Operation of Accumulator-Bank Serving Agent System Using Machine Learning

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

***Abstract*—Advancements in on-demand power management of renewable energy can be achieved by multi-agent systems. This paper proposes an innovative approach where a population of autonomous agents are able to cooperate in managing an accumulator-bank in order to effectively deliver energy in places where it is required. The distributed and adaptive multi-agent approach is able to decrease the interferences by avoiding the negative interactions and conflicts, using the cooperation among agents. Our method uses the learning ability of agents to minimize the number of communications among agents and the central unit. This adaptive behavior lets the agents minimize the time to find the optimal routes during the search. A simulation environment has also been developed for visualizing the movements of the agents and the conflict situations. The operation and the efficiency of the algorithm have been investigated using simple case studies.**

Keywords–renewable energy; agent; genetic algorithm (GA); cooperation

I. Introduction

It is widely recognized (see e.g. [3]), that ”researchers must find a sustainable way of providing the power our modern lifestyles demand.” Along this line, more and more research and development projects are aimed at modernizing energy sources e.g., [9], [4]. Hadjipascalis *et al.* [10] present an overview of the current and future energy storage technologies used for electric power applications.

The problems of applying renewable energy sources are widely and extensively investigated because of the need for sustainability of energy systems. Solar and wind power are the two main sources of renewable energy, both of them suffer from the disadvantage that they are not always readily available on demand. At the same time, the available ways of energy storing are not economic enough and are of limited capacity. One of the possible ways of storing the energy is using accumulators that can be placed in an *accumulator-bank*. An automated service of such an accumulator-bank is desirable that can be implemented by using *autonomous agents* (i.e., robots) that can cooperate with each other to provide optimal on-demand service.

Literature on learning abilities of agents: 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 to learn their behaviour from each others. In recent years, more and more researchers focus on the learning ability of agents that can improve the efficiency of a multi-agent system. Saggar *et al.* [19] developed a learning algorithm for agents to optimize walks in both speed and stability in order to improve a robot's visual object recognition. Nguyen-Thinh presented a learning algorithm for agents based on interactions with humans in conflict situations [14]. Taylor et al. [21] present an algorithm that combines transfer learning, learning from demonstration and reinforcement learning to achieve rapid learning and high performance in complex domains. Using experiments in a simulated robot in soccer domain, they show that human demonstrations transferred into a baseline policy for an agent and refined reinforcement learning significantly improve both learning time and policy performance.

Learning by using GA: Genetic algorithms (GA) are popular tools for implementing heuristic learning policies. In the context of robot movements, GA is applied for route planning using the variants of the well-known Vehicle Routing Problem (VRP) with the help of other heuristic methods. These methods are called hybrid GAs [5],[6],[12],[16],[17],[20], where the improvement can be achieved by imitating biological evolution for solutions of VRP [2],[8],[15],[18]. It is important to emphasize that hybrid GA methods are used for improving the result starting from an initial - usually not optimal - solution. Hagen et al. [11] present the implementation of a GA based path planning on RoboCup's small-size league robots. Because path planning on mobile robots is a continuous process, the path planning runs until the robot arrives at its destination. Hereby, the path is updated according to the environmental changes, such as moving obstacles.

Cooperation: Another way of improving the reactivity of an agent system is to develop the cooperation ability of its agents. A multi-agent approach was presented in [7], that uses cooperation among the agents, task decomposition and task allocation, and decentralized planning. The paper [1] proposes a solution approach of managing roadway network congestion over time based on cooperative multi-agent-based principled negotiation between agents. In our recent study [22] we proposed a cooperative optimal route planning algorithm in the accumulator-bank servicing model by using a specially constructed model that will be extended by a learning method in this paper.

