Chapter 6

Scheduling in Manufacturing Systems – Ant Colony Approach

Mieczysław Drabowski and Edward Wantuch

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/51487

1. Introduction

Scheduling problems, also in manufacturing systems [4], are described by following parameters: the processing – computing – environments comprising processor (machines) set, other resources comprising transportations and executions devices, processes (tasks) set and optimality criterion. We assume that processor set consists of m elements. Two classes of processors can be distinguished: dedicated (specialized) processors and parallel processors.

In production systems machines are regarded as dedicated rather than as parallel. In such a case we distinguish three types of dedicated processor systems: flow-shop, open-shop and job-shop. In the flow-shop all tasks have the same number of operations which are performed sequentially and require the same sets of processors. In the open-shop the order among the operations is immaterial. For the job-shop, the sequence of operations and the sets of required processors are defined for each process separately.

In the case of parallel processors each processor can execute any task. Hence, a task requires some number of arbitrary processors. As in deterministic scheduling theory [12] parallel processors are divided into three classes: identical processors – provided that all tasks are executed on all processors with that same productivity, uniform processors – if the productivity depends on the processor and on the task, and unrelated processors – for which execution speed depends on the processor and on the task. In each of the above cases productivity of the processor can be determined.

Apart from the processors the can be also a set of additional resources, each available in m_i units.

The second parameter of the scheduling problem is the tasks system. The tasks correspond to the applications for manufactured goods. We assume that the set of tasks consists of n

tasks. For the whole tasks system it is possible to determine such feature as preemptability (or nonpreemptability) and existence (or nonexistence) of precedence constraints.

Precedence constraints are represented as directed acyclic graphs (DAGs). Each task separately is described by a number of parameters. We enumerate tem in the following:

- Number of operations,
- Execution time,
- Ready time,
- Deadline,
- Resources requirements,
- Weight.

The optimality criteria constituting the third element of the scheduling problem are:

- Schedule length,
- Maximum lateness,
- Mean flow time,
- Mean tardiness.

Due to the fact that scheduling problems and their optimizations are general NP-complete [10,25] we suggest meta-heuristic approach: Ant Colony Optimization and its comparison with neural method and with polynomial algorithms for certain exemplary problems of task scheduling.

If a heuristic algorithm (such as ACO) finds an optimal solution to polynomial problems, it is probable that solutions found for NP-complete problems will also be optimal or least approximated to optimal. ACO algorithm was tested with known polynomial algorithms and all of them achieved optimal solutions for those problems.

The comparisons utilized such polynomial algorithms as [3,5,12]:

- Coffman-Graham Algorithm,
- Hu Algorithm,
- Baer Algorithm,

For non-polynomial problems of tasks scheduling ACO algorithm was tested with list algorithms [12] (HLFET, HLFNET, SCFET, SCFNET), with PDF/HIS [18] for STG tasks and neural approach [22].

http://dx.doi.org/10.5772/51487

2. Adaptation of ACO to solve the problems of scheduling

The Ant Colony Optimization (ACO) algorithm [2] is a heuristics using the idea of agents (here: ants) imitating their real behavior. Basing on specific information (distance, amount of pheromone on the paths, etc.) ants evaluate the quality of paths and choose between them with some random probability (the better path quality, the higher probability it represents). Having walked the whole path from the source to destination, ants learn from each other by leaving a layer of pheromone on the path. Its amount depends on the quality of solution chosen by agent: the better solution, the bigger amount of pheromone is being left. The pheromone is then "vapouring" to enable the change of path chosen by ants and let them ignore the worse (more distant from targets) paths, which they were walking earlier.

The result of such algorithm functioning is not only finding the solution. Very often it is the trace, which led us to this solution. It lets us analyze not only a single solution, but also permutations generating different solutions, but for our problems basing on the same division (i.e. tasks are scheduled in different order, although they are still allocated to the same processors). This kind of approach is used for solving the problems of synthesis, where not only the division of tasks is important, but also their sequence.

To adapt the ACO algorithm [24] to scheduling problems, the following parameters have been defined:

- Number of agents (ants) in the colony;
- Vapouring factor of pheromone (from the range (0; 1));

The process of choosing these parameters is important and should consider that:

- For too big number of agents, the individual cycle of algorithm can last quite long, and the values saved in the table ("levels of pheromone") as a result of addition will determine relatively weak solutions.
- On the other hand, when the number of agents is too small, most of paths will not be covered and as a result, the best solution can long be uncovered.

The situation is similar for the vapouring factor:

- Too small value will cause that ants will quickly "forget" good solutions and as a result it can quickly come to so called *stagnation* (the algorithm will stop at one solution, which doesn't have to be the best one).
- Too big value of this factor will make ants don't stop analyze "weak" solutions; furthermore, the new solutions may not be pushed, if time, which has passed since the last solution found will be long enough (it is the values of pheromone saved in the table will be too big).

The ACO algorithm defines two more parameters, which let you balance between:

- α – the amount of pheromone on the path;

- β – "quality" of the next step;

These parameters are chosen for specific task. This way, for parameters:

- $\alpha > \beta$ there is bigger influence on the choice of path, which is more often exploited,
- $\alpha < \beta$ there is bigger influence on the choice of path, which offers better solution,
- $\alpha = \beta$ there is balanced dependency between quality of the path and degree of its exploitation,
- $\alpha = 0$ there is a heuristics based only on the quality of passage between consecutive points (ignorance of the level of pheromone on the path),
- $\beta = 0$ there is a heuristics based only on the amount of pheromone (it is the factor of path attendance),
- $\alpha = \beta = 0$ we'll get the algorithm making division evenly and independently of the amount of pheromone or the quality of solution.

Having given the set of neighborhood N of the given point i, amount of pheromone on the path τ and the quality of passage from point i to point j as an element of the table η you can present the probability of passage from point i to j as [6,7]:

$$p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}} & When\, j \in N_i^k \\ 0 & Else \end{cases}$$

Formula 1. Evaluation of the quality of the next step in the ACO algorithm

In the approach presented here, the ACO algorithm uses agents to find three pieces of information:

- the best / the most beneficial division of tasks between processors,
- the best sequence of tasks,
- searching for the best possible solution for the given distribution.

Agents (ants) are searching for the solutions which are the collection resulting from the first two targets (they give the unique solution as a result). After scheduling, agents fill in two tables:

- two-dimensional table representing allocation of task to the given processor,
- one-dimensional table representing the sequence of running the tasks.

http://dx.doi.org/10.5772/51487

The process of agent involves:

1. collecting information (from the tables of allocation) concerning allocation of tasks to resources and running the tasks;
2. drawing the next available task with the probability specified in the table of task running sequence;
3. drawing resources (processor) with the probability specified in the table of allocation the tasks to resources;
4. is it the last task?

To evaluate the quality of allocation the task to processor, the following method is being used:

1. evaluation of current (incomplete) scheduling;
2. allocation of task to the next of available resources;
3. evaluation of the sequence obtained;
4. release the task;
5. was it the last of available resources?

The calculative complexity of single agent is polynomial and depends on the number of tasks, resources and times of tasks beginning.

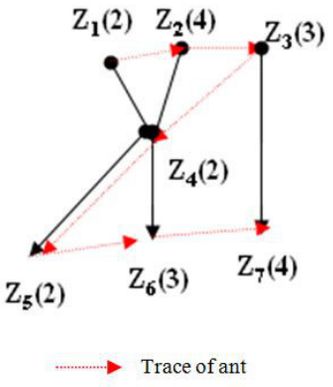
Idea of algorithm:

Algorithm:

1. Construct *G* – structure of tasks non allocation and *S* – structure of tasks, which may be allocation in next step (for ex ample: begin: G = {Z_1, Z_2,…, Z_7} and S = {Z_1, Z_2, Z_3}); update range of pheromone and consideration of vapouring factor;
2. With *S* select of tasks with the most strong of trace;
3. Allocate available of task as soon as possible and in accordance with precedence constraints;
4. Remove selected of task with *G* and *S* and to add to list of tasks in memory of ant;
5. Update range of pheromone and remain of trace;
6. If $G = \emptyset$ END of algorithm;
7. Go to 1;

Example:

Two identical processors, digraph of seven tasks $Z_i (t_i)$, where t_i = time execution.



Solution – Gantt graph

Parameters of ants' colony have been selected through experiments. Algorithm tuning is to select possibly best parameter values. This process demands many experiments which are conducted for different combinations of parameter values. For each combination of variable values, computation process has been repeated many times, and then an average result has been calculated. The same graphs type of STG, like at previous algorithms, have been applied [18,20].

Selected algorithm parameters:

- a – number of ants; for number of tasks n < 50, a = 75 and for n>= 50, a = 1,5 x n
- γ – the pheromone evaporation coefficient = 0,08.

3. Adaptation of neural method to solve the problems of scheduling

3.1. Neural network model

The starting point for defining the neural network model for solving the problems of task scheduling and resource allocation are the assumptions for the constraint satisfaction problem (CSP) [36,37]. CSP is the optimization problem which contains a certain set of variables, sets of their possible values and constraints forced on the values of these variables [14,15]. On the basis of this problem assumption a network model of the following features is suggested:

- A neural network consists of components; each of them corresponds to another variable.

http://dx.doi.org/10.5772/51487

- Each component contains such number of neurons which equals the number of possible values of each variable.
- Assigning a specified value to a variable is the process of switching on a relevant neuron (neurons) and switching off the remaining ones in the component corresponding to this variable.
- Switching on a neuron means assigning the value *"1"* to its output.
- Switching off a neuron means assigning the *"0"* to its output.
- Constraints to the network are introduced by adding a negative weight connection between neurons *('-1'*), symbolizing the variable values that cannot occur simultaneously.
- In the network there are additional neurons "the ones" that are switched on.

