = new Node("F", 33);
Node G = new Node("G", 27);
Node H = new Node("H", 24);
Node I = new Node("I", 23);
Node J = new Node("J", 19);
Node K = new Node("K", 19);
Node L = new Node("L", 19);
Node M = new Node("M", 20);
Node N = new Node("N", 16);
Node O = new Node("O", 13);
Node P = new Node("P", 13);
Node Q = new Node("Q", 9);
Node R = new Node("R", 8);
Node S = new Node("S", 6);
Node T = new Node("T", 3);
Node U = new Node("U", 3);
Node V = new Node("V", 0);
//initialize the edges
A.adjacencies = new Edge[]{
        new Edge(B, 5),
        new Edge(C, 5),};
B.adjacencies = new Edge[]{
        new Edge(D, 3),
        new Edge(C, 4),
        new Edge(F, 36)};
C.adjacencies = new Edge[]{
        new Edge(B, 4),
        new Edge(D, 7),
        new Edge(E, 7),
        new Edge(H, 8)};
D.adjacencies = new Edge[]{
        new Edge(M, 14),
        new Edge(L, 13),
        new Edge(K, 16),
        new Edge(H, 11),
        new Edge(C, 7),
        new Edge(B, 3),};
E.adjacencies = new Edge[]{
        new Edge(C, 7),
        new Edge(H, 5),
        new Edge(F, 4),
        new Edge(G, 21),};
F.adjacencies = new Edge[]{
        new Edge(E, 4),
        new Edge(G, 9),
        new Edge(A, 36),};
G.adjacencies = new Edge[]{
        new Edge(F, 9),
        new Edge(N, 12),
        new Edge(I, 6),
        new Edge(E, 21),};
H.adjacencies = new Edge[]{
        new Edge(D, 11),
        new Edge(E, 5),
        new Edge(I, 3),
        new Edge(C, 8),
        new Edge(K, 18)};
```

```
I.adjacencies = new Edge[]{
        new Edge(J, 4),
        new Edge(H, 3),
        new Edge(I, 6),};
J.adjacencies = new Edge[]{
        new Edge(P, 8),
        new Edge(N, 3),
        new Edge(I, 4)};
K.adjacencies = new Edge[]{
        new Edge(L, 5),
        new Edge(P, 5),
        new Edge(N, 7),
        new Edge(D, 16),
        new Edge(H, 18)};
L.adjacencies = new Edge[]{
        new Edge(M, 9),
        new Edge(O, 4),
        new Edge(K, 5),
        new Edge(D, 13),
        new Edge(P, 18)};
M.adjacencies = new Edge[]{
        new Edge(O, 5),
        new Edge(L, 9),
        new Edge(D, 14),
        new Edge(U, 46),};
N.adjacencies = new Edge[]{
        new Edge(K, 7),
        new Edge(P, 7),
        new Edge(G, 12),
        new Edge(J, 3),
        new Edge(R, 26)};
O.adjacencies = new Edge[]{
        new Edge(M, 5),
        new Edge(L, 4),
        new Edge(Q, 3),};
P.adjacencies = new Edge[]{
        new Edge(K, 5),
        new Edge(J, 8),
        new Edge(N, 7),
        new Edge(Q, 4),
        new Edge(R, 7),
        new Edge(L, 18),};
Q.adjacencies = new Edge[]{
        new Edge(P, 4),
        new Edge(O, 3),
        new Edge(R, 6),};
R.adjacencies = new Edge[]{
        new Edge(P, 7),
        new Edge(Q, 6),
        new Edge(S, 5),
        new Edge(T, 6),
        new Edge(N, 26),};
S.adjacencies = new Edge[]{
        new Edge(O, 8),
        new Edge(R, 5),
        new Edge(U, 4),
        new Edge(T, 3),
        new Edge(V, 12),};
T.adjacencies = new Edge[]{
        new Edge(V, 2),
        new Edge(R, 6),
        new Edge(S, 3),};
U.adjacencies = new Edge[]{
        new Edge(S, 4),
        new Edge(V, 1),
        new Edge(M, 46),};
V.adjacencies = new Edge[]{
        new Edge(U, 1),
        new Edge(T, 2),
        new Edge(S, 12),};
AstarSearch(A, V);
List<Node> path = printPath(V);
        System.out.println("Path: " + path);
    }

    public static List<Node> printPath(Node target) {
        List<Node> path = new ArrayList<Node>();
        for (Node node = target; node != null; node = node.parent) {
            path.add(node);
        }
        Collections.reverse(path);
        return path;
    }

    public static void AstarSearch(Node source, Node goal) {
        Set<Node> explored = new HashSet<Node>();
        PriorityQueue<Node> queue = new PriorityQueue<Node>(30, new Comparator<Node>() {
            //override compare method
            public int compare(Node i, Node j) {
                if (i.f_scores > j.f_scores) {
                    return 1;
                } else if (i.f_scores < j.f_scores) {
                    return -1;
                } else {
                    return 0;
                }
            }
        });
        //cost from start