Learning and cooperation in renewable energy systems: Advancements in on-demand power management of renewable energy can also be achieved by multi-agent systems. Many researchers have used this technology recently. In [23], the effectiveness of the coordination model was analysed by investigating the effect of the environmental conditions that affect the traveling time. Their approach is based on a multi-agent system for a road transportation network using supply chain management. Hrncir et al. [13] present the problem of finding parts of routes, which can be shared by several travelers with different points of departure and destinations. This is a complex multi-agent problem, for which a special method can be developed. They proposed a three-phase algorithm that utilizes performance of single-agent planners to find individual plans in a simplified domain first, then merges them using a best-response planner. This planner ensures that the resulting solutions are individually rational, and then maps the resulting plan onto the full temporal planning domain to schedule actual journeys.

The aim: Based on our cooperative optimal route planning algorithm [22], we aim at developing a novel approach that allows autonomous agents to carry out learning in conflict situations through communication with each others but without the interaction with any human during the operation. Our approach uses the learning ability of agents to minimize the number of communications among agents that is necessary to the quick service in an accumulator-bank. The learning is realized by using GA tailored to this special problem.

II. Basic notions and tools

A. The model of the accumulator bank and the basic route planning algorithm

In our recent study [22] we proposed a cooperative optimal route planning algorithm in the accumulator-bank servicing model by using a specially constructed model that guarantees the avoidance of collisions. This approach is called the basic route planning algorithm that will be briefly described here.

In order to have a simple model of the accumulator-bank, we separate the storage place into cells of equal size so that a transport agent can fit in one cell. These cells are arranged in a matrix that will be the most important helping tool for the transport of the agents: moving from cell to cell to get from one place to another, and this logical unit will also be used to avoid collision with other agents.

The basic route planning algorithm minimizes the cost that is the number of covered cells from the start to the destination. We added certain cost of every 90 degrees turns made between cells, too. (Figure 1 illustrates the cost calculation of a path with turning.)

Figure 2 shows a possible, simple cell matrix (this is the map for the accumulator-bank) with the costs in the cells. The green cell is the starting point, the blue cell is the end point and the red cells mark obstacles (wall/rack).

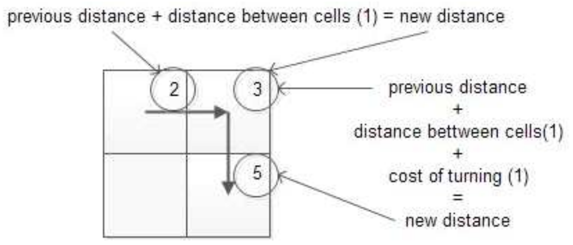
previous distance + distance between cells (1) = new distance
2
3
5
previous distance
+
distance bettween cells(1)
+
cost of turning (1)
=
new distance

Figure 1: Cost calculation

B. Communication in the basic route planning algorithm

In the basic route planning algorithm [22], each agent navigates not only avoiding collisions with each other but do

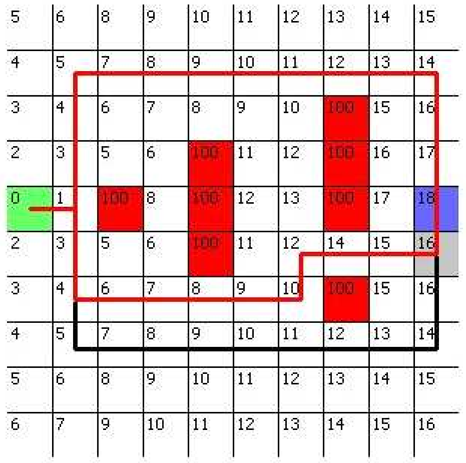

Figure 2: Matrix-based representation of a storage place of a simple example [22]

this in an optimal way. At the same time we assumed that the agents receive and send the information directly from/to a central processing unit. This is not realistic/economical in practical situations, but a wireless technology should be considered instead for communication that allows direct communication among the agents, too. (We selected the TCP protocol because this can be applied on WIFI and Bluetooth technologies, so the adaptation will be easier for future devices.)

The communication system has to deliver to the agents the data necessary for the route planning:

1) the store layout (the map), that shows, which cells it can go through and which are prohibited; and
2) the planned route of the other agents (the cell-reservations), on the basis of which it can determine, which cell is free, and when or how long it needs to wait for which other agent. We can send these information in wireless way to the agents.