Each neuron has its own table of connections and each connection contains its weight and the indicator for the connected neuron. A characteristic feature of the network is the diversity of connections between neurons, but these never applied to all neurons [22,23]. It is a consequence of the fact that connections between neurons exist only when some constraints are imposed. The constraints existing in the discussed network model may be of the following types: resources, time, order.

The method of constraints implementation shall be discussed upon examples [22].

Example 1:

Such net (Fig. 1.) blocks solution, in which $Z_1 = 1$ as well as $Z_2 = 2$ or $Z_3 = 3$ as well as $Z_4 = 2$.

Example 2:

Let us have two operations with unit execution times. The operation Z_1 arrives at the system in time t = 1 and it is to be executed before the expiry of time t = 4. The operation Z_2 arrives in time t = 1 and may be executed after the completion of operation Z_1. A fragment of the net for his case including all the connections is shown by Fig. 2.

Neuron „one" ('1') – a special neuron switched on permanently – is responsible for time constraints. Introducing connections between such neuron and the relevant network neurons excludes a possibility of switching them on when searching for the solution. Task Z_1 cannot be scheduled in moment *0* and moment *4*, which corresponds to the assumption that this task arrives at the system at moment *1* and must be performed before moment *4*. Analogical process applies to operation Z_2. The sequence constraints are executed by the connections between the network neurons. The figure shows (with dotted line) all the connections making the performance of task Z_2 before task Z_1 impossible.

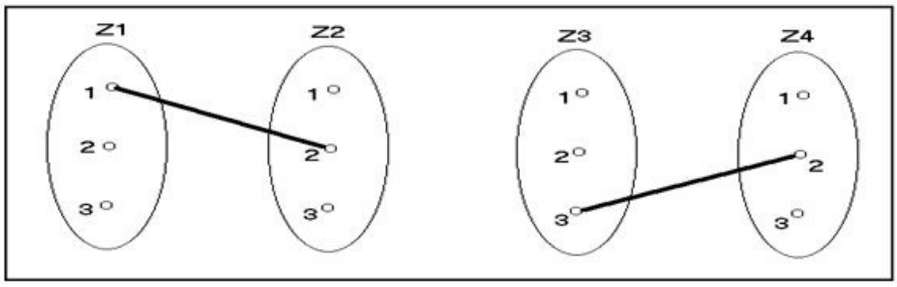


Figure 1. The example 1 of constraints.

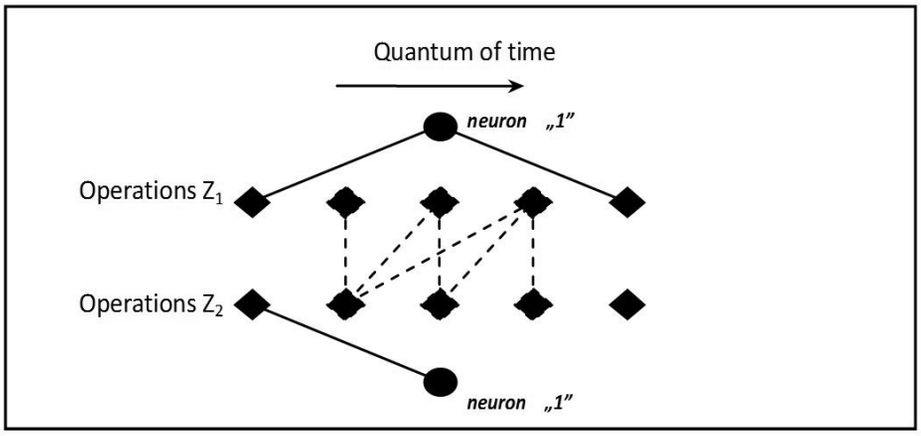


Figure 2. The example 2 of constraints.

3.2. The algorithm description

After entering the input data (the system specification), the algorithm constructs a neural network, the structure of which and the number of neurons composing it, depend upon the size and complexity of the instance of problem. We will name the part of the net allocated to this task – an area.

Constraints are introduced to the network by the execution of connections, occurring only between the neurons corresponding to the values of variables which cannot occur simultaneously.

The operation of the algorithm is the process of switching on appropriate neurons in each domain of network in order to satisfy the constraints imposed by the input data.

The algorithm course is as follows [36,38]:

1. *Allocating random values to consecutive variables.*
2. *Network relaxation:*

http://dx.doi.org/10.5772/51487

3. *Calculating the weighted sum of all neurons inputs.*
4. *Switching on the neuron with the highest input value.*
5. *Return to relaxation or – if there are no changes – exit from relaxation.*
6. *If there are connections (constraints) between the neurons that are switched on, each weight between two switched on neurons is decreased by 1 and there is a return to relaxation.*

The algorithm starts from allocating weight *'-1'* to all connections and then the start solution is generated. It is created by giving random values to the subsequent variables. This process takes place in a certain way: for each task i.e. in each area of the net such number of neurons is switched on as it is necessary for a certain task to be completed. The remaining, in the part which is responsible for its performance, neurons are being switched off. In the obtained result there are many contradictions, specified by switching on the neurons where the connections exist.

Therefore, the next step of the algorithm is the relaxation process, the objective of which is to "satisfy" the maximum numbers of limitations (backtracking). The objective is to obtain the result where the number of situations, where two switched on neurons of negative weight connection between them is the lowest. While switching on neurons with the biggest value at the start, in each area three instances may happen:

- If there is one neuron of the biggest value in the area, it is switched *on*; the remaining ones are switched *off*.
- If there are more neurons, among which there is a previously switched one, there is no change and it remains switched *on*.
- If there are more neurons, but there is no-one previously switched on, one of them is switched on randomly, the remaining ones are switched *off*.

A relaxation process finishes when the subsequent step does not bring any change and if all the requirements are met – the neurons between which a connection exist are not switched on – the right solution is found. If it is not still the case, it means that the algorithm found the local minimum and then the weight of each connection between two switched on neurons is decreased by *"1"* while its absolute value is being increased. It causes an increase in "interaction force' of this constraint which decreases the chance of switching *on* the same neurons in a relaxation process where we return in order to find the right solution.

After a certain number of iterations the network should consider all the constraints – providing that there is the right solution, it should be found. Another factor is worth pointing out: in a relaxation process such an instance may occur where changes always happen. Then, this process might never be completed. Then a problem is solved in such a way that relaxation is interrupted after a certain number of calls.

Search for a solution by algorithm consists of two stages. At the first one, which is described by the above presented algorithm, some activities are performed which lead to finding the right solution for the given specification. After finding such a solution, in consequence of

purpose function optimization there is a change of values for a certain criterion – in this case, decrease – then, the subsequent search for the right solution occur. In this case the search aims at a solution which possesses bigger constraints as the criteria value is sharper. Two criteria are taken into consideration for which a solution is being searched. It may be a cost function – where at the given time criterion, we search for the cheapest solution, or time function – where at the given cost criterion, we search for the quickest solution. Thus, the run of the algorithm is to seek a solution for smaller and smaller value of a selected criterion. However, if the algorithm cannot find the right solution for the recently modified criteria value of the algorithm, it returns to the previous criteria value for which it has found the right solution and modifies it by a smaller value.

For instance, if an algorithm has found the right solution for cost criterion which is e.g. 10, and it cannot find it for cost criteria which are 9, it tries to find a solution for cost 9.5 etc. In this way the program never finishes work, but all the time it tries to find a better solution in sense of a certain criterion. The user/designer of the system can interrupt its work at any moment if he/she considers the current solution given by an algorithm to be satisfying.

In case of time criterion minimization, optimization goes at two planes. At the first one, subsequent neurons of the right side in task part of the network are connected to the neurons *"one"*, in this way fewer and fewer quanta is available for the algorithm of task scheduling which causes moving a critical line to the left and at the same time its diminishing. However, at the second, an individual quantum of time is being diminished; at each step an individual neuron will mean a smaller and smaller time passage.

The task part:

Each area corresponds to one task (Fig. 3.). For further area, the best possible setting for the task is selected. Which setting 'wins' at the given stage and in the given area – this shall be determined by the sum of neuron outputs in the setting, i.e. the one that introduces the smaller number of contradictions. Moreover, it is checked if among the found set of the best solutions there is no previous one, then it is left.

A neuron at the *[i, k]* position corresponds to the presence of '*i*' task on the processor at the '*k*' moment. Between these neurons there are suitable inhibitory connections *(-1.0).*

If, for example, task 1 must be performed before task 2, for all the neuron pairs

[1, k], [2, m] there are inhibitory connections (denoting contradictions), if $k >= m$ and if task 8 occurs in the system at moment 2, *„one"* neuron is permanently connected to neurons *[8, 0] and [8, 1]* (neuron which has 1.0 at the start which is permanently contradictory) and guarantees that in the final solution there is no quantum at moment 0 or 1.

