A NEURAL TREE MODEL FOR CLASSIFICATION OF COMPUTING GRID RESOURCES USING PSO TASKS SCHEDULING

Jarmila Škrinárová[*], *Ladislav Huraj*[†], *Vladimír Siládi*[‡]

Abstract: This paper proposes a model of neural tree architecture with probabilistic neurons. These trees are used for classification of a large amount of computer grid resources to classes. The first tree is used for classification of hardware part of dataset. The second tree classifies patterns of software identifiers. Trees are implemented to successfully separate inputs into nine classes of resources. We propose Particle Swarm Optimization model for tasks scheduling in computer grid. We compared time of creation of schedule and time of makespan in six series of experiments without and with using neural trees. In experiments with using neural tree we gained the subset of suitable computational resources. The aim is effective mapping of a large batch of tasks into particular resources. On the base of experiments we can say that improvements have been made even for middle and small batch of tasks.

Key words: *Neural tree, PSO optimization, tasks scheduling*

Received: February 19, 2013
Revised and accepted: April 17, 2013

1. Introduction

Computational grids can be used for solving problems that require processing of large quantity of operations or data. Grid computing allows sharing of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a cohesive resource environment for executing distributed applications [1].

[*]Jarmila Škrinárová
Faculty of Natural Sciences Matej Bel University, jarmila.skrinarova@umb.sk
[†]Ladislav Huraj
Faculty of Natural Sciences Ss. Cyril and Methodius University, ladislav.huraj@ucm.sk
[‡]Vladimír Siládi
Faculty of Natural Sciences Matej Bel University, vladimir.siladi@umb.sk

Job scheduling in its different forms is computationally hard. It has been shown that the problem of finding optimal scheduling in heterogeneous systems is, in general, NP-hard [6, 8]. An application can generate several jobs which, in turn, can be composed of subtasks and the Grid system is responsible for sending each subtask to a resource to be solved. Grid systems contain schedulers that automatically and efficiently find the most appropriate machines to execute an assembly of tasks.

Task represents a computational unit which runs on a grid node. Typically it is a program and possibly associated data. A task is considered as an indivisible schedulable unit.

Job is a computational activity made up of several tasks that could require different processing capabilities and could have different resource requirements (CPU, number of nodes, memory, software libraries, etc.) and constraints, usually expressed within the job description. In the simplest case, a job could have just one task.

Grid scheduler is created by software components, which are responsible for a mapping of tasks to grid resources under multiple criteria and grid environment configurations. The computational grid, hierarchical by nature, is usually modeled as a multi-level system, which allows efficient management of geographically distributed resources and tasks scheduling under various criteria, including security and resource reliability requirements [31, 32]. The model is often a hybrid of centralized and decentralized modules. In the centralized module, there is a central authority. It is some metascheduler or meta-broker and it has knowledge of the system by monitoring the resources and interacts with local job dispatchers in order to define optimal schedules. In the decentralized module, the local schedulers interact with each other to manage the task pool. This kind of scheduler has the knowledge about the resource clusters, but they cannot monitor the whole system [38, 39, 17]. The hierarchical model addresses scalability and fault-tolerance issues. A meta-broker model is an example of the hierarchical two level grid system. In this model, grid users submit their tasks or applications to the meta-broker which uses also the information supplied by the resource owners to match the users tasks to appropriate resources [18].

This paper is organized as follows. Section 2 shortly reports on related works. Section 3 introduce Computational model for Grid scheduling and optimization criteria. Neural networks focused on multilayer perceptron (MLP) and radial basis activation function (RBF) with competitive learning rule are briefly outlined in Section 4. Neural tree is described in Section 5. There is a model of particle swarm optimization (PSO) algorithm for discrete variables introduced in Section 6. Data description and simulation tools are proposed in Section 7. Section 8 is focused on simulations and experimental results.

2. Related Works

Several stochastic and heuristic optimization methods have been proposed for job scheduling in computational grids. Monte Carlo methods, Simulated Annealing, Tabu Search, Genetic Algorithms [33, 34], among others, attempt to avoid the premature convergence to the local minima. An implementation of Simulated Annealing for grid scheduling was proposed in [25, 26]. Recently some parallel genetic algorithms frameworks have been used for designing the effective grid schedulers. Lim et al. [19] propose Grid-Enabled Hierarchical Parallel Genetic Algorithm (GE-HPGA). Another method of improving the scheduling quality is the hybridization of the heuristics with Local Search methods for the problem [20]. Particle Swarm Optimization algorithms have been implemented for grid scheduling [35, 36, 37]. A hybrid version of genetic algorithms and Tabu searching is proposed by Xhafa et al. [21]. Other approaches to the problem include the use of Fuzzy Particle Swarm Optimization [22], Artificial Neural Networks [23] and economic-based approaches [24].

3. Computational Model for Grid Scheduling and Optimization Criteria

In this part computation model for Grid scheduling is presented. The computational capacity of the node depends on its:

- Number of CPUs
- Amount of memory
- Basic storage space
- Other specifications

The node has its own processing speed, which can be expressed in the number of Cycles Per Unit Time (CPUT) [3].

A job is considered as a single set of multiple atomic operations (tasks). The task is allocated to execute on one single node without pre-emption. The task has input and output data and processing requirements in order to complete its task. The task has a processing length expressed in the number of cycles.

A schedule is the mapping of the tasks to specific time intervals of the grid nodes.

A scheduling problem is specified by:

- A set of machines
- A set of jobs
- Optimality criteria
- Environmental specifications
- Other constraints

One of the most popular optimization criteria is the minimization of the makespan. Makespan is an indicator of the general productivity of the grid system. Small values of makespan mean that the scheduler is providing good and efficient planning of tasks to resources. Makespan indicates the time when the last task finishes. Flowtime is the sum of finalization times of all the tasks.

Let J_j ($j\{1, 2, \dots, n\}$) be independent user jobs on G_i ($i\{1, 2, \dots, m\}$) heterogeneous grid nodes with an objective of minimizing the completion time and utilizing the nodes effectively.

Number of CPUT expresses speed of every node. The length of each job is specified by the number of cycles. Each job J_j has its processing requirement that can be expressed in number of cycles. A node G_i has its calculating speed and it is expressed in cycles/second. Any job J_j has to be processed in the one of grid nodes G_i until completion. Since all the nodes at each stage are identical and preemptions are not allowed, to define a schedule it suffices to specify the completion time for all tasks comprising each job [1, 3].

