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

Coherent synthesis of heterogeneous system – an ant colony optimization approach

Mieczyslaw Drabowski[1]
[1] Faculty Electrical and Computer Engineering,
Cracow University of Technology,
Warszawska 24, Krakow 31–155, Poland

Abstract. The paper presents an innovative approach to solving the problems of computer system synthesis based on ant colony optimization method. We describe algorithm realizations aimed to optimize resource, selection and task scheduling, as well as the adaptation of those algorithms for coherent synthesis realization. We then present selected analytical experiments proving the correctness of the coherent synthesis concept and indicate its practical motivations.

Keywords. Synthesis, scheduling, task, selection, resource coherent, ant colony, branch & bound

1 Introduction

The synthesis of computer systems consists in the selection of hardware (choice of resources) and allocation of specific tasks (functions of the system) to these resources, in order to find the parameters both of hardware and software, which give the best solutions of the synthesis. It is, depending on the desired criterion, either the shortest time of completion of all tasks, or the cheapest (the most economical) realization of those tasks. You may think of the synthesis as an answer to the question – what would the parameters of hardware have to be, for the problem described by the graph of tasks to be performed fastest or cheapest?

You can assume that each task performed by resources is dependent on the previous tasks and enables execution of the following tasks. Tasks are presented as directed acyclic graphs, each vertex of which represents the task and the edges – interrelations.

The problem of computer systems synthesis requires the list of available hardware resources – the set describing resources – which can be used for the synthesis of target system. This set may contain such resources as for example:

- processors (general and dedicated),
- operating memory,
- mass storage,
- communication ports, etc.

The process of synthesis is based on iterative procedures execution:

- system analysis,
- resources partitioning,
- tasks scheduling
- solution evaluation,
- system parameters modification.

Thanks to this approach, we are able to analyze in progress every solution we obtain, while the algorithm is trying to find solutions which give better results than the current ones. Furthermore, you may "teach" the algorithm to choose only these solutions, which meet our criteria (thus, besides following the principles of dependent tasks scheduling, you can expect finding solution, which requires the least resources, etc.).

The whole algorithm can be then presented in the form of diagram:

- specification of system assumptions
- definition of the set (libraries) of available resources
- definition of the set (functions) of the system
- definition of the set of system parameters
- modifications of system parameters
- analysis of system parameters in relation to assumptions
- resources partitioning
- tasks scheduling
- tasks and resources allocation
- verification and evaluation of the system obtained
- definition of the target system.

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:

- number of processors and the cost of computing power,
- size and cost of operational memory,
- size and cost of mass storage,
- 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 ant colony optimization heuristic algorithm to solve the problem of synthesis

The *Ant Colony Optimization* (ACO) algorithm 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 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 *h* 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. Formula evaluation of the quality of the next step in the ACO algorithm:

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

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:

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. task schedule realization,
5. it was the last task? (if not go to 2, if so then end).

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? (if not go to 2, if so then end).

The computational complexity of single agent process 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:

1. initiation of tables of tasks running sequence and allocation of tasks to resources,
2. completing the cycle of analysis for each agent,
3. evaluation of the best solution found in current cycle,
4. for each agent – basing on the best solution – updating the tables of tasks running sequence and allocation of tasks to resources,
5. is it the last cycle? (if not go to 2),
6. optimization/customization of system parameters.

3 Customization of the Branch & Bound greedy algorithm 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. 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.

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 three minutes 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 *Ant Colony* algorithm will be graded on the basis of solutions found by *Branch & Bound* algorithm.

$$\text{quality} = 100\% \frac{1}{criterions} \sum_{criterion=1}^{criterions} \frac{result_{B\ \&\ Bcriterion}}{result_{ACOcriterion}}$$

Formula for the quality of obtained solution is following.

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

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

The correctness of scheduling proposed by ACO and B&B algorithms was verified on the basis of the following examples.

The algorithms were given the following resources: processors, operating memory, mass storage.

Characteristic Processors:

Id	Computation power	Power consumption (active)	Power consumption (non active)
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

Power consumption optimization

Time, which has passed until solution was found and the parameters of the target system are presented in the table:

Number of tasks	*Ant Colony*				*Branch & Bound*				*Quality %*
	Time	*Length*	*Cost*	*Power*	*Time*	*Length*	*Cost*	*Power*	
5	2.0	7.5	3.00	900	7.5	7.5	3.00	900	100.0
10	2.5	7.5	9.50	1892	21.0	10	5.00	1930	102.0
15	42.5	8.0	10.00	2839	0.0	30	2.00	3000	105.6
20	32.5	12.5	10.00	3842	0.0	40	2.00	4000	104.1
25	8.5	12.5	11.50	4770	0.0	50	2.00	5000	104.8
30	11.0	15.0	16.50	5996	0.0	60	2.00	6000	100.0
35	20.5	15.0	19.00	6975	0.0	70	2.00	7000	100.3
40	32.0	20.0	19.00	7994	0.0	80	2.00	8000	100.1
45	12.0	16.5	19.00	8989	0.0	90	2.00	9000	100.1
50	22.0	21.3	19.00	9971	0.0	100	2.00	10000	100.3
55	35.5	24	19.00	11014	0.0	110	2.00	11000	99.9
60	50.5	23.8	19.00	12030	0.0	120	2.00	12000	99.8

The above example shows, that ACO algorithm has the advantage over the B&B algorithm in the wide interval. The first one gives better scheduling in many cases. that apart from very good power consumption parameters of the systems proposed by ACO algorithm, their quality is much better than the quality of systems proposed by B&B algorithm: the time of scheduling is many times shorter and their price is only a little higher.

Cost of the system optimization

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

Number of tasks	*Ant Colony*				*Branch & Bound*				*Quality %*
	Time	*Length*	*Cost*	*Power*	*Time*	*Length*	*Cost*	*Power*	
5	0.0	10	2.00	1000	0.0	10	2.00	1000	100
10	0.0	20	2.00	2000	0.0	20	2.00	2000	100
15	0.5	30	2.00	3000	0.0	30	2.00	3000	100
20	1.0	40	2.00	4000	0.0	40	2.00	4000	100
25	1.5	50	2.00	5000	0.0	50	2.00	5000	100
30	4.5	60	2.00	6000	0.0	60	2.00	6000	100
35	6.0	70	2.00	7000	0.0	70	2.00	7000	100
40	6.5	80	2.00	8000	0.0	80	2.00	8000	100
45	11.5	90	2.00	9000	0.0	90	2.00	9000	100
50	20.0	100	2.00	10000	0.0	100	2.00	10000	100
55	33.0	110	2.00	11000	0.0	110	2.00	11000	100
60	42.5	120	2.00	12000	0.0	120	2.00	12000	100

This example shows that ACO algorithm can find the solution as good as the greedy algorithm, despite the fact that this solution is an extreme case of scheduling (all tasks are allocated at the first processor).

The cost of proposed solution results from the fact, that all tasks were allocated to the first processor (with computing power equal 1) and each task declares using 1MB of operational memory and 1MB of mass storage (as a result we get the cost of ~2.00099).

Scheduling time optimization

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

Number of tasks	*Ant Colony*				*Branch & Bound*				*Quality %*
	Time	*Length*	*Cost*	*Power*	*Time*	*Length*	*Cost*	*Power*	
5	0.0	1.5	25.00	1004	0.5	1.5	24.00	989	100
10	0.0	2.5	30.51	2001	0.0	2.5	30.51	2001	100
15	0.0	4.5	30.51	3128	0.0	4.5	30.51	3094	100
20	0.5	6.0	30.51	4173	0.0	6.0	30.51	4173	100
25	1.0	7.9	30.51	5282	0.0	7.9	30.51	5282	100
30	1.5	10.0	30.51	6373	0.0	10.0	30.51	6396	100
35	2.0	11.3	30.51	7448	0.5	11.3	30.51	7448	100
40	2.5	13.5	30.51	8602	0.0	13.5	30.51	8609	100
45	3.0	15.0	30.51	9636	0.0	15.0	30.51	9614	100
50	4.5	16.5	30.51	10693	0.0	16.5	30.51	10693	100
55	6.5	18.0	30.51	11772	0.0	18.0	30.51	11772	100
60	7.0	20.3	30.51	12939	0.0	20.0	30.51	12872	98.5

This example shows that ACO algorithm is able to find very good or even optimal scheduling in the reasonable time. When the number of tasks is smaller than the number of processors, this time is shorter than the time needed by B&B algorithm. With the large number of tasks, the ACO algorithm was unable to find scheduling equally good to the one found by B&B algorithm, but its offers were very similar.

Optimization of scheduling time and system cost

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

Number of tasks	*Ant Colony*				*Branch & Bound*				*Quality %*
	Time	*Length*	*Cost*	*Power*	*Time*	*Length*	*Cost*	*Power*	
20	59.0	10	15	4007	0.0	6.0	30.50	4173	131.7
25	60.0	9.0	20.50	5054	0.0	7.9	30.51	5281	118.3
30	12.5	9.0	19.00	6057	0.0	10.0	30.51	6394	124.5
35	16.0	12.5	19.00	1004	0.5	11.3	30.51	7448	125.4
40	15.5	15.0	19.00	8010	0.0	13.5	30.51	8568	125.3
45	42.5	16.5	19.00	9011	0.0	15.0	30.51	9654	125.7
50	26.5	18.0	19.0	10024	0.0	16.5	30.51	10693	126.1
55	34.5	20.0	19.0	11009	0.0	18.0	30.51	11772	125.3
60	44.0	21.4	19.00	12003	0.0	20.0	30.51	12872	127.0

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 then the ones proposed by B&B algorithm even by 31.7%.

Power consumption and cost of the system optimization

This is another example of joint optimization, where we look for the cheapest and the most efficient system.

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

Number of tasks	*Ant Colony*				*Branch & Bound*				*Quality %*
	Time	*Length*	*Cost*	*Power*	*Time*	*Length*	*Cost*	*Power*	
10	0.5	20.0	2.00	2000	0.0	20.0	2.00	2000	100.0
15	3.5	30.0	2.00	3000	0.0	30.0	2.00	3000	100.0
20	4.5	18.0	5.00	3780	0.0	40.0	2.00	4000	72.9
25	10.0	22.0	5.00	4732	0.0	50.0	2.00	5000	72.8
30	12.5	27.0	5.00	5670	0.0	60.0	2.00	6000	72.9
35	12.0	20.0	10.00	6677	0.0	70.0	2.00	7000	62.4
40	27.0	28.0	10.00	7869	0.0	80.0	2.00	8000	60.8
45	16.5	33.0	10.00	8835	0.0	90.0	2.00	9000	60.9
50	57.5	32.0	10.00	9673	0.0	100.0	2.00	10000	61.7
55	43.5	38.0	10.00	10766	0.0	110.0	2.00	11000	61.1
60	55.5	37.5	11.50	11822	0.0	120.0	2.00	12000	59.4
10	0.5	20.0	2.00	2000	0.0	20.0	2.00	2000	100.0

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. 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). 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).

5 Conclusion

Basing on the above research you may say, that the ACO algorithm is better suitable for both one- and multi-objective analyses. The systems obtained (as a result of ACO algorithm) even in the worst case were only insignificantly worse than solutions obtained by B&B algorithm. 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 calculative experiments prove the superiority of coherent synthesis over the incoherent synthesis and heuristic algorithms over the greedy ones.

The heuristic and genetic algorithms handle the NP-complete problems much better than the greedy algorithms. It is because they approach the problem in a way that let them pre-analyze the good solutions and immediately start the optimization of bigger number of parameters in the consecutive steps. Thanks to such approach, solutions are being found in the short time and the algorithms do not require additional modifications to optimize other factors (contrary to the greedy algorithms).

Acknowledgment

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

References

1. Coffman E. G., Jr., (1976), *Computer and Job-shop scheduling theory*, John Wiley&Sons, Inc. New York.
2. Dorigo M., Di Caro G., Gambardella L., M., (1999), *Ant Algorithms for Discrete Optimization*, Artificial Life, Vol. 5, No. 2.
3. Drabowski M., (2004), Coherent synthesis of heterogeneous system – a neural approach. *Proc. of Artificial Intelligence AI'2004*, session IV (25).
4. M. Drabowski, K. Czajkowski, (2006), *A Tabu Search approach in coherent synthesis of heterogeneous system*, Studia Informatica, vol. 1/2, 2006, pp. 31-45.
5. Drabowski M., (2006). Coherent synthesis of heterogeneous system – an evolutionary approach. *Proceedings of Artificial Intelligence Studies*, vol. 3(26).
6. Drabowski M, Wantuch E., (2006). Coherent concurrent task scheduling and resource assignment in dependable computer system design. In: *Int. Journal of Reliability, Quality and Safety Engineering*, vol. 13, No. 1, 15-24.
7. M. Drabowski, K. Czajkowski, (2006), *Minimizing Cost and Minimizing Schedule Length in Synthesis of Fault Tolerant Multiprocessors Systems* [in:] Parallel Processing and Applied Mathematic, Springer-Verlag, LNCS 3911, 2006, pp. 986-993.