13

Ant Colony Optimization for Coherent Synthesis of Computer System

Mieczysław Drabowski
Cracow University of Technology
Poland

1. Introduction

The goal of high-level synthesis of computer systems is to find an optimum solution satisfying the requirements and constraints enforced by the given specification of the system. The following criteria of optimality are usually considered: costs of system implementation, its operating speed, power consumption and dependability. A specification describing a computer system may be provided as a set of interactive tasks (processes, functions).

The partition of the functions between hardware and software is the basic problem of synthesis. Such partition is significant, because every computer system must be realized as result of hardware implementation for its certain tasks. In the synthesis methods so far, the software and hardware parts were developed separately and then connected in process the co-called co-synthesis, which increased the costs and decreased the quality and reliability of the final product. The resources distribution is to specify, what hardware and software are in system and to allocate theirs to specific tasks, before designing execution details.

The problems of tasks scheduling are one of the most significant issues occurring at the procedure synthesis of operating systems responsible for controlling the distribution of tasks and resources in computer systems.

The objective of this research is to present the concept of coherent approach to the problem of system synthesis, i.e. a combined solution to task scheduling and resource partition problems. The model and approach are new and original proposals allowing synergic design of hardware and software for performing operations of the computer system. This is approach, which we called a par-synthesis (coherent co-synthesis). This research shows the results selected of computational experiments for different instances of system par-synthesis problems proving the correctness of the coherent synthesis concept and shows the methods solving these problems. Due to the fact that synthesis problems and their optimizations are NP-complete we suggest meta-heuristic approach: **Ant Colony Optimization**.

Coherent co-synthesis of computer systems, as well as synergic design methodology their structures and scheduling procedures may have practical application in developing the tools for automatic aided for rapid prototyping of such systems.

2. General model and synthesis of computer system

2.1 The classical process of computer system synthesis

The classical process co-synthesis (D'Ambrioso & Hu, 1994) – hardware and software – for computer system consists of the following stages (Fig. 2.1):

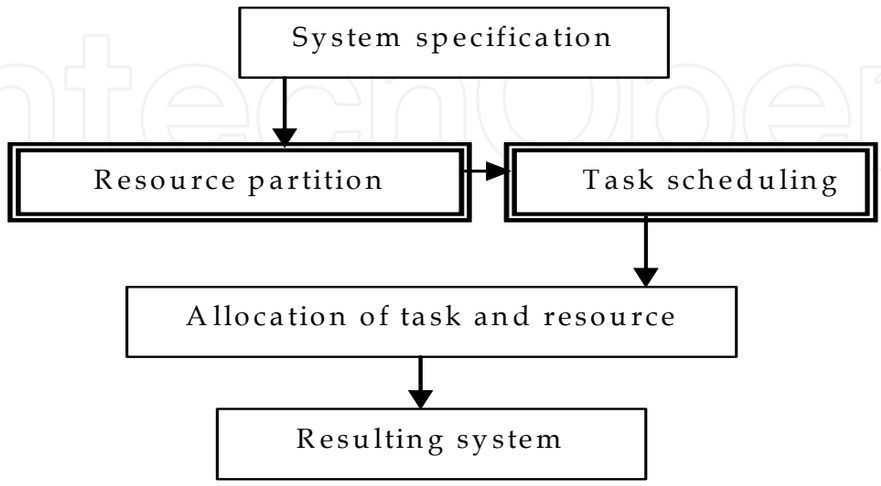


Fig. 2.1. The process co-synthesis

1. Specification of the designed system in terms functional and behavioural – requirements and constraints analysis. The system description in a high-level language, abstracting from the physical implementation.
2. Resource partition – architecture development.
3. Task scheduling – system control development.
4. Allocation the system functions to the architecture elements – generating the system modular architecture, control adaptation and the whole system integration.

The system being constructed consists of hardware elements and software components performed by selected hardware modules (Gajski, 1997) . The system is specified by a set of requirements to be met. In general, each requirement may be satisfied by hardware elements or software components executed by universal processors and memories. Obviously, at this stage of design, one must take into account appropriate system constraints and criteria of optimal system operation. Accordingly, the key issue in the synthesis is efficient partitioning of system resources due to their hardware and software implementation, providing fulfilment of all requirements and the minimum implementation cost (Sgroi et al., 2000).

Such partitioning methodology (Gupta & De Micheli, 1993) may accept, as a starting point, assignment of the hardware implementation to all system functions and further optimization of project costs, search for possibilities of replacing certain tasks realized by hardware with their software equivalents. Other methods (De Micheli, 1994) of the resources partitioning start with an exclusive software implementation and further search for implementation of certain tasks by hardware. In both approaches the objective is optimization of the implementation cost of the same tasks, i.e. in particular minimization of the execution time by specialized hardware (Axelson, 1997). Obviously the requirements and constraints, especially those regarding time and power consumption, have decisive influence upon selection of necessary hardware components.

The measure for an efficient implementation of a computer system is the degree of its modules utilization, minimized idle-time of its elements and maximized parallel operation of its elements (Schulz et al., 1998).
A non-optimum system contains redundant modules or modules that are excessively efficient in comparison to the needs defined by the tasks what, consequently, increases the system cost. In high-level synthesis, the optimization of the designed system costs, speed and power consumption is usually an iterative process, requiring both changes in the architecture and task scheduling (Steinhausen, 1993). That is, why an optimum system may be created as a compromise between the system control algorithm and its hardware organization.

2.2 The general model for the problem of system synthesis

System synthesis is a multi-criteria optimization problem. The starting point for constructing our approach to the issues of hardware and software synthesis is the deterministic theory of task scheduling (Błażewicz et al., 2007). The theory may serve as a methodological basis for multiprocessor systems synthesis.
Accordingly, decomposition of the general task scheduling model is suggested, adequate to the problems of computer system synthesis. From the control point of view such a model should take into account the tasks, which may be either preemptable or nonpreemptable Coffman, 1976). These characteristics are defined according to the scheduling theory. Tasks are preemptable when each task can be interrupted and restarted later without incurring additional costs. In such a case the schedules are called to be preemptive. Otherwise, tasks are nonpreemptable and schedules nonpreemptive .
Preemptability of tasks in our approach cannot be a feature of the searched schedule – as in the task scheduling model so far. The schedule contains all assigned tasks with individual attributes: preemptive, nonpreemptive. From the point of view of the system synthesis, the implementation of certain tasks from the given set must be nonpreemptible, for the other may be preemptible (what, in turn, influences significantly selection of an appropriate scheduling algorithm) (Węglarz, 1999). Moreover, we wish to specify the model of task scheduling in a way suitable for finding optimum control methods (in terms of certain criteria) as well as optimum assignment of tasks to universal and specialised hardware components. Accordingly, we shall discuss the system of type the complex of resources and operations (Błażewicz et al., 2000):

$$\sum = \{ \mathbf{R, T, C} \} \tag{1}$$

Where:
R – is the set of resources (hardware and software),
T – is the set of the system's tasks (operations),
C – is the set of optimization criteria for the system's behaviour and structure.
Resources. We assume that processor set $P = \{P_1, P_2,\ldots, P_m\}$ consists of m elements and additional resources $A = \{ A_1, A_2,\ldots, A_p\}$ consist of p elements.
Tasks. We consider a set of n tasks to be processed with a set of resources. The set of tasks consists of n elements $T = \{T_1, T_2,\ldots, T_n\}$. A feasible schedule is optimal, if its length is minimal and it is implemented using minimum resource cost.

Each task is defined by a set of parameters: resource requirements, execution time, ready time and deadline, attribute - preemptable or nonpreemptable. The tasks set may contain defined precedence constraints represented by a digraph with nodes representing tasks, and directed edges representing precedence constraints. If there is at least one precedence constraint in a task set, we shall refer it to as a set of dependent tasks; otherwise they are a set of independent tasks.

Optimality criteria. As for the optimality criteria for the system being designed, we shall assume its minimum cost, maximum operating speed and minimum power consumption.

The proposed model may be used for defining various synthesis problems for optimum computer systems.

The model of a system in our approach, (Drabowski et al., 2002) typical for the theory of task scheduling, consists of a set of requirements (operations, tasks) and existing relationships between them (related to their order, required resources, time, readiness and completion deadlines, preemptability/nonpreemptability, priority etc.). The synthesis procedure contains the following phases: identification of hardware and software resources for task implementation, defining the processing time, defining the conflict-free task schedule and defining the level of resource co-sharing and the degree of concurrency in task performance (Drabowski, 2008).

The synthesis has to perform the task partitioning into hardware and software resources. After performing the partition, the system shall be implemented partially by specialized hardware in the form of integrated circuits (readily available on the resources pools or designed in accordance to the suggested characteristics) (Harel, 1987). Software modules of the system are generated with the use of software engineering tools. Appropriate processors shall be taken from the resource pool. Synthesis of a system may also provide a system control, create an interface and provide synchronization and communication between the tasks implemented by software and hardware.

The system synthesis, i.e. defining system functions, identifying resources, defining control should be implemented in synergy and be subject to multi-criteria optimization and verification during implementation.

