Ant Colony and Branch & Bound algorithms in coherent synthesis of computer system

Mieczysław Drabowski

Cracow Univrsity of Technology Warszawska 24 31-155 Kraków, POLAND

Summary
The paper presents a coherent approach to solving the problems of computer system synthesis based on Ant Colony Optimization methods. We describe algorithm realizations aimed to optimize resource selection and task scheduling, as well as the adaptation of this algorithm for coherent co-synthesis realization. This is approach, which we called a par-synthesis [1]. We then present selected analytical experiments proving the correctness of the par-synthesis concept and indicate its practical motivations.

Key words:
Synthesis, Ant Colony Optimization, Branch & Bound, task scheduling, resource selection

1. Introduction

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.

2. 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 [2], [3]. 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. 1).

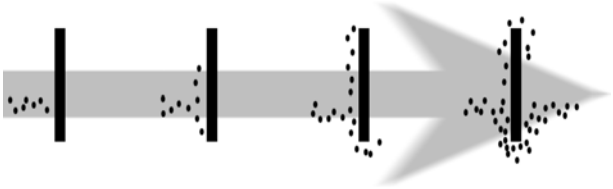


Fig. 1. The idea of algorithm – overcoming the obstacle by ants

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 partition of tasks is important, but also their schedule.
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:

Manuscript received September 5, 2010
Manuscript revised September 20, 2010

▪ 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 instance of problem. This way, for parameters:

- α > β there is bigger influence on the choice of path, which is more often exploited,
- α < β there is bigger influence on the choice of path, which offers better solution,
- α = β there is balanced dependency between quality of the path and degree of its exploitation,
- α = 0 there is a heuristics based only on the quality of passage between consecutive points (ignorance of the level of pheromone on the path),
- β = 0 there is a heuristics based only on the amount of pheromone (it is the factor of path attendance),
- α = β = 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 [4]:

$$p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k}[\tau_{ij}]^{\alpha}[\eta_{ij}]^{\beta}} & when\ \mathrm{j} \in N_i^k \\ 0 & \text{else} \end{cases} \quad (6.1.)$$

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.

The job of agent involves (Fig. 2).

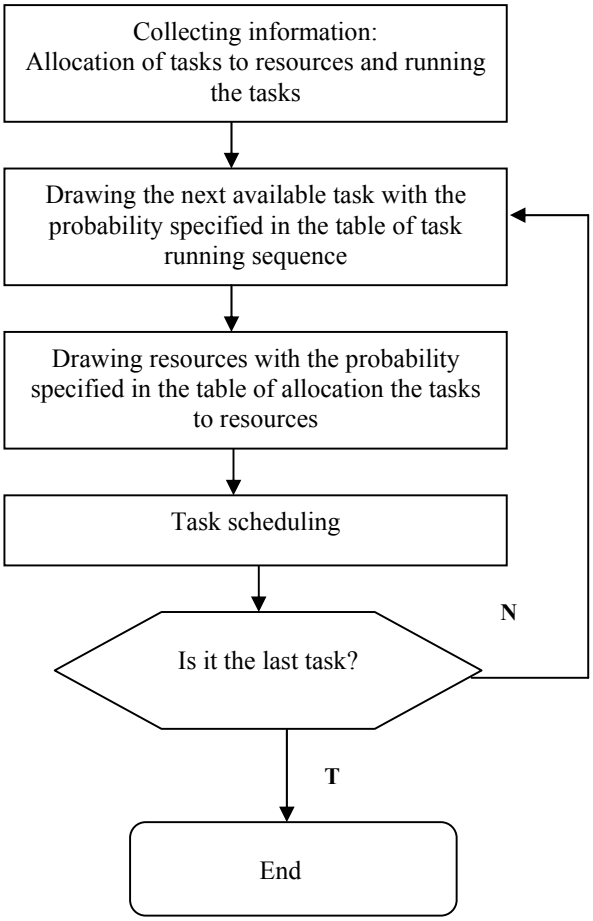


Fig. 2.Agent operation scheme

To evaluate the quality of allocation the task to processor, the following method is being used (Fig. 3).

