International Journal of Innovative
Computing, Information and Control
Volume **12**, Number **3**, June **2016**

ICIC International ©2016 ISSN 1349-4198
pp. 1005–1019

INTERFERENCE OF WAVES BASED USAGE OF AN OPTIMIZATION ALGORITHM FOR FINDING RULES WHICH CAN COPY THE LEARNED BEHAVIOR FROM LEARNING CLASSIFIER SYSTEMS AND NEURAL NETWORKS

Tomáš Cádrik, Marián Mach and Peter Sinčák

Department of Cybernetics and Artificial Intelligence
Technical University of Košice
Letná 9, Košice 04200, Slovakia
{ tomas.cadrik; marian.mach; peter.sincak }@tuke.sk

Received June 2015; revised January 2016

Abstract. *Cloud computing is getting very popular these days. It is also used with the combination of robotics. This potentially new field of robotics is called cloud robotics. The main goal in cloud robotics is to do the hard calculations on the cloud so we can spare the resources of the robot. When we have some mechanism which can be used as a controller for a robotic system, for instance, algorithms based on neural networks, evolutionary classifier systems, and other similar learning systems, we have a problem to place those controllers to the cloud. It is because they are saving the discovered data in the form of a model characteristic of this algorithm. If we want to use those in the agent domain, we need to have this method in the agent to interpret the data. Or the agent will need to access the cloud service each time step for another action. With the extension of using the cloud with a combination of robotics, it is not very effective to use this approach. So we need to build a method that can extract the data in the form of rules from the algorithms and then we can use only the extracted rules. It is also logical to use a distributed extraction mechanism because we can fully use the power of the cloud. This paper introduces a method that extracts rules from the interaction of the learned model of Zeroth Level Classifier System and neural network with the environment or the problem. The new method uses an interference of waves inspired approach based on a simple evolutionary algorithm.*

Keywords: Animat problem, Distributed calculations, Evolutionary algorithm, Interference of waves, k-multiplexer, Neural network, Rule extractor, ZCS

1. **Introduction.** The main goal of our research is to build a cloud-based learning system and database based on the cloud. The definition of cloud is in [1]. The survey around cloud computing is in [2]. The reason we want to build this system is that the resources of the robotic systems can be limited, and all of the hard calculations can be performed on the cloud without problems with different client-server architectures [3]. Also, the rules which will be learned using our system will be stored in a database on the cloud so we need to learn the behavior of the agent only once and the rules can be distributed all around the world. The robotic system will contain only a production system that will process the downloaded rules to solve a given problem. The combination of cloud and robotics is called cloud robotics [4]. The scheme of this system is showed in Figure 1.

The system will work as follows: a device (robot) will receive some task that needs to be solved. Then the device will send this task to the cloud, where there will be a database and the learning algorithms [26]. If the database does not contain the needed rules, then learning will start. The learning process will be done on a simulation mechanism. After learning of the system is finished, rules will need to be extracted from the trained state

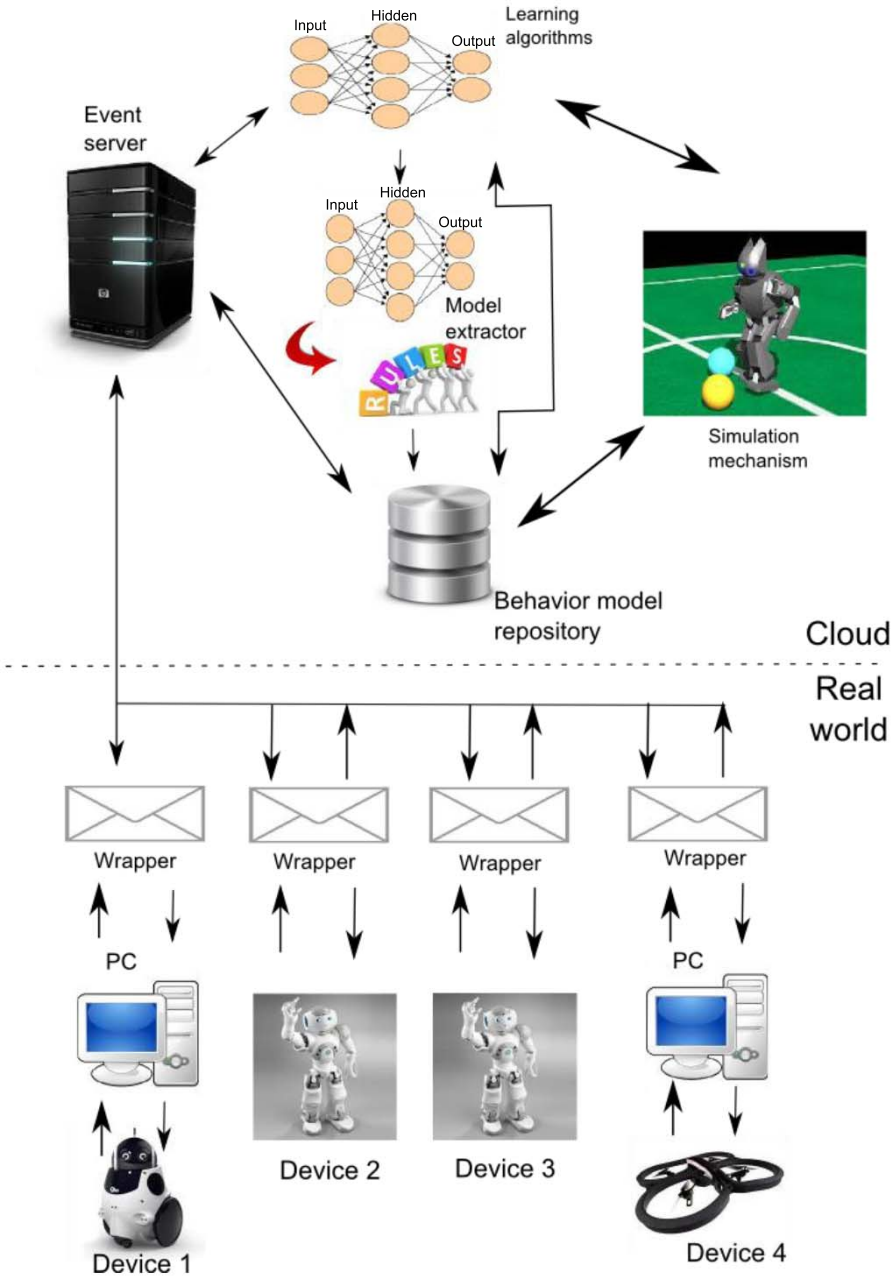

FIGURE 1. The architecture of the system that uses cloud computing from [5]

of the algorithm. After the extraction, the rules are sent to the device. The device will start to solve a given task, according to the received rules. If it is possible, the production system will be located directly on the device, so we will not need any additional external device to control it.

The goal of this paper is to describe and provide some experiments with the rule extraction. For this, we used evolutionary algorithms [6] in a wrapper, which we named interference of waves (IOW). It is called IOW because of two things. The first one is the "waves" inside each algorithm. That means it searches for the solution per part, and the parts are merged to one solution. The second reason is the "waves" represented by the algorithms. The algorithms are searching for the solution and then the solutions are merged in some way (selection of the best from the whole search or recombination of the solutions).

The IOW was used after the use of the two learned chosen methods to calculate the outputs in examples of the problems. The saved combinations of inputs and outputs from this interaction were used in the extraction process.

The paper is organized as follows. Section 2 briefly describes the state of the art about distributed use of optimization algorithms. Section 3 describes rule extraction from algorithms. Section 4 describes the ZCS classifier system and the animat problem on which the algorithm was tested. Section 5 briefly describes the artificial (NN) and the k-multiplexer problem. Section 6 describes the new interference of waves based method. Section 7 describes the experiments that show the quality of the new algorithm when extracting rules from ZCS controlling an agent in the environment of the animat problem. Section 8 describes the experiments using NN for calculating outputs in the k-multiplexer problem. Section 9 contains conclusions.

2. **Parallelism in Evolutionary Algorithms.**

Because of the increasing complexity in these days, it is almost impossible to use simple algorithms to solve it. One way how we can increase the power of the algorithm is to add parallelization to it. When we are talking about parallelism [7], we can do it in two ways. The first way is that we can add parallel functionality to the algorithm by modifying it. In [8] are multiple ways described how to parallelize evolutionary algorithms. Those types are Master-slave, Island model, Fine-grained parallel evolutionary algorithms and hybrid parallel evolutionary algorithms.

The master-slave approach has a master who works as a mechanism for fitness calculation. This node of the evolutionary algorithm distributes the fitness calculations to other cores. When the fitness calculations are completed, they are returned to the master, and master assigns the returned values to the individuals.

The island models create a set of subpopulations. Each subpopulation is an isolated "isle"; because of this, it is called the isle model. All of the operators of the EA are done separately on each subpopulation. Because of this, it is possible to use parallelization. The individuals in the subpopulations migrate between each other. In [9] are four migration strategies described: Select the best individuals and replace the worst individuals, choose random individuals and replace the worst individuals, select the best individuals and replace the random individuals, choose random individuals and replace random individuals.

The Fine-grained parallel EA is similar to the island model. However, the "isles" contain only a small number of individuals. The ideal state is when a population contains only one individual [10]. Those small subpopulations are placed into two or three-dimensional space. The environmental selection and the mating selection of a subpopulation are done using only local neighboring subpopulations.

The parallel hybrid EA is the hierarchical combination of previously mentioned techniques. The hybrid models were used for instance in [11,12].

The second choice of how to use parallelization is to use multiple algorithms which are searching the search space. So we can do optimization with multiple algorithms at the same time. The algorithms can be the same or can be different. In this paper, we are focusing on the second way, how to do parallelization.

We can do parallelization in several different ways. The first way is to use simply multiple algorithms which are the same and are searching the same search space [13]. The second choice can be that the algorithms can search different areas. This approach was used in [14]. As the optimization algorithm was the particle swarm optimization [15] used. The third method can be that we used multiple types of algorithms or algorithms with different parameters. The solution from those methods can be the best-found solution from all the algorithms.

Other ways are the usage of different fitness functions (as it is in this paper in the section when we were extracting rules from the neural network), or usage of a hybrid of the mentioned approaches. The solutions from those approaches can be the best-found solution or a combination of all solutions from each algorithm. In the last approach to multiple searching algorithms, the search algorithm needs to be modified. So we will lose the advantage to using a simple algorithm in an IOW wrapper [16], but we can obtain better results from it.

3. **Extracting Rules from Learned Behaviors.** There were some papers before which focused on the rule extraction from the learned behavior coded in some algorithm. In [17] was a method for extracting knowledge from the XCS [18] introduced. This method used preprocessing of the population set of XCS. After that, a clustering method was used for making clusters of similar individuals according to the rules they are representing. The last part of this methodology is the rule extraction process from the clusters. [19] also introduced a method for extracting rules from the classifier system; this time, it was the Zeroth Level Classifier System (ZCS). This method used the population of individuals. The first step is to sort the individuals according to the fitness values. The second part of the extraction is that the ones with the rules which have a better equivalent of the condition part are deleted.

Around extracting rules from a trained neural network (NN) were also multiple types of research done. A survey of this topic is shown in [20]. Another survey around extracting rules from trained NNs was described in [21]. [22] introduced an algorithm named X2R, which can be applied to discrete, but also to numeric data. This method created rules which had the same quality as the trained NN. In [23] Setiono and Liu presented a way to understand NNs. This understanding is done because of the extraction of rules. This was done in three phases: learning of the NN; pruning, where insignificant connections are deleted; and the last stage is extracting by recursively discretizing the hidden node activation values.

This paper introduces a new algorithm which can extract rules from the interaction of Zeroth Level Classifier System (ZCS) and NN with the environment or the problem. This algorithm uses multiple evolutionary algorithms [6] in a wrapper inspired by the interference of waves and can extract rules with binary condition parts.

4. **ZCS Evolutionary Classifier System and the Animat Problem.** Wilson introduced two classifier systems. The first one was called ZCS [24] and the second was called XCS [18,25]. These two algorithms are used to find rules that can be used to select actions in some task (environment). Those rules are coded in individuals. Each individual contains a condition part and an action part. The condition part contains symbols 0, 1 and #, where # means "do not care" symbol. The action part can include any symbol. Each individual has a fitness value that is used in the selection and when choosing children on which the genetic operators should be used. There were many applications of this system in the one-step environment (the k-multiplexer problem [24]) and in multi-step environments as the animat problem [18,24] and the corridor problem [26]. This paper focuses only on the ZCS classifier system and the animat problem.

The schematic illustration of ZCS is shown in Figure 2.

In the beginning n random individuals are created, where n is the population length of ZCS. The condition parts will contain values 0 and 1. Then each position will change with probability P# to the # symbol. The fitness is initially set to S_0. Then the learning process begins. The environment returns the actual state of the agent. Match set [M] according to the input is formed. The match set can contain individual's condition part

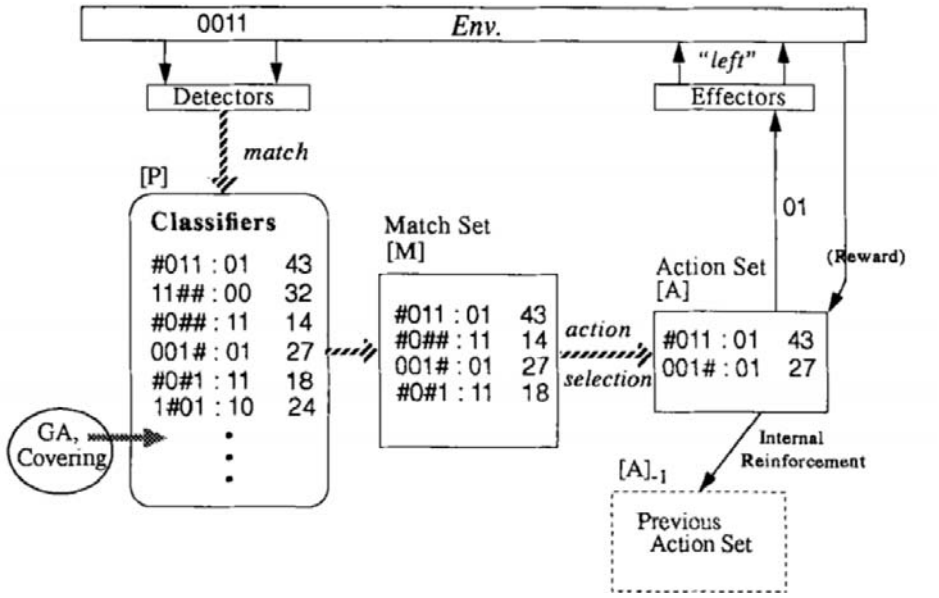

FIGURE 2. Schematic illustration of ZCS from Wilson [24]

of which matches the sensory input of the environment. An individual matches the input from the environment when all non-# positions of its condition part are the same as the input positions. The next step is the formation of the action set [A]. A roulette wheel method selects one individual from [M] according to the fitness values of the individuals. Then individuals from [M] with the same action as the selected individual are used to form the action set. Individuals from [M] which are not in [A] decrease their values of fitness by some decrements, and the decrements are put into a bucket. The selected action is sent to the environment, and the environment returns some reward. The fitness value of individuals from [A] and also individuals from the previous [A] are updated. The fitness update uses machine learning methods [26]. This update is briefly described in [24].

There are two operators in ZCS. The genetic operators start in each cycle with probability Ψ. When they start, two individuals are selected from the population with the roulette wheel according to the fitness values. One-point crossover is applied with probability χ. Then each position of the children is changed with probability μ to another symbol (i.e., if the position is 1, it can be modified to 0 or #). If a mutation is applied to the action part, it will change it to another possible action. The children are stored in the population.

When a new individual is stored in the population, one individual must be deleted from it. The roulette wheel with using the inverse fitness values is used to choose which individual will be removed.

The last operator in the ZCS classifier system is called the covering operator. This operator activates when no individual in the population corresponds to the input from the environment or when the sum of fitness values in [M] is smaller than the fraction Φ of the average fitness value of an individual in the population. The operator creates a new individual. This new individual contains the same condition part as the input from the environment and random output. Each position of this new individual is changed with probability P# to the # symbol.

The animat problem was described in [24]. The environment of the animat problem contains an agent in an environment which contains empty places, obstacles, food and has a form of a toroid. The goal is to get to the food with using the smallest number of steps. After receiving food, the agent restarts on a random position. When an agent is

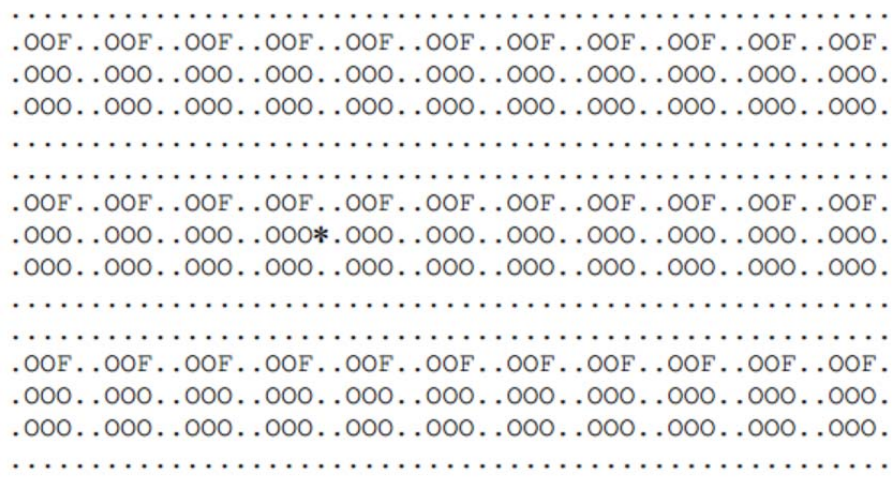

FIGURE 3. WOODS 1 environment from [24]

moving to a cell with an obstacle, it will stay in his cell, but it will lose one step. The example of an animat problem environment is in Figure 3.

"F" from Figure 3 means food, "O" means obstacle, "." means empty position and "*" means the actual position of the agent. The input from the environment has a string of characters 1 and 0 of the length 16. It represents the 8-neighbourhood of the agent, where the first two characters of the string represent the upper position of the agent. Other character pairs represent the other positions clockwise from the upper one. Food is represented by a pair 11, obstacles are represented as 10, and empty positions are represented as 00. The input from the environment according to Figure 3 will be 0000000000101011. The agent can provide one of the eight actions (moving in one direction). When the agent steps in a cell with food, ZCS receives a reward equal to 1000. Otherwise, ZCS receives reward 0.

5. **Neural Networks and the *k*-multiplexer Problem.** An artificial (NN) [28] is a massive parallel universal function approximator. It is a mechanism consisting of neurons and synaptic weights. The synaptic weights are the connections between neurons and they are used for storing the learned knowledge. There are two types of NN: recurrent and feedforward. Feed forward NN is a type where the network has three types of layers: an input layer, hidden layers (number of hidden layers can be 0) and an output layer. The input layer is the one that receives the input from the "environment". The output layer indicates the output value associated with the input. Recurrent NN is a type where the connection can be between each neuron, so we do not know which one is from which layer. The connections can be even between one neuron. In this work, we will focus only on a feed forward NN trained by the backpropagation of error learning method. The NN will be used on the k-multiplexer problem.

The k-multiplexer problem was described in [18]. The actual state of this problem is represented by a string consisting of values 0 and 1. Value k represents the number of positions with which the position of the output value is calculated. An example of the input string in k-multiplexer where k is equal to 2 is shown in Figure 4.

The first two characters are 01, which are the binary representation of the value 1. The input string in k-multiplexer corresponds to the value equal to $k + 2^k$.

The NN consists of some input neurons equal to the length of the input string. The number of output neurons is equal to 2. If the first one has a higher output, then the NN

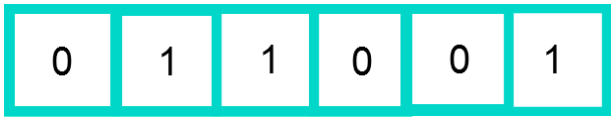

FIGURE 4. An example of the k-multiplexer problem where k is equal to 2

associates the input string with output value 0. Else it will associate it with output value 1. So that means that on the first position on the remaining string (starting from 0) is the output to this string. So the output is 0.

We have one output neuron in the NN. If the output corresponding to the input string in the k-multiplexer state is 0, the value of the output neuron should be smaller than 0.5, else it should be more than or equal to 0.5.

6. **The Interference of Waves Based Usage of Evolutionary Algorithms for Extracting Rules.** Our new method is based on the analogy of waves when a rock is dropped into the water. In the beginning, the wave is the strongest and after some time, the wave gets weaker. Then a new rock is dropped into the water, so the previous wave is reinforced. When the last wave ends and rock are not thrown into the water the interference will end.

In the first part, we must train the ZCS or NN in the given environment or problem. Then we use the trained method for solving the given problem again. After applying an output, the combination of input and chosen action is saved into a list. After that, we use this list of inputs and outputs for the rule extraction process.

The interference of waves (IOW) is used for finding the most general set of rules that represent the largest part of the input/output list as possible. Then those rules can be used as a knowledge base in a production mechanism. The scheme of the IOW method is shown in Figure 5.

The fitness value is calculated according to Figure 6.

In the beginning, the fitness value is set to the value of the number of rules that we want to use for extraction. The reason we want to use this is that we do not want the value of fitness to drop below zero, so in the future, we can use selection methods as roulette that cannot use negative fitness values. Our method contains X evolutionary algorithms. In the beginning, each individual of each evolutionary algorithm will be used to evolve only one rule.

After applying the chosen number of cycles, the wave will end, but the next wave will rise. Each individual in each evolutionary algorithm will have in his chromosome the first rule and a randomly created new rule. The evolution process will continue only with the new rule. That means that the first rule of each individual will remain the same, and the genetic operators will be used only for the second one. The fitness value of that individual is calculated by using both rules. After the second wave, the third wave starts. In the third wave, each individual will have three rules and only on the last one are the genetic operators applied. The search process continues while at least one evolutionary algorithm is not stable. If an algorithm becomes stable, it stops evolving. When all algorithms became stable, the method returns the best individual from the whole searching process. An evolutionary algorithm becomes stable when after all cycles in wave w_n is the fitness value of the best individual in this evolutionary algorithm equal or less as the fitness value of the best individual in this evolutionary algorithm in wave $w_n - 1$.

Evolutionary algorithms in our method have two genetic operators. The first one is a one-point crossover. The second one is a mutation. The mutation changes each position of the rule of the individual with a probability of mutation to a symbol which is not in

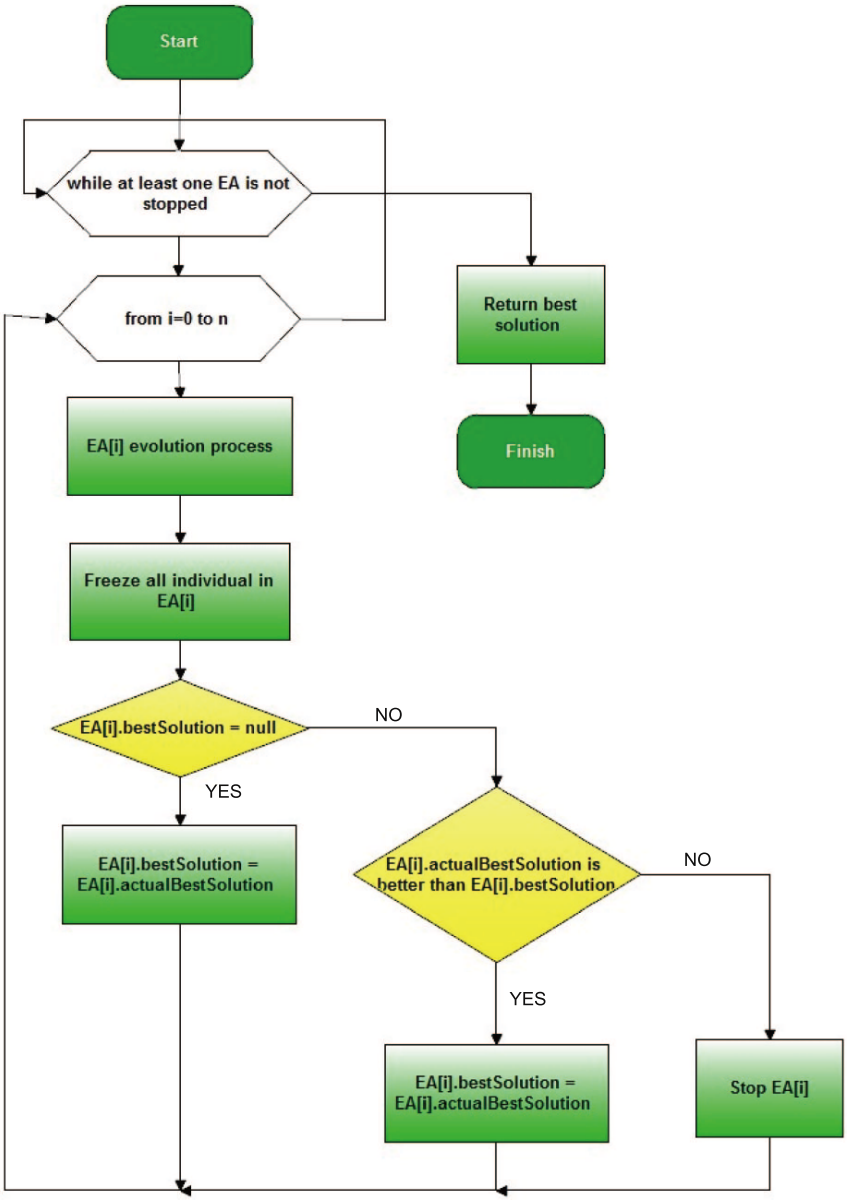

FIGURE 5. The diagram of the interference of waves based method

a given position (if the position contains 1, it will be changed to 0, and if the position contains 0, it will be changed to 1). The chromosome of an individual represents the condition part of the rule and the action part of the rule (when a mutation is applied to the action part, it changes it to another random action). The action part is coded using the same number of characters as are used when coding the actions of rules used for extraction. The condition part is two times longer than the input from the environment. One pair in the individual corresponds to one position of the condition part of the rule. If the pair is 00, the position of the condition part must be 0 if we want to say that it is the same. If the pair is 11, the position must be 1. If the pair is 10 or 01, the position of the condition part of the rule can be 0 or 1. The condition of the individual is the same as the condition part of the rule when all pairs of the condition parts of the individual correspond to the positions of the condition part of the rule. As the selection method, the q-ary tournament is used. The tournament selects the number of individuals which is equal to the number of individuals in the population. Then the pairs of individuals

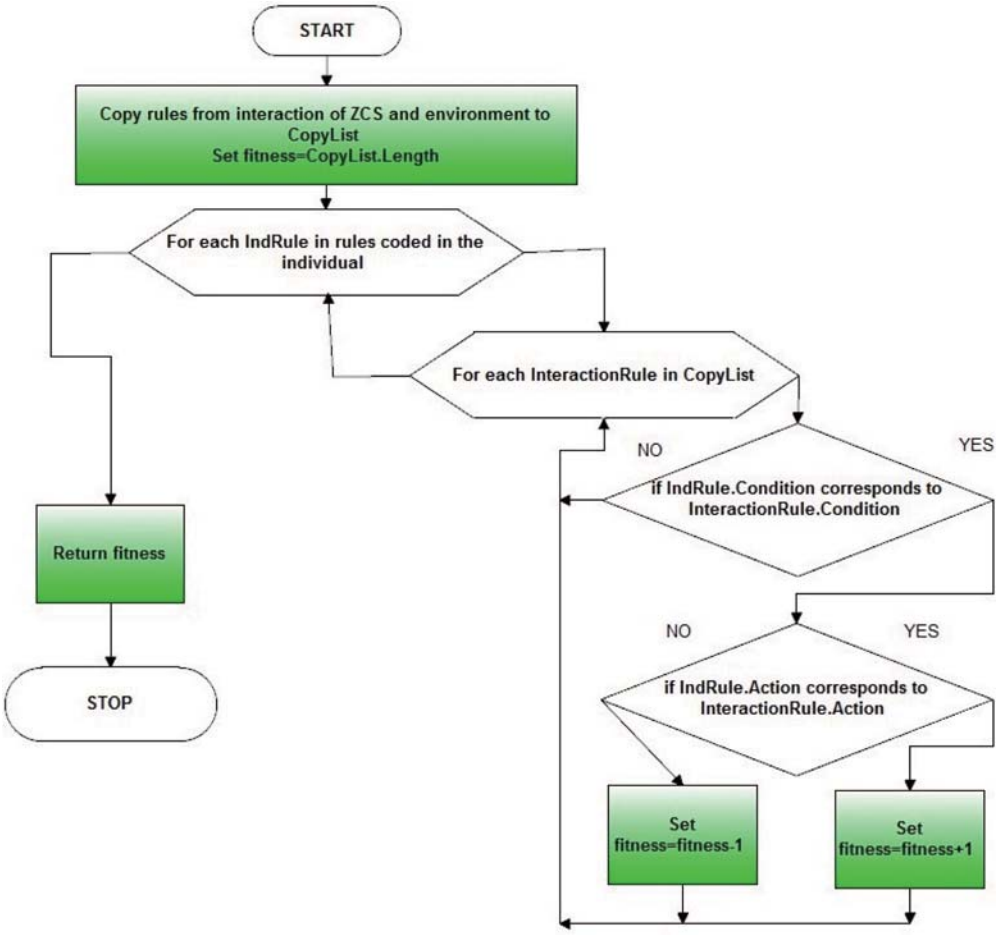

FIGURE 6. Fitness value calculation

are chosen to create children (the first parent with the second, the third parent with the fourth one, etc.). Each combination of two parents creates two children. To form a new generation, the better half of the newly created individuals replaces the worse half of the current population.

When we extract a rule set and apply them to a controller in a given environment, we must use them as follows: first the environment returns an input. We will search the rule set and the first rule in the rule set that condition part corresponds to the input from the environment is chosen. Then the action of the rule is sent to the environment. If no condition part of the rules in the rule set corresponds to the input of the environment, random actions from all the possible actions are chosen.

7. **Experiments with ZCS and the Animat Problem.** The ZCS classifier system is learned according to the parameters that were found and summarized in [29]. First the classifier system was trained with using 10000 cycles. Then 1000 cycles were used to test the learned ZCS, and the first 200 rules are selected for the extraction. The new-created algorithm has ten evolutionary algorithms, and each evolutionary algorithm contains 200 individuals. Q in the tournament of each evolutionary algorithm is set to 3. The cycle in our experiment means all the steps of the agent from the starting cell to the cell with food. The first experiment shows the comparison of ZCS and the controller containing the extracted rules in the WOODS 1 environment from Figure 3. The comparison between ZCS and the extracted rules used as a controller controlling the agent is shown in Figure 7.

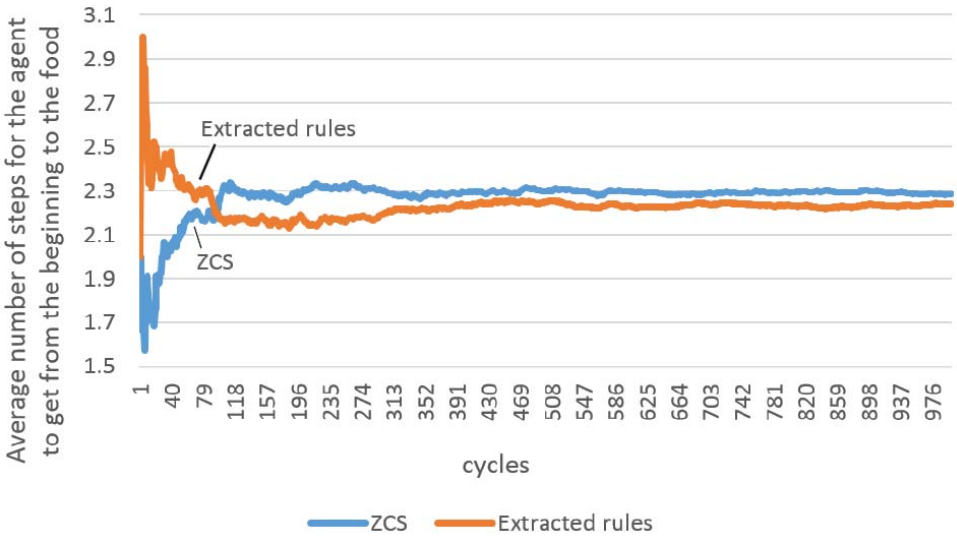

FIGURE 7. Comparison between ZCS and the extracted rules as a controller in the WOODS 1 environment

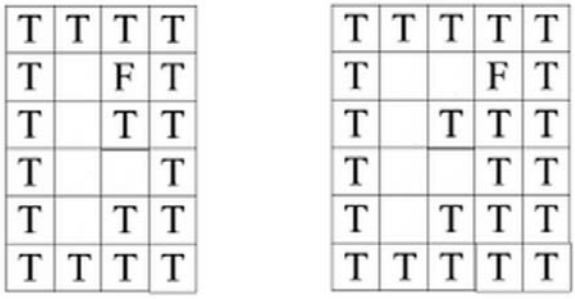

FIGURE 8. MazeF1 and MazeF2 environments

As we can see from the previous graph, the extracted rules are better than the ZCS. It is even when ZCS is learning all the time, and we used only first 200 rules from the interaction of ZCS and the environment. Even according to those two disadvantages, extracted rules are comparable with the focus on the number of steps for the agent to get the food. The second and the third experiments were done on the MazeF1 respectively on the MazeF2 environments shown in Figure 8.

T means the same as O in the WOODS 1 environment. Those two environments are more complicated as the WOODS 1 environment. It is because of two reasons. The first one is that the agent cannot get from one corner to another because there are obstacles in this environment. The second reason is that from some starting position the agent needs to provide more steps as the maximal steps from each position in the woods one environment. The comparison between ZCS and the extracted rules is shown in Figure 7 and, in Figure 8 respectively.

As we can see from the previous graphs, the extracted rules are again similar to ZCS as in WOODS 1. In the MazeF2 environment, the number of steps of the agent to get to the food using ZCS was at the end increasing. The average number of steps using the extracted rules is always stable. These experiments show that the new-created rule extractor extracts rules that are good enough for solving the problem as navigating an agent to the goal position even with using a small amount of rules for extraction.

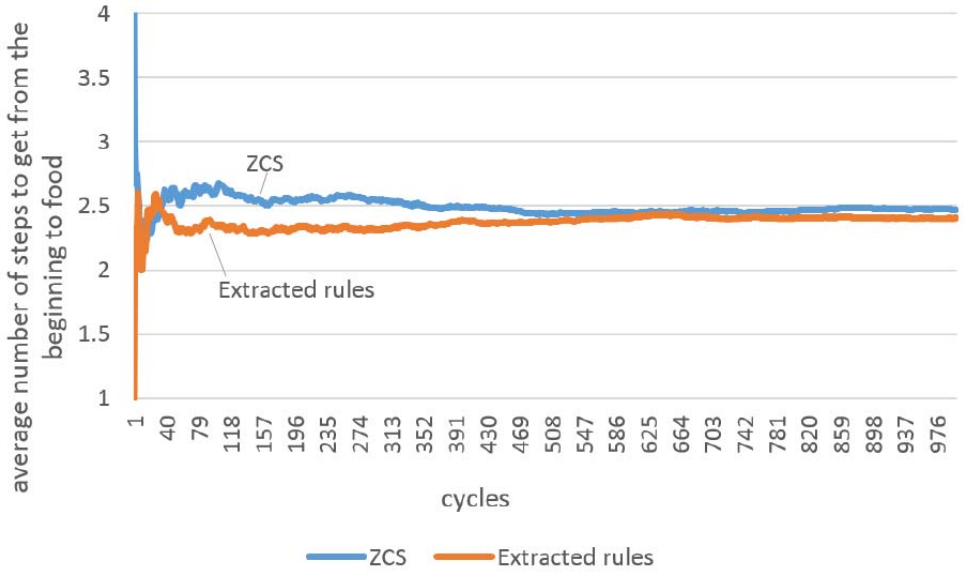

FIGURE 9. Comparison between ZCS and the extracted rules as a controller in the MazeF1 environment

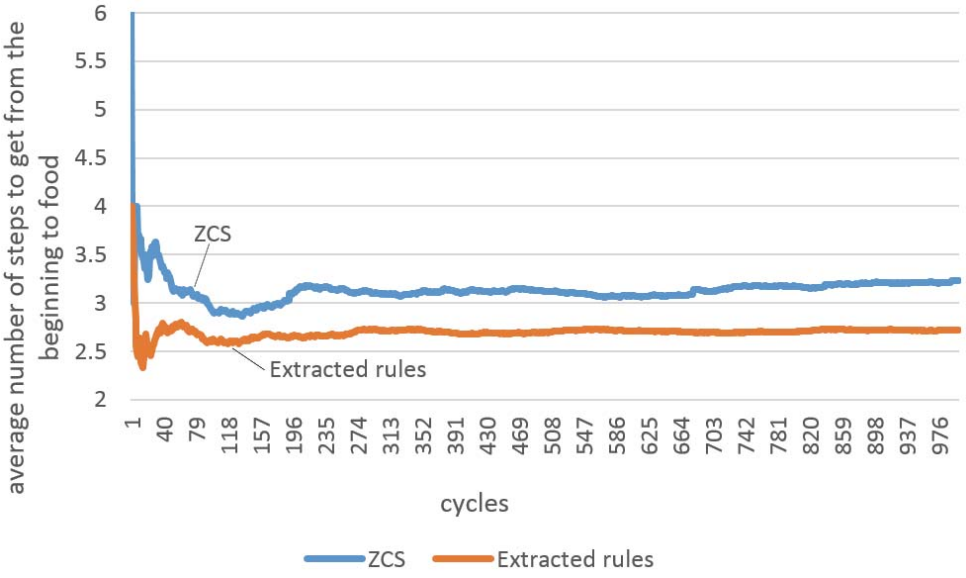

FIGURE 10. Comparison between ZCS and the extracted rules as a controller in the MazeF2 environment

Similar experiments with the rule extraction were done in [19]. A method in the mentioned paper was simpler, and it was used directly on the population of individuals. It takes the created population of individuals which are representing the rules and deletes the ones which are covered by better rules. And then when the environment returns an input, an action is added to it according to the best rule which is covering the same condition part. The input and chosen output are added to the group for that rule according to which an action is selected. For each group is a most specific rule which is covering all rules in that rule created. And those rules are used the same way as the rules created by IOW.

This method was also used in three environments. One of the experiments was on the WOODS 1 environment. The results were similar to as the results obtained with the IOW in this paper. Also, it was good enough in the next two environments. The results were

not better in the using IOW but the methodology presented in this paper had one big advantage when comparing the one presented in [19]. It can be used with the combination of any learning method, not only the methods which are coding the knowledge in a form of a population of individuals.

8. **Experiments with Neural Networks and the *k*-multiplexor Problem.** We used an NN with n input neurons, where n represents the input length in the k-multiplexer like string. It also consists of 10 hidden neurons in one layer and one output neuron. First the backpropagation of error method was used. The training was done on 400 resp. 800 randomly generated samples. This training was done using 5000 cycles. The parameter α was set to 0.5.

After the learning process, other samples were created. The trained NN was used for classifying the inputs. Those combinations of inputs and outputs are used for the rule extraction process using the IOW. After the extraction process, the created regulator with the rules is used for classifying the inputs from those samples used for the training of the NN. Also, other samples were generated. On the two sets the NN and regulator, created with IOW, were compared.

The next tables show the results where k was set to 2. The NN was trained using 400 samples. The testing was done on two samples with the amount of 400 k-multiplexer strings. For the second table, another approach was done. In that experiment, the rule finding process was note done using all 400 k-multiplexer strings. Each EA had 200 randomly chosen k-multiplexer string for learning. The set of 200 strings can also contain same multiple strings. The final solution is created by sorting all algorithms using the fitness, and then composing them to one set of rules.

TABLE 1. Results of experiments where $k = 2$ and the training set and test sets consist of 400 samples. The IOW used the same algorithms.

Method	Classification of the first samples	Classification of the first samples [%]	Classification of the second samples	Classification of the second samples [%]
NN	400	100	400	100
IOW	384	96	373	93.25

TABLE 2. Results of experiments where $k = 2$ and the training set and test sets consist of 400 samples. The IOW used algorithms with fitness functions with different rules from the interaction.

Method	Classification of the first samples	Classification of the first samples [%]	Classification of the second samples	Classification of the second samples [%]
NN	400	100	400	100
IOW	386	96.5	386	96.5

As the previous tables show, the NN was able to classify all conditions correctly from both sets of samples. In the first case, the classification mechanism created by the classified the samples with the success rate 96 resp. 93.25 percent. In the second case, the success rate was even better. Important is the fact that in the second table are the algorithms used, where the fitness is calculated with a smaller amount of conditions and actions generated from the NN. That means that the processing time was lower.

The next two tables show the results where k is set to 3, and the number of k-multiplexer strings in the samples is 800. The number of randomly chosen strings in the second case was 400.

TABLE 3. Results of experiments where $k = 3$ and the training set and test sets consist of 800 samples. The IOW used the same algorithms.

Method	Classification of the first samples	Classification of the first samples [%]	Classification of the second samples	Classification of the second samples [%]
NN	788	98.5	779	97.37
IOW	659	82.37	626	78.25

TABLE 4. Results of experiments where $k = 3$ and the training set and test sets consist of 800 samples. The IOW used algorithms with fitness functions with different rules from the interaction.

Method	Classification of the first samples	Classification of the first samples [%]	Classification of the second samples	Classification of the second samples [%]
NN	795	100	797	100
IOW	605	75.62	596	74.5

As we can see, the success rate is lower as it was in the previous experiments where k was equal to 2. Usage of multiple algorithms with the randomly chosen condition and action parts are even worst. However, we must say that the calculation time was multiple times lesser. The success rate around 75 percent is not bad, but in the future, it will be necessary to increase it. We can increase it, by using another method for recombination of the solutions from the algorithms not only the simple sorting and merging the solution.

There were many types of research done on the rule extraction from NN. However, this IOW method is specific in the case of extracting rules from the algorithm without modifying it. We were doing the extraction only from the conditions and actions chosen from the NN. Also, what is good using this methodology in comparison with other ones, is the fact that also the IOW does not need be modified. Only the algorithm need to change the representation of the solution. Those two factors are the reasons why it is interesting to focus on this methodology even when the results were not ideal. They can be increased when creating new methods for recombination of solutions and ways how to combine multiple EAs (or other optimization algorithms). We can use for instance a combination of different algorithms in one search and make a combination of the obtained solutions.

9. **Conclusions and Future Work.** We introduced in this paper a method based on the IOW. Particularly, this algorithm uses a set of evolutionary algorithms in each wave. However, it is possible to replace the evolutionary algorithms with other algorithms as PSO [14] for evolving rules with condition parts containing real values.

The algorithm was tested with the combination of two methods: ZCS and NN. The problem used for solving with ZCS was animat problem. We used three different environments. Rules were extracted from the interaction of ZCS and the environment. We tried the rule extractor only on some simple environments because there was a problem with environments in which one input can correspond to multiple outputs. This method

does not work in that type of environments. However, in those three environments the extracted rules have the same quality as the learning ZCS, sometimes even better.

The NN was tested on a problem called k-multiplexer. The rule extraction was done again on the interaction between the learning method (NN) and the problem. The IOW was able to approximate the behavior of the NN only with a limited quality. The extracted rules were usable, but they do not have the quality of the trained NN. Because of this, it will be interesting in the future to test this extraction method when we use a bigger dataset. We can use the multiple algorithms for evolving rules, where each algorithm can evolve for a different part of the dataset and then we can make a fusion of the evolved individuals and check if it is usable for our purpose.

In the future, we want also to update this work in two different ways. The first way is to use it in the cloud robotics framework mentioned in the first chapter. We want to create an AI brick (cloud service with an artificial intelligence method which can work alone, or as a part of a bigger system).

The second way is to test other EA operators, and modifications are possible. It is also possible to use this method with multiple types of algorithms and use it to other optimization problems not only the rule extraction. Interesting can be to use parallelization techniques to speed up the learning process or to increase the quality of the learned model.

When we will increase the effectiveness of the method, we want to use it on a more practical task, as a controller of a robot when dealing with a task. This task can be a cooperation of multiple robots solving some task or controlling a robot while interacting with a human.

Acknowledgment. Research supported by the National Research and Development Project Grant 1/0773/16 2016-2019 "Cloud Based Artificial Intelligence for Intelligent Robotics".

REFERENCES

[1] P. Mell and T. Grace, The NIST definition of cloud computing recommendations of the national institute of standards and technology, *NIST Spec. Publ.*, vol.145, p.7, 2011.

[2] D. Lorencik and P. Sinčák, Towards cloud robotics age, *Proc. of the 13th Scientific Conference of Young Researchers*, pp.43-46, 2013.

[3] Z. Máriás, Á. Tarcsi, T. Nikovits and Z. Halassy, Benchmark-based optimization of computational capacity distribution in a client-server web applications, *Acta Electrotechnica et Informatica*, vol.13, no.4, pp.79-84, 2013.

[4] D. Lorencik and P. Sinčák, Cloud robotics: Current trends and possible use as a service, *IEEE the 11th International Symposium on Applied Machine Intelligence and Informatics*, pp.85-88, 2013.

[5] T. Cádrik, *A Cloud-Based Multi-Robot System*, Ph.D. Thesis, 2015.

[6] M. Mach, *Evolutionary Algorithms: Elements and Principles*, Elfa, Košice, 2009.

[7] E. Alba and M. Tomassini, Parallelism and evolutionary algorithms, *Evolutionary Computation*, pp.443-462, 2002.

[8] S. Welten, *Parallelization of Evolutionary Algorithms*, Swiss Federal Institute of Technology Zurich, 2008.

[9] E. Cantú-Paz, Migration policies, selection pressure, and parallel evolutionary algorithms, *Journal of Heuristics*, vol.7, no.4, pp.311-334, 2001.

[10] S. Park, J. Kim and C. Lee, Topology and migration policy of fine-grained parallel evolutionary algorithms for numerical optimization, *Congress on Evolutionary Computation*, vol.1, pp.70-76, 2000.

[11] A. A. Tantar, N. Melab, E. G. Talbi, B. Parent and D. Horvath, A parallel hybrid genetic algorithm for protein structure prediction on the computational grid, *Future Gener. Comput. Syst.*, vol.23, no.3, pp.398-409, 2007.

[12] W. F. Punch, R. C. Averill, E. D. Goodman, S. Lin and Y. Ding, Using genetic algorithms to design laminated composite structures, *IEEE Expert: Intelligent Systems and Their Applications*, vol.10, no.1, pp.42-49, 1995.
[13] T. Cádrik and M. Mach, The method based on interference of waves for solving real value optimisation problems, *Mendel*, pp.27-30, 2015.
[14] T. Cádrik and M. Mach, Parallel usage of multiple optimization algorithms for searching different candidate spaces, *IEEE the 14th International Symposium on Applied Machine Intelligence and Informatics: Proceedings*, Herl'any, Slovakia, Danvers, pp.293-297, 2016.
[15] R. Poli, J. Kennedy and T. Blackwell, Particle swarm optimization, *Swarm Intelligence*, vol.1, pp.33-57, 2007.
[16] T. Cádrik, M. Mach and P. Sinčák, Interference of waves based usage of an optimization algorithm for finding rules in an agent system, *SCIS and ISIS*, Danvers, pp.1068-1072, 2014.
[17] F. Kharbat, M. Odeh and L. Bull, New approach for extracting knowledge from the XCS learning classifier system, *Int. J. Hybrid Intell. Syst.*, vol.4, no.2, pp.49-62, 2007.
[18] S. W. Wilson, Classifier fitness based on accuracy, *Evolutionary Computation*, vol.3, no.2, pp.149-175, 1995.
[19] T. Cádrik and M. Mach, Extracting rules from ZCS evolutionary classifier system for the purpose of future usage in robotic systems, *Mendel*, pp.393-396, 2014.
[20] R. Andrews, J. Dietrich and A. B. Tickle, Survey and critique of techniques for extracting rules from trained artificial neural networks, *Knowledge-Based Systems*, vol.8, no.6, pp.373-389, 1995.
[21] S. M. Kamruzzaman and M. M. Islam, Extraction of symbolic rules from artificial neural networks, *Engineering and Technology*, vol.10, 2005.
[22] H. Liu and S. T. Tan, X2R: A fast rule generator, *Proc. of IEEE International Conference on Systems, Man and Cybernetics*, Vancouver, CA, 1995.
[23] R. Setiono and H. Liu, Understanding neural networks via rule extraction, *Proc. of the 14th International Joint Conference on Artificial Intelligence*, pp.480-485, 1995.
[24] S. W. Wilson, ZCS: A zeroth level classifier system, *Evolutionary Computation*, vol.2, pp.1-18, 1994.
[25] M. V. Butz and S. W. Wilson, An algorithmic description of XCS, *Adv. Learn. Classif. Syst.*, vol.6, pp.267-274, 2001.
[26] K. W. Tang and R. A. Jarvis, Is XCS suitable for problems with temporal rewards?, *Int. Conf. Intell. Agents, Web Technol. Internet Commer.*, vol.2, 2005.
[27] K. Machová, *Machine Learning: Principles and Algorithms*, Elfa, Košice, 2002.
[28] P. Sinčák and G. Andrejková, *Neural Networks Vol. 2*, Elfa, Košice, 1996.
[29] T. Cádrik and M. Mach, Evolutionary classifier systems, *Electrical Engineering and Informatics IV: Proc. of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice*, Košice, pp.168-172, 2013.