2.3 The coherent process of system synthesis

Modeling the joint search for the optimum task schedule and resource partition of the designed system into hardware and software parts is fully justified. Simultaneous consideration of these problems may be useful in implementing optimum solutions, e.g. the cheapest hardware structures. Synergic approach enables also performing of all assigned tasks with the minimum schedule length. With such approach, the optimum task distribution is possible on the universal and specialized hardware and defining resources with maximum efficiency.

We propose the following schematic diagram of a coherent process of systems synthesis (Drabowski & Czajkowski, 2005), (Fig. 2.2).

The suggested coherent synthesis consists of the following steps:

1. specification of requirements for the system to be designed and its interactions with the environment,
2. specification of tasks, including evaluation of task executive parameters using available resources (e.g. execution times),
3. assuming the initial values of resource set and task scheduling – initial resource set and task schedule should be admissible, i.e. should satisfy all requirements in a non-optimum way,

4. task scheduling and resource partitioning,
5. evaluating the operating speed and system cost, multi-criteria optimization,
6. the evaluation should be followed by a modification of the resource set, a new system partitioning into hardware and software parts (step 4).

Iterative calculations are executed till satisfactory design results are obtained – i.e. optimal (or sub-optimal) system structure and schedule. The designed system should be fast, cheap and small of power consumption.

3. Ant Colony and Branch & Bound methods in coherent synthesis of computer systems

The synthesis based on two algorithms behaving in totally different ways lets you not only find the (sub-)optimal solution, but also verify this solution by algorithm searching through all possible solutions.

Presented algorithms let us find the solution, but at the same time they let us evaluate the algorithms themselves. This way we can tell which of the algorithms is faster in finding better and better solutions, which algorithm is more tolerant to modifications of system parameters, and also which of them enables fast adaptation to new parameters, while the system changes dynamically.

If we assume that solution is changing dynamically, it would be a big obstacle for greedy algorithms, because modification of single parameter (giving eventually better parameters) forces another verification of the full set of solutions.

In our approach, the obtained solutions are considered allowing for the following parameters:

- size and cost of operational memory,
- size and cost of mass storage,
- number of processors and the cost of computing power,
- the time needed for scheduling the tasks.

To evaluate obtained solution, we use the method of weighted average: evaluated are all parameters considered during the analysis with appropriate weights; if the final grade of the new solution is better than the grade of the previous one, the new solution is being saved.

3.1 Adaptation of ACO to solve the problems of synthesis

The Ant Colony Optimization (ACO) algorithm is a heuristics using the idea of agents (here: ants) imitating their real behavior (Blum, 2005). 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 (Fig. 3.1).

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

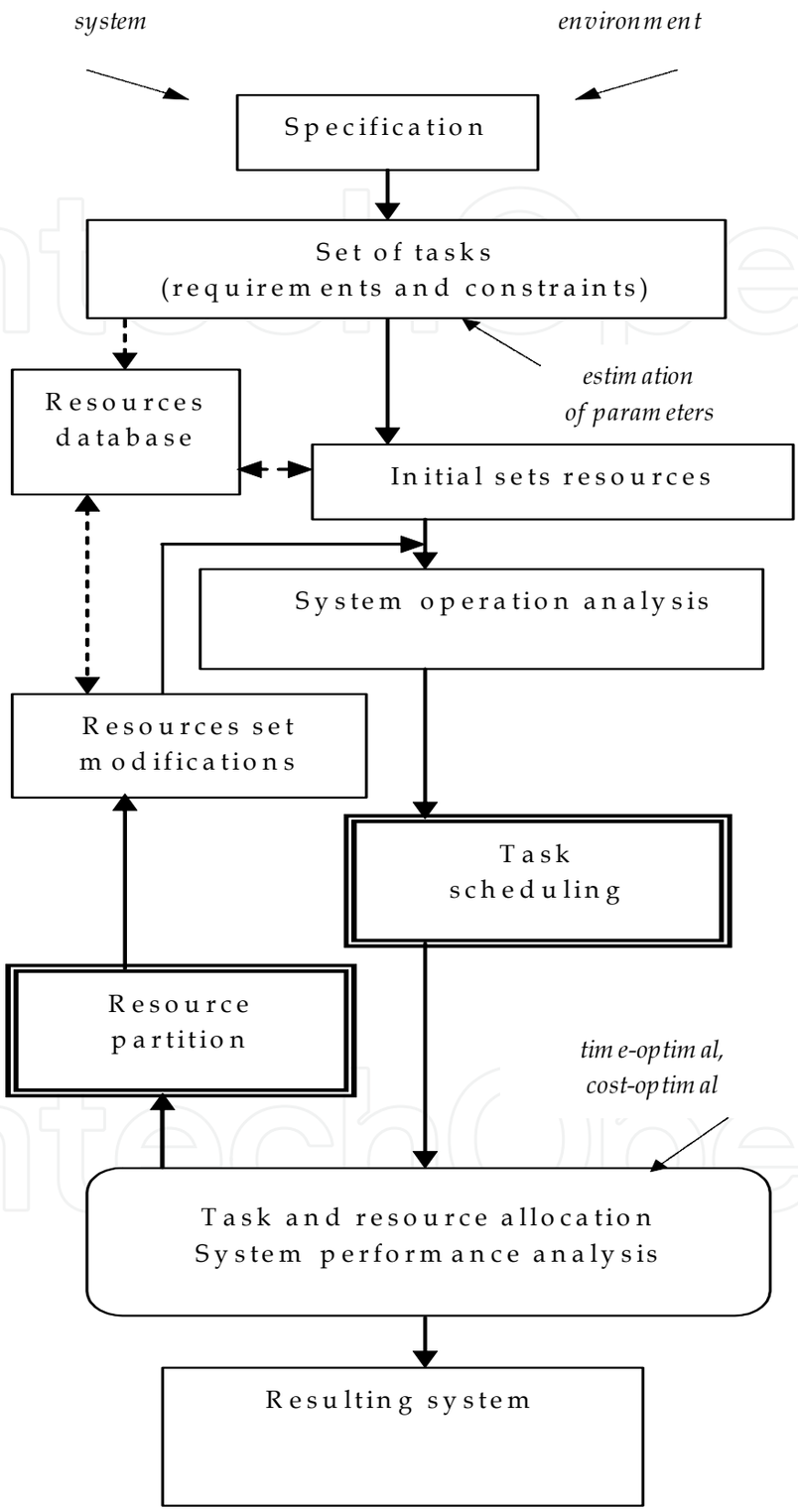


Fig. 2.2. The coherent process of computer system synthesis

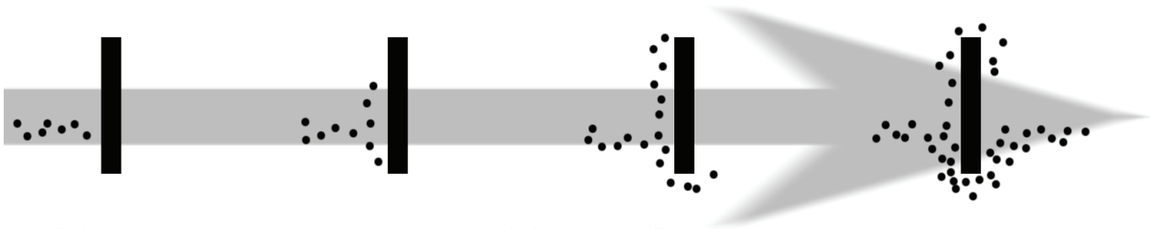


Fig. 3.1. The idea of algorithm – overcoming the obstacle by ants

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 (Montgomery et al., 2006).
To adapt the ACO algorithm to synthesis 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, and
- 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:

$$p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}} & \text{when } j \in N_i^k \\ 0 & \text{else} \end{cases} \tag{3.1}$$

Formula 3.1. 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.

The job of agent involves (Fig. 3.2):

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

To evaluate the quality of allocation the task to processor, the following method is being used (Fig. 3.3):

- evaluation of current (incomplete) scheduling
- allocation of task to the next of available resources
- evaluation of the sequence obtained
- release the task
- 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.

After initiating the tables (of allocation and sequence) for each agent, the algorithm starts the above cycle, after which the evaluation of solutions takes place. Having completed the particular number of cycles, the parameters are being updated and algorithm continues working (Fig. 3.4):

- initiation of tables of tasks running sequence and allocation of tasks to resources
- completing the cycle of analysis for each agent
- evaluation of the best solution found in current cycle
- for each agent – basing on the best solution – updating the tables of tasks running sequence and allocation of tasks to resources
- is it the last cycle?
- Optimization/customization of system parameters.

3.2 Customization of B&B to synthesis problems solving

Branch & Bound (B & B) algorithm is a greedy algorithm browsing the set of solutions and "pruning" these branches, which give worse solutions than the best solution already found (Mitten, 1970). This kind of approach often significantly reduces the number of solutions, which must be considered. However in the worst case scenario, "pruning" the branches is impossible and as a result, the B & B algorithm analyzes the complete search-tree.
Both forms (DFS and BFS) of B & B algorithm were used for synthesis. It let us comprehend the problem of analysis of three different kinds of optimization (cost, power, time) without discrediting any of the problems.
B&B algorithm investigates the problem by:

- choice of the task,
- definition of initial time to which you can schedule the task,
- choice of processor on which the task will be allocated.