C. Synchronization

Because every agent should communicate with the central unit, it is practical to build up a connection when the agent connects the system, this connection is reserved continuously after that. Every agent needs a personal identifier. When an agent starts its operation, it connects to a predefined server, from where the agent asks a serial number and a connection identifier. On the server side a separate thread waits for the agent on a specific IP-address and port. After connection this thread manages the communication between the agent and the central unit.

In this process, synchronization plays an important role. If two agents join the name-server at the same time, one of the connection requests is forced to wait until serving the other, but we assume that this waiting time is negligible compared to the operation times.

Figure 3 illustrates the exchange of the necessary information to build up the connection and to plan the route. The new agent - placed in the system - is connected to the name-server, after that it queries its destined connector reach (IP-address

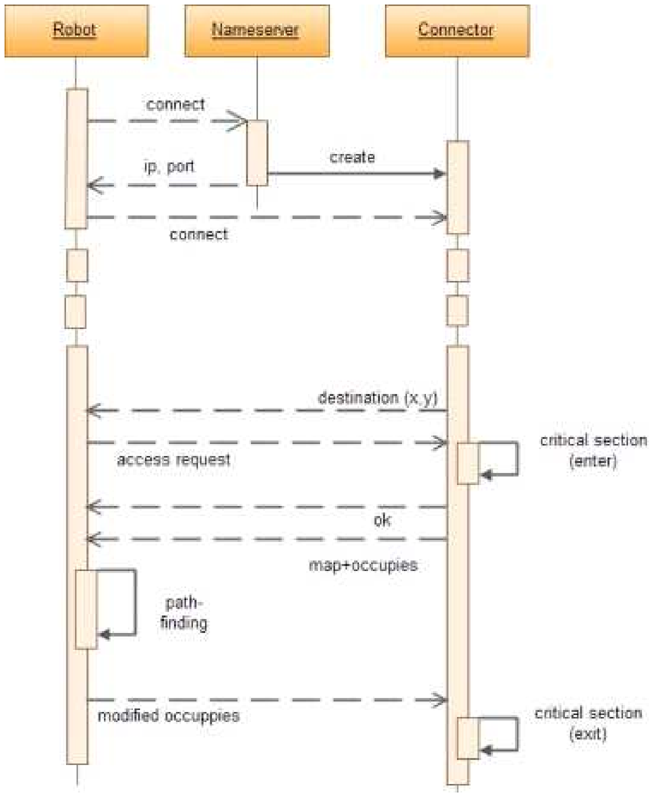

Figure 3: Building up of the connection and the exchange of information that is necessary for planning

and port) then it closes this connection. The agent builds up another connection with its connector then it waits for its task. The connector gives a task to the agent submitting its goal (with x and y coordinate pair). After that the agent asks for an access to the map for reading. Then, the first synchronized function is executed on the connector side, which induces the agent to wait until the map is used by another agent. As soon as the resource gets free, the connector grants the request and sends the necessary data: the map and the seizing. Then the agent can plan its route while taking all other agents' planned movements into consideration. As soon as the agent has performed planning it sends back the modified seizing. As a result the connector logs out of the critical section and the other agents can reach the map again. It is important that the agent can only reach the map for a given short time period.

There may be a situation in which two agents both read the seizing and thereafter they both load their calculated results back but these are in conflict (because neither could plan the other agent's route at the retrieving of the seizing). That is why we apply mutual exclusion, which forces agents to wait, but each agent has to plan their route only once. Finally, the agent sends back its route when it reaches the destination, so it no longer seizes the common resources.