We also take critical lines into account, which stand for time constraints that cannot be exceeded by any allocated tasks – connecting 'one' will apply to the neurons of the right side of the network outside the critical line.

http://dx.doi.org/10.5772/51487

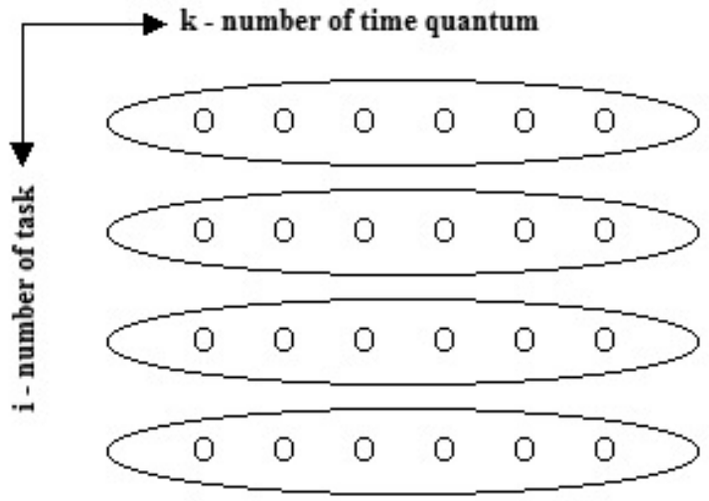


Figure 3. The task part for scheduling problems.

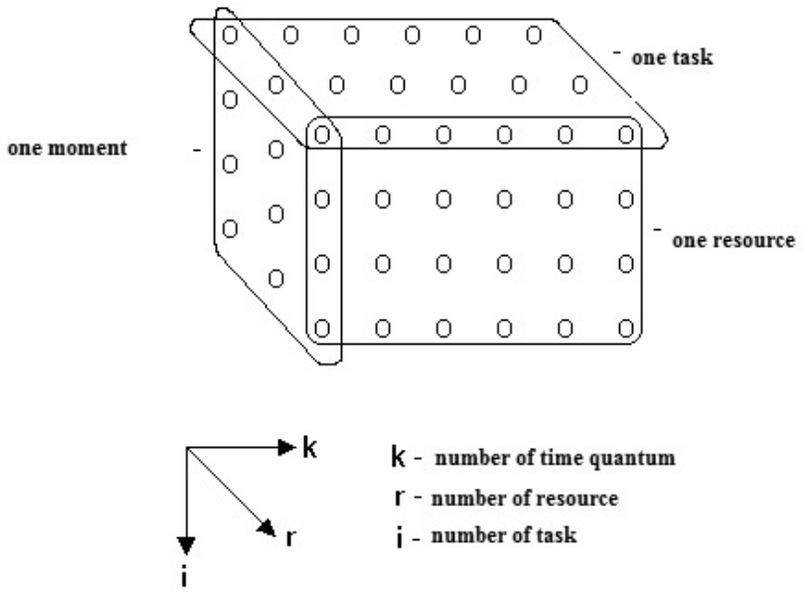


Figure 4. The resource part for scheduling problems.

The resource part:

Before selecting the quanta positions in the areas, algorithm has to calculate inputs for all the neurons. The neurons of the resource part are also connected to these inputs, as the number and the remaining places in recources have an impact on the setting which is going to "win" at a certain stage of computation. Thus, before an algorithm sets an exact task, it calculates the value of neuron inputs in resource part. The [r, i, k] neuron is switched on if at 'k' moment the resource 'r' is overloaded (too many tasks are using t), or it is not overload-

ed, but setting the task of part 'i' at the moment defined by *'k'* would result in overloading. Neurons in resource part (Fig. 4.) respond by their possible connection, resource overloaded, if part of the task were set and at moment *'k'*; therefore, neurons of resource part are connected to task inputs.

When in the resource part the neuron *" 'r' resource overload'* is switched *on*, as task *'i'* is set at moment *'k' "*, its signal *(1.0)* is transferred by weight (-1.0) to the neuron existing in the task part, which causes the negative input impulse (-1.0 * 1.0) at the input which results in a contradiction.

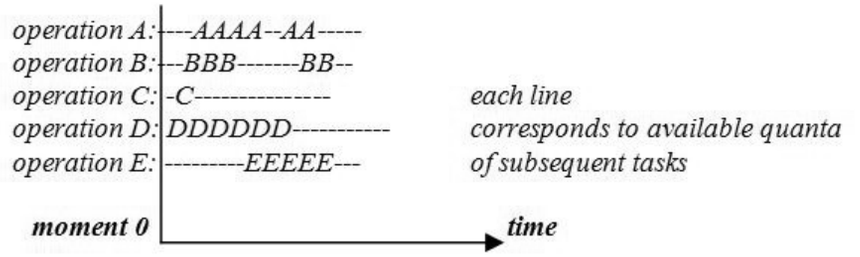
In other words – it "disturbs" function *'compute_in'* to set the task and at moment "k". Thus, in the network there are subsequent illegal situations implemented (constraints).

Each neuron *[r, i, k]* of the resource part is connected with neuron *[i, k]* from the task part, so a possibility of task existence at a given moment with concurrent resource overloading is 'inhibited'.

Example:

Let us assume that there are five operations *A, B, C, D, E.*

Task part works as follows (a letter means a neuron switched on, sign '-' means a switched off neuron):



These operations should be allocated to a certain number of processors, so that one only operation would be performed on one processor at an exact moment:

1. Algorithm allocates (at moment 0) fragment DDDDDD, adds a new processor (the first) and allocates on it:

   DDDDDD----------

2. Allocation -C: for this moment (1) there is no place on the first processor, so algorithm adds the next processor and allocates an operation:

   DDDDDD----------

   -C---------------

3. Allocation BBB: there is place on the second processor:

   DDDDDD----------

   -C-BBB----------

http://dx.doi.org/10.5772/51487

4. Allocation AAAA: there is no place at quantum 4 –algorithm adds the third processor and allocates:

DDDDDD----------

-C-BBB----------

----AAAA---------

5. Allocation EEEEE: there is place on the first processor :

DDDDDD---EEEEE---

-C-BBB----------

----AAAA---------

6. Allocation AA there is place on the second processor

DDDDDD---EEEEE---

-C-BBB----AA------

----AAAA---------

7. Allocation BB: there is place on the second processor:

DDDDDD---EEEEE---

-C-BBB----AA-BB—

----AAAA---------

The result of the operations on the processors is as follows:

P1:DDDDDD---EEEEE---

P2:-C-BBB----AA-BB--

P3: ----AAAA---------

Computational complexity of neural algorithm for task scheduling

An algorithm gives the right solution for the problems of known multi-nominal algorithms and also may be used for problems NP-complete. The complexity of one computation step may be estimated as follows:

$$i * \left(1 + p * k + k * \left(1 + k * p\right) * m + k * r * i * p\right) + p \quad (2)$$

Where:

i – Number of tasks.

p – Number of processors.

k – Number of time quanta.

m – Number of all consecutive depend abilities between tasks.

r – Number of resources.

The largest complexity is generated by the process of increasing the number of tasks and an increase in the number of time quanta. Also, maximum number of processors and number of constitutive depend abilities in the introduced graph have a powerful effect on computation. It is a pessimistic estimation; in practice, real complexity may be slightly smaller, but proportional to that. An algorithm itself is convergent i.e. step by step generates better and better solutions.

4. Tests of task scheduling algorithms

4.1. The comparison with polynomial algorithms

To show convergence of ACO algorithm towards optimum, one can compare their results with optimal results of already existing, precise, polynomial algorithms for certain exemplary problems of task scheduling. If a heuristic algorithm finds an optimal solution to polynomial problems, it is probable that solutions found for NP-complete problems will also be optimal or at least approximated to optimal. Heuristic algorithm described herein was tested with known polynomial algorithms and all of them achieved optimal solutions for those problems. The comparisons utilized such polynomial algorithms as:

- Coffman – Graham Algorithm,
- Hu Algorithm,
- Baer Algorithm,

Comparisons of ACO solutions with selected precise polynomial algorithms will be presented as an example.

Coffman and Graham algorithm

Scheduling of tasks which constitute a discretionary graph with singular performance times on two identical processors in order to minimize $C_{max.}$ Calculation complexity of the algorithm is $O(n^2)$.

Test problem no 1:

- 2 identical processors (a), 3 identical processors (b).
- 15 tasks with singular performance times.
- Graph with tasks:

http://dx.doi.org/10.5772/51487

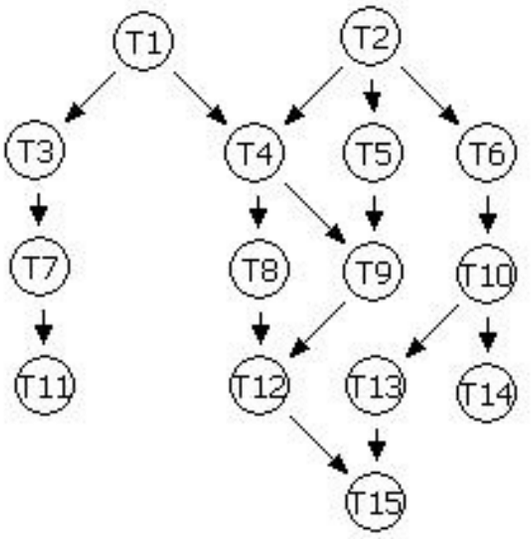


Figure 5. Graph of tasks used for the comparison of ACO algorithm with Coffman and Graham algorithm (test problem no 1).

- Optimal scheduling for two processors obtained as a result of Coffman and Graham algorithm use (1a).