Because allocating the chosen task in the first available time unit or on the first available processor is not always the best idea, all available time units and processors are being considered. As a result, calculative complexity of algorithm changes exponentially when new tasks are added or polynomial after addition of new processors. Although B&B algorithm operation process is relatively simple, the number of solutions, which must be examined, is huge.

Example

In scheduling of ten independent tasks on 4 different processors and on 2 additional resources is the full tree which included more than 10^{18} potential solutions!

3.3 Calculative experiments

Because one algorithm creates unlimited cycle and the other one takes a very long time to finish in many cases, the results given in the tables' present state of the system after not more than given time limit of analysis (Drabowski, 2007). Depending on the solution criterion, there were used both forms of B&B – DFS and BFS – for the algorithm to be able to find a good solution in time. Each solution given by Ant Colony algorithm will be graded on the basis of solutions found by Branch & Bound algorithm.
Formula for the assessment of obtained solution is following:

$$assessment = ASS = 100\% \cdot \frac{1}{criterions} \cdot \sum_{criterion=1}^{criterions} \frac{result_{B\&B}}{result_{ACO}} \tag{3.2}$$

Formula 3.2. Assessment of solutions

The final grade is influenced only by these parameters, which were being optimized by algorithms: cost, power and time of scheduling (Drabowski, 2009).
The assessment of proposed system includes all three parameters (scheduling time, cost and power consumed by the system):

- the assessment higher than 100% means that ACO algorithm has found better solution than B&B,
- the assessment equal 100% means that both algorithms have found equally good solutions,
- the assessment less than 100% means that B&B algorithm has found better solution.

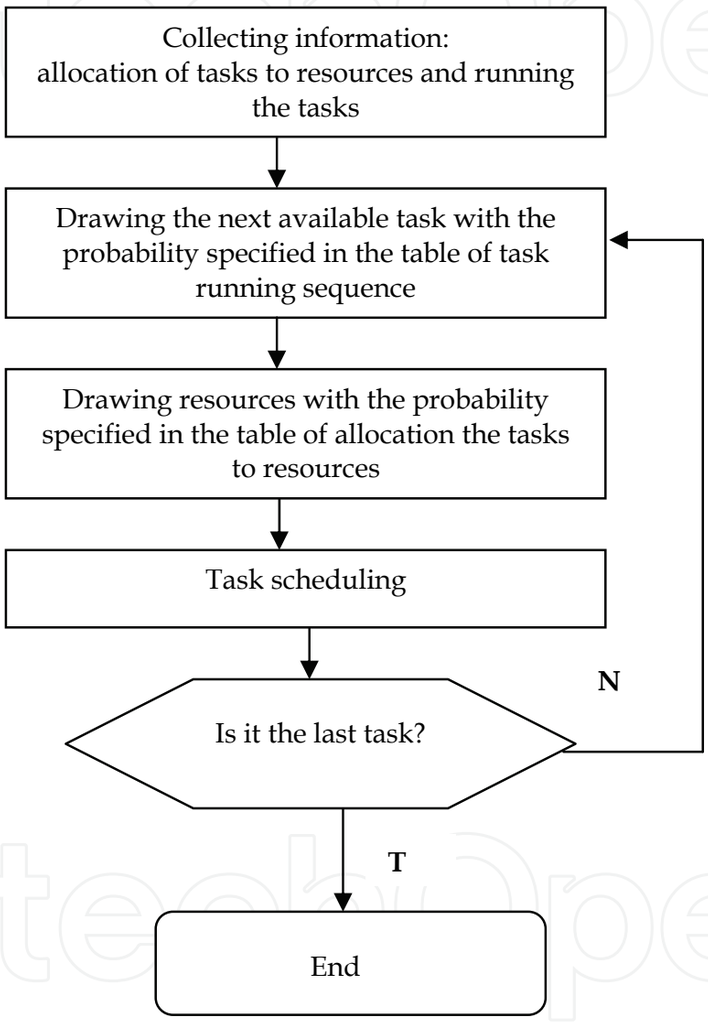


Fig. 3.2. Agent operation scheme

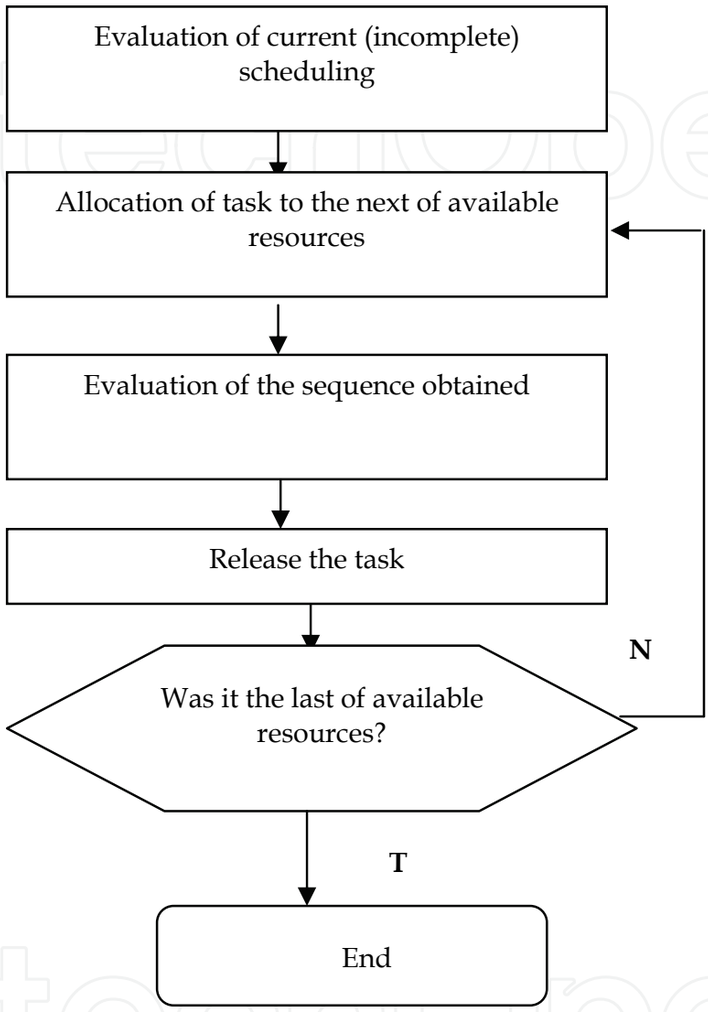


Fig. 3.3. The principle of path evaluation

3.3.1 Verification of tasks schedule

Task schedule proposed by ACO and B&B algorithms was verified on the basis of the following examples.

For the simplicity of tasks descriptions, the *(n: i, j)* scheme was adopted, where *n* – name of the task, *i* – constant time (independent of the speed of processor), *j* – time dependent on the speed of processor.

