Static Homogeneous Multiprocessor Task Graph Scheduling Using Ant Colony Optimization

Hamid Reza Boveiri[1,*] and Raouf Khayami[2]
[1] Sama Technical and Vocational Training College, Islamic Azad University, Shoushtar Branch, Shoushtar, Iran.
[2] Information Technology Department, Shiraz University of Technology, Shiraz, Iran.
e-mail: boveiri{@shoushtar-samacollege.ir or @ieee.org}
[*] Corresponding Author: Hamid Reza Boveiri

Received April 3, 2016; revised August 11, 2016; revised November 5, 2016; accepted November 15, 2016; published June 30, 2017

Abstract

Nowadays, the utilization of multiprocessor environments has been increased due to the increase in time complexity of application programs and decrease in hardware costs. In such architectures during the compilation step, each program is decomposed into the smaller and maybe dependent segments so-called tasks. Precedence constraints, required execution times of the tasks, and communication costs among them are modeled using a directed acyclic graph (DAG) named task-graph. All the tasks in the task-graph must be assigned to a predefined number of processors in such a way that the precedence constraints are preserved, and the program's completion time is minimized, and this is an NP-hard problem from the time-complexity point of view. The results obtained by different approaches are dominated by two major factors; first, which order of tasks should be selected (sequence subproblem), and second, how the selected sequence should be assigned to the processors (assigning subproblem). In this paper, a hybrid proposed approach has been presented, in which two different artificial ant colonies cooperate to solve the multiprocessor task-scheduling problem; one colony to tackle the sequence subproblem, and another to cope with assigning subproblem. The utilization of background knowledge about the problem (different priority measurements of the tasks) has made the proposed approach very robust and efficient. 125 different task-graphs with various shape parameters such as size, communication-to-computation ratio and parallelism have been utilized for a comprehensive evaluation of the proposed approach, and the results show its superiority versus the other conventional methods from the performance point of view.

Keywords: Ant colony optimization (ACO), metaheuristics, multiprocessor task-scheduling problem, task-graph, parallel and distributed systems.

http://dx.doi.org/10.3837/tiis.2017.06.014 **ISSN : 1976-7277**

1. Introduction

Today, most application programs are too time-consuming to be executed on a regular single CPU machine. On the other hand, hardware costs are decreasing constantly, and on these bases, the trends to utilize multiprocessor environments such as parallel and distributed systems have been increased a lot. In such architectures, each program is divided into the smaller and maybe dependent segments named tasks. Some tasks need the data generated by the other tasks; hence, there will be precedence constraints among tasks, and the problem can be modeled using a directed acyclic graph (DAG) so-called task graph. In a task graph, nodes are tasks, and edges indicate precedence constraints among them. In the static scheduling, all the parameters such as required-execution-times of the tasks, communication costs/delays, and precedence constraints are determined during the program's compilation step. All the tasks in the task graph should be mapped into a number of processors with respect to their precedence so that the overall finish time of the given program is minimized.

Multiprocessor task scheduling is an NP-hard problem from the time complexity point of view [1]; therefore, different intelligent heuristic and metaheuristic approaches and algorithms have already been introduced in the literature to find suboptimal solutions in a timely manner [2]. Most of the conventional task-scheduling approaches, such as HLFET (Highest Level First with Estimated Time) [3], ISH (Insertion Scheduling Heuristic) [4], CLANS (which uses the cluster-like CLANs to partition the task graph) [5], LAST (Localized Allocation of Static Tasks) [6], ETF (Earliest Time First) [7], DLS (Dynamic Level Scheduling) [8], and MCP (Modified Critical Path) [9], are based on the list-scheduling technique. That is, these approaches make a list of ready tasks at each stage and assign them some priorities. The ready tasks are either those without any parents or without any unscheduled ones. Then, the task with the most priority in the ready-list is selected to be assigned to the processor that allows the earliest start time (EST), until all the tasks in the task graph are scheduled.

The makespans (finish-times) achieved by such methods are dominated by two major factors: first, which order of tasks should be selected (sequence subproblem), and second, how the selected sequence should be assigned to the processors (assigning subproblem). The author was the first who explicitly distinguished and presented these two disparate-in-nature subproblems, and introduced two different ACO-based approaches in [10] and [11], to tackle the sequence and assigning subproblems, respectively. In this paper, we combine these two robust approaches together, and propose a hybrid method in which two different artificial ant colonies cooperate to solve the multiprocessor task-scheduling problem; one colony works to tackle the sequence subproblem, and try to find the most appropriate sequence of tasks, and another to cope with assigning subproblem, and maps the selected sequence to the existing processor elements.

Ant colony optimization (ACO) is a metaheuristic approach simulating social behavior of real ants. Ants always find the shortest path from the nest to the food and vice versa. Artificial counterparts try to find the shortest solution of the given problem on the same basis. Dorigo *et al.* was the first who utilized ant algorithm as a multi-agent approach to solve the traveling sales man problem (TSP) [12], and after that, it has been successfully used in order to solve a large number of difficult discrete optimization problems [13]. We believe that ACO is one of the best methods to cope with such kinds of problems presented by a graph because the ants, to find the shortest path to solve the given problem, are using an indirect local communication called stigmergy. Stigmergy actually retains the experiences faced by all the previous ants,

and lets the ACO to be fast and efficient in comparison with other metaheuristic and evolutionary algorithms.