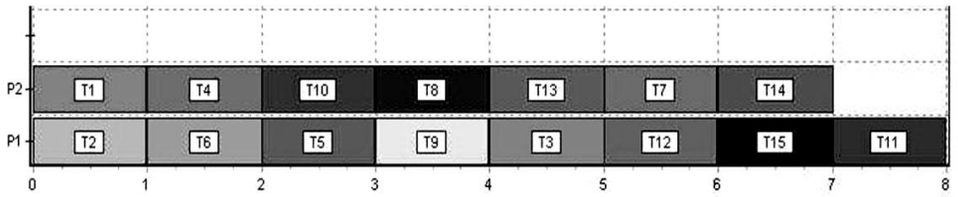


Figure 6. Optimal scheduling for two processors - Coffman and Graham algorithm (1a).

- Optimal scheduling for two processors obtained as a result of ACO algorithm use (1a).

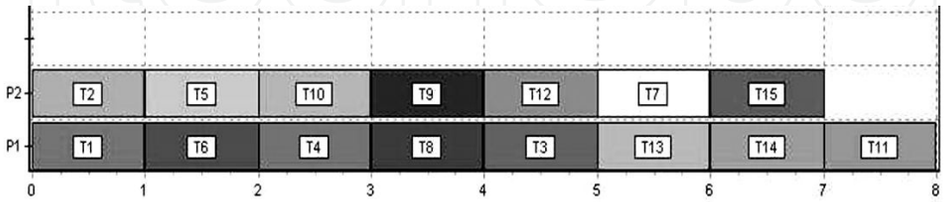


Figure 7. Optimal scheduling for two processors - ACO algorithm (1a).

- Scheduling for three processors obtained as a result of Coffman and Graham algorithm use (1b).

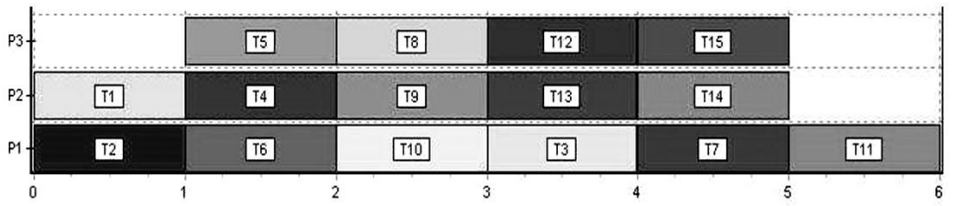


Figure 8. Problem scheduling for 3 processors - Coffman and Graham algorithm use (1b).

- Scheduling for three processors obtained as a result of ACO algorithm use (1b).

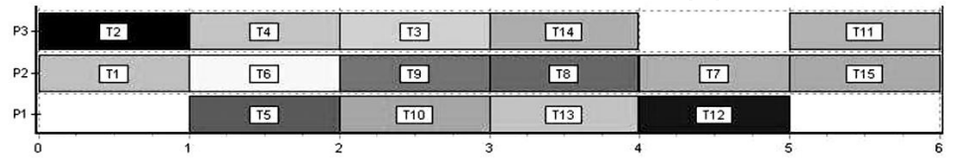


Figure 9. Optimal problem scheduling for 3 processors – ACO algorithm (1b).

For two processors (1a) ACO algorithm identical to Coffman and Graham algorithm obtained optimal scheduling. It was the same in the case of three processors (1b) – both algorithms obtained the same scheduling. Coffman and Graham algorithm is optimal only for two identical processors. For task graph under research it also found optimal scheduling for 3 identical processors.

Another test problem is shown by the non-optimality of Coffman and Graham algorithm for processor number greater than 2.

Test problem no 2:

- 2 identical processors (a), 3 identical processors (b)
- 12 tasks with singular performance times.
- Graph of tasks:

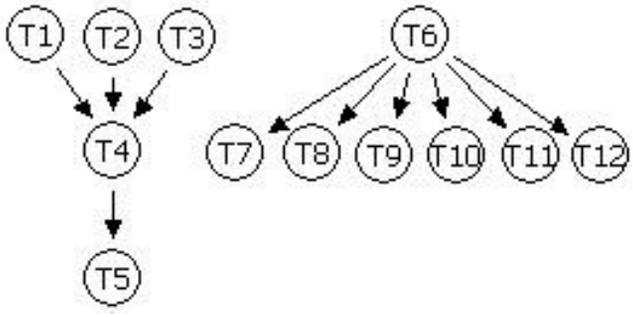


Figure 10. Graph of tasks used for the comparison of ACO algorithm with Coffman and Graham algorithm (test problem no 2).

http://dx.doi.org/10.5772/51487

- Optimal scheduling for two processors obtained as a result of Coffman and Graham algorithm use (2a).

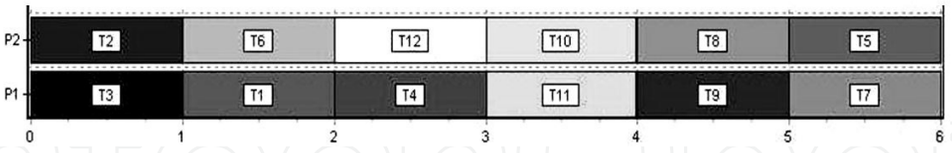


Figure 11. Optimal scheduling for 2 processors - Coffman and Graham algorithm (2a).

- Optimal scheduling for two processors obtained as a result of ACO algorithm use (2a).

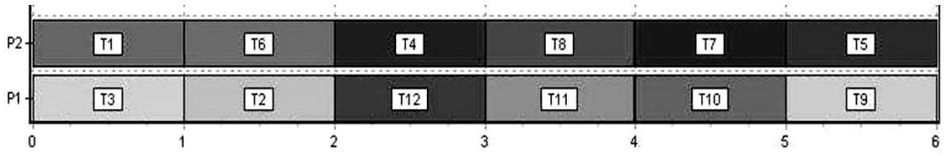


Figure 12. Optimal scheduling for 2 processors - ACO algorithm (2a).

- Non-optimal scheduling for three processors obtained as a result of Coffman and Graham algorithm use (2b).

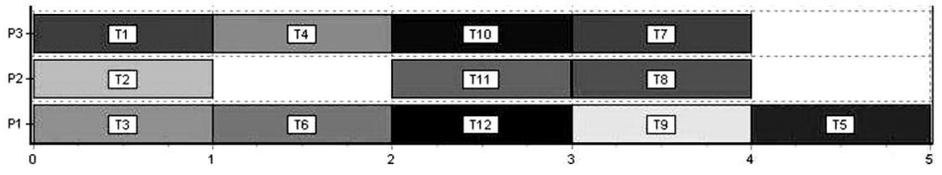


Figure 13. Non-optimal scheduling for 3 processors – Coffman and Graham algorithm (2b).

- Optimal scheduling for three processors obtained as a result of ACO algorithm use (2b).

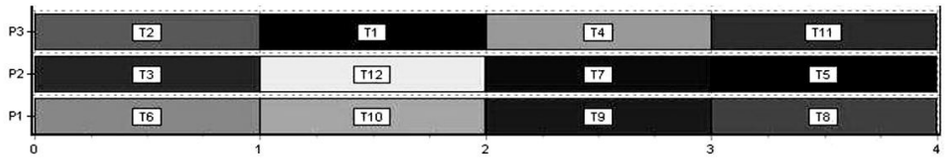


Figure 14. Optimal scheduling for 3 processors - ACO algorithm (2b).

For the problem of two processors (2a) both algorithms obtained optimal scheduling. In the case of three processors (2b) the Coffman and Graham algorithm did not find optimal scheduling, whereas the ACO algorithm did find it without any difficulty.

In another test example both algorithms were compared for the problem of task scheduling on two identical processors with singular and different performance times.

Test problem no 3:

- 2 identical processors.
- 5 tasks with singular performance times (a), 5 tasks with different performance times (b)
- Graph of tasks:

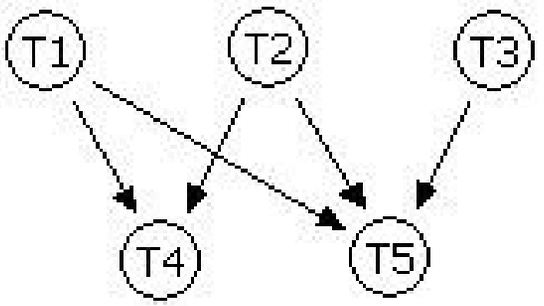


Figure 15. Graph of tasks used for the comparison of ACO algorithm with Coffman and Graham algorithm (test problem no 3)

- Optimal scheduling for singular task performance times obtained as a result of Coffman and Graham algorithm use (3a).

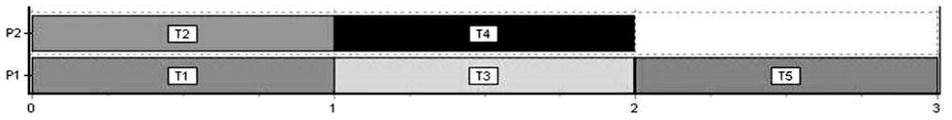


Figure 16. Optimal problem scheduling for singular task performance times – Coffman and Graham algorithm (3a)

- Optimal scheduling for singular task performance times obtained as a result of ACO algorithm use (3a).

