STUDIA INFORMATICA
Nr 2(9) Systemy i technologie informacyjne 2007

LCS Approach to Tasks Scheduling Problem in the Two Processor System

Katarzyna Wasielewska[1], Franciszek Seredyński[2,3]
[1] Institute of Computer Science, Higher State School of Vocational Education, ul. Wojska Polskiego 1, 82-300 Elbląg, Poland
[2] Institute of Computer Science, University of Podlasie, ul. Sienkiewicza 51, 08-110 Siedlce, Poland
[3] Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland

Abstract. In this paper we propose an approach to solve multiprocessor scheduling problem with use of rule-based learning machine – Learning Classifier System (LCS). LCS combines reinforcement learning and evolutionary computing to produce adaptive systems. We interpret the multiprocessor scheduling problem as multi-step problem, where a feedback is given after some number steps. We show that LCS is able to solve scheduling tasks of a parallel program in the two processor system.

Keywords. Learning classifier systems, scheduling problem, evolutionary technique

1 Introduction

The multiprocessor scheduling problem still remains a challenge for researches in the whole world. This problem is formulated as optimization problem in which a set of tasks (with a given processing times) has to be assigned to a set of processor. The aim of this assigning is minimizing the total execution time. It can be considered as optimal allocation of the tasks in a parallel architecture. This scheduling problem is known to be NP-hard (i.e. algorithms have exponential time complexity) combinatorial optimization problem. This means that finding a solution of large problem is almost impossible. It exists some important algorithms of scheduling (e.g. list scheduling, clustering, dynamic critical-path), but these heuristics do not guarantee an optimal solution of the problem for general case. Some scheduling algorithms applies techniques derived from the nature (e.g. genetic algorithms, cellular automata, ant colony). These algorithms try to find near-optimal solutions.

In the other hand, existing heuristics for the multiprocessor scheduling problem can produce good solutions but still have high time complexity which often

comes from the necessity of searching the large spaces of solutions and has the natural influence on the behaviour of the realistic scheduling systems. Some approach to reducing the time complexity is parallelization of a scheduling algorithm [1].

In this paper we propose to use the learning classifier system XCS [9] to solve the multiprocessor scheduling problem. The XCS represents genetic-based machine learning architecture. We investigate a possibility applying of XCS as agent which migrates in the program graph and makes decisions about the choice of node and allocation it on the system graph.

The remainder of the paper is organized as follows. The next section presents the multiprocessor scheduling problem. Section 3 is an overview of learning classifier systems. Section 4 explains proposed application of XCS to solve the scheduling problem. Experimental results are presented in Section 5 and last section contains conclusions.

2 Tasks scheduling problem

A *multiprocessor system* is represented by an undirected unweighted graph $G_S = (V_S, E_S)$ (called a *system graph*), where V_S is the set of n_S nodes representing processors and E_S is the set of edges representing bi-directional channels between processors.

A parallel program is represented by a weighted directed acyclic graph $G_S = (V_S, E_S)$ (called a *program graph* or a *precedence task graph*), where V_S is the set of N_p nodes of the graph representing elementary tasks and E_P is the set of edges of the program graph describing the communication time between the tasks.

Figure 2.1 shows an example of the system graph which represents a multiprocessor system consisting of two processors: P0 and P1 and an example of the program graph with four nodes (words: tasks and nodes we will use interchangeably) and connections between them.

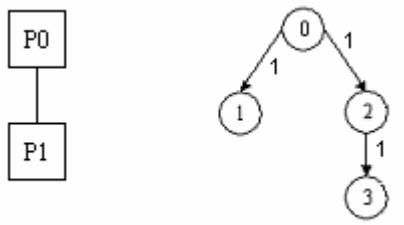


Figure 2.1. Example of a system graph (left) and a program graph (right)

The weight b_k (marked on every node) of the node k describes the processing time needed to execute a task k on some processor of the system. The weights a_{kl} of the edges (marked near edge) describe a communication time between pairs of tasks k and l, when they located in the neighbor processors. If the tasks k and l are located in the same processor, then the communication time between them id equal to 0. The

arrows indicate direct communication between a predecessor and a successor pair of tasks; and represent dependence and time precedence in parallel scheduling.

