Cooperative Optimal Route Planning of Accumulator-Bank Servicing Robots

Ágnes Werner-Stark
and Tibor Dulai
Department of Electrical
Engineering and Information Systems
University of Pannonia
Egyetem str. 10.
Veszprém, Hungary
Email: werner.agnes@virt.uni-pannon.hu
dulai.tibor@virt.uni-pannon.hu

Katalin M. Hangos
Computer and Automation
Research Institute HAS
Budapest, Hungary
and Department of Electrical
Engineering and Information Systems
University of Pannonia
Egyetem str. 10.
Veszprém, Hungary
Email: hangos@scl.sztaki.hu

***Abstract*—Renewable energy storage originating from solar energy is possible in an accumulator-bank, from where the demands and utilization may be provided by robots. A novel optimal route planning algorithm is proposed in this paper that is based on the cooperation of the robots implemented as agents. The interaction between the agent-robots is learned to enhance the stability of the system, and in such a way more efficient operation can be achieved. A special model is developed for describing the operation of the multi-agent system formed by the robots, and the route planning algorithm determines the optimal route considering the cooperation of the robots in special situations. The operation and properties of the proposed algorithm is illustrated using simple examples with robots in different conflict situations.**

Keywords-cooperation; renewable energy; accumulator bank; multi-agent system

I. Introduction

Nowadays, all problems related to the application of renewable energy sources enjoy an increased attention and popularity. Beside of their advantageous properties from environmental and sustainability point of view, these energy sources suffer from limited and unpredictable availability. A solar panel, for example, can produce enough energy during the day, but during the night (or just on cloudy days) it proves unusable. Therefore, a sufficient amount of electrical energy storage capacity should be provided along with each renewable energy source to ensure the availability of sufficient energy on demand. One of the easiest ways is to use accumulators as energy storage, but the price and the storage place they need is too large compared to their capacity. If we would like to store energy in large volumes, we would need to place the accumulators in very large storage parks. The service of this storage (accumulators out and to transport) is a big logistics task. It is then very important to solve the problem of efficient place utilization and the quick service. We developed a system of self-service for an accumulator-bank. For this purpose self-propelled robots are required which are able to transport the accumulators, and can perform independent decision making as well as reacting to certain environmental events. In such a distributed setting, the cooperation among the robots may significantly enhance the performance of the system.

The above accumulator-bank servicing problem is much similar to some well investigated problems in traffic management and control, and logistics. An important approach to solve these problems is to use *multi-agent* techniques. A multi-agent approach to design in the transportation domain is presented in [4]. It presents three important instances for distributed artificial intelligence techniques that proved to be useful in the transportation applications: cooperation among the agents, task decomposition and allocation, and decentralized planning. They can be used to obtain good initial solutions for complex resource allocation problems. As another example, one can consider real-time approaches to manage roadway network congestion over time and space, that is a difficult problem. A solution approach based on cooperative negotiation between agents based on multi-agent principles is proposed in [1].

In one of our earlier works [8], we dealt also with autonomous agents, as we considered such circumstances that make autonomy important, such as extreme high or low temperatures and closeness of dangerous materials. These circumstances had the need of applying robots, they had to solve their problems self-sufficiently, without any direct human intervention.

In the field of logistics, operations research approaches deal with Vehicle Routing Problem (VRP) ([3], [7]) and its solutions that help the companies in their logistic tasks as well. Because of the huge application area of VRP, lots of variants of the problem have born. Some of them include additional constraints (e.g., [9]), while other variants modify the basic tasks (e.g., [5]). The cooperation of vehicles has proven to be useful in this problem class, too [2], where we proposed a method of choosing the directions of the routes of the VRP solution which has the best answer (the minimal extra route) in case of an immediate event supposing cooperative agents. As an immediate event may happen at different phases of the completion of the transportation task, the event's effect has to be taken into account on average.

Usually, in a multi-agent system the agents have specific pre-defined abilities to perform a certain task. One of the challenges of a multi-agent system is to develop agents with the ability of learning from each others' behavior. The aim of this paper is to present an algorithm that allows autonomous agents to use cooperation in conflict situations through communication with other robots. The agents are not in interaction with humans during operation.

The paper is organized as follows. First, we describe the domain of the multi-agent system (Section 2). Then, we introduce our algorithm that can be applied in the context of the multi-robot system example (Section 3). The testing of the algorithm is presented in Section 4. Section 5 concludes the paper.

II. Optimal route planning in the accumulator-bank

This section describes the simple model that is used to design the route of the robots in the accumulator-bank.