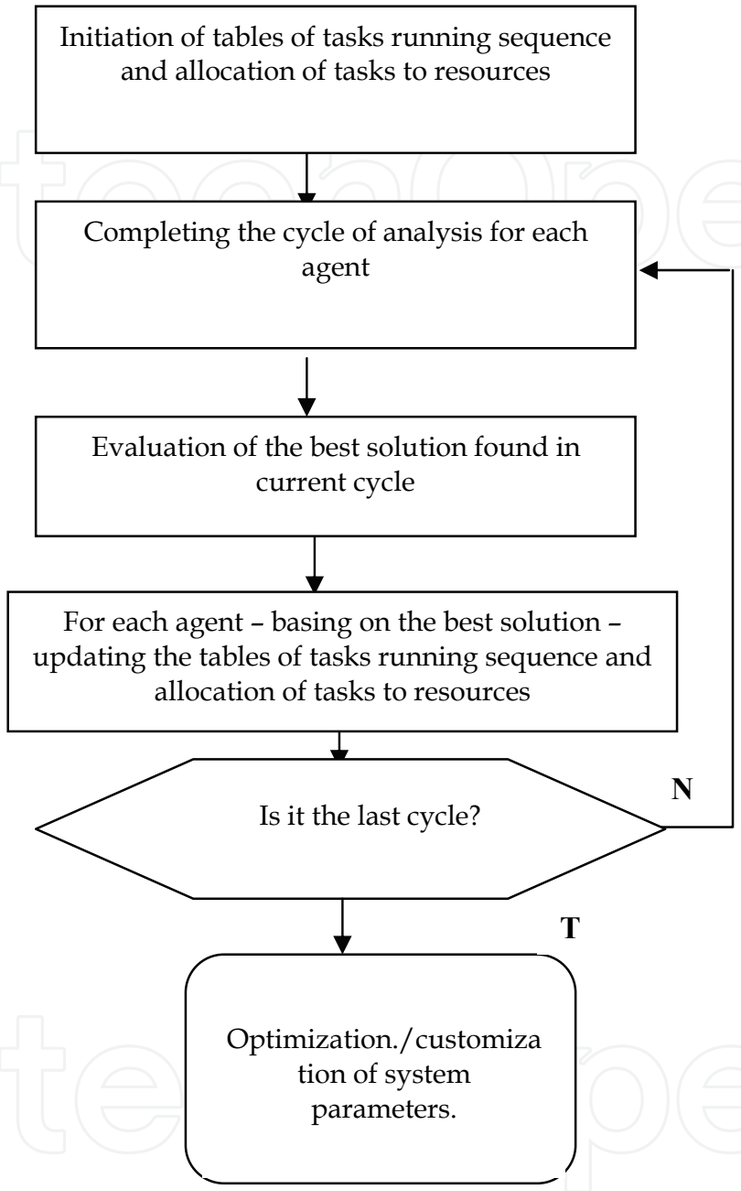


Fig. 3.4. The principle of ACO algorithm operation

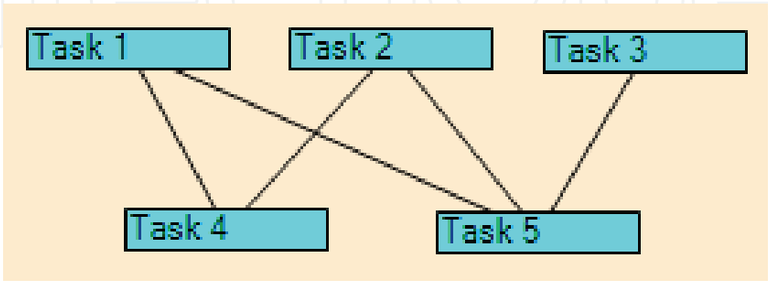
Example 1

Parameters of the problem:

- 5 tasks.
- 2 identical, universal processors.
- Additional resources (memory, storage): without of constraints.
- Parameters of tasks:

Tasks	Time	memory	storage
Task1	1	2	1
Task2	1	2	1
Task3	2	1	1
Task4	1	1	1
Task5	1	2	1

- Relations between tasks are shown on the figure:



Scheduling obtained by both algorithms is identical.

- Total time of scheduling: 3 units,
- Use of resources: 2 units.

The algorithms have found solutions immediately after their activation. Obtained scheduling is presented on the figure (Fig. 3.5):

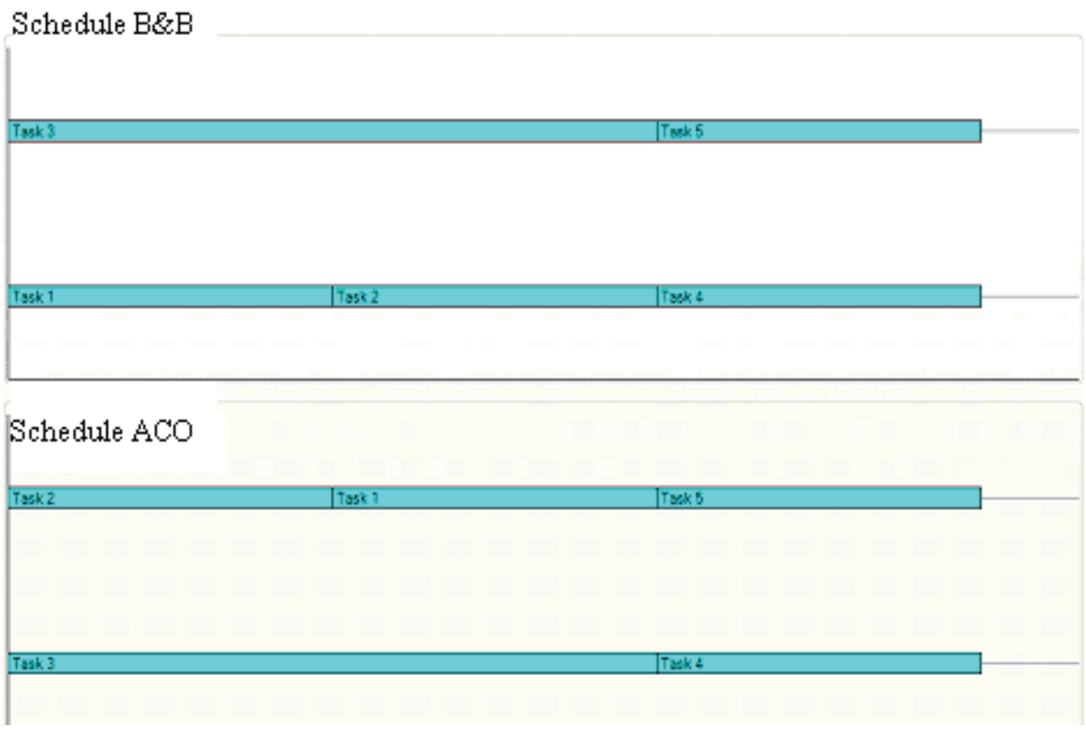
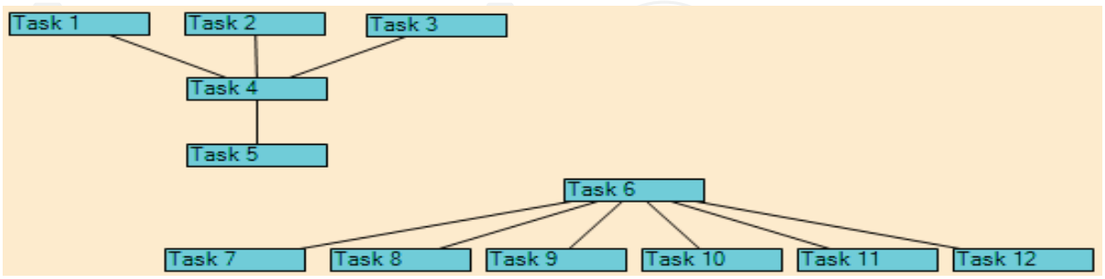


Fig. 3.5. Schedules - results of operations of algorithms

Example 2

Parameters of the problem:

- 12 identical tasks UET (time equal 1); Unit Execution Tasks.
- 2 identical, universal processors.
- Relations between tasks are shown on the figure:



Scheduling obtained by both algorithms is identical: 6 total time of scheduling.
The algorithms have found solutions immediately after their activation. Obtained scheduling is presented on the figure (Fig. 3.6):

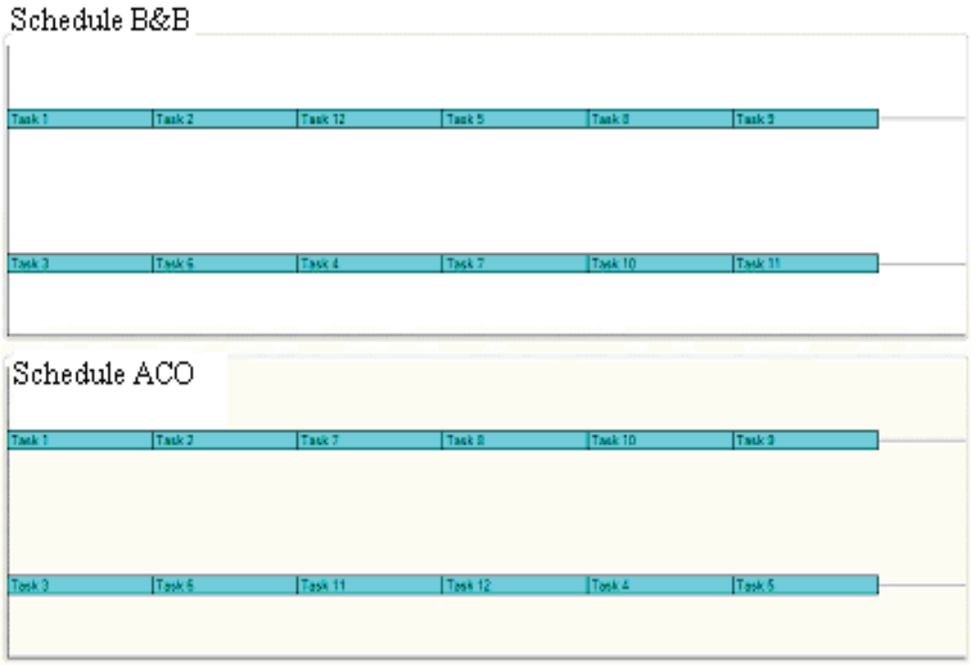


Fig. 3.6. Schedules - results of operations of algorithms

Example 3

Example from link STG (*Standard Graph Set:* task 000 with packet RNC50), [<http://www.kasahara.elec.waseda.ac.jp/schedule/index.html>].
Parameters of the problem:

- 50 dependent tasks about difference parameters.
- 2 identical, universal processors.
- Additional resources (memory, storage): without of constraints.