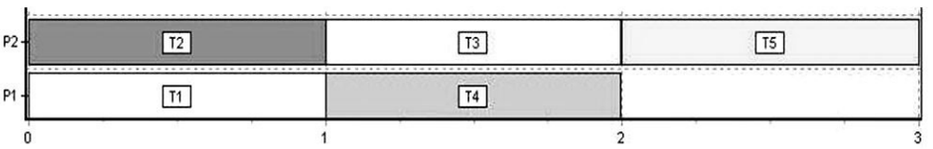


Figure 17. Optimal problem scheduling for singular task performance times – ACO algorithm (3a)

- Non-optimal scheduling for irregular task performance times obtained as a result of Coffman and Graham algorithm use (3b).

http://dx.doi.org/10.5772/51487

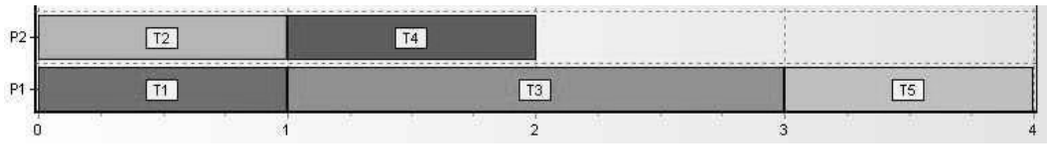


Figure 18. Non-optimal problem scheduling for irregular task performance times – Coffman and Graham algorithm (2b)

- Optimal scheduling for irregular task performance times obtained as a result of ACO algorithm use (3b):

Figure 19. Optimal problem scheduling for irregular task performance times – ACO algorithm (3b)

Both compared algorithms obtain optimal scheduling for the problem with regular (singular) task performance times (3a). For different task performance times (3b) the Coffman and Graham algorithm does not obtain optimal scheduling, whereas the ACO algorithm does obtain.

Hu algorithm

Scheduling of tasks with singular performance times which create a digraph of anti-tree type on identical processors in order to minimize C_{max}. Algorithm complexity is O(n).

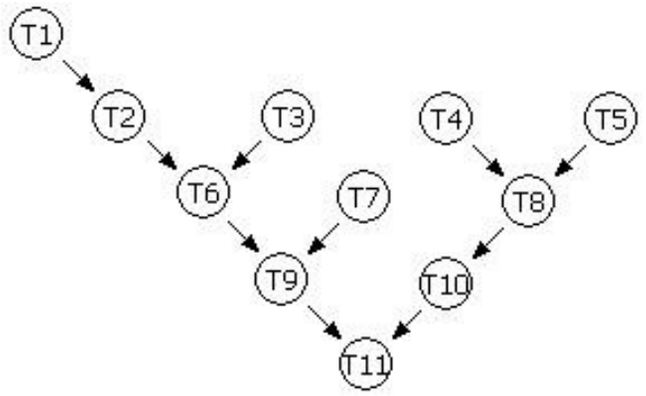


Figure 20. Graph of tasks used for the comparison of ACO and Hu algorithms.

Test problem no 1:

- 3 identical processors,
- 11 tasks with singular performance times,

- Graph of tasks (anti-tree):
- Optimal scheduling for problem 1 obtained as a result of Hu algorithm use:

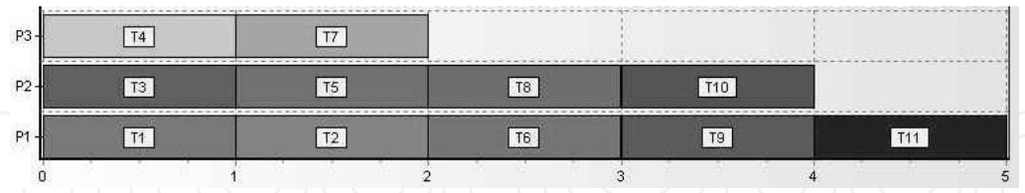


Figure 21. Optimal scheduling for problem 1 solved with Hu algorithm.

- Optimal scheduling for problem 1 obtained as a result of ACO algorithm use.

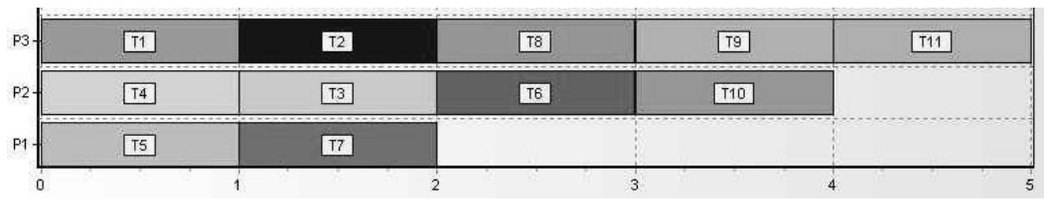


Figure 22. Optimal scheduling for problem 1 solved with ACO algorithm.

Test problem no 2:

- 3 identical processors,
- 12 tasks with singular performance times,
- Graph of tasks (anti-tree):

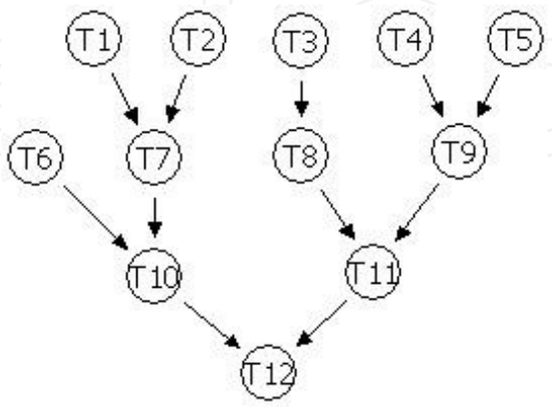


Figure 23. Graph of tasks used for the comparison of ACO and Hu algorithms (test problem no 2)

- Optimal scheduling for problem 2 obtained as a result of Hu algorithm use.

http://dx.doi.org/10.5772/51487

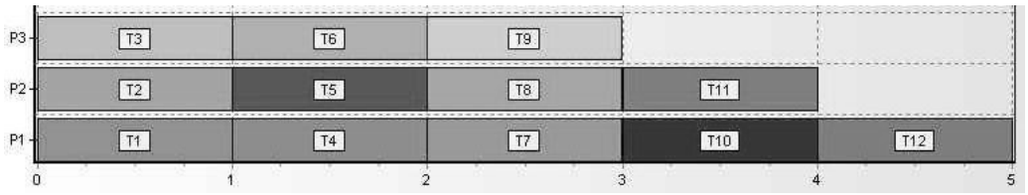


Figure 24. Optimal scheduling for problem 2 solved with Hu algorithm

- Optimal scheduling for problem 2 obtained as a result of ACO algorithm use.

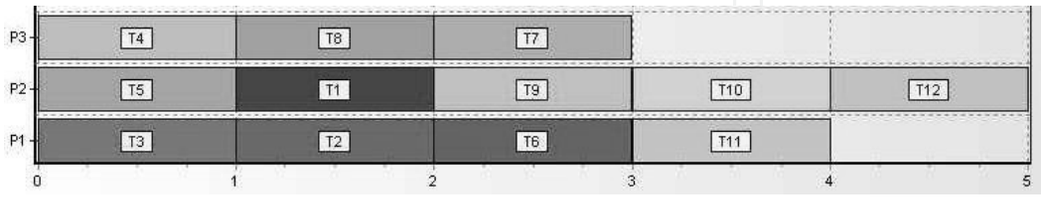


Figure 25. Optimal scheduling for problem 2 solved with ACO algorithm

Both problems solved with Hu algorithm were also solved easily by ACO algorithm. Scheduling obtained is optimal.

Baer algorithm

Scheduling of indivisible tasks, with singular performance times, which create a graph of anti-tree type on two uniform processors in order to minimize C_{max}.

Test problem:

- 2 uniform processors with speed coefficients $b_1 = 2$, $b_2 = 1$.
- 11 tasks with singular performance times.
- Graph of tasks (anti-tree):

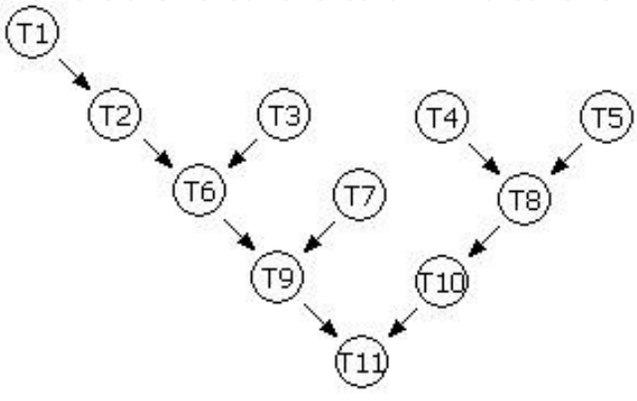


Figure 26. Graph of tasks used for the comparison of ACO and Baer algorithms.

- Optimal scheduling for the problem solved with Baer algorithm, obtained as a result of ACO algorithm use.

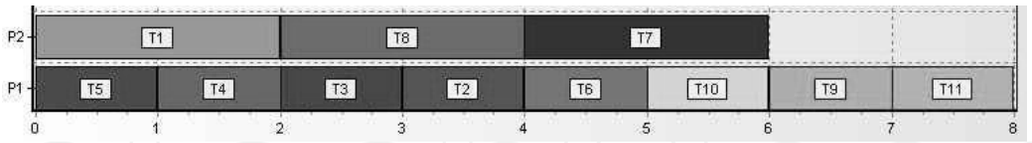


Figure 27. Optimal scheduling for the problem solved with Baer algorithm, obtained as a result of ACO algorithm use

For the problem optimized with Baer algorithm, the ACO algorithm also obtains optimal solution.

4.2. Comparison of algorithms for non-polynomial problems of task scheduling

4.2.1. NP- complete problem no 1:

Scheduling *nonpreemptive*, independent tasks on identical processors for C_{max} minimization.