D. Inter-agent communication

Until now, a simple policy of ordering the route planning of agents has been followed: the agent that gets access earlier can send back its seizing first, and from that point it cannot be changed by the later arriving agents. So if the first agent seized

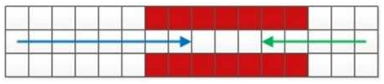

Figure 4: An example of the priority giving

itself a route, the other agents had to respect it. However, this policy may not be an optimal one. For example, it may happen that if an earlier planning agent were yielding precedence to a later arriving agent, than the agents could reach their destinations quicker. Figure 4 shows an example of such a situation. The first agent planned earlier (the blue arrow marks its motion) so this agent can pass through passage first. The second agent (the green arrow marks it) passes through this passage too, however it has to wait until the first agent leaves the passage. We suppose that they start at the same time from the starting point of the arrow representing them. The second agent is nearer to the entrance of the passage so the agent waits more for the first agent than the first agent would wait for the second agent if they had reserved their route in reverse order. Negotiating is optimal when the agents can directly exchange data with each other: passing data through the central server takes up twice as much time than sending the data directly.

E. Unexpected events

If an agent cannot continue the way for some reason, but the communication unit is operable, it has to report its break down to the central unit. The central unit then deletes every seizing of the broken down agent, thereafter it informs every agent whose path passes through that cell on which the agent is broken down and waiting. The affected agents then redesign their route and bring it again to the central unit.

Of course, a braking down failure can occur in such a way that causes the complete stop of the agent and it cannot signal its state. We will not deal with this problem in this paper.

F. Advance planning

An agent can move on one route, but we need to pay attention to certain situations meanwhile, at the same time. For example:

1) There may be situations in which it is simply not enough to avoid another agent because for example the agent 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 agent than to choose a bypass route.
2) There can be some narrow passages in the storage for the sake of better utilization of space, therefore we also need to deal with them. In these passages there can be one agent at a time, this can cause a traffic-jam. If two agents approach the passage at its opposite ends then the route search algorithm can sense only the character of the problem before the collisions.

The possible collisions can be detected in advance, not locally. This requires communication among agents but if every agent communicates with each other then it is a lot of time that can slow down the operation. The learning ability of agents may help in such situations.

III. Advanced algorithm with machine learning

The proposed algorithm that uses machine learning implemented as a GA problem is described in this section that enables the agents to learn in order to minimize the number of communications.

A. Learning ability of agents

The basic route planning algorithm provided means on how two agents can yield precedence to each other [22]. With the cooperative communication ability of the agents available in the advanced algorithm, they can determine which agent is worthwhile to contact. If every agent established contact with every other agent at the planning, it would increase the planning time $n-1$ fold in case of n agents. The learning ability of the agents is used to minimize the need for communication, i.e., to limit the number of negotiations among agents. For this it is necessary to determine, which agents have a potential conflict against which other ones that should be contacted.

For this purpose we store the following *data* about a *collision*:

- *coordinate*: where the collision occurred
- *agentID*: identifier of the collided agent
- *distance_covered*: the time of the traveled route to collision (an expected value)
- *waiting*: how much the agent waited in the cell when the collision occurred
- *shut_down*: it is true, if the thread was shut down because of the collision; it is false, if we have to wait longer to avoid collision
- *on_route*: Is the coordinate on one of the optimal route? (true/false)
- *manh_distance*: manhattan distance from the target
- *estimation*: a specific value of the collision, which is calculated by weighting parameters (equation (1))

These variables representing the collision are stored in a data structure. The route planner builds up a list for these in the course of running.

B. The GA problem

Using these data, we use one of the methods of machine learning, this is the GA to determine from the list of collisions which agents should be contacted. For this we need to prioritize the collisions. This is stored in the *estimation* field, and calculated as:

$$\begin{aligned}&estimation = distance\_covered * A + waiting * B + \\ &+ shut\_down * C + on\_route * D + \\ &+ manh\_distance * E \end{aligned} \tag{1}$$

The numbers A, B, C, D, E are the so-called weights to which we give initial values, and their future values are determined by the GA. With their help we are able to estimate, which agents are willing to give us priority with high possibility in case of a collision.