The algorithms have found solutions 15 minutes after their activation. Scheduling obtained by both algorithms is identical: schedule length: 131 units (optimum by STG, too).
Scheduling obtained by both algorithms is identical: 131 total time of scheduling, are presented on the figure (Fig. 3.7):

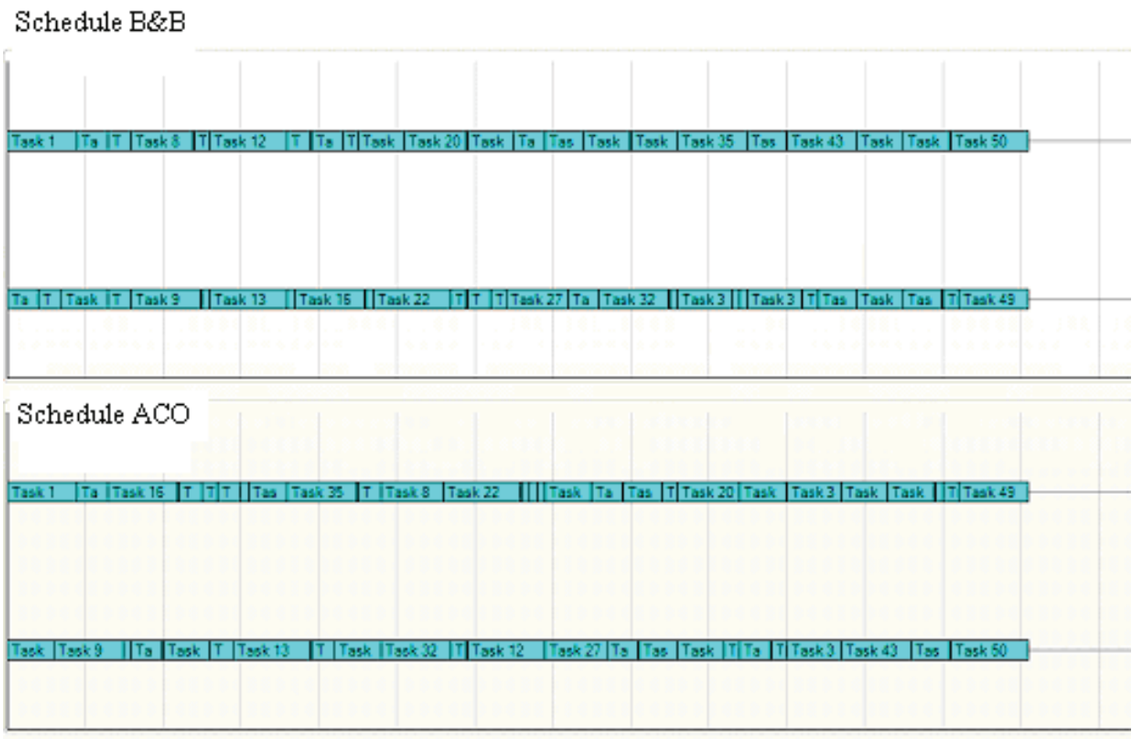


Fig. 3.7. Schedules - results of operations of algorithms

3.3.2 Verification of resources partition

Resources partition proposed by ACO and B & B algorithms were verified on the basis of the following examples.

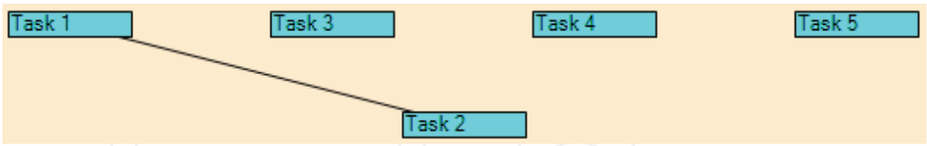
Example 1

Parameters of the problem:

- 5 tasks.
- 2 identical, universal processors.
- Additional resources: 3 units of memory, 3 units' storage.
- Parameters of tasks:

Tasks	Time	Memory	Storage
Task1	1	2	1
Task2	3	2	1
Task3	2	1	1
Task4	1	1	1
Task5	1	2	1

- Relations between tasks are shown on the figure:



The algorithms have found solutions immediately after their activation.
Scheduling obtained by both algorithms is identical: 5 total time of scheduling.
Obtained scheduling is presented on the figure (Fig. 3.8):

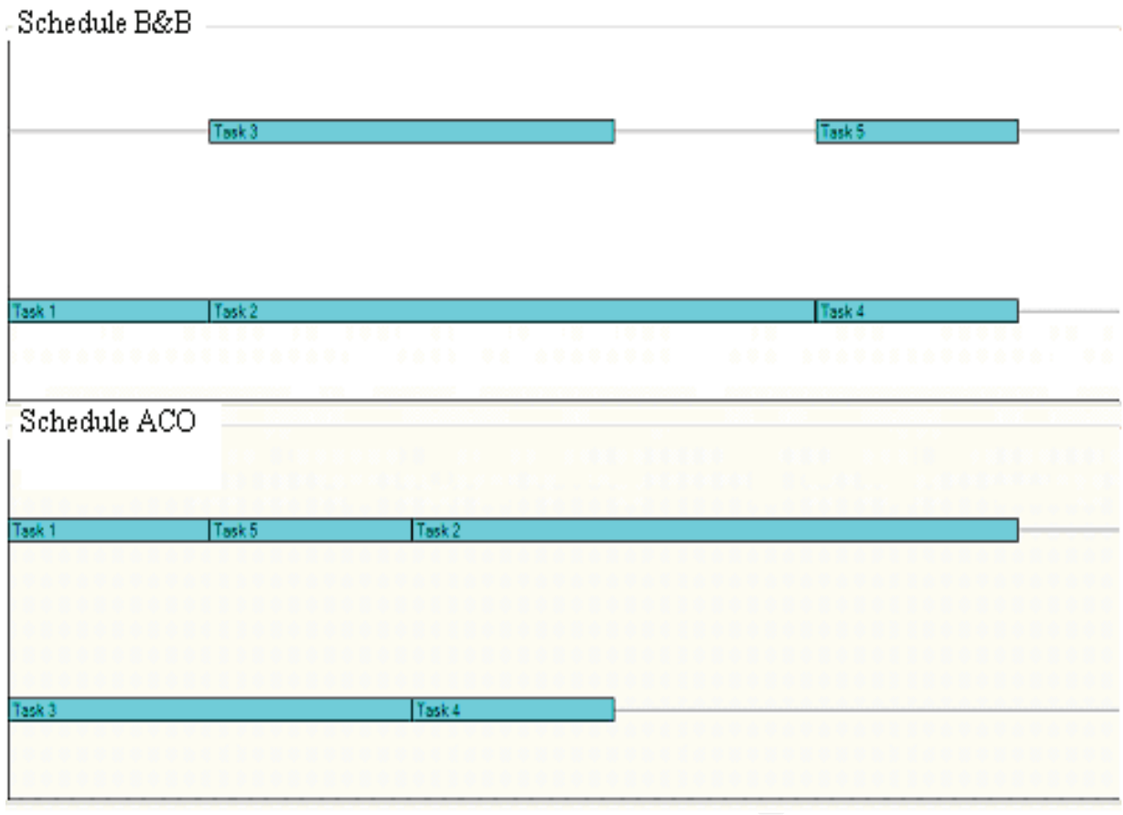


Fig. 3.8. Schedules - results of operations of algorithms

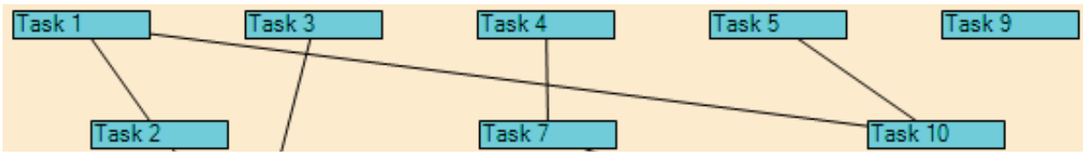
Example 2

Parameters of the problem:

- 10 tasks.
- 2 identical, universal processors.
- Additional resources: 3 units of memory, 3 units of storage.
- Parameters of tasks:

Tasks	Time constants	Memory	Storage
Task1	1	2	1
Task2	3	2	1
Task3	2	1	1
Task4	1	1	1
Task5	1	2	1
Task6	1	2	3
Task7	3	2	2
Task8	2	1	1
Task9	1	3	1
Task10	1	1	1

- Relations between tasks are shown on the figure:



The algorithms have found solutions immediately after their activation.
Scheduling obtained by both algorithms is identical: 10 total time of scheduling.
Obtained scheduling is presented on the figure (Fig 3.9):