Number of tasks	Number of processors	Cmax Neural Algorithm	Cmax ACO Algorithm
5	3	4	4
10	3	9	8
10	6	4	4
20	3	15	16
20	6	9	8
20	8	7	6

Table 1. Scheduling nonpreemptive, independent tasks on identical processors.

For all problems under research algorithms found similar solutions. Only neural algorithm did worse – for the problem of scheduling 10 tasks on 3 identical processors, 20 tasks on 6 processors and 20 tasks on 8 processors as well ACO algorithm for the problem of scheduling 20 tasks on 3 identical processors.

4.2.2. NP-complete problem no 2:

List scheduling with various methods of priority allocation

Because in general case the problem of scheduling dependent, nonpreemptable tasks is highly NP-complete, in some applications one can use polynomial approximate algorithms. Such algorithms are list algorithms.

http://dx.doi.org/10.5772/51487

In the chapter five types of list scheduling rules were compared: HLFET (Highest Levels First with Estimated Times), HLFNET (Highest Levels First with No Estimated Times), RANDOM, SCFET (Smallest Co-levels First with Estimated Times), SCFNET (Smallest Co-levels First with No Estimated Times) [12].

The number of cases, in which the solution differs less than 5% from optimal solution, is accepted as an evaluation criterion for the priority allocation rule. If for 90% of examined examples the sub-optimal solution fit in the above range, the rule would be described as "almost optimal". This requirement is met only by HLFET rule, which gives results varying from optimum by 4,4% on average.

Example:

- 2 identical processors.
- 12 tasks with different performance times: (Z0,1), (Z1,1), (Z2,7), (Z3,3), (Z4,1), (Z5,1), (Z6,3), (Z7,2), (Z8,2), (Z9,1), (Z10,3), (Z11,1).
- Graph of tasks:

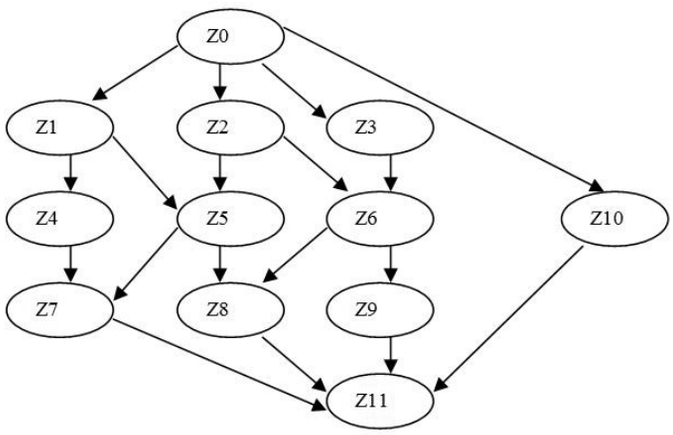


Figure 28. The graph of tasks used for the comparison of ACO and list algorithms

Scheduling obtained as a result of ACO algorithm operation.

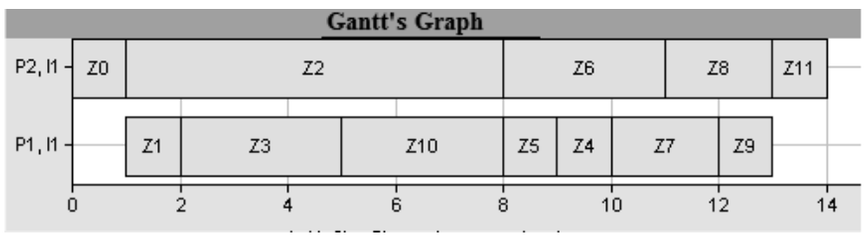


Figure 29. Scheduling obtained with ACO algorithm.

The length of obtained scheduling is compliant with the scheduling which was obtained by means of the best list scheduling available for this case and which is HLFET (”almost optimal”).

4.2.3. Comparison with PDF/HIS algorithm

For research purposes a set of graphs was utilized from the website below: http://www.kasahara.elec.waseda.ac.jp/schedule/index.html. Task graphs made available therein were divided into groups because of the number of tasks. Minimum scheduling length was calculated by means of PDF/HIS algorithm (Parallelized Depth First/ Implicit Heuristic Search) for every tasks graph. STG graphs are vectored, a-cyclic tasks graphs. Different task performance times, discretionary sequence constraints as well as random number of processors cause STG tasks scheduling problems to be NP-complete problems. Out of all solved problems heuristic algorithms under research did not find an optimal solution (assuming this is the solution obtained with PDF/IHS algorithm) only for three of them. However, results obtained are satisfactory, because the deviation from optimum varies from 0,36% to 4,63% (table Tab 2).

STG	**Num-ber of tasks**	**Number of processors**	**PDF/ IHS**	**Ant colony**			**Neural**		
			C_{max}	C_{max}	Number of iterations	Diffe-rence [%]	C_{max}	**Number of itera-tons**	Diffe-rence [%]
rand0008	50	2	281	281	117	0	281	80	0
rand0038	50	4	114	114	1401	0	114	818	0
rand0107	50	8	155	155	389	0	155	411	0
rand0174	50	16	131	131	180	0	131	190	0
rand0017	100	2	569	569	171	0	569	92	0
rand0066	100	4	253	253	4736	0	257	3644	1,58
rand0106	100	8	205	205	861	0	205	927	0
rand0174	100	16	162	162	265	0	162	216	0
rand0020	300	2	827	846	5130	2,30	830	4840	0,36
rand0095	300	8	382	394	5787	3,14	384	5253	0,52
rand0136	300	16	324	339	2620	4,63	324	3067	0

Table 2. Comparison with PDF/IHS algorithm – the influence of tasks number

http://dx.doi.org/10.5772/51487

Algorithms were investigated by scheduling tasks represented with the same graph (50 STG tasks) on a different number of processors.

Number of tasks	Number of processors	PDF/IHS	Ant colony		Neural	
		C_{max}	C_{max}	Number of iterations	C_{max}	Number of iterations
50	2	228	228	132	228	92
50	4	114	114	1401	114	925
50	8	57	61	4318	58	4442
50	16	48	48	58	48	33

Table 3. Minimization of C_{max} of dependent tasks (STG rand0008.stg)

Number of tasks	Number of processors	PDF/IHS	Ant colony		Neural	
		C_{max}	C_{max}	Number of iterations	C_{max}	Number of iterations
50	2	267	267	388	267	412
50	4	155	157	4487	160	3339
50	8	155	154	89	155	112
50	16	155	155	10	155	8

Table 4. Minimization of C_{max} of dependent tasks (STG rand0107.stg)

In all researched problems algorithms under comparison found optimal solution. The only difference can be observed in the number of iterations needed to find an optimal solution. ACO algorithm needed less iterations than neural one to find the solution.

5. Comparing ACO algorithm and neural algorithm

For multiple criteria optimization in the following tests comparisons were made of compromise solutions for ACO algorithm with the results of neural algorithm. Optimization criteria were: time, cost and power consumption. Additional requirements and constraints were adopted: maximum number of processors – 5, maximal cost – 3, maximal time – 25.

Number of tasks	Ant colony			Neural		
	Cost	Time	Power consumption	Cost	Time	Power consumption
5	1,75	6,75	9,26	1,00	3,90	4,39
10	1,50	6,20	35,47	1,50	8,50	11,61
15	2,75	18,00	22,96	2,00	16,00	17,85
20	1,75	12,83	35,45	2,00	22,50	20,31
25	2,00	14,50	51,25	2,00	22,00	28,93
30	2,75	16,90	63,58	2,50	23,00	35,01
35	2,00	18,00	78,30	2,50	24,67	36,12
40	2,75	17,75	104,68	2,50	17,00	72,52
45	2,25	21,75	99,50	2,50	18,67	79,02
50	2,25	23,88	113,26	2,50	21,00	88,57
55	2,50	25,00	164,58	2,50	22,50	95,33

Table 5. Comparison of Ant Colony and neural for minimization of time, cost and power consumption.

Results were illustrated on the following charts – Chart: 30, 31, and 32.

When comparing solutions obtained by the algorithms one cannot provide an unequivocal answer which of the optimization methods is better. Greater influence on the quality of offered solutions has the algorithm itself, especially its exploration capacity of admissible solutions space. When analyzing the graphs of interdependence between cost and task number, it appears that neural algorithm is more stable i.e. attempts to maintain low cost, despite an increase in the number of tasks. This results in worse task performance time what is very visible on the graph where time is contingent on the number of tasks. From power consumption analysis it is evident that ACO algorithm solutions are more beneficial.