The *list of collisions* contains data about all collision of an agent. An element of this list consists of two numbers: the serial number of the agent that is collided and the *estimation* field of that agent. From this list, we select N agents with the highest ”estimation” values (this is called the *agent list*) and we give back the list of these to the agent to which the list belongs to. The value of N is typically about 5: smaller value may cause that we do not succeed, i.e., we don't get priority from any agent, in case of bigger value we may spend too much time on communication. The order of the agent list has the most important role: if we get priority from an agent of the list, it has effect on the complete route search. In this case the total collision list voids and it is not worth to begin discussions with other agents.

GA aims at minimizing the number of communications among cooperating agents and yet to achieve the best possible result, i.e., to determine optimal routes for the agents. This is served by the estimation of the collisions, the amendment was done by the weights ($A-E$). A GA will be responsible for the determination of these weights. Following a route plan the corresponding values are calculated and are added to the agent to store, and these values are used to calculate the estimation of the collisions during the next route planning.

In two cases, the GA can be left out:

1) if we got priority from the agent that is the first in the agent list, or
2) if we didn't get priority by any of the agents meaning that the order is irrelevant.

C. GA parameters

We carried out several tests of the algorithm that we can determine the main parameters of GA, which affect to the results. We performed tests with different map size and different numbers of agents in our program simulation environment. Some important parameters of the used GA are

- maximum number of populations: 100. This is necessary to ensure that we find the optimal solution.
- mutation rate: 0.07
- size of population: 20 entities
- selection: roulette wheel selection
- recombination: we used one point crossover for generating the first third of the population, two points crossover for the second third of the population and uniform crossover for the third third of the population
- coding: binary, 5 bits per weight. So the values of the weights are placed in the $[-15, 16]$ interval.

D. Fitness function

The fitness calculation is based on noting, which was the *first element of the agent list that has succeeded* in the previous route planning. Based on this, the elements of the agent list are classified as:

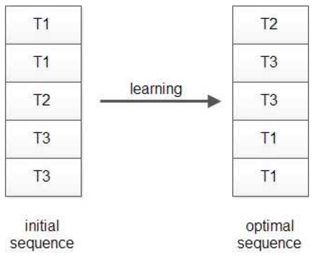

Figure 5: A possible instance of the original and optimal sequence

- priority was requested but we didn't receive (T_1 instance)
- priority was requested and we received (T_2 instance)
- priority wasn't requested because we already got it at a former element of the list (T_3 instance)

The goal of the GA is that the *estimation* field value of the T_2 instance becomes the highest, so next time the algorithm uses it earlier, this way we can save time.

Therefore, the fitness function is determined based on how much the agent list counted again by the new weights $(A, \ldots, E)$, which were stored by some individuals of the GA, approximating the optimal sequence. Optimal sequence is considered to be the one where the T_2 instance is in the first place, under it the T_3 instances (about these we do not know if the outcome had been right or wrong), and at the bottom are the T_1 instances (because these are proven wrong), as Figure 5 shows.

To be able to calculate the fitness of the new agent list - given by the GA - the agents of the existing agent list and their types ($T_1 - T_3$) should be noted. We retrieve the agents of the new list from this list, after that we execute the following algorithm (where type is type of the given agent, serial number is its position in the list).

```
$result := 0$
**if** $type = T_1$ **then**
  $result := result + serial\_number * 2$
**end if**
**if** $type = T_2$ **then**
  $result := result + (5 - serial\_number) * 5$
**end if**
**if** $type = T_3$ **then**
  $result := result + (5 - serial\_number) * 1$
**end if**
**return** result
```

Thus, better results are obtained when

- disadvantageous cases (T_1 type) are placed higher in the list,
- advantageous cases (T_2 type) are placed at the top of the list,
- neutral cases (T_3 type), preferably should be in the lower positions of the list. (Fig. 5)

The multipliers of positions (2, 5, 1) give the weights of importance.