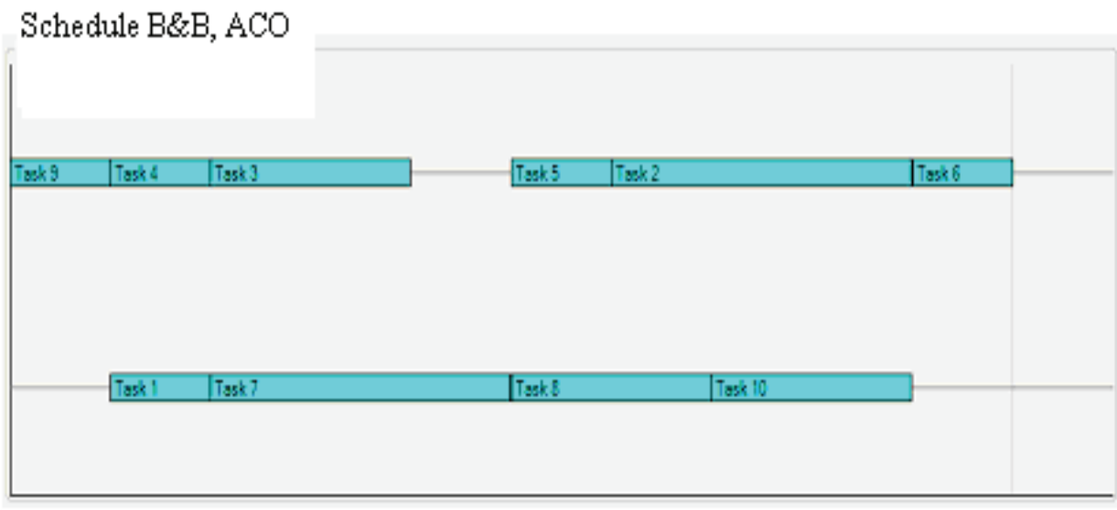


Fig. 3.9. Schedules - results of operations of algorithms

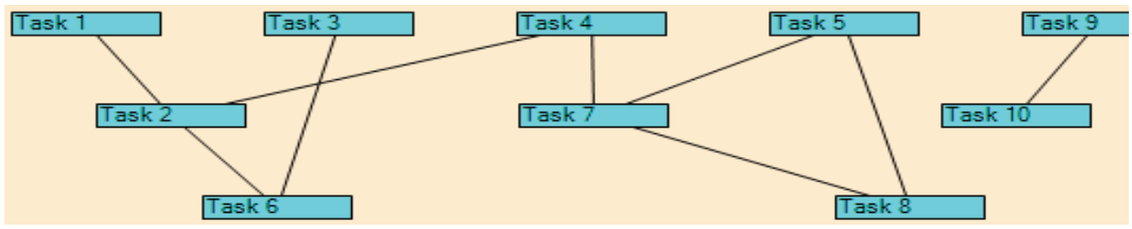
Example 3

Parameters of the problem:

- 10 tasks.
- 2 identical, universal processors.
- Additional resources: 3 units memory, 3 units storage.
- Parameters of tasks:

Tasks	Time constants	Memory	Storage
Task1	1	2	1
Task2	3	2	1
Task3	2	1	1
Task4	1	1	1
Task5	1	2	1
Task6	1	2	3
Task7	3	2	2
Task8	2	1	1
Task9	1	3	1
Task10	1	1	1

- Relations between tasks are shown on the figure:



The algorithms have found solutions immediately after their activation.
Scheduling obtained by both algorithms is identical: 10 total time of scheduling
Obtained scheduling is presented on the figure (Fig. 3.10):

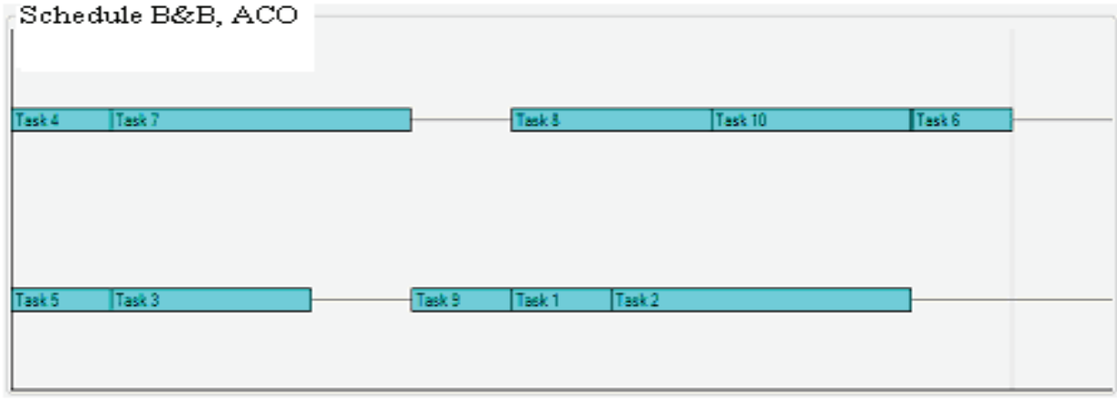


Fig. 3.10. Schedules - results of operations of algorithms

3.3.3 Comparison of coherent and non-coherent synthesis

Coherent synthesis is based on recurring division and scheduling tasks, in order to define the best set of hardware and scheduling for the system. As a result, the systems proposed by coherent synthesis may be better than the ones obtained as a result of incoherent synthesis (which makes division at the beginning of synthesis process) not only in relation to

optimized parameters, but also in general (eventually, the system can enable much faster tasks completion at the same or even lower energy consumption, etc.). The results obtained by coherent and incoherent synthesis will be presented on the basis of the following examples.

Example 1

- 25 independent tasks with different completion times.
- 3 identical processors.
- Criterion of optimization: power.

The time of algorithm operation until finding the solution, length of scheduling, cost and power consumption of the system as well as the quality of solution obtained as a result of coherent synthesis are presented in the table (Tab. 3.1).

Algorithm	Coherent				Non-coherent				ASS (%)
	Time	Length	Cost	Power	Time	Length	Cost	Power	
ACO	45.0	42.0	7.00	691.5	12.0	32.0	8.00	692.7	100.2
B&B	45.0	69.0	6.00	690.0	12.0	63.0	8.00	702.0	101.7

Table 3.1. Results of coherent and non-coherent synthesis – Example 1

Systems obtained as a result of coherent synthesis consume less energy and are cheaper. In the case of B&B algorithm, system obtained as a result of coherent synthesis is generally better than the one obtained by incoherent synthesis (assessment = 108.8%).

Example 2

- 25 independent tasks with different completion times.
- 3 identical processors.
- Criterion of optimization: cost.

The time of algorithm operation until finding the solution, length of scheduling, cost and power consumption of the system as well as the quality of solution obtained as a result of coherent synthesis are presented in the table (Tab. 3.2).

Algorithm	Coherent				Non-coherent				ASS (%)
	Time	Length	Cost	Power	Time	Length	Cost	Power	
ACO	2.0	45.0	7.00	692.1	3.0	38.0	8.00	694.5	114.3
B&B	2.0	69	6.00	690.0	3.0	65	8.00	702.6	133.3

Table 3.2. Results of coherent and non-coherent synthesis – Example 2

Similarly how in previous case, systems for coherent synthesis are clearly cheaper and quicker.

Example 3

- 25 identical, independent tasks.
- 5 identical processors.
- Criterion of optimization: cost.

The time of algorithm operation until finding the solution, length of scheduling, cost and power consumption of the system as well as the quality of solution obtained as a result of coherent synthesis are presented in the table (Tab. 3.3).

Algorithm	Coherent				Non-coherent				ASS (%)
	Time	Length	Cost	Power	Time	Length	Cost	Power	
ACO	12.5	28.0	4.00	500.6	4.5	20.0	8.00	505.0	200.0
B&B	12.5	50.0	2.00	500.0	4.5	50.0	6.00	520.0	300.0

Table 3.3. Results of coherent and non-coherent synthesis – Example 3

In presented examples is visible the considerable superiority of coherent synthesis with non-coherent. Except improvement of the costs, the power consumption improved also. The larger number of processors was eliminated as well as the demand lowered of memory and storage too. In result of the assessment of system for algorithm the ACO is equal 124.1 % and for algorithm B & B is equal 168.0 %.

Example 4

- 25 identical, independent tasks.
- 5 identical processors.
- Criterion of optimization: power consumption.

The time of algorithm operation until finding the solution, length of scheduling, cost and power consumption of the system as well as the quality of solution obtained as a result of coherent synthesis are presented in the table (Tab. 3.4).

Algorithm	Coherent				Non-coherent				ASS (%)
	Time	Length	Cost	Power	Time	Length	Cost	Power	
ACO	8.5	10.0	10.00	500.0	29.5	10.0	10.00	500.0	100.0
B&B	8.5	50.0	2.00	500.0	29.5	44.0	7.00	517.0	103.4

Table 3.4. Results of coherent and non-coherent synthesis – Example 4

Systems for coherent synthesis are clearly cheaper and quicker. The difference is visible in case of algorithm B&B: the assessment of solution for coherent synthesis is higher though the assessment of proposed solutions in both cases is considerably worse than in case of solutions proposed by algorithm the ACO (206.7 %).

Example 5

- 25 identical, independent tasks.
- 5 unrelated processors.
- Criterion of optimization: power consumption.

The time of algorithm operation until finding the solution, length of scheduling, cost and power consumption of the system as well as the quality of solution obtained as a result of coherent synthesis are presented in the table (Tab. 3.5).