The computational 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. 4).

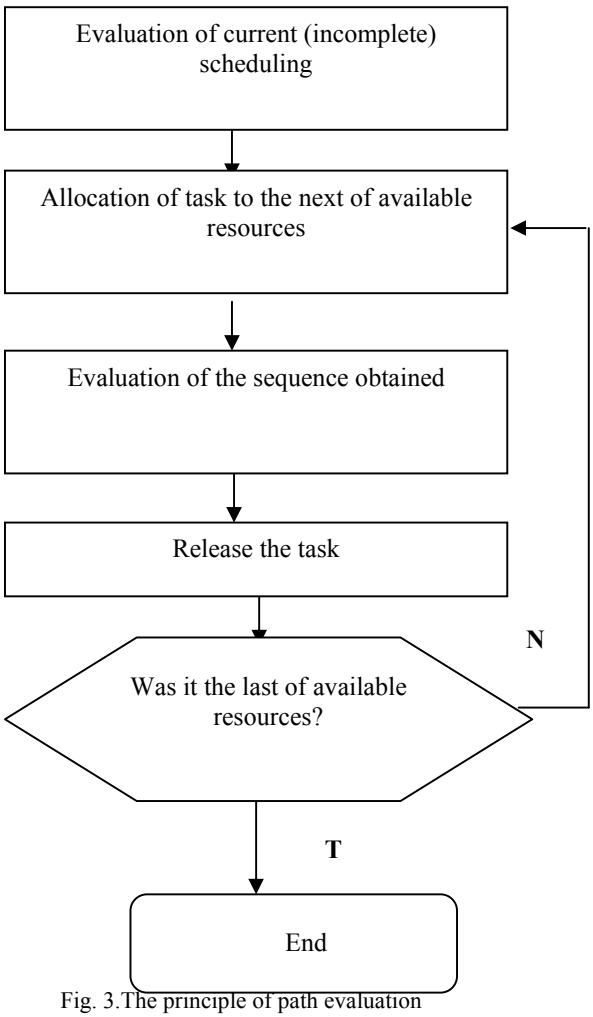


Fig. 3.The principle of path evaluation

3. Customization of the 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 [5], [6]. 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 schedule 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. B&B algorithm is relatively simple, but 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!