        source.g_scores = 0;
        queue.add(source);
        boolean found = false;
        while ((!queue.isEmpty()) && (!found)) {
            //the node in having the lowest f_score value
            Node current = queue.poll();
            explored.add(current);
            //goal found
            if (current.value.equals(goal.value)) {
                found = true;
            }
            //check every child of current node
            for (Edge e : current.adjacencies) {
                Node child = e.target;
                double cost = e.cost;
                double temp_g_scores = current.g_scores + cost;
                double temp_f_scores = temp_g_scores + child.h_scores;
  /*if child node has been evaluated and
   the newer f_score is higher, skip*/
                if ((explored.contains(child)) &&
                        (temp_f_scores >= child.f_scores)) {
                    continue;
                } else if ((!queue.contains(child)) ||
                        (temp_f_scores < child.f_scores)) {
                    child.parent = current;
                    child.g_scores = temp_g_scores;
                    child.f_scores = temp_f_scores;
                    if (queue.contains(child)) {
                        queue.remove(child);
                    }
                    queue.add(child);
                }
            }
        }
    }
}
class Node {
    public final String value;
    public double g_scores;
    public final double h_scores;
    public double f_scores = 0;
    public Edge[] adjacencies;
    public Node parent;

    public Node(String val, double hVal) {
        value = val;
        h_scores = hVal;
    }

    public String toString() {
        return value;
    }
}
class Edge {
    public final double cost;
    public final Node target;

    public Edge(Node targetNode, double costVal) {
        target = targetNode;
        cost = costVal;
    }
}
```

As a result of planning a route A → V the program presents path: [A, B, D, L, O, Q, R, T, V], which is optimal for a certain time interval (10 s). These data will be transmitted to the IR-driver from CTMS until situation on the IR-driver's route is irremovable. Otherwise, the program calculates a new route. The same algorithm works for all vehicles of the city. As a result, complete synchronization of traffic flows will occur, which will result in complete disappearance of congestions in a transport network and will allow each driver to arrive at the destination within a minimal period of time.

The testing of the program has been performed on graph with number of vertices up to 150 and number of edges 430. The execution time is 4s 580ms. An algorithmic complexity of A*-algorithm is O (n), (n – input data volume) which makes it quite effective for programs with a large amount of input data.

It should be noted that information obtained from GPS is an important factor in improving the reliability of data derived from input and output sensors. The fact is that almost every driver has a GPS navigator or a smartphone with the Google Maps program. Therefore GPS data can be used as an additional source of information. The situation is described in details in [11-15]. Taking into account the specifics of task and statistical nature of received data there is no need to determine precisely a position of each vehicle: only statistical character of the data is important. Why is it importantly to combine the data from piezoelectric sensors with GPS data? The data received from the sensors – incoming and outgoing – located between adjacent intersections play the main role in our program. But the considered technology involves using GPS-navigators by IR-drivers. It is therefore logical to use the information synthesized from both GPS and sensors data. An expediency of combining stationary and mobile data received from GPS-devices or smartphones is indicated in [16], for example. The similar algorithm of using GPS data for planning optimal routes is discussed in details in [17, 18].

3. Conclusions

The aim of the work is to create solution for the urban traffic problems. It is planned to use this technology for practical purposes.

ЛІТЕРАТУРА

1. Богуто Д.Г., Волинець В.І., Ніколюк П.К., Ніколюк П.П. Автоматизована система керування рухом транспортних засобів в межах міста. *Вісник ХНУ серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»*.2017. Вип.35. С.5 – 12.
2. Богуто Д.Г., Комаров В.Ф., Ніколюк П.К., Ніколюк П.П. Інтелектуальний алгоритм управління міським трафіком. *Вісник ХНУ серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління»*. 2017. Вип.38. С.46 – 57.
3. *Piezoelectric RoadTraxB*. URL:http://www.irdinc.com/pcategory/axle-sensors-accessories/ piezoelectric-roadtrax-bl.html. (дата звернення: 15.05.2019).
4. *Vehicle Sensing: Ten Technologies to Measure Traffic*. URL: https://www. azosensors. com/article.aspx?ArticleID=95. 06.06.2018 (дата звернення: 16.05.2019).
5. Jun Liu, Chuan-Wei Liang, Min Li, Ke Jian, Lan Qin, Jing-Cheng Liu. Principle Research on a Novel Piezoelectric 12-DOF Force/Acceleration Sensor. *J. of Sensors*. ID 2836365. 2017. P.116 – 124.
6. Yingfeng Cai, Ze Liu, Xiaoqiang Sun, Long Chen, Hai Wang, and Yong Zhang. Vehicle Detection Based on Deep Dual-Vehicle Deformable Part Models. *J. of Sensors*. ID 5627281. 2017. P.103 – 117.
7. Jiyuan Tan, Xiangyun Shi, Zhiheng Li, Kaidi Yang, Na Xie, Haiyang Yu, Li Wang, Zhengxi Li. Continuous Discrete-Time Optimal Controls for an Isolated Signalized Intersection. *Journal of Sensors*. ID 6290248. 2017. P. 11 – 19.
8. Ilya Ioslovich, Jack Haddad, Per-Olof Gutman, David Mahalel. Optimal traffic control synthesis for an isolated intersection. *Control Engineering Practice*. 2011. V.19, №8. P. 900-911.
9. Pang-wei Wang, Hong-bin Yu, Lin Xiao, Li Wang. Online Traffic Condition Evaluation Method for Connected Vehicles Based on Multisource Data Fusion. *J. of Sensors*. ID 7248189. 2017. P. 123 – 131.
10. *Java Programming Examples on Graph Problems & Algorithms*. URL: https://www. sanfoundry.com/ java-programming-examples-graph-problems -algorithms (дата звертання: 06.04.2019).
11. Mohammed Quddu, Simon Washing-ton. Shortest path and vehicle trajectory aided map-matching for low frequency GPS data. *Transportation Research. Part C.* 2015. V.55. P. 328-339.
12. Sina Dabiri, Kevin Heaslip. Inferring transportation modes from GPS trajectories using a convolutional neural network. *Transportation Research, Part C.* 2018. V.86. P. 360–37.

13. Feilong Wang, Cynthia Chen. On data processing required to derive mobility patterns from passively-generated mobile phone data. *Transportation Research, Part C.* 2018. V.87. P. 58–74.
14. X. Ma, H. Yu, Y. Wang, Y. Wang, J. Gomez-Gardenes. Large-scale transportation network congestion evolution prediction using deep learning theory. *PLoS ONE.* 2015. V.10. №3. P. 171–185.
15. Mahdi Hashemi, Hassan A. Karimi. A weight-based map-matching algorithm for vehicle navigation in complex urban networks. *J. of Intel. Transp. Systems*. 2016. V.20, №6. P. 45-76.
16. Mahmood Rahmani, Eric Jenelius, Harilaos N. Koutsopoulos. Floating car and camera data fusion for non-parametric route travel time estimation. *Procedia Comp.* Science. 2014. V.37. P. 390-395.
17. Masoud Fadaei Oshyaniv, Marcus Sundberg, Anders Karlström. Consistently estimating link speed using sparse GPS data with measured errors. *Procedia – Social and Behavioral Sciences.* 2014. V.111. P. 829-838.
18. Ashish Kumar Patnaika, Prasanta Kumar Bhuyan, K.V. Krishna Rao. Divisive Analysis (DIANA) of hierarchical clustering and GPS data for level of service criteria of urban streets. *AEJ.* 2016. V. 55. P. 407-418.

REFERENCES

1. D.G.Bohuto, V.I.Volynets, P.K.Nikoluk, P.P.Nikolyuk, "Automated management system of vehicles movement within the city". *Bulletin of the Kharkov University, series "Math. Mod., Inform. Technol., Automated Cont. Syst.",* vol.35, pp. 5 –12, 2017. [in Ukrainian]
2. D.G.Bohuto, V.F.Komarov, P.K. Nikoluk, P.P. Nikolyuk, "Intelligent urban traffic management algorithm". *Bulletin of the Kharkov University, series "Math. Mod., Inform. Technol., Automated Cont. Syst."*, vol.39, pp. 51-62, 2018. [in Ukrainian]
3. Piezoelectric RoadTraxBL, http://www.irdinc.com/pcategory/axle-sensors-accessories/ piezoelectric-roadtrax-bl.html. [May 06.2018].
4. Vehicle Sensing: Ten Technologies to Measure Traffic, https://www. azosensors. com/article.aspx?ArticleID=95. 06.06.2018.
5. Jun Liu, Chuan-Wei Liang, Min Li, Ke Jian, Lan Qin, and Jing-Cheng Liu, "Principle Research on a Novel Piezoelectric 12-DOF Force/Acceleration Sensor". *J. of Sensors*, ID 2836365, pp. 116 – 124, 2017.
6. Yingfeng Cai, Ze Liu, Xiaoqiang Sun, Long Chen, Hai Wang, and Yong Zhang, "Vehicle Detection Based on Deep Dual-Vehicle Deformable Part Models". *J. of Sensors*, ID 5627281, pp. 103 – 117, 2017.
7. Jiyuan Tan, Xiangyun Shi, Zhiheng Li, Kaidi Yang, Na Xie, Haiyang Yu, Li Wang, Zhengxi Li, "Continuous and Discrete-Time Optimal Controls for an Isolated Signalized Intersection". *Journal of Sensors,* ID 6290248, pp.11 – 19, 2017.
8. Ilya Ioslovich, Jack Haddad, Per-Olof Gutman, David Mahalel, "Optimal traffic control synthesis for an isolated intersection", *Control Engineering Practice*, vol.19, no.8, pp. 900 –911, 2011.
9. Pang-wei Wang, Hong-bin Yu, Lin Xiao, Li Wang, "Online Traffic Condition Evaluation Method for Connected Vehicles Based on Multisource Data Fusion". *J. of Sensors*, ID 7248189, pp. 123–131, 2017.
10. Java Programming Examples on Graph Problems & Algorithms, https://www. sanfoundry.com/ java-programming-examples-graph-problems -algorithms.[May 06.2018].
11. Mohammed Quddu, Simon Washing-ton, "Shortest path and vehicle trajectory aided map-matching for low frequency GPS data". *Transportation Research Part C*, vol.55, pp. 328-339, 2015.
12. Sina Dabiri, Kevin Heaslip, "Inferring transportation modes from GPS trajectories using a convolutional neural network". *Transportation Research, Part C*, vol.86, pp. 360–37, 2018.
13. Feilong Wang, Cynthia Chen, "On data processing required to derive mobility patterns from passively-generated mobile phone data". *Transportation Research, Part C,* vol.87, pp. 58–74, 2018.
14. X. Ma, H. Yu, Y. Wang, Y. Wang, and J. Gomez-Gardenes, "Large-scale transportation network congestion evolution prediction using deep learning theory". *PLoS ONE,* vol.10, no.3, pp. 171 – 185, 2015.
15. Mahdi Hashemi, Hassan A. Karimi, "A weight-based map-matching algorithm for vehicle navigation in complex urban networks". *J. of Intel. Transp. Systems*, vol. 20, no.6, pp. 45-76, 2016.

16. Mahmood Rahmani, Eric Jenelius, Harilaos N. Koutsopoulos, "Floating car and camera data fusion for non-parametric route travel time estimation". *Procedia Comp. Science*, vol.37, pp. 390-395, 2014.