Algorithm	Coherent				Non-coherent				ASS (%)
	Time	Length	Cost	Power	Time	Length	Cost	Power	
ACO	87.5	50.0	2.00	500.0	14.0	48.0	7.00	539.0	107.8
B&B	87.5	50.0	2.00	500.0	14.0	50.0	6.00	520.0	104.0

Table 3.5. Results of coherent and non-coherent synthesis – Example 5

Algorithm ACO for coherent synthesis finds good solution, better than solution for non-coherent. We have again the superiority of coherent synthesis. Solutions for non - coherent synthesis are weak, assessment 75% for ACO as well as 76% for B & B.

3.3.4 Optimization of scheduling length and cost

Optimizing two aspects of system is much more difficult for the algorithms than minimizing a single parameter. As a condition of the choice we can take the assessment of obtained solution. The set of resources used for the optimization of the time and cost of system

- Memory (max. 100, cost 1mpower/unit).
- Storage (max. 100, cost 1spower[/unit).
- Processors, cost 1ppower/unit.

Type	Speed	Power consumption (action)	Power consumption (idle)
Processor 1	1	100	10
Processor 2	2	120	12
Processor 3	4	150	15
Processor 4	8	200	20
ASIC 1	1	80	8
ASIC 2	2	110	11
ASIC 3	4	150	15
ASIC 4	8	180	18

Time, which has passed until solution was found and the parameters of the target system are presented in the table (Tab. 3.6).

Number of tasks	Ant Colony				Branch & Bound				ASS (%)
	Time	Length	Cost	Power	Time	Length	Cost	Power	
20	59.0	10.0	15.00	4007	≥ 59.0	6.0	30.50	4173	131.7
25	60.0	9.0	20.50	5054	≥ 60.0	7.9	30.51	5281	118.3
30	12.5	11.3	19.00	6057	≥ 12.5	10	30.51	6394	124.5
35	16.0	12.5	19.00	7004	≥ 16.0	11.3	30.51	7448	125.4
40	15.5	15.0	19.00	8010	≥ 15.5	13.5	30.51	8568	125.3
45	42.5	16.5	19.00	9011	≥ 42.5	15	30.51	9654	125.7
50	26.5	18.0	19.00	10024	≥ 26.5	16.5	30.51	10693	126.1
55	34.5	20.0	19.00	11009	≥ 34.5	18	30.51	11772	125.3
60	44.0	21.4	19.00	12003	≥ 44.0	20	30.51	12872	127.0

Table 3.6. The results of schedule length of tasks and cost optimization

In the multi-objective optimization it is clear that ACO algorithm exceeds the greedy algorithm B & B in relation to the quality of solutions: solutions proposed by ACO algorithm are better than the ones proposed by B & B algorithm even by 31.7%. Interesting are the graphs presenting solutions, which were found by ACO algorithm before the final result was obtained (Chart 3.1). As a result of such operation of algorithm, the quality (comparing to the first solution found) was changing as follows (Chart 3.2).
Apart from better quality of the solution it proposed by ACO algorithm, we should notice that the total quality of the system is also very high (Chart 3.3).

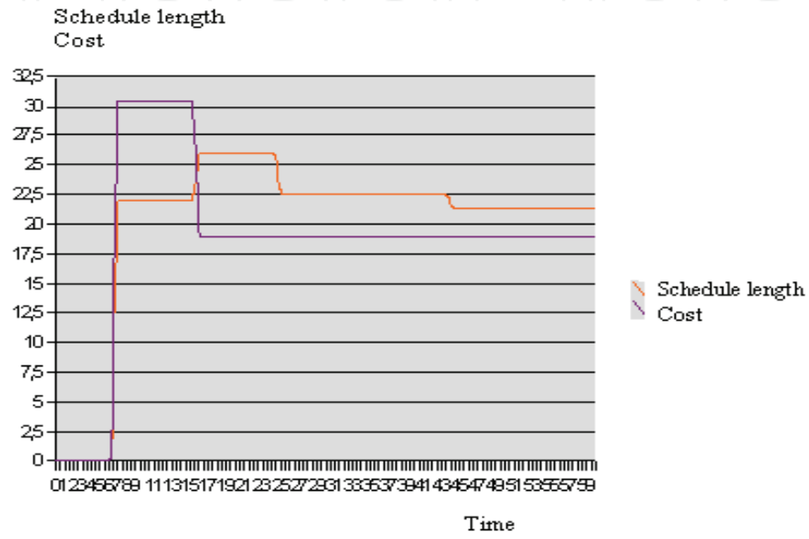


Chart 3.1. The change of cost and scheduling length in time

3.3.5 Optimization of power consumption and cost

The set of resources used for the optimization of the time and cost of system

- Memory (max. 100, cost 1 mpower/unit).
- Storage (max. 100, cost 1 spower/unit).

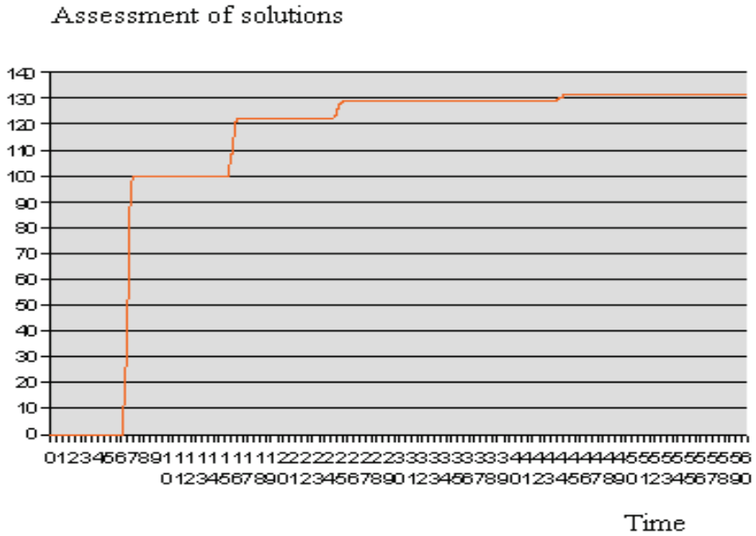


Chart 3.2. Improvement of the assessment of consecutive solutions in time function

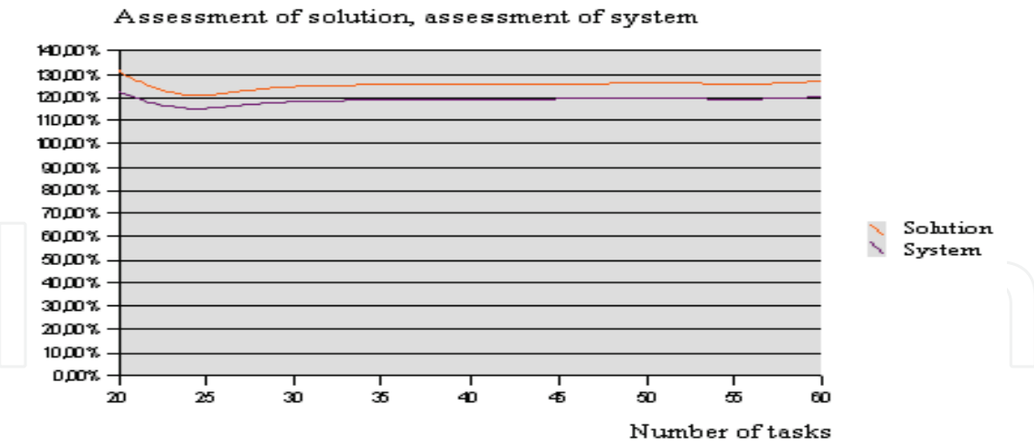


Chart 3.3. The assessment of proposed solutions and systems in relation to the number of tasks

- Processors, cost 1 ppower/unit.

Type	Speed	Power consumption (action)	Power consumption (idle)
Processor 1	1	100	10
Processor 2	2	120	12
Processor 3	4	150	15
Processor 4	8	200	20
ASIC 1	1	80	8
ASIC 2	2	110	11
ASIC 3	4	150	15
ASIC 4	8	180	18

Time, which has passed until solution was found and the parameters of the target system are presented in the table (Tab. 3.7).