A. Plan of routes

Let us divide the storage place into cells of equal size such that a transport robot fits in them. It is very important that we use the available place in the storage the best possible way. The resulted matrix is used as a tool for describing the traffic of the robots: they move from cell to cell to get from one place to another. Obviously, if we use the smallest possible units, then more condition examination and much more calculation have to be performed. There is a trade-off between achieving the best possible result and the efficiency of the algorithm.

Basic assumptions for the route planning algorithm are as follows.

- The capacity of each robot is one unit as is the weight and size of every accumulator, too.
- The orientation is based upon a grid of cells which allows the robots to drive only among the neighboring cells (but not diagonally). In every cell there is at most one robot at a time.
- Every robot moves a unit distance in a unit time, i.e. they move only to the closest neighboring cell. The 90 degrees turn takes a unit time, too.
- One accumulator fits into one storing cell of the storage place.

B. Identification of the optimal route

The robots move along the cells of a grid between the neighboring cells with a one cell per time unit velocity, and the 90 degrees rotation takes a unit time, too. A widely-used path search algorithm has been modified for the identification of the optimal route. This is a popular version of Dijkstra's graph-based algorithm, that was developed by Hart et al. [6], where they described how heuristic information from a problem domain can be incorporated into a formal mathematical theory of graph search and demonstrated an optimal property of a class of search strategies. The algorithm stores the path length from the starting point to the points of graph on the graph's points, that is used again when the recursive algorithm re-visits this point. This re-visit is easily detectable, and the stored value is used to prevent continued counting on the given branch (because we found an existing shorter way) or we can stop the run of the branch because at this point we have already found a more efficient path.

The main modification is that we reduced the cost by reducing the distance between cells. Because the turning of robots requires time, too, we have to record from which direction the robot arrived in to the examined cell and to which direction it continued the search. We add the cost of every 90 degrees turns made between cells as a unit virtual distance. Another modification serves the route which makes traceability easier: when we get a smaller value in a point than the former ones and we overwrite until now the smallest approaching cost, then we note it too, from where (from which direction) the robot comes to a given point. So we can determine easier the compliant route after the filling of a table.

Fig. 1 shows an example of the distance table with the distances in the cells. The green cell is a start point, the blue cell is an end point and the red cells mark obstacles (wall/rack).

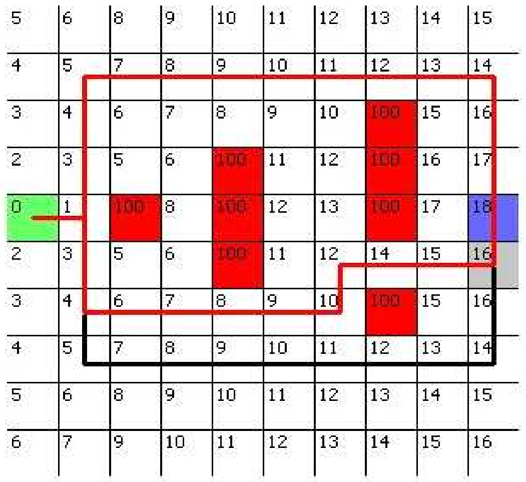

Figure 1. Matrix-based orientation - the problem of listing all the routes

The pseudo code of the proposed basic route search algorithm can be seen in Fig. 2.

The determination of the shortest route is happening backward: it starts at the end point and determines the desired route unanimously. For this purpose one has to record from which cell the robot arrived (source cell) when the length of the shortest route is modified. The easiest method for doing this is to build a new data unit in the cells of matrix (source cell). With this we have a structure similar to a chained list.

It occurs often that there are some routes with equal length between two given points. It is advisable to process each of them, so a crisis situation could be avoided in the future. The routes of equal length present a problem, as the source cell is not enough to store a single value in the cells when the robot can arrive to a cell from several sides after driving the same route length. In order to process the case of equal route lengths properly, it is very important that we consider the following.

- If the robot arrives in a cell and the covered distance is less than the smallest distance until now, we have

```
Date structures:
Map_element: record
                distance: int
                object: wall/empty
                previous: point ((x,y) coordinate pair)
Map: 2 dimensional array, that consists of map_elements

Procedure:
procedure Next(current_point, distance, previous_point)
if current_point is on the map then
    if Map[current_point].object ≠ wall then
        if Map[current_point].distance > distance then
            Map[current_point].distance:=distance
            Map[current_point].previous:=previous_point
            loop for each "new" neighboring cell
                Next(new, distance + 1 + cost of turning, current_point)
            end loop
end procedure
Startup: Next(goal, 0, null)
```

Figure 2. Pseudo code of the proposed basic route search algorithm

to cancel the list of source cells (this information is not relevant).