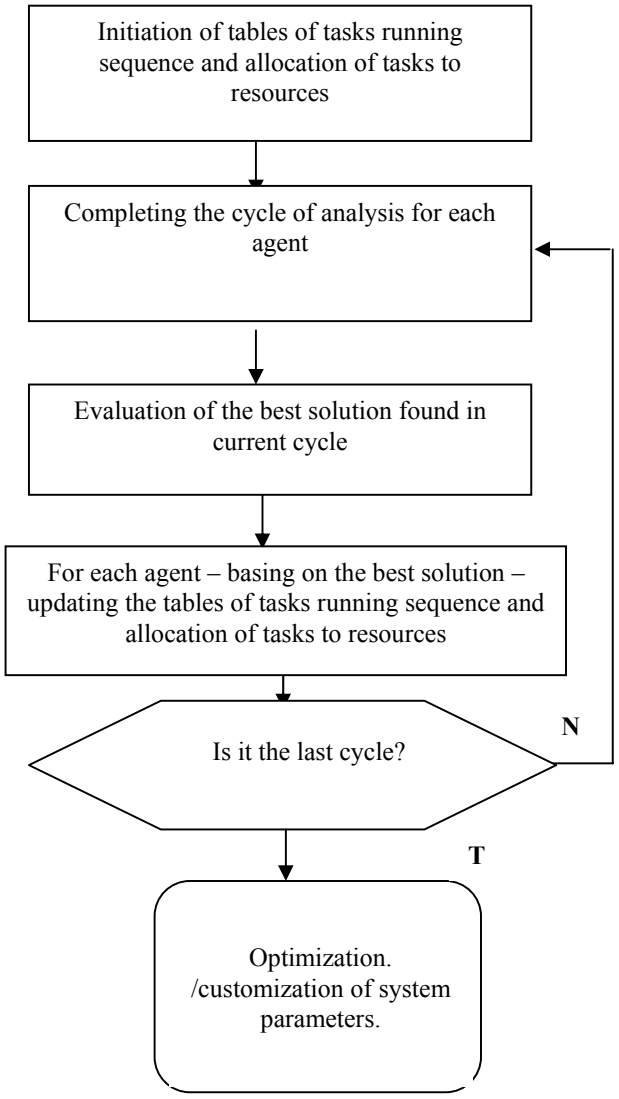


Fig. 4.The principle of ACO algorithm operation

4. 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. 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 ACO algorithm will be graded on the basis of solutions found by Branch & Bound algorithm.
Formula for the assessment of obtained solution is following [4]:

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

(Formula 6.2.)

Formula 2. Assessment (AS) of solutions

The final grade is influenced only by these parameters, which were being optimized by algorithms: cost, power and schedule length (speed). The assessment of proposed system includes all three parameters (schedule length, 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.

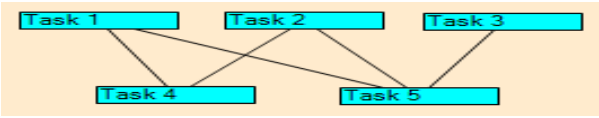
4.1. Scheduling of tasks

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.

Example 1

Parameters of the problem:

- 5 tasks: (Task1: 1, 0), (Task2: 1, 0), (Task3: 2, 0), (Task4: 1, 0), (Task5: 1, 0),
- 2 identical, universal processors,
- additional resources (memory, storage): without of constraints.
- 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.

Obtained scheduling is presented on the figure (Fig. 5):

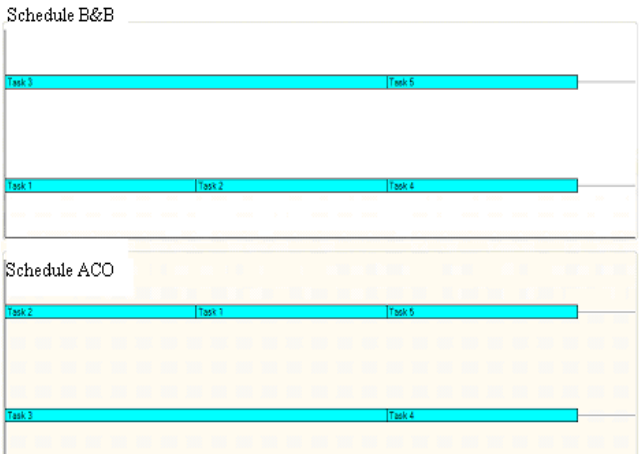
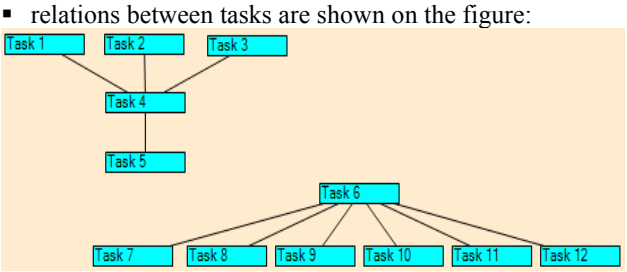


Fig5.Schedules - results for example 1

Example 2

Parameters of the problem:

- 12 identical tasks UET (time equal 1unit); Unit Execution Tasks
- 2 identical, general processors,
- additional resources (memory, storage): without of constraints,
- relations between tasks are shown on the figure:



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

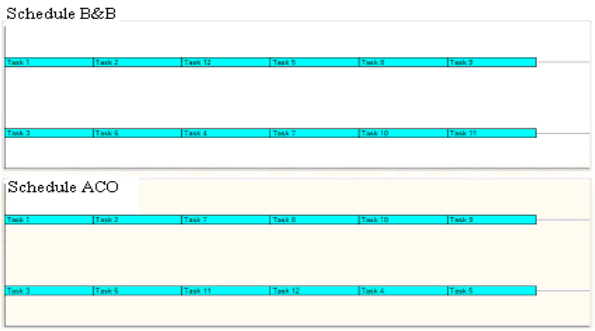
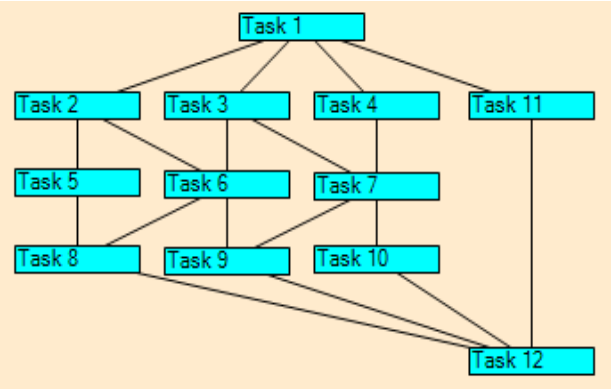


Fig. 6.Schedules; results for example 2

Example 3
Parameters of the problem:
- 12 task: (Task1: 1, 0), (Task2: 1, 0), (Task3: 7, 0), (Task4: 3, 0), (Task5: 1, 0), (Task6: 1, 0), (Task7: 3, 0), (Task8: 2, 0), (Task9: 2, 0), (Task10: 1, 0), (Task11: 3, 0), (Task12: 1, 0)
- 2 identical, general processors,
- additional resources (memory, storage): without of constraints,
- relations between tasks are shown on the figure:



Scheduling obtained by both algorithms is identical: 14 unit - time of scheduling. Obtained scheduling is presented on the figure (Fig. 7):

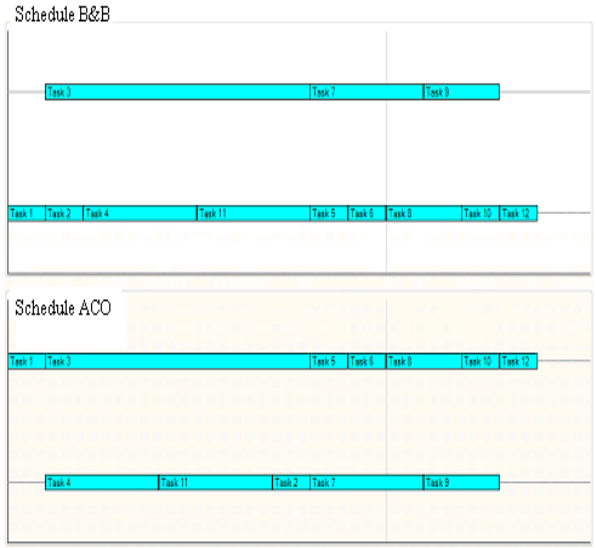


Fig. 7.Schedules - results for example 3

Example 4
Example from link STG (*Standard Graph Set:* task 000 RNC50) [7].
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 unit (optimum by STG, too). Schedules are presented on the figure (Fig. 8):

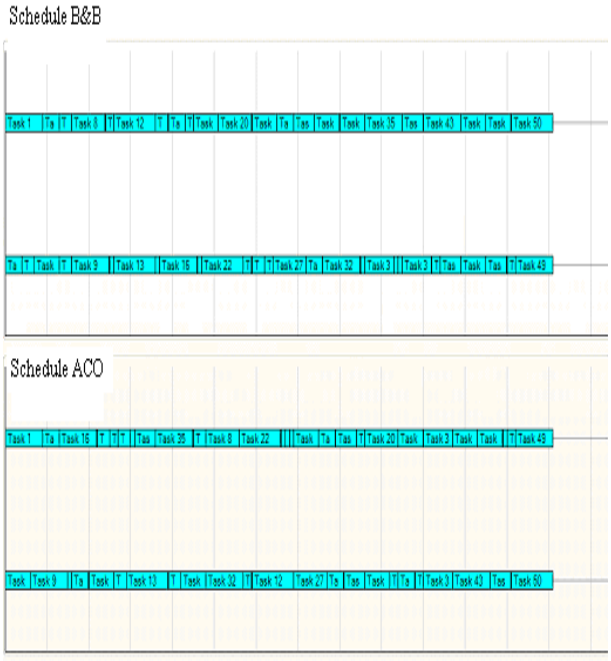


Fig. 8.Schedules - results for example 4

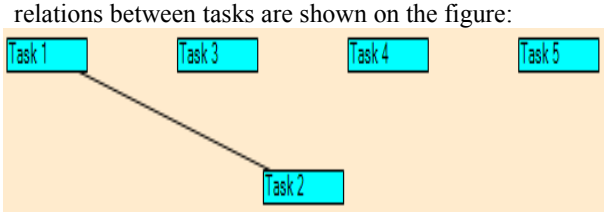
4.2. Partition of resources

Solves of resources partition problems 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, general processors,
- additional resources: 3 units of memory, 3 unit's 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 optimum solution immediately after their activation. Schedules obtained by both algorithms are identical: 5 unit schedule length, 3 unit memory, 3 unit storage. Obtained scheduling is presented on the figure (Fig. 9):

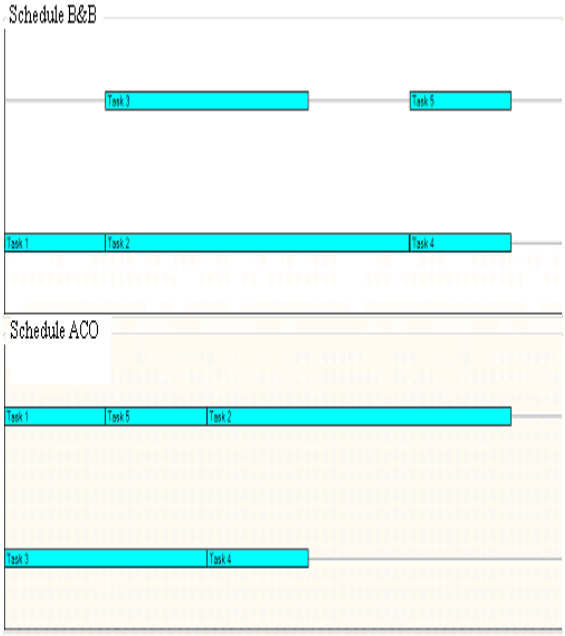


Fig. 9.Schedules - results for example1

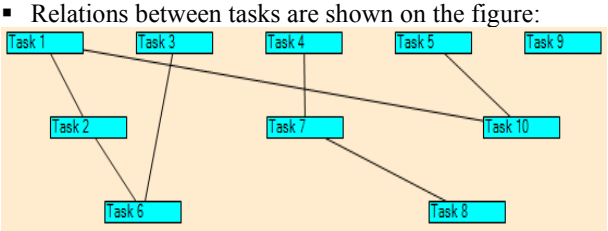
Example 2

Parameters of the problem:

- 10 tasks,
- 2 identical, general processors,
- additional resources: 3 units of memory, 3 units of 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
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:



Schedules obtained by both algorithms are identical: 10 unit schedule length, 3 unit memory, 3 unit storage. Obtained scheduling is presented on the figure (Fig. 10):

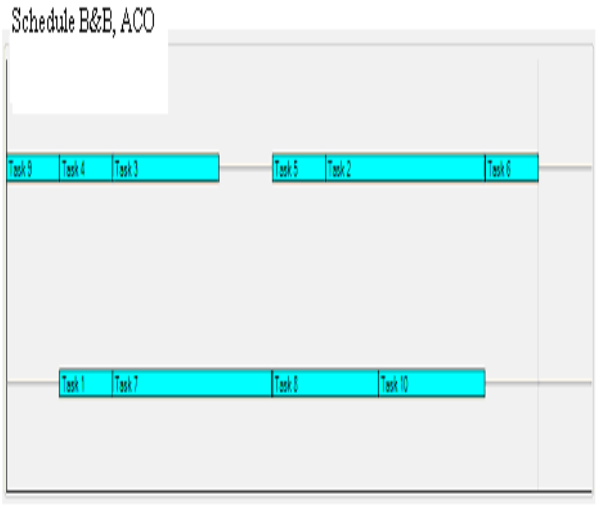


Fig. 10.Schedules - results for example 2

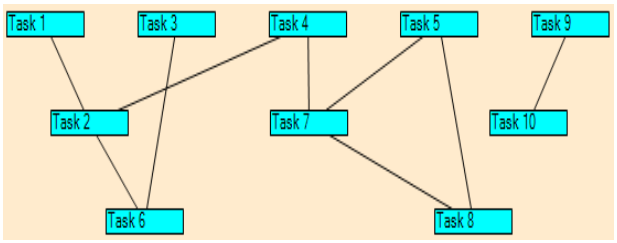
Example 3

Parameters of the problem:

- 10 tasks,
- 2 identical, general processors,
- additional resources: 3 unit's memory, 3 unit's 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
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. Schedules obtained by both algorithms are identical: 10 unit schedule length, 3 unit memory, 3 unit storage. Obtained scheduling is presented on the figure (Fig. 11):

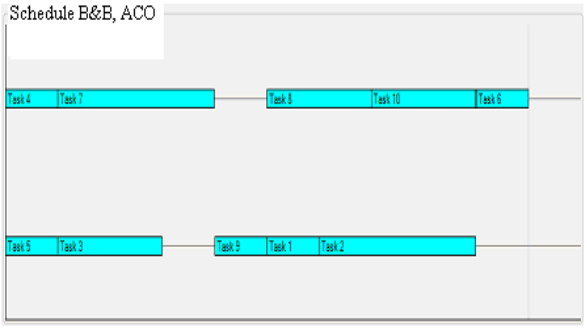


Fig. 11.Schedules - results of operations of algorithms for example 3

Example 4

Parameters of the problem:

- 25 tasks,
- 3 identical, general processors,
- additional resources: 5 units of memory, 5 units of storage.
- parameters of tasks:

Tasks	Time	Memory	Storage
Task1	1	3	2
Task2	3	2	4
Task3	3	2	2
Task4	5	4	1
Task5	2	1	4
Task6	4	2	2
Task7	1	2	3
Task8	2	5	1
Task9	3	0	0
Task10	3	0	3
Task11	1	3	4
Task12	10	1	1
Task13	1	3	2
Task14	3	2	1
Task15	3	0	1
Task16	4	2	4
Task17	3	1	1
Task18	5	1	1
Task19	1	2	3
Task20	1	2	2
Task21	4	1	4
Task22	1	3	1
Task23	3	1	3
Task24	1	2	2
Task25	1	1	4

Algorithms presented in a schedule in time till 15 minutes from starting. Algorithm B&B did not find in this time to find optimum. It following results was received was:

- schedule length: 33 unit for B & B, 30 unit for ACO.

Obtained scheduling is presented on the figure (Fig 12):

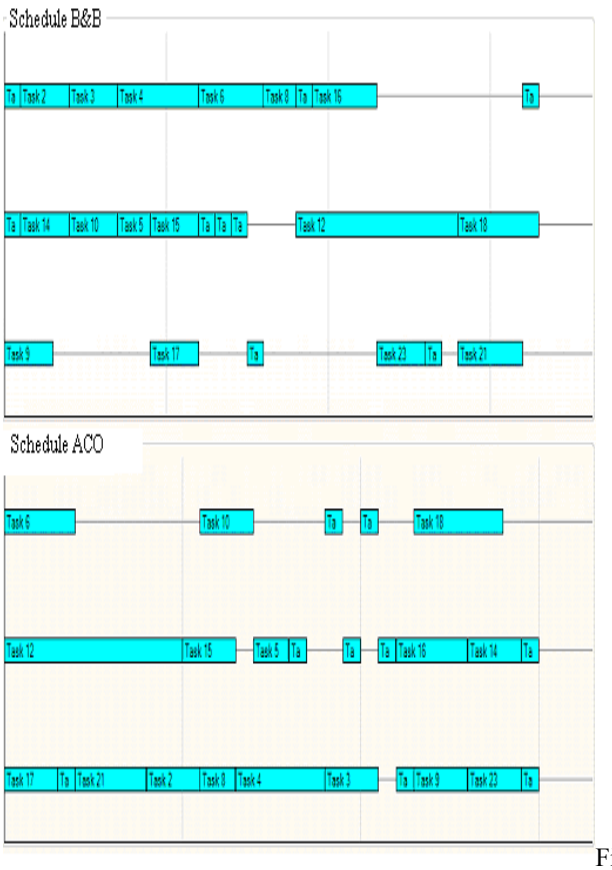


Fig. 12. Schedules - results for example 4

4.3. 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. 1).

Tab.1. Results of coherent and non-coherent synthesis: Example 1

Algorithm	Coherent				Non-coherent				AS
	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

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. 2).

Tab. 2. Results of coherent and non-coherent synthesis – Example 2

Algorithm	Coherent				Non-coherent				AS
	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

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).

Tab. 3. Results of coherent and non-coherent synthesis – Example 3

Algorithm	Coherent				Non-coherent				AS
	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

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. 4).

Tab. 4. Results of coherent and non-coherent synthesis – Example 4

Algorithm	Coherent				Non-coherent				AS
	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

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. 5).

Tab. 5. Results of coherent and non-coherent synthesis – Example 5

Algorithm	Coherent				Non-coherent				AS %
	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

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.

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.

Acknowledgments

This work was supported by the Polish Ministry of Science as a 2007-2010 research project.

References

[1] Drabowski M., (2008), Par-synthesis of multiprocessors parallel systems, International Journal of Computer Science and Network Security, Vol. 8, No. 10, 90-96.
[2] Blum C., (2005), Beam-ACO – Hybridizing ant colony optimization with bean search: An application to open shop schedling, Comput. Oper. Res. 32, 1565-1591.
[3] Montgomery J., Fayad C., Petrovic S., (2006), Solution representation for job shop scheduling problems in ant colony optimization, LNCS 4150, 484-491.
[4] 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.
[5] Mitten L.G., (1970), Branch-and-bound methods: general formulation and properties, Oper. Res. 18, 24-34.
[6] 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.
[7] http://www.kasahara.elec.waseda.ac.jp/schedule/index.html

Mieczyslaw Drabowski, Assistant Professor of Department of Computer Engineering, Faculty of Electrical and Computer Engineering, Cracow University of Technology, received the M. Sc. degree in automatic control and communication from AGH University of Science and Technology, graduated mathematic from Jagiellonian University in Krakow and received the Ph. D. degree (with honors) in computing science from Poznan University of Technology, in 1977, 1979 and 1986, respectively.
Currently he is member of several editorial boards, among others Scientific Journals International, International Association for Development of the Information Society (IADIS), and International Association of Science and Technology for Development (IASTED) on Artificial Intelligence and Soft Computing.
His research interests include schedule, assignment and allocation for tasks and resources, dependable and fault tolerant systems, artificial intelligence, operating systems and software engineering, author and co-author of 3 monographs and over 60 papers in major professional journals and conference proceedings.
Dr. Drabowski is a member of the council of the Polish Information Processing Society.