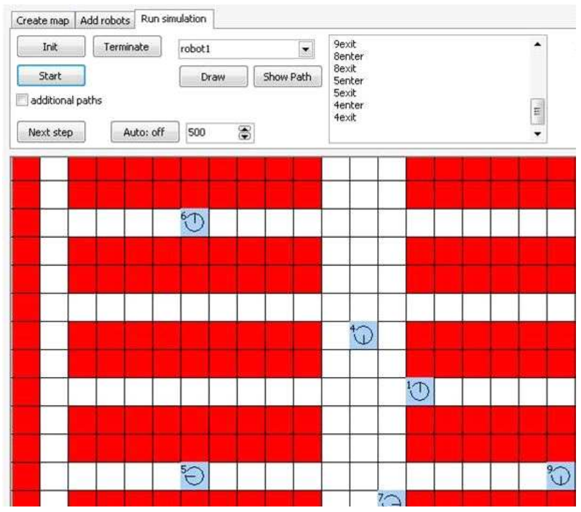

Figure 6: Screen shot part from the simulation environment

TABLE I: THE EFFECT OF THE AGENT NUMBER ON THE RUNNING TIME (BASIC ALGORITHM)

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

IV. CASE STUDIES

Simple case studies were used to test the efficiency of the proposed learning by GA. It is important to emphasize that the proposed learning by GA method is used for *improving* the result (usually not an optimal solution).

The simulation environment has been developed in Delphi programming language (see Fig. 6 for a screen shot part), by which we could test the agents' movements, and we could compare the operation with and without learning by GA.

A. Efficiency test

In order to test and compare the route planning algorithm we recorded the full running time of the algorithms and examined how this value changed with the increasing complexity of the planning problem.

1) *Effect of the number of agents using the basic route planning algorithm:* In case of the first test the agents were arranged randomly in a $25x25$ cell map-file. Five program runs were performed with each agent number value, and the running times were averaged. Table I shows the simulation results without learning, i.e., by using the basic algorithm [22].
2) *Effect of the number of agents using the advanced algorithm with GA:* In order to test the effect of learning, the agents were arranged randomly in a $25x25$ cell map-file, too. Five program runs were performed with each agent number value, and the

TABLE II: THE EFFECT OF THE AGENT NUMBER ON THE RUNNING TIME (LEARNING BY GA)

Number of agents	Time of route planning (ms)	Time of planning/ agent(ms)
5	13,9	2,78
10	26,6	2,66
15	44,25	2,95
20	59,6	2,98

running times were averaged. Table II shows the simulation results.

It can be seen from the results that the system integrates the new agents well, the agent per-planning time is below 3 ms independently of the number of agents. This important result shows that the GA scales up well with the size and complexity of the problem, thus offering an efficient service of the accumulator-bank.

V. CONCLUSION AND FUTURE WORK

A novel GA-based learning method is proposed in this paper for optimal cooperative route planning of autonomous agents moving in an accumulator-bank. The agents calculate their best possible route in a distributed way giving precedence to other agents to avoid conflict situations, and communicate with each other and the central unit. The agents are equipped with a learning ability in order to minimize the number of communications with the other agents. The adaptive behaviour lets the agents minimize the time to find the optimal routes during the search.

The effect of learning on the performance of the system has been investigated using simple case studies, and substantial improvement has been observed. At the same time, it was observed that the GA used for the learning scales up well with the size and complexity of the problem.

In the future, we will test our method in different situations and we plan to build a simulation environment, in which the agent-robots' motion as well as the unexpected events can be tested.

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, 2002, pp. 433-454.

[2] E. Alba and B. Dorronsoro, ”Solving the vehicle routing problem by using cellular genetic algorithms”, Evolutionary Computation in Combinatorial Optimization (EvoCOP), Lecture Notes in Computer Science, Vol. 3004, Springer-Verlag, Berlin, 2004, pp. 11-20.

[3] M. Armand, and J. M. Tarascon, ”Building better batteries”, Nature, Vol. 451, 7 February 2008, pp. 452-457.

[4] J. Baker, ”New technology and possible advances in energy storage”, Energy Policy, Vol. 36, 2008, pp. 4368-4373.

[5] J. Berger, M. Sassi and M. Salois, ”A hybrid genetic algorithm for the vehicle routing problem with time windows and itinerary constraints”, In: Proceedings of the Genetic and Evolutionary Computation Conference, Orlando, USA, 1999, pp. 44-51.

[6] O. Bräysy, ”Genetic Algorithms for the Vehicle Routing Problem with Time Windows”, Special Issue on Bioinformatics and Genetic Algorithms, 2001, pp. 33-38.

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

[8] D. E. Goldberg, ”Genetic Algorithms in Search Optimization and Machine Learning”, Addison Wesley, USA, 1989.

[9] A. Göllei, P. Görbe, and A. Magyar, ”Modeling and optimization of electrical vehicle baterries in complex clean energy systems”, Journal of Cleaner Production, 34, 2012, pp. 138-145.

[10] I. Hadjipaschalis, A. Poullikkas, and V. Efthimiou, ”Overview of current and future energy storage technologies for electric power applications”, Renewable and Sustainable Energy Reviews, 13, 2009, pp. 1513-1522.

[11] B. Hagen and S. Ralf, ”Implementation of Path Planning using Genetic Algorithms on Mobile Robots”, Evolutionary Computation, CEC 2006. IEEE Congress on, 2006, pp. 1831-1836.

[12] W. Ho, G. T. S. Ho, P. Ji and H. C. W Lau, ”A hybrid genetic algorithm for the multi-depot vehicle routing problem”, Engineering Applications of Artificial Intelligence 21, 2008, pp. 548-557, 2008.

[13] J. Hrncir and M. Rovatsos, ”Applying Strategic Multiagent Planning to Real-World Travel Sharing Problems” 7th International Workshop on Agents in Traffic and Transportation, AAMAS, 2012 http://arxiv.org/abs/1301.0216

[14] L. Nguyen-Thinh and N. Pinkwart, ”Strategy-based Learning through Communication with Humans”, Jezic et.al. (Eds.): KES-AMSTA 2012, LNAI 7327, 2012, pp. 54-64.

[15] H. Nazif and L. S. Lee, ”Optimized Crossover Genetic Algorithm for Vehicle Routing Problem with Time Windows”, American Journal of Applied Sciences 7, 2010, pp. 95-101.

[16] Y. B. Park, ”A hybrid genetic algorithm for the vehicle scheduling problem with due times and time deadlines”, International Journal of Productions Economics, 2001, pp. 175-188.

[17] J. Y. Potvin, and S. Bengio, ”The Vehicle Routing Problem with Time Windows Part II: Genetic Search”, INFORMS Journal on Computing 8, 1996, pp. 165-172.

[18] S. Ronald, and S. Kirkby, ”Compound optimization solving transport and routing problems with a multi-chromosome genetic algorithm” In The 1998 IEEE International Conference on Evolutionary Computation, ICEC’98, 1998, pp. 365-370.

[19] M. Saggar, T. D’Silva, N. Kohl, and P. Stone, ”Autonomous learning of stable quadruped locomotation”, In: Lakemeyer, G., Sklar, E., Sorrenti, D.G., Takahashi, T. (eds.) ToboCup2006: Robot Soccer World Cup X. LNCS (LNAI), vol. 4434, 2007, pp. 98-109.

[20] S. Sahli, and R. J. Petch, ”A GA based heuristic for the vehicle routing problem with multiple trips”, J. Math. Model. Algorithms 6, 2007, pp. 591-613.

[21] M. E. Taylor, H. B. Suay, and S. Chernova, ”Integrating reinforcement learning with human demonstrations of varying ability”, Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems, 2011, pp. 617-624.

[22] Á. Werner-Stark, T. Dulai, and M. K. Hangos, ”Cooperative optimal route planning of accumulator-bank servicing robots”, Proceedings of ICSEA 2013: The Eighth International Conference on Software Engineering Advances, 2013, pp. 408-413.

[23] M. Zolfpour-Arokhlo, A. Selamat, and S. Z. M. Hashim, ”Route planning model of multi-agent system for a supply chain management”. Expert Syst. Appl. 40, 5, 2013, pp. 1505-1518.