The goal of *scheduling* is to distribute the tasks among the processors in such a way that the precedence constraints are retained and the total execution time *T* (the *response time*) is minimized. The response time T for a given program graph depends on the tasks allocation in the system graph and some scheduling policy applied in individual processors:

$$T = f(allocation, scheduling\_policy)$$

We will assume that a scheduling policy is the same for all processors of the system. A schedule is often represented by a *Gantt* chart which shows allocation of tasks on processors and times when a given task is executed.

To formulate multiprocessor scheduling problem we used the models of a multiprocessor system and a parallel program which are described in [5].

3 Learning classifier system XCS

The *Learning Classifier System* (LCS) is a rule-based learning machine introduced by John Holland [4] in the 1970s. This technique combines a reinforcement learning and evolutionary computing to produce adaptive systems. The LCS is the system that learns a syntactically simple string rules (called *classifiers*). Each classifier consists of two parts: <condition>:<action>. This rule means: "if a current observed state of the environment matches the *condition*, then execute the *action*".

The problems that LCS has to solve within are divided into two classes: *a single-step* and *a multi-step*. In single-step problems an environmental feedback is returned on each step of the LCS. The second class contains problems where a feedback is given after some number steps. The multi-step environment requires a chain of actions before a feedback is received.

An overview and applications of the LCS can be found in [4,6].

In 1995 Wilson introduced the eXtended Classifier System (XCS) [9]. The goal of the XCS is to form a complete and accurate mapping of the problem space through efficient generalizations [10]. The XCS uses standard Q-learning algorithm [8] to update the parameters of the classifiers. Figure 3.1 shows an example of this classifier. Classifiers of XCS have three parameters: *prediction*, *prediction error* and *fitness*. These parameters are just updated by Q-learning technique.

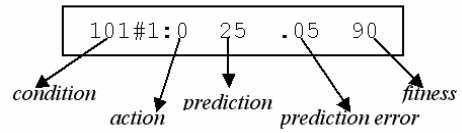


Figure 3.1. An example of a classifier of XCS

At each time step the system receives a message from the environment. The system compares this message with conditions of classifiers from population of all classifiers [P] and creates a *match set* [M] containing classifiers from the population

whose condition part matches the current input. If the [M] is empty a new classifier is created through *covering* mechanism. Then for each possible action a_i the system prediction $P(a_i)$ is computed and prediction array P(A) is created. The value $P(a_i)$ gives an evaluation of the expected reward if action a_i is performed. Then, action selection is performed. The classifiers in [M] (which propose a selected action) are placed in the *action set* [A]. The selected action is sent to the environments. And an immediate reward is returned to the system. For multi-step problems, XCS creates the *previous cycle's action set* $[A]_{-1}$. Figure 3.2 shows an illustration of the XCS cycle.

The reward is used to update the parameters of the classifiers in the action set corresponding to the previous time step $[A]_{-1}$. $[A]_{-1}$ is updated using the sum of the previous cycle's reward and the discounted maximum of P(A).

GA is applied to the action set. It selects two classifiers with probability proportional to their fitnesses, copies them and performs crossover on the copies and mutates each allele with some probability.

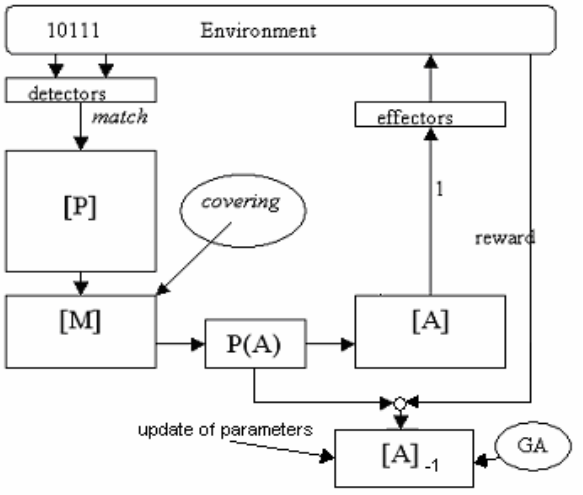


Figure 3.2. A schematic illustration of XCS for a multi-step problem

The XCS works in two modes: exploration and exploitation. In *exploration mode* the action is selected randomly from the rules with non-zero prediction within [M]. In *exploitation mode* the action with highest value of prediction is selected and the GA is no active.

4 An approach to scheduling problem based on XCS

We propose a multi-step approach to tasks scheduling based on decisions of an agent-learning classifier system of the XCS. Agent is migrating in the program graph, interpreting it as an environment, and taking decisions about allocation of the chosen tasks to processors. The goal of the agent is finding of optimal allocation of program tasks into processors.

In the beginning the agent receives an information from the environment about number of node which can be first allocate in the system graph. Then, the agent receives a message from the environment describing a position of all nodes (tasks of a parallel program) from point of view of its current position of the node where it stays. A length of a message is equal to 2*n*, where *n* is the number of tasks in the considered program graph. Coding the message is following:

- values 11 - these values on a given *i-th* position of a message says that the node *i* is a current position of the agent;
- values 10 - these values on a *i-th* position says that the *i-th* task is a successor of a task where the agent currently stays;
- values 01 – these values on a *i-th* position says that the *i-th* task is a predecessor of a task where the agent currently stays;
- values 00 - these values on a *i-th* position says that the *i-th* task is neither a successor or a predecessor of a task where the agent currently stays.

An action of a classifier of XCS has two components: a label of a task to where the agent will move from the current position after execution of this action, and a label of a processor to which the proposed task will be allocated. Coding of the action is following:

- action 0 – the agent will choose the task 0 and this task will be allocated into processor P0;
- action 1 – the agent will choose the task 1 and this task will be allocated into processor P0;
- action 2 – the agent will choose the task 2 and this task will be allocated into processor P0;
- action 3 – the agent will choose the task 3 and this task will be allocated into processor P0;
- action 4 – the agent will choose the task 0 and this task will be allocated into processor P1;
- action 5 – the agent will choose the task 1 and this task will be allocated into processor P1;
- action 6 – the agent will choose the task 2 and this task will be allocated into processor P1;
- action 7 – the agent will choose the task 3 and this task will be allocated into processor P1;

For example, the classifier of <11 10 10 00>:<2> can be interpreted in the following way: the agent location is the task 0; this task has two successors: 1 and 2; and the task 3 task is neither a successor or a predecessor of a current task, but agent can choose task 3 if its all predecessors were allocated; and the action: “go to the task 2 and allocate it into processor P0”.

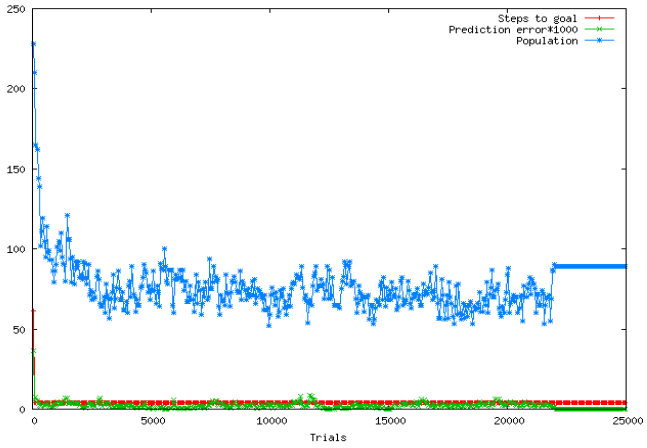
After execution of the action the agent moves to proposed node and allocates this task on proposed processor. Then the agent reads a perception (new actual position) and executes successive action. The agent moves under control of XCS until it visits last node. We don’t suppose any order constraints. The agent should learn some scheduling policy. This is the chain of steps. In this sequential problem of scheduling the positive reward will arrive on last step, i.e. the agent is estimated for the cycle of the actions.

5 Experiments results

The goal of experiments is to verify our approach to tasks scheduling problem based on XCS. We will focus our attention on the system graph from Figure 2.1 and the program graph from Figure 2.2. All weights of this program graph are set to 1. The response time *T* for this program graph in the two processors system is equal 3.

For each problem the agent migrates under control of XCS until it visits all four nodes and receives the reward of 1000. The performance is computed as the average number of steps to goal in the last 50 exploitation problems. The standard statistic of LCS to multi-step problems: *steps to goal* means here the number of steps which the agent needs to realize the program graph. In this problem an optimal number of steps to goal is equals 4. Each results presented in this paper is averaged on five experiments.

a)



b)

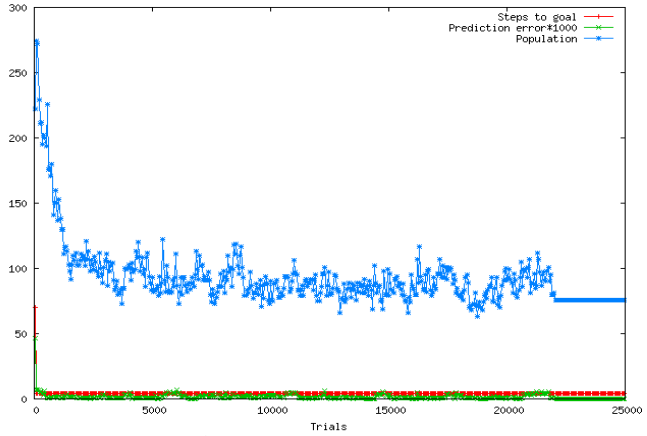
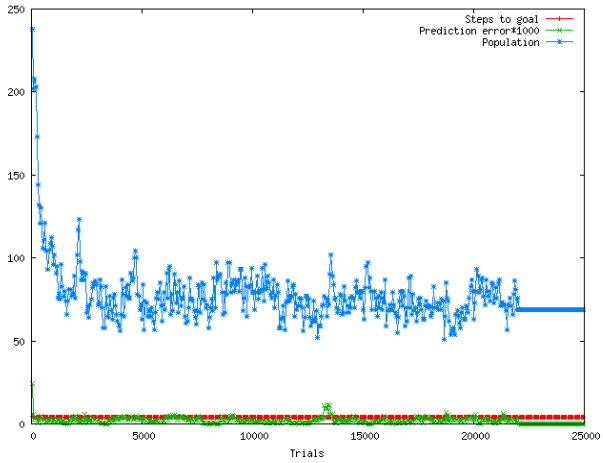


Figure 5.1. The statistics of XCS: steps to goal, population size and prediction error for the values of parameters: $P_{\#}$ = 0.8 and N=400 (a) and N=800 (b)

The figures below show the results of experiments. We can see on these pictures following statistics: the number of steps to goal, the population size and the prediction error. The parameters values of XCS were set as by Wilson in [9]: $\beta = 0.2$, $\gamma = 0.71$, $\theta = 25$, $\varepsilon_0 = 0.01$, $\alpha = 0.1$, $\chi = 0.8$, $\mu = 0.01$, $\delta = 0.1$, $\Phi = 0.5$, $p_I = 10.0$, $\varepsilon_I = 0.0$, $F_I = 10.0$. Our experiments consist in modification of population size parameter and $P_{\#}$. The parameter of $P_{\#}$ is a probability of using a don't care symbol in an allele during covering. We show below results of experiments for the values of population size parameter set to 400 and 800 and $P_{\#}$ set as following: $P_{\#} = 0.8$ (figure 5.1), $P_{\#} = 0.5$ (figure 5.2) and $P_{\#} = 0$ (Figure 5.3).

We can observe that agent learn the problem during first 50 trials. Our experiments showed that the XCS finds the optimal solutions: the steps to goal = 4 and the response T = 3 during initial trials. The prediction error decreases to zero during first 50 trials too. The minimal response time T we calculate in the following way: in every trial we count the total execution time T_E and if $T_E \leq T$ then $T = T_E$.

a)



b)

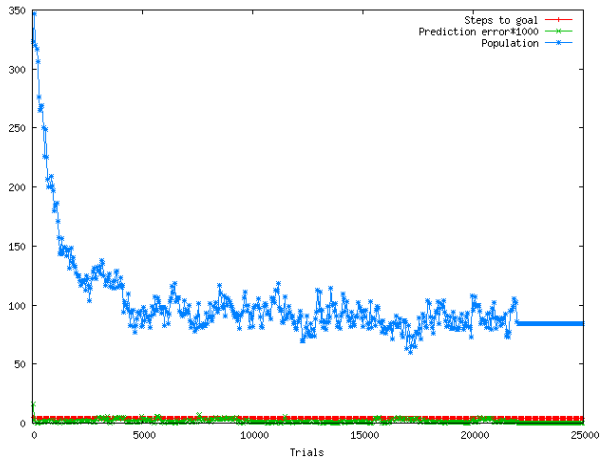
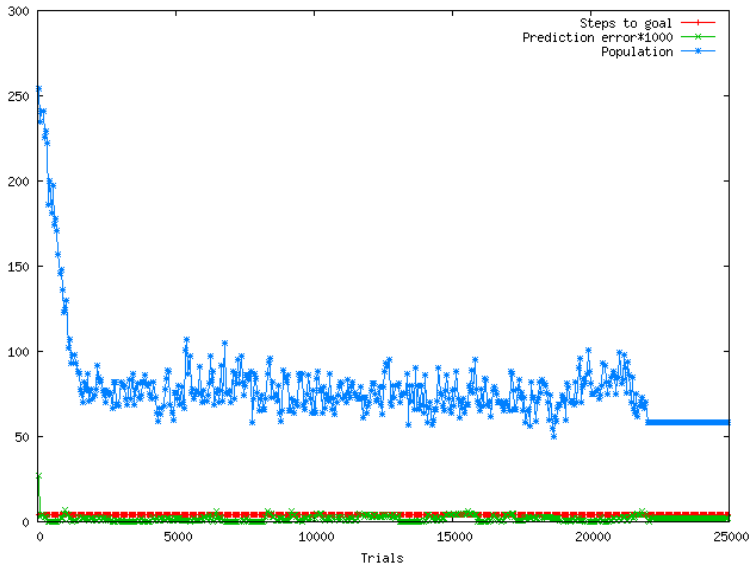


Figure 5.2. The statistics of XCS: steps to goal, population size and prediction error for the values of parameters: $P_{\#} = 0.5$ and N=400 (a) and N=800 (b)

a)



b)

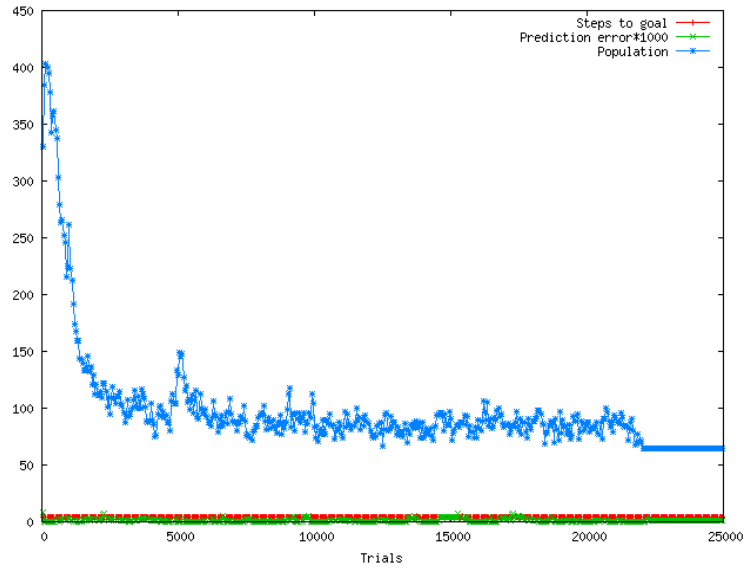
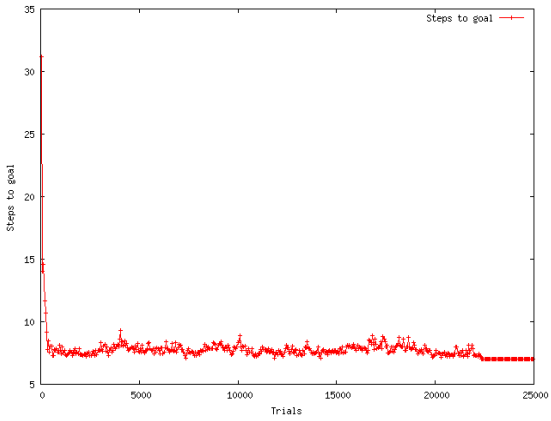


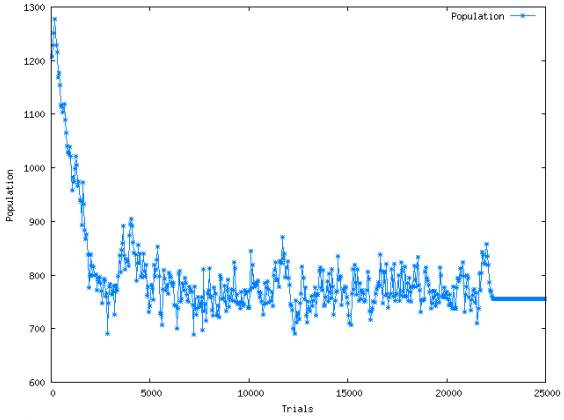
Figure 5.3. The statistics of XCS: steps to goal, population size and prediction error for the values of parameters: $P_{\#} = 0$ and N=400 (a) and N=800 (b)

During last 3000 problems exploration is turned off. We can see that values of prediction error are near equal to 0. However, figures show that a population of classifiers has tendency to keeping a population of large number of classifiers. This can indicate that generalization is not operating.

a)



b)



c)

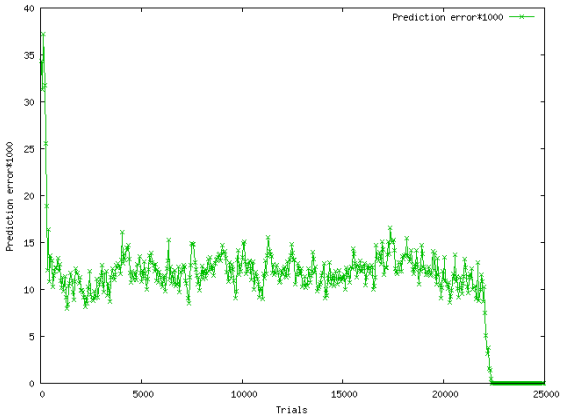


Figure 5.4. The statistics of XCS: *steps to goal* (a), *population size* (b), *prediction error* (c) for N=1600 and the standard parameters ($\beta = 0.2$, $\gamma = 0.71$, $\theta = 25$, $\varepsilon_0 = 0.01$, $\alpha = 0.1$, $\chi = 0.8$, $\mu = 0.01$, $\delta = 0.1$, $\Phi = 0.5$, $p_I = 10.0$, $\varepsilon_I = 0.0$, $F_I = 10.0$, $P_{\#} = 0.5$)

We also tested the approach where the message from the environment contains additional information about actual situation of the scheduling. In this approach the agent has the knowledge about the allocation individual tasks. The agent receives a message from the environment describing a position of all nodes (tasks of a parallel program) from point of view of its current position of the node where it stays and an information about the tasks which were allocated. A length of a message is equal to *3n*, where *n* is the number of tasks in the considered program graph. The coding of the message is as follows:

- values 11 – these values on a given *i-th* position of a message says that the node *i* is a current position of the agent;
- values 10 – these values on a *i-th* position says that the *i-th* task is a successor of a task where the agent currently stays;
- values 01 – these values on a *i-th* position says that the *i-th* task is a predecessor of a task where the agent currently stays;
- values 00 – these values on a *i-th* position says that the *i-th* task is neither a successor or a predecessor of a task where the agent currently stays;
- value 0 on *(2n+i)-th* position says that the *i-th* task wasn't allocated;
- value 1 on *(2n+i)-th* position says that the *i-th* task was allocated.

First results we have presented in [7]. We tested the program graph referred as *tree7* in the 2-processor system. Our experiments showed that the system can learn the optimal solution in short time, but the population of classifiers is not ideal. The XCS found the optimal solutions: the steps to goal = 7 and the response T = 5. The parameters values of XCS were set as by Wilson in [9]: $\beta = 0.2$, $\gamma = 0.71$, $\theta = 25$, $\varepsilon_0 = 0.01$, $\alpha = 0.1$, $\chi = 0.8$, $\mu = 0.01$, $\delta = 0.1$, $\Phi = 0.5$, $p_I = 10.0$, $\varepsilon_I = 0.0$, $F_I = 10.0$, $P_{\#} = 0.5$. We show below results of experiment for the value of population size parameter set to 1600. We can see on these pictures following statistics: the number of steps to goal (Figure 5.4a), the population size (Figure 5.4b) and the prediction error (Figure 5.4c). We have observed that agent learn the problem during first 400 trials. These results are averaged on five experiments.

We used Martin Butz version of XCS [2] available free over the web from the IlliGAL site.

6 Conclusion

In this paper we have presented an approach to tasks scheduling problem in the two processors system and the results of experiments. In our approach the LCS solves one multiprocessor scheduling problem per one experiment. Our exercises showed that the scheduler based on LCS founds the optimal solution during initial generations. The final population is not ideal because the LCS encounters the problems of combinatorial optimization theory. We have shown that learning classifier system can be promising technique to solving the scheduling problem. But the questions in this area are still open. For example, how this approach will solve more difficult scheduling problem. Future works will concern verification this approach for bigger program graphs and improvement of the scheduler in the sense of the generalization.

References

1. Ahmad I., Kwok Y.K, (1999). On Parallelizing the Multiprocessor Scheduling Problem, IEEE Transactions on Parallel and Distributed Systems, 10(4), 414-432.
2. Butz M.V., Wilson S.W., (2000). An algorithmic description of XCS, Technical Report 2000017, Illianois Genetic Algorithms Laboratory.
3. Holland J.H., Reitman J., (1978). Cognitive systems based on adaptive algorithms. In Waterman D., Hayess-Roth F. (Eds), Pattern-directed Inference Systems. Academic Press, New York.
4. Lanzi P.L., Riolo R.L., (2000). A Roadmap to the Last Decade of Learning Classifier System Research. In: Learning Classifier Systems. From Foundations to Applications, Lanzi P.L., Stolzmann W., Wilson S.W. (Eds), LNAI 1813. Springer, 33-62.
5. Swiecicka A., Seredynski F., Zomaya A., (2006). Multiprocessor scheduling and rescheduling with use of cellular automata and artificial immune system support, IEEE Trans. On Parallel and Distributed Systems, vol. 17, N3, 253-262.
6. Wasielewska K., Seredynski F., (2006). Learning Classifier Systems: a way of reinforcement learning based on evolutionary techniques. In: Evolutionary computation and global optimization, Arabas J. (Ed.), OWPW, 385-395.
7. Wasielewska K., Seredynski F., (2007). LCS approach to multiprocessor scheduling. In Grzech A. (Ed.), Proceedings of the 16th International Conference on Systems Science, OWPW, Wroclaw, Vol. 2,463-469.
8. Watkins C.J.C.H., (1989). Learning from Delayed Rewards. PhD Thesis, Cambridge University.
9. Wilson S.W., (1995). Classifier Fitness Based on Accuracy. Evolutionary Computation, 3(2), 149-76.
10. Wilson S.W., (1998). Generalization in the XCS classifier system. Proc. of the Third Annual Conference, 665-674.

System control