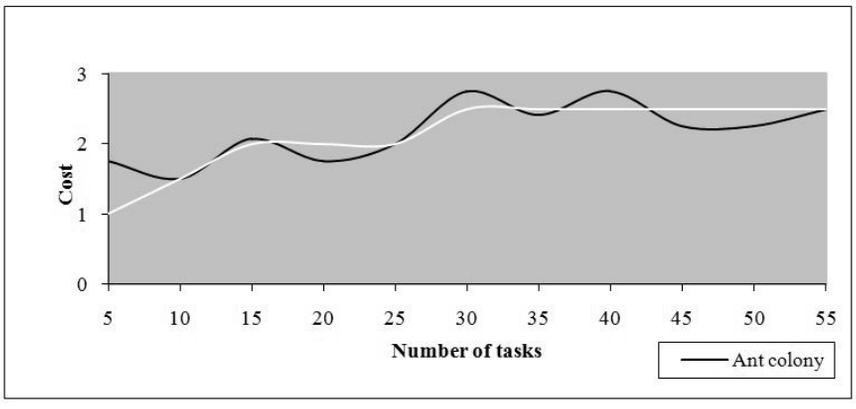


Chart 30. Influence of number tasks on cost – minimization of time, cost and power consumption .

http://dx.doi.org/10.5772/51487

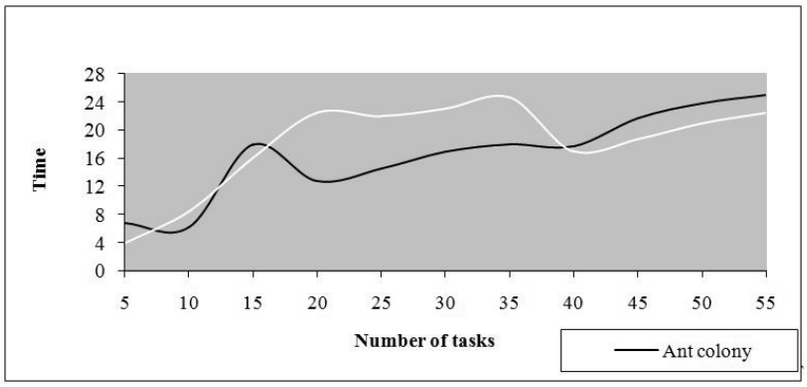


Chart 31. Influence of number of tasks on time – minimization of time, cost and power consumption.

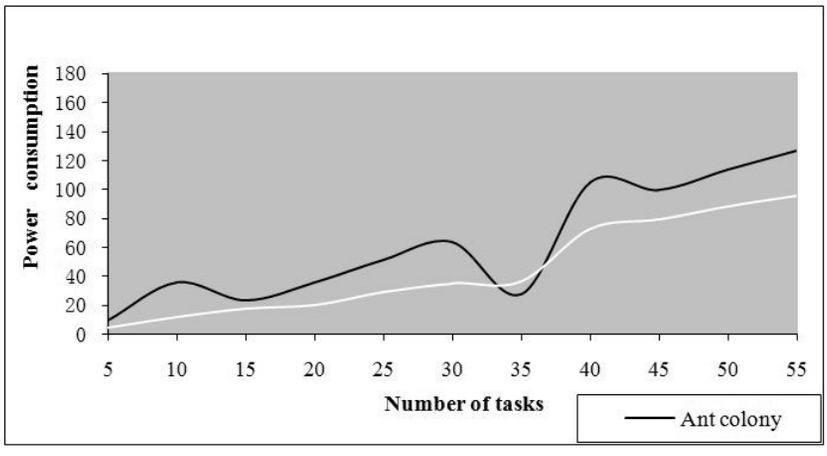


Chart 32. Influence of number of tasks on power consumption – minimization of time, cost and power consumption.

Additional requirements and constraints were adopted: maximum number of processors: 5, maximal cost: 8, maximal time: 50.

Results were illustrated in the following charts - Chart: 33, 34, and 35.

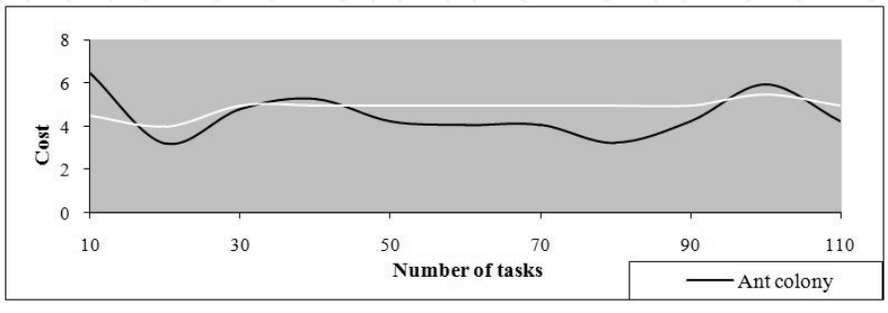


Chart 33. Influence of number of tasks on cost – minimization of time, cost and power consumption with of cost of memory.

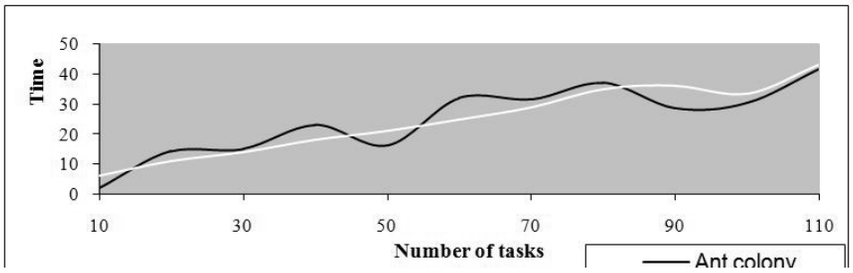


Chart 34. Influence of number of tasks on time – minimization of time, cost and power consumption with of cost of memory.

Number of tasks	Ant colony			Neural		
	Cost	Time	Power consumption	Cost	Time	Power consumption
10	6,50	2,00	37,99	4,50	6,00	7,52
20	1,50	18,50	33,15	4,00	11,00	19,07
30	5,90	23,00	82,41	5,00	14,00	30,98
40	7,00	23,00	121,56	5,00	18,00	37,33
50	4,25	16,20	186,05	5,00	21,00	49,99
60	2,50	32,00	175,24	5,00	25,00	60,20
70	2,50	38,00	167,59	5,00	29,00	69,35
80	3,25	37,00	183,67	5,00	32,00	79,19
90	4,25	28,60	328,73	5,00	36,00	98,39
100	6,75	30,33	336,36	5,50	39,00	101,62
110	4,25	41,80	435,77	5,00	43,00	115,53

Table 6. Comparison of Ant colony and neural for minimization of time, cost and power consumption.

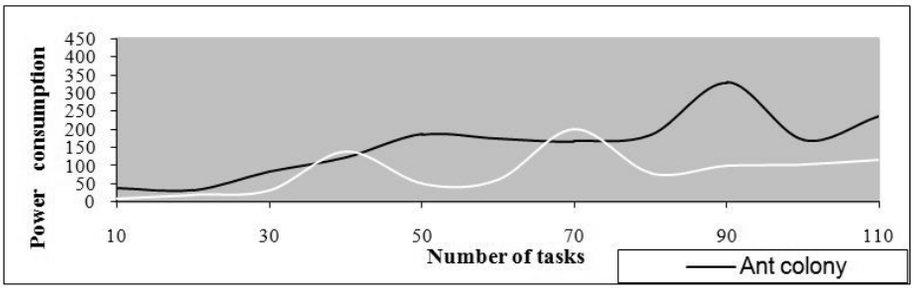


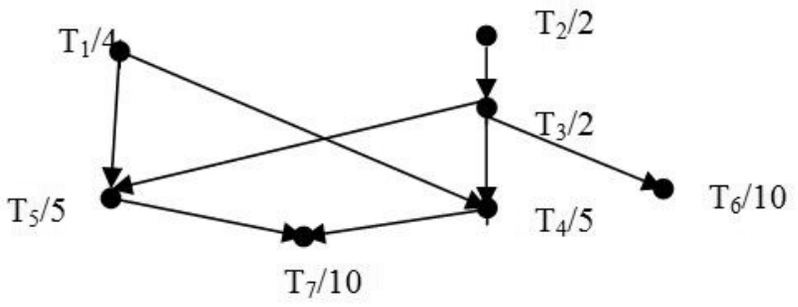
Chart 35. Influence of number of tasks on power consumption – minimization of time, cost and power consumption with memory cost

http://dx.doi.org/10.5772/51487

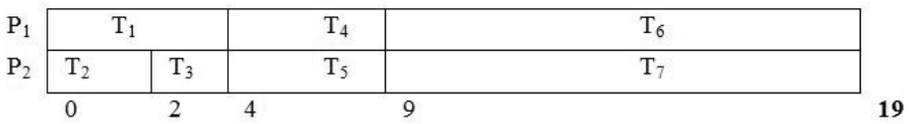
6. Conclusions

Conducted research shows that presented algorithms for task scheduling obtain good solutions - irrespectively of investigated problem complexity. These solutions are considered optimal or sub-optimal whose deviation from optimum does not exceed 5%. Heuristic algorithms proposed for task scheduling problems, especially ACO, should be a good tool for supporting planning process.

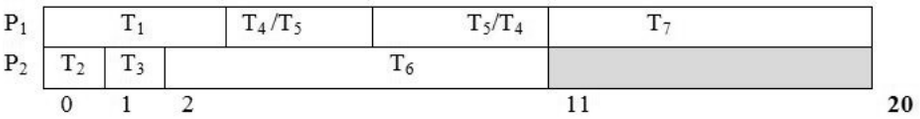
One should indicate a possible and significant impact of anomalies in task scheduling on the quality of the obtained results. The following examples [12] show a possibility of appearing such anomalies. Take an example of this digraph of tasks:



•



- Diminishing of performance time for all the tasks $t_i' = t_i - 1$ and the scheduling is longer than optimum scheduling (independently from choice list!):



For different problem instances, particular algorithms may achieve different successes; others may achieve worse results at different numbers of tasks. The best option is to obtain results of different algorithms and of different runs.

The goal of this scheduling is to find an optimum solution satisfying the requirements and constraints enforced by the given specification of the tasks and resources as well as criteria.