17. Masoud Fadaei Oshyaniv, Marcus Sundberg, Anders Karlström, "Consistently estimating link speed using sparse GPS data with measured errors". *Procedia – Social and Behavioral Sciences*, vol. 111, pp. 829-838, 2014.
18. Ashish Kumar Patnaika, Prasanta Kumar Bhuyan, K.V. Krishna Rao, "Divisive Analysis (DIANA) of hierarchical clustering and GPS data for level of service criteria of urban streets". *AEJ*, vol.55, pp. 407-418, 2016.

Boguto Denys Gennadiyovich *– post graduate student of physical and engineering department of Vasil' Stus Donetsk National University, Vinnytsia-21, 600-richchia str, 21, 21021; e-mail: d.boguto@donnu.edu.ua; ORCID: 0000-0001-6288-3070.*

Богуто Денис Геннадійович *– аспірант фізико-технічного факультету Донецького національного університету імені Василя Стуса, Вінниця-21, вул. 600-річчя, 21, 21021; e-mail: d.boguto@donnu.edu.ua; ORCID: 0000-0001-6288-3070.*

Boguto Denys Gennadiyovich *– аспирант физико-технического факультетатДонецкого национального университета имени Василия Стуса, Винница-21, ул. 600-летия, 21, 21021; e-mail: d.boguto@donnu.edu.ua; ORCID: 0000-0001-6288-3070.*

Kadomskiy Kirilo Kostantinovich *– senior lecturer of physical and engineering department of Vasil' Stus Donetsk National University, Vinnytsia-21, 600-richchia str, 21, 21021; e-mail: k.kadomsky@donnu.edu.ua; ORCID: 0000-0002-6163-3704.*

Кадомський Кирило Костянтинович *– старший викладач фізико-технічного факультету Донецького національного університету, Вінниця-21, вул.600-річчя, 21, 21021; e-mail: k.kadomsky@donnu.edu.ua; ORCID: 0000-0002-6163-3704.*

Кадомский Кирилл Константинович *– старший преподаватель физико-технического факультета Донецкого национального университета имени Василия Стуса, Винница-21, ул. 600-летия, 21, 21021; e-mail: k.kadomsky@donnu.edu.ua; ORCID: 0000-0002-6163-3704.*

Nikolyuk Peter Karpovich *– doctor of physical and mathematical sciences, professor of computer technologies chair of Vasil' Stus Donetsk National University, Vinnytsia-21, 600-richchia str, 21, 21021; e-mail: nikolyuk54@gmail.com; ORCID: 0000-0002-0286-297X.*

Ніколюк Петро Карпович *– доктор фізико-математичних наук фізико-технічного факультету Донецького національного університету імені Василя Стуса, Вінниця-21, вул. 600-річчя, 21, 21021; e-mail: nikolyuk54@gmail.com; ORCID: 0000-0002-0286-297X.*

Николюк Петр Карпович *– доктор физико-математических наук, проффесор кафедры компьютерных технологий йфизико-технического факультета Донецкого национального университета имени Василия Стуса, Винница-21, ул. 600-летия, 21, 21021; e-mail: nikolyuk54@gmail.com; ORCID: 0000-0002-0286-297X.*

Pidgurska Anastasia Igorivna *– student of third course of physical and engineering department of Vasil' Stus Donetsk National University, Vinnytsia-21, 600-richchia str, 21, 21021; e-mail: pidhurska.a@donnu.edu.ua; ORCID: 0000-0001-5349-0689.*

Підгурська Анастасія Ігорівна *– студентка третього курсу фізико-технічного факультету Донецького національного університету імені Василя Стуса, Вінниця-21, вул.600-річчя, 21, 21021; e-mail: pidhurska.a@donnu.edu.ua; ORCID: 0000-0001-5349-0689.*

Пидгурская Анастасия Игоревна *– студентка третьего курса физико-технического факультета Доецкого национального университета имени Василия Стуса, Винница-21, ул.600-летия, 21, 21021; e-mail: pidhurska.a@donnu.edu.ua; ORCID: 0000-0001-5349-0689.*

Надійшла у першій редакції 15.07.2019, в останній –13.09.2019.