- If the values of the two distances are equal we have to record it in the source cell to the list, making sure not to overwrite the past values.

C. Constructing the list of all routes

The specification of every optimal route and taking into consideration the turning cost presents a problem when constructing the list of all routes. For clarity, let us consider the example in Fig. 1. The values in the cells mean the distance values and these are determined by the above algorithm. For example, the red lines are the shortest routes (with the same lengths), but the black line has the same length as the other two red lines (not all routes are drawn in the figure). The source of the problem is hidden in the grey colored cell. When the robot approaches the goal from bottom (black line), the distance value of the cell is 17. But when the robot comes from the left-hand side (red line) the distance value is 16, because the cell value is overwritten with the larger 17 value (this information is correct). However, we have to turn 90 degrees to achieve the target following the red line, that would add one time unit at this point and the length of the two routes are in fact equal (17), that we lost by overwriting it.

The easiest way to resolve this problem takes place during the building of the route. We examine all cells and compare the directions either with rotation or a straight route. Since a rotation is not straight, we calculate where the next cell is if we go on straight. If the direction value of this deviates by maximum 1 from the direction value of the current cell then we add this cell to the list of the previous routes. Because in each cell the shortest route to get there and its length are stored, it is enough if we make up the connection only from the overwritten cell, the recursive route construction will bring us to the start cell.

III. Cooperative route searching of the robots

While a single robot navigates in the storage, it can use the previously described algorithm. However, in the case of more than one robot, it is important that we deal with prevention of conflicts, e.g., with the collision of two robots. The possible collision can be detected in advance, not locally. The robots can cross-check the routes in advance so they can search for another route at the start moment instead of waiting for another robot if a possible collision is forecasted. For this reason, the robots make a note to each cell when they pass through it, and notify the other robots about this event. In order to reduce the load of the communication channel, in certain cases the robots may communicate indirectly to each other. In this situation we install a central computer that is able to store the collected information of the storage of the cells and it can pass these information at the request of the robots.

Fig. 3 (a) shows a situation when two robots starting from the points $A1$ and $A3$, respectively, may have a collision.

We have got two possibilities to avoid the collision.

- The robots go to the meeting point and after that one of the robots goes round the other robot. This route will be longer than the pre-planned route because of the turns. This can be seen in Fig. 3 (b).
- If the robots plans their routes in advance then the roundabout route may be shorter, this can be seen in Fig. 3 (c).

When a robot plans a route for itself then it reserves specific cells for itself at pre-planned time instances. Because each navigating robot uses the same time unit, we consider the time unit to be the time step of the system (we suppose that every robot pushes on in synchrony). When the next robot plans its route, it queries the data of the previous robots so it knows exactly what the first robot (or all previous robots) reserved: exactly when and which cells they intend to visit. Now the robot in turn can take this into consideration during route searching, therefore it can decide what is more profitable in case of crossing routes: waiting for the passing of another robot or looking for another route.

A. Waiting for other robots

There may be situations in which it is simply not enough to avoid another robot because for example the robot takes up a bottleneck passage and the other passage is too far. At that time it is more appropriate to wait for the passing of another robot than to choose a bypass route.

In order to handle such situations properly, we should modify the route search algorithm so that the algorithm deals not only with the travelled distance but also with the latency. For this purpose we must note the latency in every cell together with the exact current shortest distance, and when the robot comes into a new cell, we need to compare the sum of the two values with the entered value. If the following cell is reserved at the moment of arrival we must wait until the cell will be empty. During the latency we need to pay attention to the current cell (in which the robot waits) so that no other robot traverses it. If this is to happen, the waiting is not possible.

B. Passages

There can be some narrow passages in the storage for the sake of the better utilization of space, therefore we also need

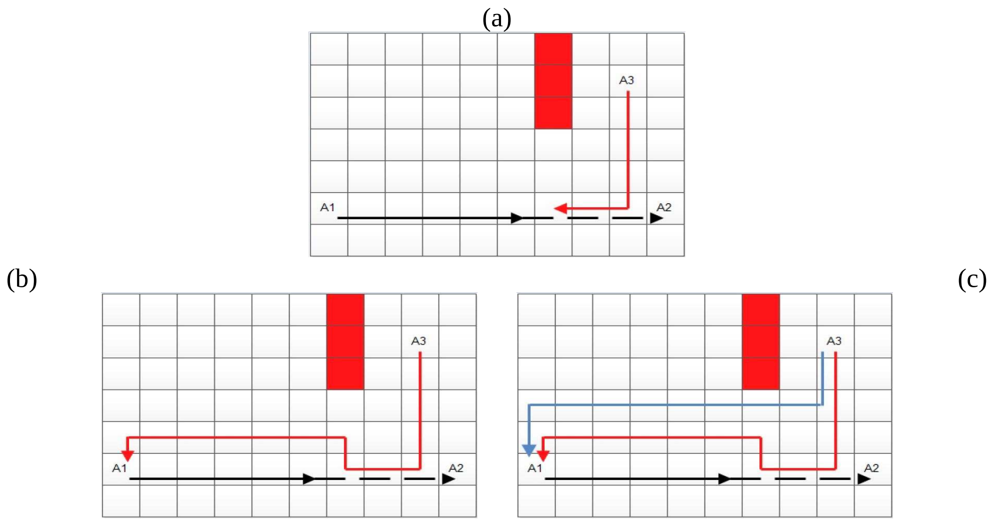

Figure 3. Comparison of the local collision detection and pre-planning: (a) A possible collision, (b) The unplanned route, (c) The pre-planned route

to deal with them. In these passages there can be one robot at a time, this can cause a traffic-jam. If two robots approach the passage at its opposite ends then the route search algorithm can sense only the character of the problem before the collisions.

In order to handle passages in a proper way, a new seizing method had to be developed. When a robot comes into a passage it places a separate seizing at the end of the passage (i.e., at the cell after the last cell of the passage), which is valid not only for the duration when the robot will pass through the cell but it already starts before entering the passage and keeps until when the robot steps out from the passage.

A simple example of a passage situation is seen in Fig. 4. The white squares mark empty cells and the red squares mark obstacles (wall/rack). The robot marked with the blue arrow tries to get out from the passage, his route being reserved. During the route planning of the other robot the recursive algorithm goes in regularly on the red marked route, it senses the collision with the first robot, because this branch stops (in this direction is not any route temporarily). The other green routes are open though, but with different conditions: on the dark green route we have not got to wait for the first robot supposing that we leave the cell before the robot arrives at the end of passage; on the light green route we have to wait in any case for the first robot (or else we find ourselves face to face with the robot and one of them has to turn back). The problem is that the waiting for a given cell (in our case it is $B3$) depends on to which cell we want to go to later.

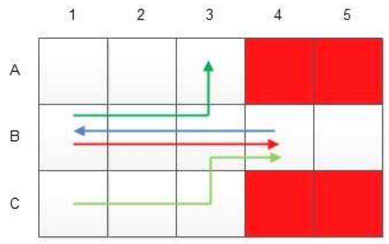

Figure 4. Comparison of the local collision detection and advance planning: The problem of stepping in the passage

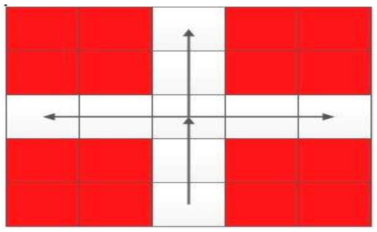

Figure 5. Comparison of the local collision detection and advance planning: Different stopping required in various passages

Unfortunately, however, a more complicated situation can also arise (as it is depicted in Fig. 5. In this case it may happen, that with each of the three different further directions we need to wait for a different duration. This can be resolved if we examine separately every case in the course of route searching. If we perceive a special seizing before entering a cell we have to examine to which passage it is allocated because the rate of waiting will depend on this. For every touched passage we need to create a separate branch and to examine the waiting time of them before we can go over to this special cell. We need to attend to the given branch with the individual waiting time in the direction of only one given cell.

C. Cells multiple visited

There may be cases in which the optimal route passes through a cell twice or even more times. This situation presents a problem to the proposed route planning method, because the cells between two visits can not be clearly defined, and the other robots can not decide on the direction they should proceed. One such example is shown in Fig. 6. The first robot (black arrow) is planning to pass for the first time so the route of this is specific: the robot goes straight from cell E11 to cell E6. If we assume that the robot can pass through a cell only once, the second robot (red arrow) is forced to make a long roundabout way. The shortest route will be the blue route, i.e. the robot shuns the front of the other robot in cell F7 and waits while the other robot passes and then continues on it is way. The original route planning algorithm can not process

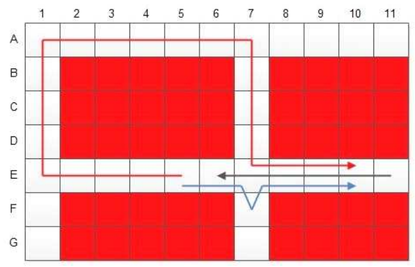

Figure 6. Comparison of the local collision detection and advance planning: The strategy of the stand aside option in contradiction to the roundabout way

this route satisfactorily, because we leave the end points of the passages (here it is cell E7) and we register the following information: the previous value of the cell F7 will be E6 and the previous value of the cell E8 will be F7. Thus, there will be a stoppage in the course of decryption of the route.

A similar problem may arise where a robot decides to pass through the cell E7 twice because there are two different distance values and waiting values. The different waiting value case is more important. When the second robot passes through the cell for the first time, the algorithm records 0 waiting, then a positive value for the second time (assuming that we need to wait some time units for the first robot). These values should also be noted separately in a suitable data structure. Three parameters are needed for this: the previous and next cells (these identify where the robot came from and where it is going to when it passes through the cell) and the waiting value. This permits us to record the difference in the duration of waiting times before the robot steps into the different passages.

IV. Case studies

Simple case studies were used to test the operation and efficiency of the proposed cooperative route planning algorithm. For this purpose an implementation of the algorithm has been developed in Delphi programming language (we used also Indy (Internet Direct) component package to the communication), and this simulation environment can visualize the robots' movements.

Every measurement result was verified with the following configuration:

- CPU: Inter Celeron 560@2.13 GHz
- Operating system: Microsoft Windows XP SP2

A. Route planning tests

Each case study had a few cooperating robots and the topology of the accumulator-bank storage place was also different. The planning order of the robots was the same in each case, and it corresponded to their serial number (in ascending order).

Fig. 7 illustrates the starting situation and the movements of the robots in the following two examples. The grey cells show the actual positions of the robots, where both the robots' serial number and their actual direction are indicated. The cells with a number denote the goal of the robot with the same serial number.

TABLE I. The effect of the map size on the running time

Size of map	Time of route planning (ms)	Time of planning/ robot(ms)
25x25	62,6	3,13
25x50	175	8,75
50x50	334,8	16,74

TABLE II. The effect of the robot number on the running time

Number of robots	Time of route planning (ms)	Time of planning/ robot(ms)
5	14,4	3,08
10	28	2,8
15	46,8	3,12
20	62,6	3,13

- **Example 1: Passages**
  There are four robots in the storage and they know to which cells they need to get to. We can see in Fig. 7 (a) that every robot stands in compliance with his forward direction. Robot3 waits till robot1 and robot2 pass through the passage, thereafter robot3 goes on in the direction of its goal. Robot4 has attained the goal in the meantime because its route has not crossed the others. Fig. 7 (b) illustrates the movement of the robots in this situation.
- **Example 2: Getting out of the way**
  This example illustrates the getting out of the way: robot1 planned first, it has priority, so robot2 gets out of its way in the other passage. Thereafter robot2 continues on its way when robot1 passes before it. We can see the starting situation in Fig. 7 (c), and the movements in Fig. 7 (d).

B. Efficiency test

In order to test the efficiency of the proposed algorithm, we recorded the full running time of the algorithm and noticed how this value changed with the increasing complexity of the planning problem.

Effect of the map size: In the first test we examined how the time of planning changes by increasing the size of the map applying the same number of robots (in the present instance 20 robots). The robots were randomly placed on the map. The results are collected in Table I.

The average length of the randomly placed robots' route doubled in case of doubling the map size, so the route search algorithm had to explore the space with twice as large radius in this case.

Effect of the number of robots: In case of the other test the robots were arranged randomly in a $25x25$ of size map-file. Five program running was performed with each robot number value, and the running times were averaged. Table II shows the simulation results.

It can be seen from the results that the system integrates the new robots well, the robot pre-planning time is about 3 ms independently of the number of robots. This important result

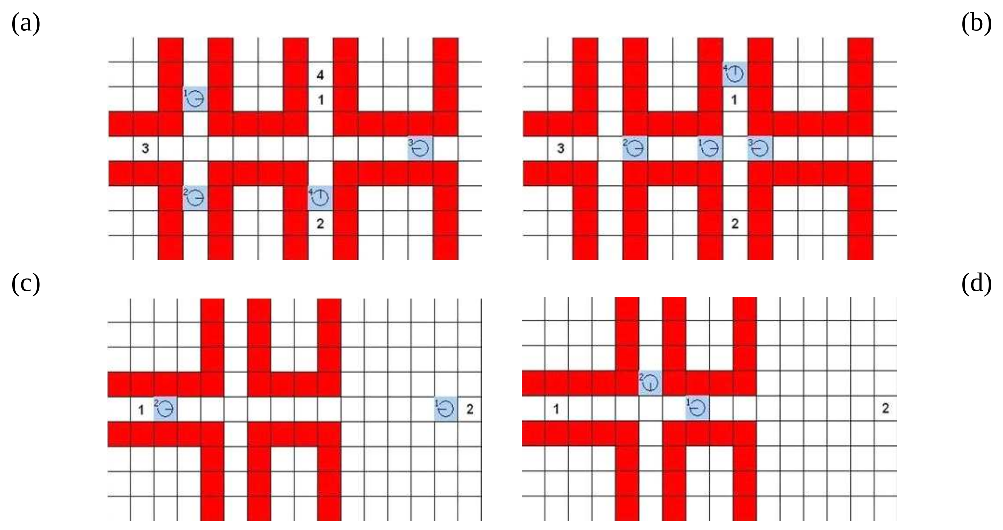

Figure 7. (a) Example1, starting situation, (b) Example1, movements of the robots, (c) Example2, starting situation, (d) Example2, movements of the robots,

shows that the proposed algorithm scales up well with the size and complexity of the problem, thus offering an efficient service of the accumulator-bank.

V. Conclusion

A novel optimal route planning algorithm is proposed in this paper that is based on the cooperation of the robots implemented as agents. The basic version of the algorithm uses a special data structure that is arranged according to the matrix-type grid of the cells defined in the storage place.

The robots use the same route planning algorithm in turn, and take into account the plans of the other robots in order to avoid collision. This way they can detect and avoid collision in advance and not locally. Special conflicting situations, including waiting, passage handling and multiple visiting of cells are also investigated.

The operation and properties of the proposed algorithm are illustrated using simple examples with robots in different special conflict situations.

For optimizing the navigation of the robots, we aim at enriching their communication process with learning capabilities.

Acknowledgment

We acknowledge the financial support of the Hungarian State and the European Union under the TAMOP-4.2.2.A-11/1/KONV-2012-0072. This publication/research has been supported by the European Union and Hungary and co-financed by the European Social Fund through the project TAMOP-4.2.2.C-11/1/KONV-2012-0004 - National Research Center for Development and Market Introduction of Advanced Information and Communication Technologies.

References

[1] J.L. Adler and V.J. Blue, ”A cooperative multi-agent transportation management and route guidance system”, Transportation Research Part C: Emerging Technologies 10(5-6). 2002, pp. 433-454.

[2] T. Dulai and Á. Werner-Stark, ” Immediate event-aware routing based on cooperative agents”, Proceedings of Factory Automation 2012, Veszprém, 21-22 May. 2012, pp. 144-148.

[3] B. Eksioglu, A.V. Vural, and A. Reisman, ”The vehicle routing problem: A taxonomic review”, Computers & Industrial Engineering 57, 2009, pp. 1472-1483.

[4] K. Fischer, J.P. Müller, and M. Pischel, ”Cooperative transportation scheduling: An application domain for DAI”, Applied Artificial Intelligence: An International Journal, 10(1), 1996, pp. 1-34.

[5] M. Gendreau, F. Guertin, J.Y. Potvin, and R. Séguin, ”Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries”, Transportation Research Part C 14, 2006, pp. 157-174.

[6] P.E. Hart, N.J. Nilsson, and B. Raphael, ”A formal basis for the heuristic determination of minimum cost paths”, IEEE Transactions on Systems Science and Cybernatics, 1968, pp. 100-107.

[7] G. Laporte, ”The vehicle routing problem: An overview of exact and approximate algorithms”, European Journal of Operational Research, No. 59, 1992, pp. 345-358.

[8] Z. Szabó, B. Lájer, and Á. Werner-Stark, ”Automata Depository Model with Autonomous Robots”, Acta Cybernetica 19, 2010, pp. 655-660.

[9] D. Zhenggang, C. Linning, and Z. Li, ”Improved Multi-Agent System for the Vehicle Routing Problem with Time Windows”, Tsinghua Science and Technology, 14(3), 2009, pp. 407-412.