Number of tasks	Ant Colony				Branch & Bound				ASS (%)
	Time	Length	Cost	Power	Time	Length	Cost	Power	
10	0.5	20.0	2.00	2000	≥ 0.5	20.0	2.00	2000	100.0
15	3.5	30.0	2.00	3000	≥ 3.5	30.0	2.00	3000	100.0
20	4.5	18.0	5.00	3780	≥ 4.5	40.0	2.00	4000	72.9
25	10.0	22.0	5.00	4732	≥ 10.0	50.0	2.00	5000	72.8
30	12.5	27.0	5.00	5670	≥ 12.5	60.0	2.00	6000	72.9
35	12.0	20.0	10.00	6677	≥ 12.0	70.0	2.00	7000	62.4
40	27.0	28.0	10.00	7869	≥ 27.0	80.0	2.00	8000	60.8
45	16.5	33.0	10.00	8835	≥ 16.5	90.0	2.00	9000	60.9
50	57.5	32.0	10.00	9673	≥ 57.5	100.0	2.00	10000	61.7
55	43.5	38.0	10.00	10766	≥ 43.5	110.0	2.00	11000	61.1
60	55.5	37.5	11.50	11822	≥ 55.5	120.0	2.00	12000	59.4

Table 3.7. The results of power consumption and system cost optimization

This example illustrates that ACO algorithm isn't better than greedy algorithms for all kinds of problems. The reason of such weak results is a very difficult choice for the algorithm between power and cost. To illustrate the problem we will try to analyze the scheduling of three first tasks (Drabowski, 2007). Even scheduling the first task causes some kind of dilemma: you can do this cheaper, but the scheduling will be longer and at the same time more power consuming, or you can do this at the higher cost, but with less power consumption (on the faster processor, the task will be completed sooner). If the algorithm chooses the second processor – the choice of slower processor in the next step will turn out more expensive as well as more demanding, while staying with the faster one will let us keep the same cost and limit the power (comparing to the slower processor). Also scheduling time will reduce significantly (what was presented in the table above) (Drabowski & Wantuch, 2006). The final quality of the system is then difficult to determine during the whole cycle – it is possible to determine only when you know the total scheduling length (and thus the power consumed by system, in other words – after the end of the whole cycle).

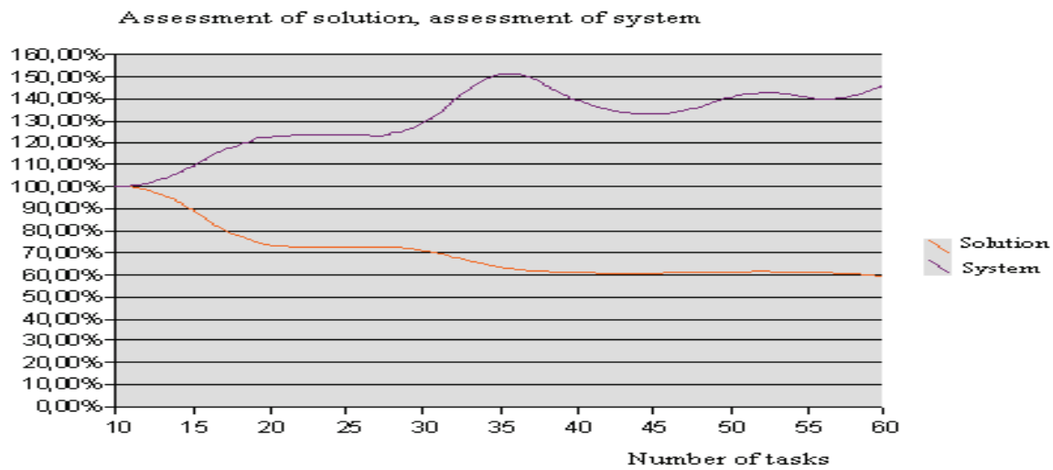


Chart 3.4.The assessment of solution and the systems proposed in relation to the number of tasks

When the number of tasks grows, the quality of solution decreases more and more, but you cannot say the same about the quality of system; the ACO algorithm shows, that at the higher expenditure you can obtain solution which is economical and fast at the same time (Drabowski, 2009). The graphs illustrating the quality of solution and system are presented on the chart (Chart 3.4).

4. Conclusions

We may say, basing on the above research, that the ACO algorithm is better suitable for both one- and multi-objective analyses of optimization of computer systems. Furthermore, the use of coherent analysis significantly improved the quality of obtained solutions. In the case of multi-objective synthesis, heuristic algorithm gave comparable results for optimized parameters and at the same time, the final grade of the systems it proposed was much better. The computational experiments prove the superiority of coherent synthesis over the incoherent synthesis and heuristic algorithms over the greedy ones.

Solutions of this method are better both, for their cost, as and of time of executing the tasks and of optimization of multi-criterions.

5. References

D'Ambrioso J., Hu X., (1994), *Configuration Level Hardware/Software Partitioning for Real-Time Systems,* Proc. of the Int. Workshop on Hardware/Software Codesign, Vol. 14, 34-41.

Gajski D., (1997), *Principles of Digital Design*, Prentice Hall, Upper Saddle River, NJ.

Sgroi M., Lavagno L., Sangiovanni-Vincentelli A., (2000), *Formal Models for Embedded System Design, IEEE Design&Test of Computers,* Vol. 17, No. 2, 14-27.

Gupta R.K., De Micheli G., (1993), *Hardware-Software Co-synthesis for Digital Systems,* IEEE Design&Test of Computers, Vol. 10, No. 3, 29-41.

De Micheli G., (1994), *Computer-Aided hardware-Software Codesign,* IEEE Micro, Vol. 14, No. 4, pp. 10-24.

Axelson J., (1997), *Architecture Synthesis and Partitioning of Real-Time Systems: A Comparison of Three Heuristic Search Strategies,* Proc. of the Int. Workshop on Hardware/Software Codesign, 161-165.

Schulz S., Rozenbilt J.W., Mrva M., Buchenrieder K., (1998), *Model-Based Codesign*, IEEE Computer, Vol. 31, No. 8, 60-67.

Steinhausen U., (1993), *System Synthesis Using Hardware/Software Codesign*, Proc. of the Int. Workshop on Hardware-Software Co-Design.

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

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

Węglarz J., (1999), *Project Scheduling – Recent Models, Algorithms and Applications,* Kluwer Academic Publ.

Błażewicz J., Ecker K., Plateau B., Trystram D., (2000), *Handbook on Parallel and Distributed Processing,* Spinger-Verlag, Berlin – Heidelberg.

Drabowski M., M., Rola M, Roślicki A., (2002), Algorithm in control of allocation tasks and resources in frameworks, 2rd International Congress of Intelligent Building Systems, INBus2002, Kraków, 45-52.

Drabowski M., (2008), *Par-synthesis of multiprocessors parallel systems,* International Journal of Computer Science and Network Security, Vol. 8, No. 10, 90-96.

Harel D., (1987), *Statecharts: A Visual Formalism for Complex Systems,* Science of Computer Programming, Vol. 8, No. 3, 231-274.

Drabowski M., Czajkowski K., (2005), *Task scheduling in coherent, co-synthesis of computer system,* Advanced Computer Systems – Computer Information Systems and Industrial Management Application (ACS-CISIM 2005), in. Image Analysis, Computer Graphics, Security Systems and Artificial Intelligence Applications, vol. 1, 53-60.

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

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

Mitten L.G., (1970), Branch-and-bound methods: general formulation and properties, Oper. Res. 18, 24-34.

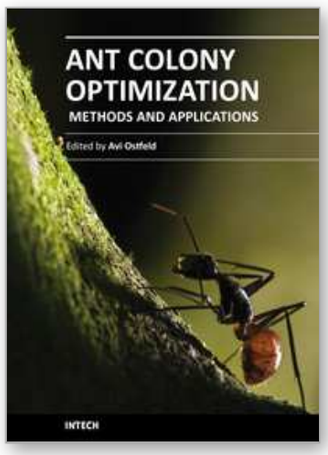
Drabowski M., (2007), *Coherent synthesis of heterogeneous system – an ant colony optimization approach,* Studia Informatica, vol. 2/9, 9-18.

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

Drabowski M., (2007), *An Ant Colony Optimization to scheduling tasks on a grid,* Polish Journal of Environmental Studies, vol. 16, No. 5B, 16-21.

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, vol. 13, no. 1. World Scientific Publishing, 15-24.

Drabowski M., (2009), *High-level synthesis of self-testing parallel multiprocessors computer systems*, in: Proceedings of the IASTED International Conference on Computational Intelligence, Honolulu, USA, ACTA Press, Anaheim, USA, 127-133.



Ant Colony Optimization - Methods and Applications

Edited by Avi Ostfeld

ISBN 978-953-307-157-2
Hard cover, 342 pages
Publisher InTech
Published online 04, February, 2011
Published in print edition February, 2011

Ants communicate information by leaving pheromone tracks. A moving ant leaves, in varying quantities, some pheromone on the ground to mark its way. While an isolated ant moves essentially at random, an ant encountering a previously laid trail is able to detect it and decide with high probability to follow it, thus reinforcing the track with its own pheromone. The collective behavior that emerges is thus a positive feedback: where the more the ants following a track, the more attractive that track becomes for being followed; thus the probability with which an ant chooses a path increases with the number of ants that previously chose the same path. This elementary ant's behavior inspired the development of ant colony optimization by Marco Dorigo in 1992, constructing a meta-heuristic stochastic combinatorial computational methodology belonging to a family of related meta-heuristic methods such as simulated annealing, Tabu search and genetic algorithms. This book covers in twenty chapters state of the art methods and applications of utilizing ant colony optimization algorithms. New methods and theory such as multi colony ant algorithm based upon a new pheromone arithmetic crossover and a repulsive operator, new findings on ant colony convergence, and a diversity of engineering and science applications from transportation, water resources, electrical and computer science disciplines are presented.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen