An ACO-based Approach for Scheduling Task Graphs with Communication Costs

Markus Bank Udo Hönig Wolfram Schiffmann
FernUniversität Hagen
Lehrgebiet Rechnerarchitektur
58084 Hagen, Germany
{Markus.Bank, Udo.Hoenig, Wolfram.Schiffmann}@FernUni-Hagen.de

Abstract

In this paper we introduce a new algorithm for computing near optimal schedules for task graph problems. In contrast to conventional approaches for solving those scheduling problems, our algorithm is based on the same principles that ants use to find shortest paths between their nest and food sources. Like their natural counterparts, artificial ants cooperate by means of pheromone trails where information about the quality of the possible solution's building blocks is stored. Based on this common communication structure, new solutions emerge by means of cooperative interaction between the ants. In the paper we demonstrate how this basic principle can be adapted to solve scheduling problems. We also evaluated the performance of the proposed ANTLS-algorithm (Ant List Scheduler) by means of a comprehensive test bench with more than 30,000 test cases. Compared to two conventional and two nature-inspired approaches it performed very well.

1. Introduction

The number of networked computers world wide is steadily increasing. If one has to conduct complex computations it is therefore useful to distribute the working load over multiple computers. In this way one can not only accelerate the computation but also improve the usage of the available computers. To solve a specific problem, we need a parallel algorithm whose components will be executed on a homogeneous or heterogeneous network of more or less spatial distributed computers (e.g. cluster or grid computers, respectively).

Dependencies between the parts of a parallel algorithm can be described by means of a directed acyclic graph (DAG), also known as *task graph*. In general, the nodes and edges of a task graph are weighted. The work load of a specific task i is indicated by the node's weight w_i. It corresponds to the time needed by a task to compute new outputs using the results of preceding tasks as inputs. The cost of communication between two tasks i and j is specified by an edge weight c_{ij}. This cost must be considered only if the communicating tasks will be mapped to different processing elements.

If we want to execute an algorithm which is specified by a task graph on a parallel machine like a cluster or grid computer, we have to find a schedule that determines both, optimal sequencing and optimal mapping of the tasks to the available processing elements. Most often, the objective of solving this *task graph scheduling problem* is to minimize the overall computing time by making efficient use of the parallel processing elements. In this paper, we suppose a homogeneous computing environment. Thus we can use a greedy mapping strategy and can concentrate on solving the sequencing (sub)problem. But despite the restriction to identical processing elements, the problem to determine an optimal schedule – apart from some restrained cases – has been proven to be NP-complete [1]. Thus, most researchers use heuristic approaches to solve the problem for reasonable sizes of the task graph. Three categories can be distinguished: list–based, clustering–based and duplication–based heuristics. List–based heuristics assign priority levels to the tasks and map the highest priority task to the best fitting processing element [2]. Clustering–based heuristics embrace heavily communicating tasks and assign them on the same processing element, in order to reduce the overall communication overhead [3]. Duplication-based heuristics also decrease the amount of communication while simultaneously the amount of (redundant) computation will be increased. It has been combined with both list-based [4] and cluster-based approaches [5].

In this paper we will introduce a list-based approach that concentrates on finding a near optimal sequence for the tasks. The mapping subproblem is solved by a so called *greedy mapper* that simply assigns the next task to be scheduled to that processing element where it can be started as soon as possible. In order to find a nearly optimal sequence, we use a quite new approach that models the behavior of ant colonies.

In contrast to other list-based heuristics, our algorithm mimics the successful strategies of ants in finding solutions to the complex routing problems the ants are faced with in real life. We compare the results found by this so called *Ant Colony Optimization* (ACO) approach with the performance of other well known conventional and also nature-inspired heuristics. This comparison is based on a comprehensive test bench of task graph problems that provides the actual *optimal* solutions to a collection of various task graph problems [6]. By means of this test bench we can easily evaluate and analyze the strengths and drawbacks of the proposed ACO-approach. Even though the ACO-approach has been applied to various optimization problems, e.g. the Traveling Salesman Problem (TSP), in this contribution it has been for the first time applied to task graph scheduling. By the introduction of problem dependent rules for computing the pheromone concentrations, the stability of the algorithm and the quality of its results have been improved.

The paper is organized as follows: In the next section the ACO-metaheuristic will be described. Then, we introduce into the details of our ant list scheduler algorithm (ANTLS) for computing near optimal sequences to a given task graph scheduling problem by means of the ACO-metaheuristic. Then, we compare the results achieved by ANTLS with those of two conventional and two nature-inspired approaches. We also analyze the influence of different parameter settings and the size of the task graphs to the achieved performance.

2. The Ant Colony Optimization Metaheuristic

As a result of the considered task graph scheduling problem's NP-completeness, researchers developed a large number of deterministic and stochastic heuristics in order to find near optimal schedules in reasonable time, many of them basing on metaheuristics (e.g. [7, 8]). A *metaheuristic* is a generic, to numerous different problems adaptable algorithm that guides the processing of at least one subordinated (local) heuristic which is specialized in the given problem. A detailed description of general metaheuristics' characteristics as well as a survey of the currently most popular methods can be found in [9]. Some metaheuristics imitate natural processes and are therefore called *nature-inspired*. Among Simulated Annealing, Genetic Algorithms and many others, the class of metaheuristics also comprises a method called *Ant Colony Optimization* (ACO)[10, 11].

ACO is inspired by the indirect communication of a foraging ant swarm. By means of this communication behavior, the swarm is enabled to find shortest paths between food sources and the nest. While moving, every ant deposits a trail of a chemical substance, called *pheromone*, on its way. The more ants pass by, the higher is the pheromone concentration at a location. Although every ant selects its direction at random, a path's probability of being chosen raises with its pheromone concentration. Since every ant leaves a pheromone trail at the selected path, it reinforces the already deposited pheromone and therefore raises the path's attractiveness for later ants. Without any limitation mechanism, this positive feedback would lead to a soon convergence and the selection of a probably poor path. In nature, this limitation is achieved by *evaporation* which reduces the pheromone concentration by-and-by, if the path is not frequently used. This way, an ant colony will 'forget' a path if it is of no further use.

The ACO metaheuristic applies the described behavior of natural ants to optimize constructive search processes for combinatorial problems. By using *artificial pheromone* trails, a swarm of independent agents, the *artificial ants*, is guided through the investigated search space. The general structure of the ACO metaheuristic, including some of the later described optional extensions is shown in Figure 1. While a problem dependent termination criterion is not satisfied, the following steps are conducted: Firstly, every ant constructs a solution to the given problem. Next, the found solutions are evaluated and the artificial pheromone trails are updated. The third step is optional and deals with so-called *daemon actions* which are centralized actions that need global knowledge.

```
while termination criterion is not satisfied{
1.      ants construct solutions
2.      pheromone trail update
                online delayed pheromone update (opt.)
                evaporation
3.      daemon actions (opt.)
                elitism (opt.)
                offline pheromone update (opt.)
}
```

Figure 1. The ACO-metaheuristic.

The termination criterion is usually a combination of a demand on the found solution's quality and an upper bound of allowed loop iterations. If the search gets stuck in a local extremum, this bounding ensures that ACO terminates within a reasonable period of time.

An artificial ant performs its walk through the search space by incrementally completing a solution of the given problem. In every construction step, the ant has to select a building block that should be integrated in its current partial solution next. This decision problem corresponds to the problem of natural ants to select a suitable direction. Like their natural counterparts, artificial ants can use the pheromone trails of preceding ants as a guideline for this decision. In addition, a local heuristic provides further information that can be used to guide each ant.

To fulfill its task, every artifical ant is equipped with enough memory to remember its path through the search space. This information can be required to find a valid solution, to evaluate the found solution or to deposit the ant's pheromone trail afterwards.

In contrast to nature, where every moving ant deposits a continuous and constant pheromone trail, ACO permits several alternative ways. The trail update can either be conducted by the ant itself (*online update*) or a deamon process (*offline update*). In case of an online update, the ant can deposit the pheromone immediately at every construction step (*online step-by-step pheromone update*) or after finishing the whole construction process (*online delayed pheromone update*). A delayed trail generation enables the consideration of a solution's quality when selecting the amount of pheromone to deposit. This way, the further search can be biased more strongly to promising areas of the search space.

Like in nature, without any appropriate protection mechanism the artificial ants' positive feedback would lead to an early convergence of the algorithm with possibly bad results. For this reason, ACO-algorithms usually implement some kind of *evaporation*, allowing the artificial ants to 'forget' and therefore favoring the exploration of new search space areas.

Many ACO-algorithms implement an 'elitism'-function. Among plenty of possible alternatives, elitism can – for example – be realized by considering only the best solution of every run for the trail update [12], or by means of an deamon action, that deposits an extra amount of pheromone upon the global best ant's path every run [13].

3. ANTLS (Ant List Scheduling)

In ANTLS, ants build sequences of tasks by visiting the nodes of a DAG in a topological order. The scheduling lists are used as an input of a greedy list scheduler which maps the tasks to the processors. The length of the resulting schedule is then used as a quality measure of the generated solution.

Building a scheduling list for a DAG requires the consideration of the predefined precedence constraints. If they are not taken into account, it is most likely that the mapper can not create a feasible schedule. This does not only limit the set of starting nodes, but also the set of nodes that are allowed to be scheduled at each step. In order to create feasible scheduling lists, every ant's memory has to store the following information:

- The current node the ant is. This is the last scheduled node.

- A set of nodes, the so-called *free* nodes, which are ready for scheduling. This set will be updated after every step.

- The number of unscheduled predecessors for each node. This is used by the ant to detect when a node becomes ready for being processed. The ant needs to keep track of this information because a node must not be scheduled if it has unscheduled predecessors.

- The scheduling list which is build by the ant.

At the beginning, the set of free nodes will be preset with the entry nodes. The ant will be placed at a pseudo node $v_0 \notin V$, V denoting the set of the DAG's nodes, which will not be scheduled. The purpose of this pseudo node is to let the ants decide which entry node should be scheduled first. The number of unscheduled predecessors for each node will be initialized and the ants scheduling list is empty.

We now define the relationship between pheromone and scheduling lists. For that purpose we introduce a pheromone value $\tau(v, w)$ that represents the benefit of scheduling a node w directly after node v, where $v, w \in V$. If a scheduling list which schedules a node w directly after a node v results in a short schedule, this fact should be saved for succeeding ants by depositing some pheromone between v and w. Later ants will then be attracted to schedule the nodes in an order that resulted in a short schedule. The initial amount of pheromone at the start of the algorithm will be initialized to $\tau(v, w) = \tau_0$ for each pair of nodes (v, w), so at the beginning there will be no guidance by pheromone.

Additionally, the ants use a local heuristic, supporting them to build good scheduling lists. At each step an ant has to decide between several nodes to schedule next. Classical task scheduling heuristics, such as HLFET [14] and MCP [15] are making such a decision by assigning priorities to the nodes. These priorities are often based on levels implied by the structure of the DAG, for example:

- $blevel(v)$ which is the length of the longest path from v to an exit node, excluding the weight of node v.

- $sblevel(v)$ which is the same as $blevel(v)$ but without considering edge weights.

We compared the results of using $sblevel$ and $blevel$ ordering and found that scheduling nodes in descending $sblevel$-order gives slightly better results than scheduling in $blevel$-order. For this reason, we define a local heuristic to schedule a node w directly after a node v (which is in fact independent of v) as:

$$\eta(v,w) = sblevel(w) \tag{1}$$

We will now describe, how an ant k selects the next node to be scheduled. Assume the ant is at node Pos_k and its set of free nodes is denoted as $Free_k$. At first, the ant decides between the exploitation of a known good scheduling order or the exploration of different orderings. The balance of exploitation and exploration is controlled by a parameter $q_0 \in [0,1]$ which indicates the probability that an ant decides to exploit a good order and schedules the node with the highest attractiveness. This is the one, which possesses the best combination of a high $sblevel$ and a large amount of pheromone.

$$Next_k = \underset{u \in Free_k}{argmax} \left\{ \tau(Pos_k,u) \cdot \eta^{\beta}(Pos_k,u) \right\} \tag{2}$$

where β is a parameter that controls the influence of the local heuristic. If the currently considered ant decided to explore, it selects one node $w \in Free_k$ by using a so called *random-proportional* decision rule. The probability the ant selects node w is:

$$p(w) = \frac{\tau(Pos_k,w) \cdot \eta^{\beta}(Pos_k,w)}{\sum\limits_{u \in Free_k} \tau(Pos_k,u) \cdot \eta^{\beta}(Pos_k,u)} \tag{3}$$

After the ant has selected a node $Next_k$, it moves to this node and updates its set of free nodes by decrementing the number of unscheduled predecessors for each successor of $Next_k$. If a successor's number of unscheduled predecessors reaches zero, it is added to the set of free nodes. After maintaining its internal state, every ant immediately applies a local pheromone update using the following rule:

$$\tau(Pos_k, Next_k) \leftarrow (1-\rho) \cdot \tau(Pos_k, Next_k) + \rho \cdot \tau_0 \tag{4}$$

In the above rule, the parameter $\rho \in [0,1]$ controls the ratio of pheromone evaporation and reinforcement. Unlike other ant algorithms, e.g. *Ant Colony System* (ACS) for the TSP [13], we decided to update the pheromone directly after an ant made its step. Since all ants start at the same pseudo node and therefore the set of allowed nodes to schedule is the same for all ants at the start of an iteration, without an immediate update the probability that all ants choose the same node would be quite high, leading to a soon convergence.

After all ants have built their scheduling lists, these lists are evaluated by mapping the nodes to the processors. We used a greedy mapper which maps a node to the processor that allows its earliest starting time. The ant whose scheduling list resulted in the shortest schedule is called the *iteration best ant*. Analogous we keep a *global best ant*, which is the best ant over all iterations.

At the end of an iteration, a global pheromone update is applied by allowing the global best ant to deposit some extra pheromone between the nodes it has visited. This will attract ants in later iterations to schedule the nodes in a similar order which might also result in a short schedule. Let $(v_{i_1}, \ldots, v_{i_n})$ be the global best ant's scheduling list. Pheromone will be updated for each $j \in \{1, \ldots, n-1\}$ by means of the following rule:

$$\tau(v_{i_j}, v_{i_{j+1}}) \leftarrow (1-\alpha) \cdot \tau(v_{i_j}, v_{i_{j+1}}) + \alpha \cdot \Delta_\tau \tag{5}$$

Similar to the local pheromone update, a parameter $\alpha \in [0,1]$ controls the ratio of evaporation and reinforcement. In [13] the amount of reinforcement is determined by ${L_{gb}}^{-1}$, where L_{gb} is the global best ant's tour length. In ANTLS we chose a different approach to compute the reinforcement. In order to formalize this approach, we have to introduce a lower bound of an optimal schedule length first.

Assume scheduling a DAG on an unbounded number of processors. Then, the length of a critical path without considering communication costs CP_{len} will be a first lower bound of an optimal schedule length which is implied by the structure of the DAG. A second lower bound can be achieved by assuming scheduling the nodes to a bounded number of processors p. If all nodes are equally distributed to all processors and there are no idle times, the schedule length is bounded by:

$$SL_{min} = \left\lceil \frac{\sum\limits_{v \in V} w(v)}{p} \right\rceil \tag{6}$$

By using these two bounds, we get a lower bound SL_{lb} of an optimal schedule length by:

$$SL_{lb} = \max\{CP_{len}, SL_{min}\} \tag{7}$$

Now, let SL_{gb} be the length of the global best ant's schedule. We define the amount of reinforcement as:

$$\Delta_\tau = \frac{1}{SL_{gb} - SL_{lb} + 1} \tag{8}$$

Subtracting the lower bound SL_{lb} makes the reinforcement independent of the actual DAG. Assume an optimal schedule has length SL_{opt}. If $SL_{opt} = SL_{lb}$, the reinforcement of an ant that created an optimal scheduling list will always be 1.

As a consequence, the amount of pheromone deposited by the global best ant will be much higher as if a reinforcement of ${SL_{gb}}^{-1}$ was used. This needs to be considered when computing the initial amount of pheromone τ_0 at the start of the algorithm. In ANTLS we therefore initially deposit

$$\tau_0 = \frac{1}{|V|} \tag{9}$$

between each pair of nodes (v,w).

4. Results

In order to conduct an unbiased analysis of our algorithm, we used the test bench proposed in [6] as test set. This test bench comprises 36000 task scheduling problems with task graphs ranging from 7 to 24 nodes and target architectures ranging from 2 to 32 parallel computers. Since the optimal schedules comprised by this test bench are not yet available for all test cases, the interim version used for the presented analysis is limited to 30511 task graphs with known optimal solutions. To our knowledge there exists no comparable test bench for the evaluation of heuristic scheduling algorithms. Although in [16] a test bench for task graphs with up to 5000 tasks is provided these task graphs do not take communication overhead into account. Also, it is not guaranteed that the minimum schedule length is given for every task graph problem, because of the NP-completeness the run time of the search algorithm was limited to 10 minutes. In [17] a performance study of 15 heuristic scheduling algorithms is presented. In contrast to the test set used, the number of task graphs is much lower ($\approx$ 350) and for most of the investigated task graphs (250) the optimal schedules are not known.

Like other metaheuristics, ACO allows for a wide range of alternative options for several parameters. Our first investigation was therefore to find a setting, enabling the algorithm to find as many optimal schedules as possible. Since ACS performed well considering the Traveling-Salesman-Problem, we decided to realize an ACS-like implementation (see fig. 2(A)) as starting point for our search. The number of iterations was set to 60, with an iteration-size of 17 ants, totaling 1020 ants. We selected

- the local heuristic,
- the amount of reinforcement Δ_τ,
- the initial amount of pheromone τ_0 and
- the timing of the pheromone update's conduction

for variation. Figure 2 can show only some of the obtained results. Implementations using the $sblevel$ as local heuristic are slightly superior to those using the $blevel$ (fig. 2(A) and (B)). Increasing Δ_τ from ACS's default to $\Delta_\tau = \frac{1}{SL_{gb}-SL_{lb}+1}$ produces worse results if the initial amount of pheromone (τ_0) is unchanged (see fig. 2(C)). In combination with an increased initial amount of pheromone ($\tau_0 = \frac{1}{|V|}$), this value for Δ_τ achieves much better results (fig. 2(D)). In comparison to ACS's pheromone update, an early pheromone update after every single ant's move shows no observable difference as can be seen in figure 2, (D) and (E). This way, we found the implementation described in section 3, which will be further analyzed.

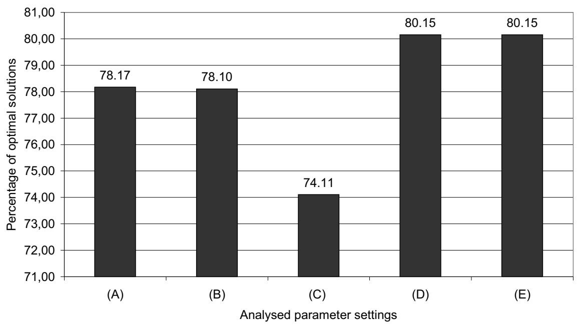

Figure 2. Results of several variations of our ACO-algorithm.

Our next aim is to investigate the influence of the ant/iteration-ratio on the quality of the achieved results. The number of total ants is fixed to a value of about 1024. As can be seen in table 1, the results of different ant/iteration-ratios differ by a maximum of only 1,47%. Although the results of [13] are approximately confirmed, at least for small task graphs our algorithm seems to be quite stable here.

While the ratio of ants and iterations is of minor importance considering the test set used, the number of ants that move accumulated over all iterations has a strong influence on the quality of ANTLS's results. For a closer investigation, we varied this parameter within a range of 1 and 2048 ants and analyzed the effects with respect to several task graph properties. Exemplarily, we will now focus on the task graphs' sizes. Figure 3 shows the achieved results for task graphs consisting of 7, 15 and 24 tasks as well as the overall test set with all task graph sizes. It is obvious that the algorithm finds the optimal schedules more often, the smaller the considered task graphs are. Independent of the task graphs' size, the benefit of adding more ants to the search process declines soon and finally reaches a level where it is almost zero. The bigger the task graph is, the more ants are required to reach this level.

The compared heuristics were originally all designed for static scheduling. Although the runtime of an algorithm is less important in static scheduling environments, a too extensive overhead would make it inapplicable for larger task graphs. For this reason, we investigated the scaling behaviour of our ANTLS algorithm and compared the obtained results with those of other heuristics. The experiment was conducted upon a PC with an Athlon-1000 MHz processor. Figure 4 shows the runtimes of the considered nature inspired heuristics. The deterministic algorithms are not considered since their runtimes are all within a few seconds and would therefore appear as flat lines only. ANTLS, SA and GA obtain similar results, with the GA perform-

Table 1. Results of different ant/iteration-ratios

Number of Ants	1	2	4	8	16	21	32	64	128	256	512	1024
Number of Iterations	1024	512	256	128	64	49	32	16	8	4	2	1
Optimal Results (in %)	78.78	79.26	79.55	79.91	80.11	80.25	80.09	80.15	79.86	79.45	79.21	79.03

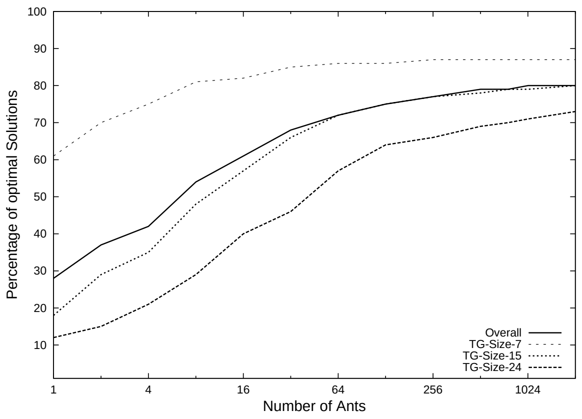

Figure 3. Influence of the number of ants involved in the search process.

ing the best and ANTLS the worst. Although this overhead prevents these heuristics from being applied in dynamic scheduling environments, the shown performance is sufficient for static scheduling and analysis purposes.

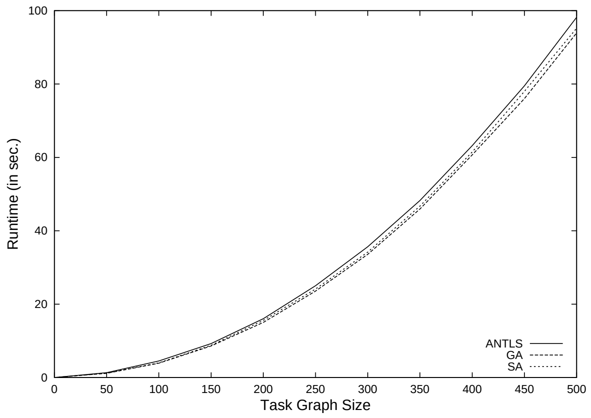

Figure 4. Comparison of the nature inspired algorithms' scalability.

ANTLS achieves promising results when being compared to other nature-inspired stochastic as well as to deterministic scheduling heuristics. As can be seen in figure 5, it performs best with exception of Simulated Annealing (SA). In general, the nature-inspired stochastic algorithms perform much better than the deterministic ones.

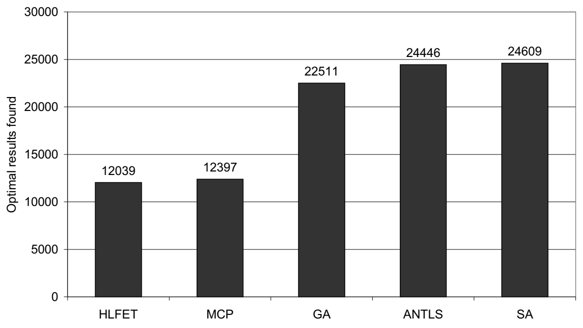

Figure 5. Comparison of several scheduling heuristics.

These observations hold to a large extend, when considering larger task graphs with a maximum of 250 tasks. ANTLS is still the second best after SA. The better deterministic algorithms close up to the Genetic Algorithm (GA) but are far-off to SA and ACO. A more detailed comparison can be found in [18]. Although our algorithm's current version already provides good results, we nevertheless expect a further improvement when exchanging our algorithm's greedy mapping heuristic by an ACO-based mapping.

5. Conclusion and future work

In this paper we demonstrated how the emergent behavior of cooperating ants can be simulated to solve task graph scheduling problems. We proposed the ANTLS-algorithm and evaluated its performance by means of a comprehensive test bench with over 30,000 test cases. By the introduction of problem dependent rules for computing the pheromone concentrations, the stability of the algorithm and the quality of its results have been improved. ANTLS performs very good compared to several other investigated nature-inspired as well as conventional approaches. We plan for the future work to improve the greedy mapping part of the ANTLS-algorithm. Especially, it would be interesting to investigate if this component can also be realized by an integrated ACO-based approach.

References

[1] Kwok, Y.-K., Ahmad, I.: Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Computing Surveys, Vol. 31, No. 4, 1999, pp. 406–471

[2] Radulescu, A., van Gemund, J.C.:Low-Cost Task Scheduling for Distributed-Memory Machines, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, No. 6, June 2002

[3] Aguilar, J., Gelenbe E.: Task Assignment and Transaction Clustering Heuristics for Distributed Systems, Information Sciences, Vol. 97, No. 1& 2, pp. 199–219, 1997

[4] Bansal, S., Kumar, P., Singh, K.: An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems, IEEE Transactions on Parallel and Distributed Systems, Vol. 14, No. 6, June 2003

[5] Park, C.-I., Choe, T.Y.: An optimal scheduling algorithm based on task duplication, IEEE Transactions on Computers, Vol. 51, No. 4, April 2002

[6] Hönig, U., Schiffmann, W.: A comprehensive Test Bench of optimal Schedules for the Evaluation of Scheduling Heuristics, Proceedings of the sixteenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), Cambridge, U.S.A., 2004

[7] Greenwood, G.W., Gupta, A., McSweeney, K.: Scheduling Tasks in Multiprocesor Systems Using Evolutionary Strategies, International Conference on Evolutionary Computation, pp. 345–349, 1994

[8] Kwok, Y.-K., Ahmad, I.: Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors Using a Parallel Genetic Algorithm, Journal of Parallel and Distributed Computing, Vol. 47, No. 1, pp. 58–77, 1997

[9] Blum, C., Roli, A.: Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, ACM Computing Surveys, Vol. 35, No. 3, 2003, pp. 268–308

[10] Dorigo, M., Maniezzo, V., Colorni, A.: Ant System: Optimization by a colony of cooperating Agents, IEEE Transactions on Systems, Man and Cybernetics, Part B, Vol. 26, No. 1, 1996, pp.1–13

[11] Dorigo, M., Stützle, T.: The Ant Colony Optimization Metaheuristic: Algorithms, Applications and Advances, In F. Glover and G. Kochenberger (Eds.), Handbook of Metaheuristics, volume 57 of International Series in Operations Research & Management Science, pp. 251–285, Kluwer Academic Publishers, Norwell, MA, 2002

[12] Stützle, T.: An Ant Approach to the Flow Shop Problem, Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98), 1998, Vol. 3, pp. 1560–1564

[13] Dorigo, M., Gambardella, L. M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem, IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, pp. 53–66, 1997

[14] Adam, T. L., Chandy, K. M., Dickson, J. R.: A comparison of list schedules for parallel processing systems, Communications of the ACM, Vol. 17, No. 12, pp. 685–690, 1974

[15] Wu, M.Y., Gajski, D. D.: Hypertool: A Programming Aid for Message-passing Systems, IEEE Transactions on Parallel and Distributed Systems, Vol. 1, No. 3, pp 330-343, 1990

[16] Kasahara Laboratory, http://www.kasahara.elec.waseda.ac.jp/schedule/index.html, 2004

[17] Y.-K. Kwok, I. Ahmad, Benchmarking the Task Graph Scheduling Algorithms, *Proceedings of the 12th International Parallel Processing Symposium*, Orlando, U.S.A., 1998, 531-537

[18] Hönig, U., Schiffmann, W.: Comparison of nature inspired and deterministic scheduling heuristics considering optimal schedules, In B. Ribeiro et al. (Eds.), Adaptive and Natural Computing Algorithms, ISBN 3-211-24934-6, Springer-Verlag, 2005