As for the optimality criteria for the manufacturing system for better control, we shall assume its minimum cost, maximum operating speed and minimum power consumption.

We will apply multi-criteria optimization in sense of Pareto. The solution is optimized in sense of Pareto if it is not possible to find a better solution, regarding at least one criterion without deterioration in accordance to other criteria. The solution dominates other ones if all its features are better. Pareto ranking of the solution is the number of solutions in a pool which do not dominate it. The process of synthesis will produce a certain number of non-dominated solutions. Although non-dominated solutions do not guarantee that they are an optimal Pareto set of solutions; nevertheless, in case of a set of suboptimal solutions, they constitute one form of higher order optimal set in sense of Pareto and they give, by the way, access to the problem shape of Pareto optimal set of solutions.

Let's assume that we want to optimize a solution of two contradictory requirements: the cost and power consumption Fig. 36.

While using a traditional way with one optimization function, it is necessary to contain two optimal criteria in one value. To do that, it is advisable to select properly the scales for the criteria; if the scales are selected wrongly, the obtained solution will not be optimal. The chart in the illustration shows where, using linearly weighed sum of costs, we will receive the solution which may be optimizes in terms of costs.

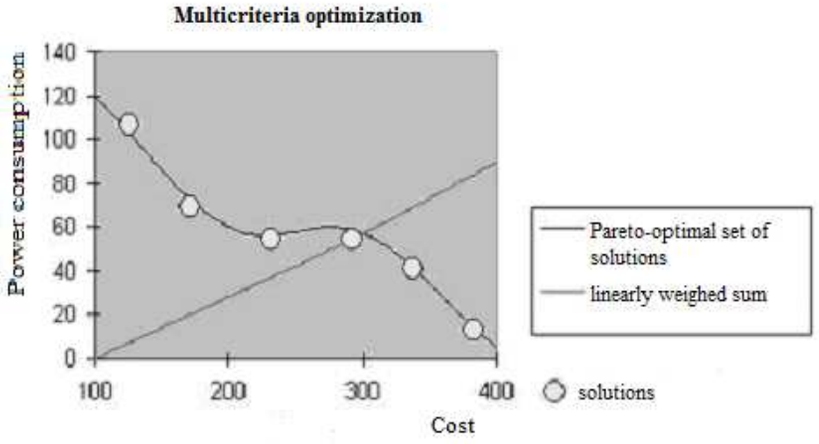


Figure 36. Set of optimal solutions in sense of Pareto.

Cost optimization, power and time consumption in the problem of scheduling is, undoubtedly, the problem where the potential number of solutions in sense of Pareto is enormous.

Future research: others of instances of scheduling problems, and additional criteria, especially in sense of Pareto and for dependable systems, are still open and this issue is now studied.

http://dx.doi.org/10.5772/51487

Author details

Mieczysław Drabowski[1*] and Edward Wantuch[1,2]

*Address all correspondence to: drabowski@pk.edu.pl

1 Cracow University of Technology, Poland

2 AGH University of Science and Technology, Poland

References

[1] Aggoune, R. (2004). Minimizing the makespan for the flow shop scheduling problem with availability constraints. *Eur. J. Oper., Res.*, 153, 534-543.

[2] Ostfeld, Avi. (2011). Any Colony Optimization. *Rijeka, Croatia, InTech*.

[3] Błażewicz, J., Drabowski, M., & Węglarz, J. (1984). Scheduling independent 2-processor tasks to minimize schedule length. *Inform. Proce. Lett.*, 18, 267-273.

[4] Błażewicz, J., Ecker, K., Pesch, E., Schmidt, G., & Węglarz, J. (1996). Scheduling Computer and Manufacturing Processes. *Springer*.

[5] Błażewicz, J., Ecker, K., Pesch, E., Schmidt, G., & Węglarz, J. (2007). Handbook on Scheduling, From Theory to Applications. *Springer-Verlag Berlin Heidelberg*.

[6] Blum, C. (2005). Beam-ACO- Hybridizing ant colony optimization with bean search: An application to open shop schedling. *Comput. Oper. Res.*, 32, 1565-1591.

[7] Blum, C., & Sampels, M. (2004). An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modeling and Algorithm*, 3, 285-308.

[8] Breit, J., Schmidt, G., & Strusevich, V. A. (2003). Non-preemptive two-machine open shop scheduling with non-availability constraints. *Math. Method Opr. Res.*, 57(2), 217-234.

[9] Brucker, P. (2004). Scheduling Algorithms. *Springer*.

[10] Brucker, P., & Knust, S. (2006). Complex Scheduling. *Springer*.

[11] Cheng, T. C. E., & Liu, Z. (2003). Approximability of two-machine no-wait flowshop scheduling with availability constraints. *Opr. Res. Lett.*, 31, 319-322.

[12] Coffman, E. G. Jr. (1976). Computer and Job-shop scheduling theory. *John Wiley&Sons, Inc. New York*.

[13] Colak, S., & Agarwal, A. (2005). Non-greedy heuristiad augmented neural networks for the open-shop scheduling problem. *Naval Res. Logist.*, 52, 631-644.

[14] Dechter, R., & Pearl, J. (1988). Network-based heuristic for constraint satisfaction problems. *Artificial Intelligence*, 34, 1-38.

[15] Dorndorf, U., Pesch, E., & Phan-Huy, T. (2000). Constraint propagation techniques for disjunctive scheduling problems. *Artificial Intelligence*, 122, 189-240.

[16] Drabowski, M., & Wantuch, E. (2006). Coherent Concurrent Task Scheduling and Resource Assignment in Dependable Computer Systems Design,. *International Journal of Reliability Quality and Safety Engineering, World Scientific Publishing,*, 13(1), 15-24.

[17] Drabowski, M. (2007). Coherent synthesis of heterogeneous system- an ant colony optimization approach. *Proceedings of Artificial Intelligence Studies, Vol.4 (89)/2007, supported by IEEE, Siedlce*, 65-74.

[18] Drabowski, M. (2007). The ant colony in par-synthesis of computer system. *Proceedings of the 11th IASTED International Conference on Artificial Intelligence and Soft Computing, Palma de Mallorca, ACTA Press, Anaheim, USA*, 244-249.

[19] Drabowski, M. (2007). Coherent synthesis of heterogeneous system- an ant colony optimization approach. *Studia Informatica*, 2.

[20] Drabowski, M. (2007). An Ant Colony Optimization to scheduling tasks on a grid. *Polish Journal of Environmental Studies*, 16(5B).

[21] Drabowski, M. (2008). Solving Resources Assignment and Tasks Scheduling Problems using Neural Networks. *Artificial Intelligence Studies*, 2.

[22] Drabowski, M. (2008). Neural networks in optimization scheduling resources and processes for management on a grid. *Polish Journal of Environmental Studies*, 17(4C).

[23] Drabowski, M. (2009). Ant Colony and Neural method for scheduling of complex of operations and resources frameworks- comparative remarks. *Proceedings of the IASTED International Conference on Computational Intelligence, Honolulu, USA, ACTA Press, Anaheim, USA*, 91-97.

[24] Drabowski, M. (2011). Ant Colony Optimization for coherent synthesis of computer system. *Ostfeld A., (ed.) Ant Colony Optimization, InTech, Croatia, Austria, India*, 179-204.

[25] Garey, M. R., & Johnson, D. S. (1979). Computers and intractability: A guide to the theory of NP-completeness,. *San Francisco, Freeman*.

[26] Ha, S., & Lee, E. A. (1997). Compile-Time Scheduling of Dynamic Constructs in Dataflow Program Graphs,. *IEEE Trans. On Computers*, 46(7).

[27] Leung, J. Y. T. (2004). Handbook on Scheduling: Algorithms, Models and Performance Analysis,. *Chapman&Hall, Boca Raton*.

[28] Lee, C. Y. (1996). Machine scheduling with an availably constraint. *J. Global Optim.*, 9, 363-384.

http://dx.doi.org/10.5772/51487

[29] Lee, C. Y. (2004). Machine scheduling with availably constraints. *Leung J.Y.T. Handbook of Scheduling, CRC Press*, 22, 1-22.

[30] Meseguer, P. (1989). Constraint satisfaction problems: An overview. *AICOM*, 2, 3-17.

[31] Montgomery, J., Fayad, C., & Petrovic, S. (2006). Solution representation for job shop scheduling problems in ant colony optimization. *LNCS*, 4150, 484-491.

[32] Morton, T. E., & Pentico, D. W. (1993). Heuristic Scheduling System. *Wiley, New York*.

[33] Nuijten, W. P. M., & Aarts, E. H. L. (1996). A computational study of constraint satisfaction for multiple capacitated job shop scheduling. *European J. Oper. Res.*, 90, 269-284.

[34] Pinedo, M. (2001). Scheduling Theory, Algorithms, and Systems,. *Prentice Hall, Englewood Cliffs, N.J.*

[35] Taillard, E. (1993). Benchmarks for basic scheduling problems. *European J. Oper. Res.*, 64, 278-285.

[36] Tsang, E. (1993). Foundations of Constraint Satisfaction. *Academic Press, Essex*.

[37] Wang, C. J., & Tsang, E. P. K. (1991). Solving constraint satisfaction problems using neural-networks,. *IEEE Second International Conference on Artificial Neural Networks*.

[38] Xu, J., & Parnas, D. L. (1993). On Satisfying Timing Constraints in Hard-Real-Time Systems. *IEEE Trans. on Software Engineering*, 19(1), 70-84.

IntechOpen