To formulate our objective, we define:

- completion time $C_{i,j}(i\{1, 2, \dots, m\}, j\{1, 2, \dots, n\})$ that the grid node G_i finishes the job J_j, represents the time that the grid node G_i finishes all the jobs scheduled for itself,
- makespan can be expressed as the $C_{max} = max\{C_1, C_2, \dots, C_n\}$ and it is the maximum completion time of all jobs J_j,
- mean flowtime is $\sum_{j=1}^{n} C_j$ and it is the sum of completion times of all n jobs.

An optimal schedule will be the one that optimizes the flowtime and makespan.

In our contribution we will minimize C_{max}. It guaranties that no job takes too long. For minimization we choose PSO algorithm.

4. MLP and RBF Neural Networks

Multiple Layer Perceptron (MLP) network [2] is a feed-forward structure composed of an input layer of neurons, one or more hidden layers and an output layer. Neurons in the nearest layers can be interconnected by weighted connections. Given is an input vector, the output vector is computed by a forward pass which computes the activity levels of each layer in turn using the already computed activity levels in the earlier layers.

A non-linear activation function is one in which the output of a unit is a non-decreasing and differentiable function of the network total output.

$$net_k = \sum_j w_{kj} a_j. \tag{1}$$

In relation (1) a_j is an output from j-th neuron and weights w_{kj} are weighting signals between neuron k and neuron j in lower level.

The output from the k-th neuron is $a_k = f(net_k)$ and f is an activation function.

An error of the p-th pattern is given as:

$$E_p = \frac{1}{2} \sum_k (t_{pk} - a_{pk})^2. \tag{2}$$

The system first uses the input vector to produce its own output vector and then compares this with the desired output. If there is no difference, no learning takes place. Otherwise, the weights are changed to reduce the difference in the direction from the output layer to the input layer. This way of learning is called error back-propagation algorithm. There are several modifications of the learning algorithms. The known modifications are the gradient descent algorithm, gradient descent with momentum, conjugate gradients algorithms (e.g. Powell-Beale) or Levenberg-Marquardt algorithms.

Radial-basis-function RBF network is a feed-forward structure composed of an input layer, a hidden layer with RBF neurons and a linear output layer [15]. Adjustable weights among the neurons are only between hidden and output layer. The inputs are directly connected to the neurons in the hidden layer via unity weights. A neuron I in RBF network in the first (hidden) layer [16] has the following transfer function:

$$z(x) = r(\|x - p\|)/\sigma_i. \tag{3}$$

The element $z(x)$ is i-th output of the neuron in the hidden layer. The argument x is the input vector $x = [x_1, x_2, \ldots x_q]^T$, $r(.)$ is a radial basis function and p_i, σ_i are the center and the width for the i-th RBF neuron. Output y_i of the i-th linear neuron in the output layer assuming the number of RBF neurons n is

$$y_i = \sum_{j=1}^{n} w_{ij} z(x). \tag{4}$$

Equation (4) represents linear operation.

Typical radial-basis-function is Gaussian function, in this case the transfer function is

$$z_i(x) = e^{\frac{\|x-p_i\|^2}{\sigma^2}}. \tag{5}$$

Elements p_i, σ_i are the center and the standard deviation in relation (5). Parameter σ_i is also called the spread parameter.

Probabilistic neural network is a special kind of RBF network. In the probabilistic neural networks the linear layer is replaced by a competitive layer. Competitive learning rule consists of three basic elements:

- A set of neurons that are all the same except for some randomly distributed synaptic weights, and which, therefore, respond differently to a given set of input patterns.
- A limit imposed on the "strength" of each neuron.
- A mechanism that permits the neurons to compete for the right to respond to a given subset of inputs, such that only one output neuron, or only one neuron per group, is active at the time. The neuron that wins the competition is called *winner-takes-all.*

According to the standard competitive learning rule, the change w_{ij} can be defined by [15]:

$$\Delta w_{ij} = \begin{cases} \eta(x_i - w_{ji}), & \text{if neuron } j \text{ wins the competition} \\ 0, & \text{if neuron } j \text{ loses the competition} \end{cases} \tag{6}$$

where η is learning rate parameter.

5. Neural Tree

A neural tree is a decision tree with a simple perceptron for each intermediate or non-terminal node [16]. Two main phases can be distinguished:

- Training
- Classification

The training phase
The role of this phase is to construct the neural tree. The tree is generated recursively by partitioning a training set consisting of feature vectors and their corresponding class labels. There are two kinds of nodes in the tree (intermediate nodes and leaf nodes). The procedure of constructing the tree involves three steps:

- Computing internal nodes
- Determining leaf nodes
- Labeling leaf nodes

An intermediate node classifies the input patterns into different classes. The leaf node gives input patterns into a single class which is used as the label for the leaf node. The neural tree arranges recursively partition of the training set such that each generated path ends with a leaf node. The training algorithm, along with the computation of the tree structure, calculates the connection's weights for each node. Each connection's weight is adjusted by minimizing a cost function as mean square error or another error function. These weights are used during the classification phase to classify unseen patterns.

The neural tree training algorithm consists of particular steps:

- First node (root) is created.
- The patterns of the training set are presented to the root node. The node is trained to divide the training set into subsets. The process stops after reaching some condition, for example mean square error of net.
- When a subset is homogeneous, a leaf node is associated and labeled as the corresponding class (Fig. 1).
- When some of subsets is not homogeneous, a new node is added to the neural tree at the next level. We can continue by second step. The training process is progressing and all subsets gradually become homogenous (Fig. 2).
- Training process is finished when all nodes become the leaf nodes (Fig. 3).

The classification phase
In the phase of classification, unknown patterns are presented to the root node. The class is obtained by going through the tree from root to the leaf nodes. The activation values f are computed for each node, on the base of each connection's weight. The activation values of the current node determine the next node to consider until a leaf node is reached. In every node the *winner takes-all rule* (6) is applied.

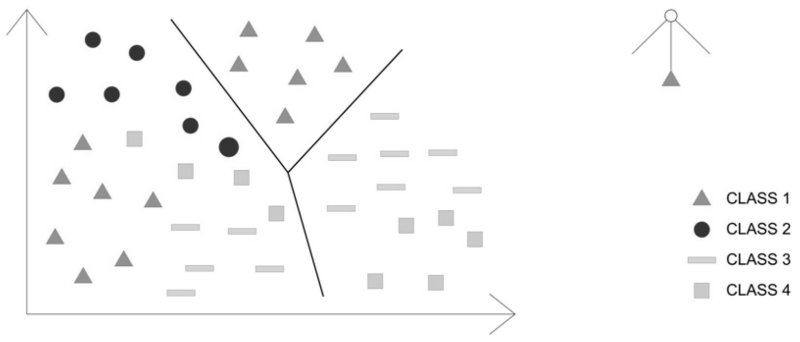

Fig. 1 *Partition of the training set by the root node.*

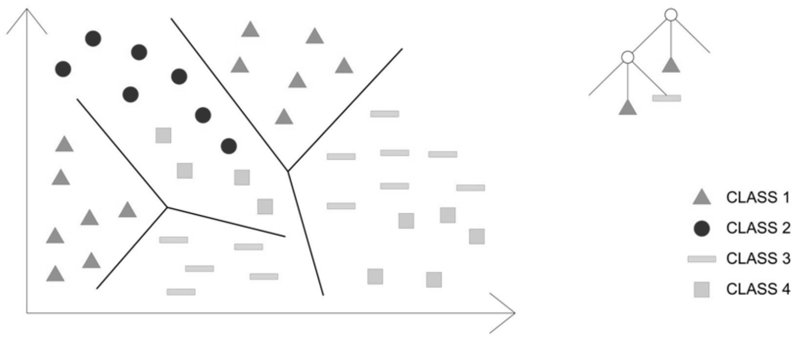

Fig. 2 *Partition by the internal node.*

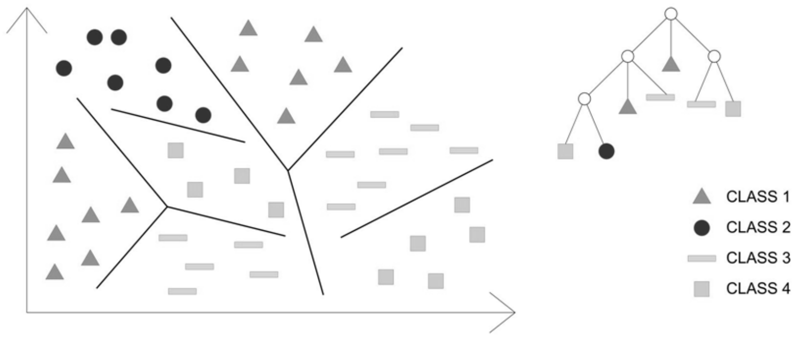

Fig. 3 *Final neural tree.*

6. The Model of Particle Swarm Optimizer Algorithm for Discrete Variables

The PSO algorithm was invented by Kennedy, Eberhart and Shi [4]. It is a population-based algorithm with fewer parameters to implement. The PSO algorithm was first applied to the optimization problems with continuous variables. Recently, it has been used to solve optimization problems with discrete variables [5, 7]. The optimization problem with discrete variables is a combination optimization problem which obtains its best solution from all possible variable combinations. The scalar S includes all permissive discrete variables arranged in ascending sequence. Each element of the scalar S is given a sequence number to represent the value of the discrete variable correspondingly. It can be expressed as follows $S_d = \{X_1, X_2, \ldots X_j, \ldots X_p\}, 1 \leq j \leq p$.

A mapping function $h(j)$ is selected to index the sequence numbers of the elements in set S and represents the value X_j of the discrete variables correspondingly $h(i) = X_j$.

Thus, the sequence numbers of the elements will substitute the discrete values in the scalar S. This method is used to search for an optimum solution, and makes the variables to be searched for in a continuous space.

The PSO algorithm includes a number of particles. The particles are initialized randomly in the search space. The position of the i-th particle in the space can be described by a vector x_j, $x_i = \left(x^{1_i}, x^{2_i}, \ldots, x^{d_i}, \ldots, x^{D_i}\right)$, $1 \leq d \leq D$, $i = 1, \ldots, n$, where D is the dimension of the particle, and n is the number of dimensions. The scalar $x^{d_i} = \{1, 2, \ldots, j, \ldots, p\}$ corresponds to the discrete variable set $\{X_1, X_2, \ldots, X_j, \ldots, X_p\}$ by the mapped function $h(j)$. Therefore, the particle flies through the continuous space, but only stays at the integer space. In other words, all the components of the vector x_i are integer numbers. The positions of the particles are updated based on each particle's personal best position as well as the best position found by the swarm in all iterations. The objective function is evaluated for each particle, and the fitness value is used to determine which position in the search space is the best of the others. The swarm is updated by the relations (7) and (8)

$$V_i^{k+1} = \omega V_i^{(k)} + c_1 r_1 \left(P_i^{(k)} - x_i^{(k)}\right) + c_2 r_2 \left(P_g^{(k)} - x_i^{(k)}\right). \tag{7}$$

$$x_i^{(k+1)} = INT\left(x_i^{(k\ )} + V_i^{(k+1)}\right), \tag{8}$$

where $1 \leq I \leq n$, represent the current position and the velocity of each particle at the k-th iteration, respectively, $P_i^{(k)}$ and $P_g^{(k)}$ the best global position among all the particles in the swarm (called gbest), r_1 and r_2 are two uniform random sequences generated from $(0, 1)$, and ω the inertia weight used to discount the previous velocity of the particle persevered [8, 9, 10].

7. Data Description and Simulation Tools

In this paper, we use the data based on the parameters of the real grid system Nordugrid. See `http://www.nordugrid.org/monitor/loadmon.php`. There are about 70 computing resources with a different number of CPUs (from 2 to 9856). Grid equipment involves a wide range of hardware. Available are multiprocessors SMP with shared memory based on architecture MIPS and multicomputer clusters using 32-bit and 64-bit processors. Internal connections among nodes are realized using special networks Myrinet (2.5Gb/s) and Infiniband (20Gb/s) for some selected nodes, and Gigabit Ethernet (1Gb/s) otherwise. Clusters use mostly 1Gb/s Ethernet or Infiniband for their internal communication. They can have different capacity of data storages, operating systems, compilers and application software. We can widely classify resources into 3 categories on the base of numbers of CPUs:

- S – small (1 – 100 CPUs)
- M – middle (100 – 1000 CPUs)
- L – large (1000 – 10000 CPUs)

User tasks include more required parameters of resources. For that reason the final characteristic of computing resources can be classified by different kinds of parameters:

- Numbers of CPUs are from the range 1 – 10000 and we classify it (1 – 3)
- Capacity of data storage is from 1 – 40000 and it corresponds to (1 – 3)
- Bandwidth between CPUs and data storages is classified (1 – 3)
- Type of operating system (1 – 5)
- Type of compiler (0 – 4)
- Type software for applications (0 – 9)

Every computing resource is denoted by a unique number and 6 previous parameters (three from the hardware and three from the software point of view). Every pattern is characterized by vector of 7 parameters. This is the way of coding each computing element.

On the other hand, we need to code a task and its characteristics. These attributes are considered the most important for our analysis:

- Length – length of the calculation in millions instructions or another unit which can relatively measure task size (1 – 100000).
- Input file size – size of the input file, before the task execution starts (1 – 3)
- Output file size – size of the output file, after the task execution ends (1 – 3)
- Type of operating system (1 – 5)
- Type of compiler (0 – 4)
- Type software for applications (0 – 9)

We have designed a model of scheduling system which consists of two basic parts. The first part is created by the neural tree. The goal of the neural tree is to match the users' tasks to an appropriate class of resources. We modeled this resource broker as a neural tree created in Matlab software tools.

The second part of our scheduling system is modeled using *GridSim* toolkit – grid simulation tool for modelling resources and scheduling applications for parallel and distributed systems. The aim of this part is to design a model that finds optimal schedule (or schedule very close to optimal) for running tasks on group of resources. As an optimization criterion we use makespan *Cmax*. For minimization makespan we choose and implement PSO algorithm.

The *GridSim* toolkit allows modeling and simulation of entities in parallel and distributed computing systems-users, applications, resources, and schedulers for design and evaluation of scheduling algorithms. This level of scheduling system uses algorithms for mapping jobs to resources to optimize system or user objectives depending on their goals. *GridSim* toolkit is written in Java and is available on-line [11].

One computing task is represented by a *Gridlet* class which contains several attributes defining character of the task (length, input file size and output file size).

We assume that we know task complexity, and we can express it in the same units as computer efficiency.

Grid resources are presented by a *GridResource* class which describes characteristics of the resource. It also contains *ResourceCharacteristic* object which contains a list of all computers in the cluster, number of their processors and their efficiency. The *GridSim* works only with the part of resources that was chosen by neural tree. To be able to compare the efficiency of all CPUs, we chose to use a simple measure unit which presents the number of floating point operations per second. Evaluation of every processor used is based on data taken from website of Geekbench benchmark [12]. We created web application tool on the base of *GridSim* for creating models of grid scheduling algorithm and evaluating these models in process of simulation. There are implemented PSO algorithms, too [33, 34, 35, 36, 37].

8. Simulations and Experimental Results

We have designed a neural tree on the base of probabilistic neural network and known structure of data. Structure of data is described in the previous part of this paper.

The training process of neural tree

Patterns were prepared from the data sets. Every pattern has 7 elements. The training set was presented to the root node and subsequently its parts to other created nodes. The tree was trained to divide the training set into three subsets that include some portion of numbers of CPUs, capacity of data storage and bandwidth. We used elements 2, 3 and 4 from every pattern (Fig. 4). These categories represent grid performance from the hardware point of view. We measured sum square error (Fig. 6) and mean square error of net as the stop criterion. After that we gained corresponding class 1, 2 or 3 of grid resources.

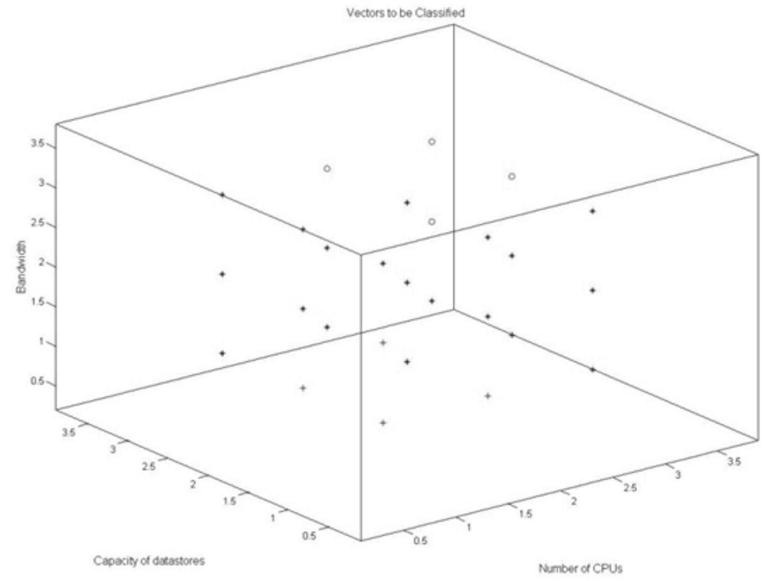

Fig. 4 *Part of input pattern.*

In a similar way, we trained a new neural tree for every class (1 − 3) of grid hardware resources with the goal to divide every class into new 3 classes from the software point of view. In such a way we reached nine classes of the grid resources. In the training phase of the second tree we used patterns elements that correspond with the operating system, compilers and application software. We used elements 5, 6 and 7 from every pattern. Output y_i of i-th linear neuron was computed by relation 4, and weights were updated by relation 6. These weights represent dividing planes between separated input subsets. One of the weight sets can be seen in Fig. 5. Colors of weights are changed according to their size.

The classification phase
Similarly as in training, in the phase of classification, unknown patterns were presented to the root node. Every pattern has 7 elements. The elements of patterns that were considered by passing over the tree again correspond with hardware characteristics (numbers of CPUs, capacity of data storage and bandwidth). We used elements 2, 3 and 4 from every pattern. The one of three classes was obtained by going through the tree from root to the leaf nodes. A second tree was used for classification patterns based on the software criteria (operating system, compilers and application software). We used elements 5, 6 and 7 from every pattern.

We repeated these processes 6 times for 500, 450, 400, 350, 300 and 250 input patterns in the training process.

There is relation between the numbers of patterns in the training process and accuracy of classification in Fig. 7, and additional information about the tree is in Tab. I.

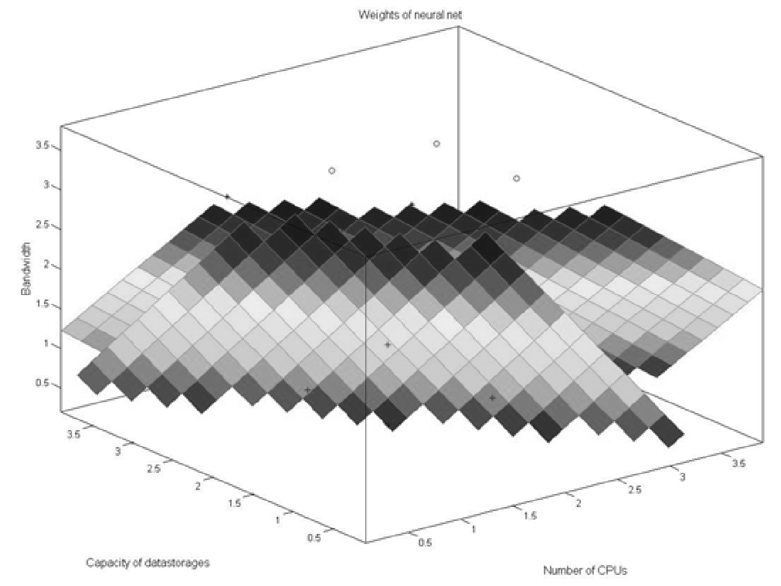

Fig. 5 *Evolution of weight in the process of training.*

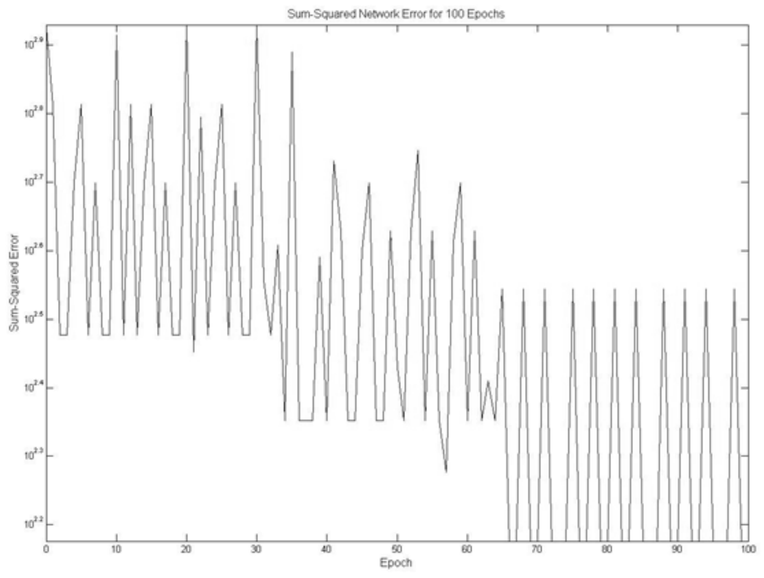

Fig. 6 *Performance of sum square error in the process of training tree.*

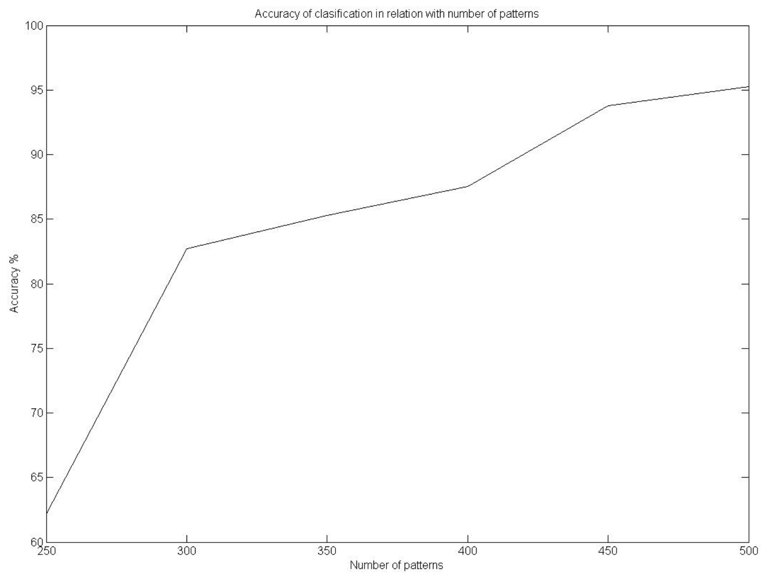

Fig. 7 *Relation between numbers of patterns in the training process and accuracy of classification.*

Number of patterns	Percentage average accuracy of classification	Depth of tree	Total nodes
500	95.25	8	143
450	93.76	8	135
400	87.51	7	124
350	85.27	7	108
300	75.73	5	92
250	62.14	4	35

Tab. I *Measured data after classification phase of the first neural tree.*

After using neural trees, we have separated grid resources into nine classes. The patterns that correspond with the nine classes have been implemented to the *GridSim* toolkit in *GridResource* class which describes characteristics of the resource and contains *ResourceCharacteristic.*

We executed a total of 6 series of experiments with different tasks (see Tab. II). The first and second series of tasks were aimed at scheduling a very large number of tasks. The first simulation input has 1001 – 5000 tasks with a large size of their length. The second batch consists of small tasks. Series 3 and 4 use 101 – 1000 tasks and series 5 and 6 are designed for 1 – 100 tasks. We assume a small diversity of length of tasks, which means that all task lengths can be distributed into an input interval. All tasks were generated randomly providing only minimum and maximum values of length, and file size. Other hardware or software parameters of tasks are associated to every series. On the base of all the values and parameters tasks are mapped only on the one class of grid resources we have gained from the neural trees.

Series	Number of tasks	Size of task	Type of input files size	Type of output files size
Series 1	1001 – 5000	10 001 – 100 000	1 – 3	1 – 3
Series 2	1001 – 5000	1 – 10 000	1 – 3	1 – 3
Series 3	101 – 1000	10 001 – 100 000	1 – 3	1 – 3
Series 4	101 – 1000	1 – 10 000	1 – 3	1 – 3
Series 5	1 – 100	10 001 – 100 000	1 – 3	1 – 3
Series 6	1 – 100	1 – 10 000	1 – 3	1 – 3

Tab. II *Main characteristics of tasks series.*

First, we applied the PSO scheduling algorithm 10 times in all series (see Tab. II) of experiments without using neural tree classification. All schedules were successfully created. Subsequently, we used PSO scheduling algorithm again 10 times in all series of experiments while using neural tree classification of grid resources. All schedules were successfully created again.

Experimental results of average time of creating schedule (ms) and average time of makespan of schedules (s) with and without neural network tree are in the table (see Tab. III). There is a comparison of three algorithms (PSO, Hill climbing and Round Robin) in Figs. 8 and 9. There are values of schedules makespan (s) in these figures. For this paper we take into account only simulation time of PSO optimization of scheduling. The best experimental case of schedule creating without using neural tree for series 1 is in the figure (see Fig. 8). The worst experimental case of schedule creating when using neural tree for series 1 is in figure (see Fig. 9).

We have used three different variants of this algorithm that differ in algorithm parameters. PSO-1 variant uses 10 particles and 300 iterations for series 5 and 6. Variant PSO-2 uses 20 particles and 300 iterations for series 3 and 4. Variant PSO-3 uses 30 particles and 500 iterations for series 1 and 2.

There is a comparison of average time of schedule creation without and with neural tree in figure (see Fig. 10). We can see comparison of time of makespan of schedules without and with using neural tree in the figure (see Fig. 11). There are vertical axes in logarithmic scale to base 10 in the figures (see Fig. 10 and Fig. 11). We have assumed improvement in scheduling in systems with large portions of tasks for high performance computing. This assumption was confirmed. The experiments show that there are also improvements in cases of middle and relatively small portions of tasks (up to 100).

The average improvement of the makespan criterion for schedules by using NNT with a large number of tasks, large task length (series 1) was 7.7 times, and improvement for a large number of relatively small tasks (series 2) was 3.33 times.

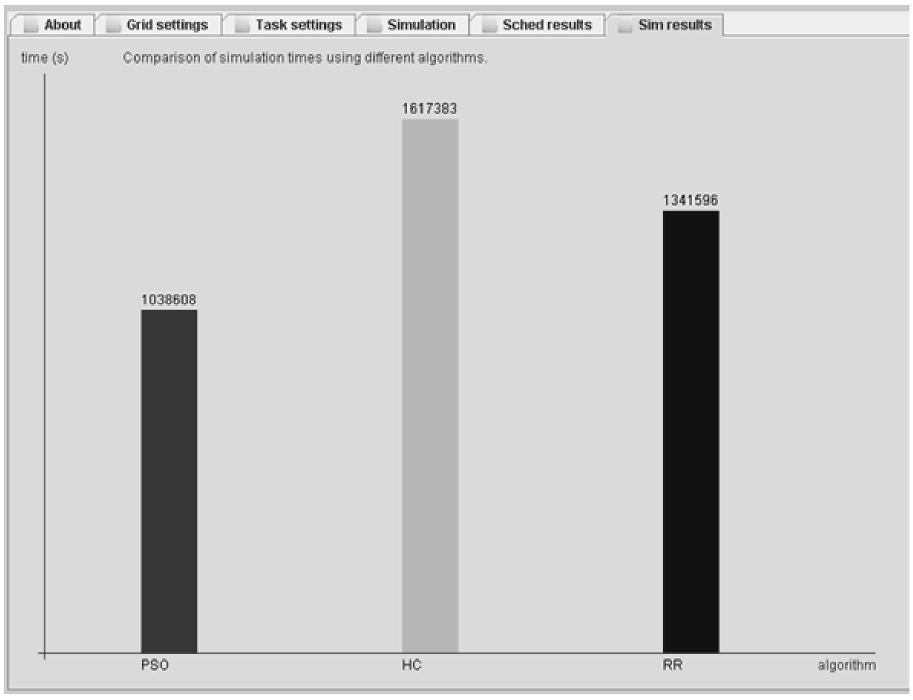

Fig. 8 *The best case of experimental results of makespan of schedules without using classification by NNT (s) from series 1.*

Series	Average number of tasks	Average size of task	Average time of creation schedule (ms)	Average time of creation schedule with using NNT (ms)	Average makespan (s)	Average makespan with using NNT (s)
Series 1	3582	50024	25015	4438	2287429	297228
Series 2	3380	5200	11828	2562	101720	30520
Series 3	426	48320	1579	683	92367	21178
Series 4	683	5782	781	831	9662	1042
Series 5	62	62040	159	104	9005	3156
Series 6	49	3560	94	77	1041	251

Tab. III *Experimental results.*

Improvements for series 3 and 4 that use 101 – 1000 tasks were 4.26 and 9.27 times, respectively, and for series 5 and 6 that are designed for 1 – 100 tasks were 2.85 and 4.15 times, respectively.

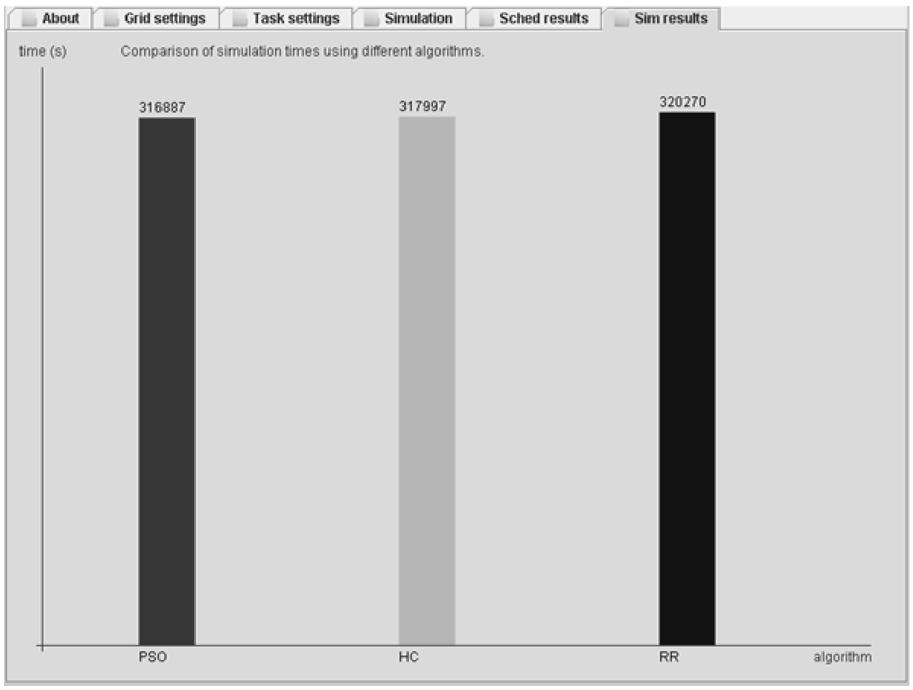

Fig. 9 *The worst case of experimental results of makespan of schedules with using classification by NNT (s) from series 1.*

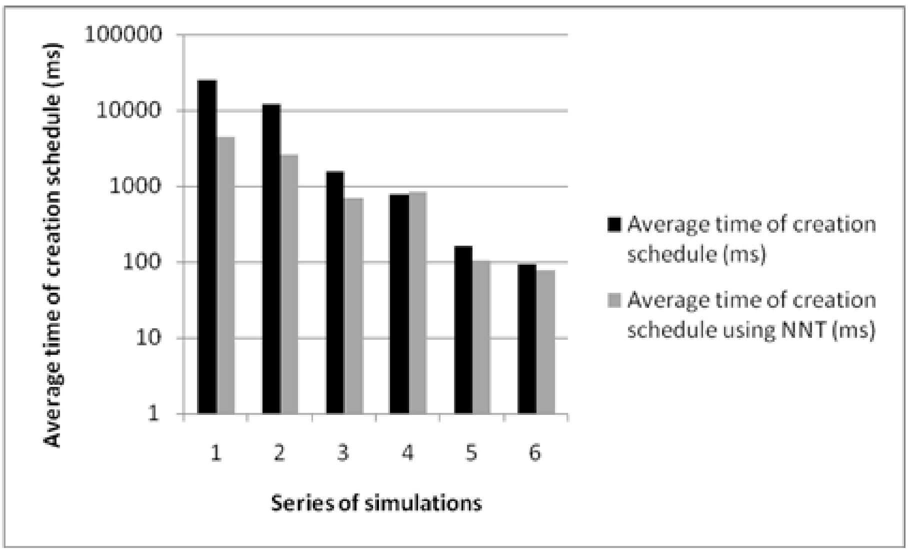

Fig. 10 *Comparison of average time of schedule creation without and with NNT.*

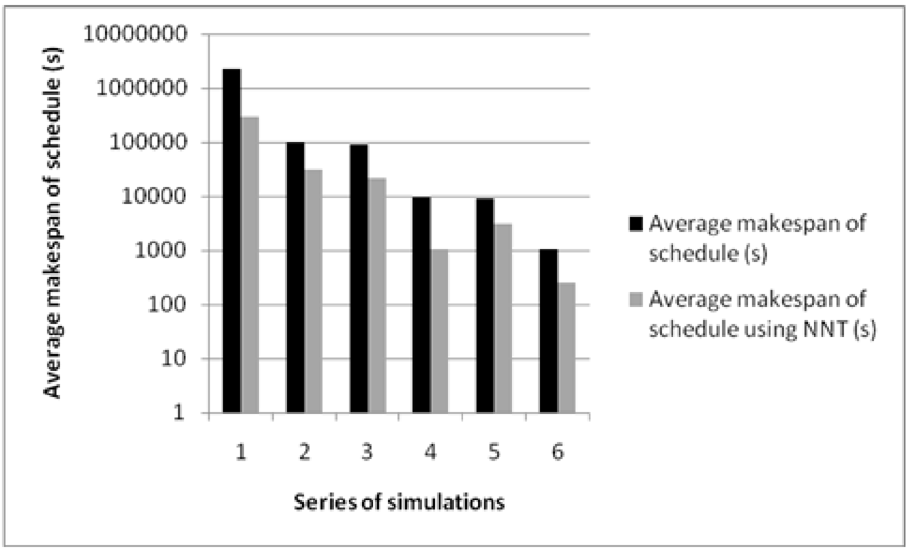

Fig. 11 *Comparison of time of makespan of schedules without and with NNT.*

9. Conclusion

Job scheduling is computationally hard. It has been shown that the problem of finding optimal scheduling in heterogeneous systems is in general NP-hard.

This paper proposes a model of neural tree architecture with probabilistic neurons. These trees are used for classification of a large amount of computer grid resources to classes. The first tree is used for classification hardware part of dataset. The second tree classifies patterns of software identifiers. The trees are implemented to successfully separate inputs into nine classes of resources. We proposed Particle Swarm Optimization algorithm for tasks scheduling in computer grid. We compared time of creation of schedule and time of makespan for each of six series of experiments without and with NNT. In the experiments using NNT we gained the subset of suitable computational resources. The aim was effective mapping of large tasks into particular resources. On the base of our experiments we can say that improvements have been made even for middle and small batch of tasks. Average improvement for each used structure batch of tasks is 5.28 when using NNT for classification of resources against the experiments without neural tree.

In future works we plan to apply wave probabilities [27, 28], Heisenberg's uncertainty limit [29] or information circuits [30] in our neural net models to speed up classification or to eliminate indefinites.

Acknowledgement

This work is partially supported by the High Performance Computing Centre of Matej Bel University in Banská Bystrica, Slovakia (HPCC UMB).

References

[1] Xhafa F., Abraham A.: Computational models and heuristic methods for Grid scheduling problems. Future Generation Computer Systems, 26, 2010, pp. 608–621.

[2] Rummelhardt D. E., Hinton G. E., Williams R. J.: Learning Representations by Back-propagation error. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, MA: MIT Press, **1**, 1986, pp. 318–362.

[3] Liu H., Abraham A., Hassainen A.: Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Generation Computer Systems, 2009.

[4] Kennedy J., Eberhart R. C., Shi Y.: Swarm intelligence, Morgan Kaufman, San Francisco.

[5] Li L. J., Huang Z. B., Liu F.: A heuristic particle swarm optimizer (HPSO) for optimization of pin connected structures. Computing Structure, 85, 2007, pp. 340–349.

[6] Škrinárová J., Melicherčík M.: Measuring concurrency of parallel algorithms. 1st International Conference on Information Technology Gdansk: IEEE computer society, IEEE Catalog Number: CFP0825E-PRT, 2008, pp. 289–292, ISBN 978-1-4244-2244-9.

[7] Parsopoulos K. E., Vrahatis M. N.: Recent approaches to global optimization problems through particle swarm optimization. Natural Computing, 12, 2002, pp. 235–306.

[8] Shi Y., Eberhart R. C.: A modified particle swarm optimizer. Proceedings of IEEE congress on evolutionary computation, 1997, pp. 303–308.

[9] Li L. J., Huang Zy B., Liu F.: A heuristic particle swarm optimization method for truss structures with discrete variables. Computers & Structures, **87**, 2004, pp. 435–443.

[10] He S., Wu Q. H., Wen J. Y.: A particle swarm optimizer with passive congregation. Biosystems, 2004, 78, pp, 135–47.

[11] Buyya R., Murshed M. M.: GridSim: a toolkit for the modelling and simulation of distributed resource management and scheduling for Grid computing. Concurrency and Computation: Practice and Experience, 2002, pp. 1175–1220.

[12] Geekbench, Geekbench result browser 2011 http://browse.geekbench.ca/, 2010.[Online; accessed 19–March–2012].

[13] Škrinárová J., Krnáč M.: E-learning course for scheduling of computer grid, International Conference Interactive Collaborative Learning ICL 2011. Wien : International Associaton of Online Engineering, IEEE Computer Society, 2011, pp. 352–356, ISBN 978-1-4577-1746-8.

[14] Luger G. F.: Artificial Intelligence: Structures and Strategies for Complex Problem Solving, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2001.

[15] Haykin S.: Neural Networks, a comprehensive foundation. Prentice Hall, 1994.

[16] Micheloni Ch., Rani A., Kumar S., Foresti G. L.: A balanced neural tree for pattern classification. Neural Networks, 2012, 27, pp. 81–90.

[17] Martincová P., Grondžák K.: Effective distributed query processing on the grid. In: 2nd International Workshop on Grid Computing for Complex Problems – GCCP 2006. November 2006 IISAS Bratislava, Slovakia, pp. 129–135, ISBN 978-80-969202-6-6.

[18] Kołodziej J., Xhafa F: Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population. In: Future Generation Computer Systems, **27**, 8, October 2011, pp. 1035–1046.

[19] Lim D., Ong Y.-S., Jin Y.: Efficient hierarchical parallel genetic algorithms using grid computing, Future Generation Computer Systems, **23**, 4, 2007, pp. 658–670.

[20] Ritchie G., Levine J.: A fast effective local search for scheduling independent jobs in heterogeneous computing environments. Tech. Rep., Centre for Intelligent Systems and their Applications, University of Edinburgh, 2003.

[21] Xhafa F., Gonzalez J. A., Dahal K. P., Abraham A.: A GA(TS) hybrid algorithm for scheduling in computational grids. HAIS, 2009, pp. 285–292.

[22] Liu H., Abraham A., Hassanien A. E.: Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm. Future Generation Computer Systems, **26**, 8, 2010, pp.1336–1343.

[23] Kalantari M., Akbari M. K.: A parallel solution for scheduling of real time applications on grid environments. Future Generation Computer Systems, **25**, 7, 2009, pp. 704–716.

[24] Buyya R.: Economic-based distributed resource management and scheduling for grid computing, Ph.D. Thesis, Monash University, Australia, 2002.

[25] Schaefer R., Kołodziej J.: Genetic search reinforced by the population hierarchy, Foundations of Genetic Algorithms VII, Morgan, 2003, pp. 369–385.

[26] Yarkhan A., Dongarra J.: Experiments with scheduling using simulated annealing in a grid environment. In: Proc. of the 3rd International Workshop on Grid Computing, 2002, pp. 232–242.

[27] Svítek M.: Wave probabilities and quantum entanglement. Neural Network World, **18**, 5, 2008, pp. 401–406, ISSN 1210-0552.

[28] Svítek M.: Wave Probabilistic Models. Neural Network World. **17**, 5, 2007, pp. 469–481, ISSN 1210–0552.

[29] Svítek M.: Investigation to Heisenberg's uncertainty limit. Neural Network World, **8**, 6, 2008, pp. 489–498, ISSN 1210–0552.

[30] Svítek M., Votruba Z., Moos P.: Towards Information Circuits, Neural Network World, **20**, 2, 2010, pp. 241–247, ISSN 1210–0552.

[31] Huraj L., Reiser H.: VO Intersection Trust in Ad hoc Grid Environments. In: Fifth International Conference on Networking and Services (ICNS 2009), Valencia, Spain, IEEE Computer Society, April 2009, pp. 456–461.

[32] Huraj L., Siládi V., Škrinárová J.: Towards a VO Intersection Trust model for Ad hoc Grid environment: Design and simulation results. IAENG International Journal of Computer Science 2013, ISSN: 1819–9224.

[33] Škrinárová J., Krnáč M.: Particle Swarm Optimization Model for Grid Scheduling. Second International Conference on Computer Modelling and Simulation, CSSIM 2011. Brno, Czech Republic, September 5-7, 2011. Brno: Brno University of Technology, pp. 146–153, ISBN 978-80-214-4320-4.

[34] Škrinárová J., Krnáč M.: Particle Swarm Optimization for Grid Scheduling. Informatics 2011: Eleventh International Conference on Informatics, Rožňava, November, 2011 Košice: Faculty of Electrical Engineering and Informatics of the Technical University, 2011, pp. 153-158, ISBN 978-80-89284-94-8.

[35] Škrinárová J., Zelinka F.: The grid scheduling. GCCP 2010 6th International Workshop on Grid Computing for Complex Problems, Bratislava, November, 2010. Bratislava: II SAS, pp. 100–107, ISBN 978-80-970145-3-7.

[36] Škrinárová J.: TSP Models for cluster computing on the base of genetic algorithm. 5th International Workshop on Grid Computing for Complex Problems. GCCP 2009. October, 2009. Bratislava: II SAS, pp. 110–117, ISBN 978-80-970145-1-3.

[37] Škrinárová J., Zelinka F.: TSP Model of genetic algorithm on the base of Cluster computing. Informatics 2009 Tenth International Conference on Informatics, Herlany, November, 2009. Košice: Technical University of Košice, 2009, pp. 326–331, ISBN 978-80-8086-126-1.

[38] Martincová P.: Performance of simulated Grid scheduling algorithms. Journal of Information, Control and Management Systems, **5**, 2/1, 2007, pp. 261–270, ISSN 1336-1716.

[39] Martincová P.: Global scheduling on the grid. InfoTech-2008 Proceedings of the International Conference on Information Technologies. September 2008, Varna St. Constantine and Elena resort, Bulgaria. Sofia: Technical University, 2008, pp. 75–82, ISBN 978-954-9518-56-6.