It should be emphasized that each scheduling problem, such as job-shop scheduling, flow-shop scheduling, project scheduling, task-scheduling in heterogeneous environments (or grid computing), task scheduling in cloud, multi-core task-scheduling and so on, has its own review, taxonomy, definition, datasets and literature varying based on the different natures of the environment in-use and the objectives to be considered. That is, a certain heuristic or metaheuristic approach proposed for one of these scheduling problems cannot be applied for the others but with some/huge number of modifications in the encoding/decoding mechanism, input datasets and utilized algorithm. In addition, an approach efficient for one scheduling problem is not necessarily as so for the others, and we need a comprehensive study and various sets of experiments to proof that. On these bases, each comparison study can be made on its own category; for example, we are not able to compare an ACO-based approach introduced for job-shop scheduling or introduced for task-scheduling in grid with our approach proposed for static homogeneous multiprocessor task-graph scheduling. We believe the most contributions of this study are as follows:

- The paper formulates static task-graph scheduling in homogeneous multiprocessor environments in a comprehensive way using an easy-to-understand taxonomy, notations and definitions.
- A novel high-performance hybrid algorithm based on the meta-heuristic ACO has been proposed to tackle the problem, which properly use the background knowledge about the problem (priority measurements stated in the Section II) and is very competitive with other conventional methods in terms of performance and time-complexity.
- Different sets of experiments on various input samples with different shape parameters have been conducted, and diverse results and conclusions have been made. These conclusions not only will better introduce the proposed approach but also will better reveal the behavior, strengths and weaknesses of other conventional methods in this scheduling category.

The organization of the rest of the paper is as follows. In the following Section, multiprocessor task scheduling problem is surveyed in detail. Ant colony optimization is discussed in the Section III. Section IV introduces the proposed approach. Section V is devoted to implementation details and results, and finally, the paper is concluded in the last Section.

2. Multiprocessor Task Scheduling

A directed acyclic graph $\mathbf{G} = \{\mathbf{N}, \mathbf{E}, \mathbf{W}, \mathbf{C}\}$ named task graph is used to model the multiprocessor task scheduling problem, where $\mathbf{N} = \{n_1, n_2, \ldots, n_n\}$, $\mathbf{E} = \{(n_i, n_j) \mid n_i, n_j \in \mathbf{N}\}$ $\mathbf{W} = \{w_1, w_2, \ldots, w_n\}$, $\mathbf{C} = \{c(n_i, n_j) \mid (n_i, n_j) \in \mathbf{E}\}$, and n are a set of nodes, a set of edges, a set of the weights of the nodes, a set of the weights of the edges, and the number of nodes respectively.

Fig. 1 shows the task graph of a real application program comprised of nine different tasks. In such graph, nodes are tasks and edges specify precedence constraints among them. Each edge such as $(n_i, n_j) \in \mathbf{E}$ demonstrates that the task n_i must be finished before the starting of the task n_j. In this case, n_i is called a parent, and n_j is called a child. Nodes without any parents and nodes without any children are called “entry-nodes” and “exit-nodes”, respectively. Each

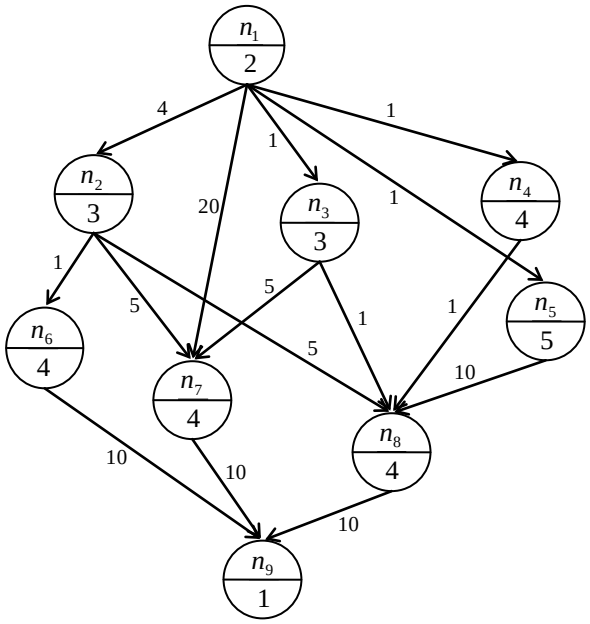

Fig. 1. The task graph of a program with nine tasks inside [16].

node-weight such as w_i is the necessary execution-time for the task n_i, and each weight-of-edge such as $c(n_i, n_j)$ is the time required for data transmission from the task n_i to the task n_j identified as communication cost/delay. If both of the tasks n_i and n_j are executed on the same processor, the communication cost will be zero between them. In static scheduling, execution times of the tasks, communication costs, and precedence constraints among them are generated during the program's compiling-stage. Tasks should be mapped into the given m processor elements such as $\mathbf{P} = \{p_1, p_2, \ldots, p_m\}$ according to their precedence so that the overall finish-time (or makespan) of the given program would be minimized.

Most of the scheduling algorithms are based on the so-called list-scheduling technique. The basic idea behind the list-scheduling is to make a sequence of nodes as a list by assigning them some priorities, and then, repeatedly removing the most priority node from the list, and allocating it to the processor that allows the earliest-start-time (EST), until all the nodes in the graph are scheduled.

If all the predecessors (parents) of the task n_i were executed on the processor p_j, $EST(n_i, p_j)$ would be $Avail(p_j)$ that is, the earliest time at which p_j is available to execute the next task; otherwise, the earliest-start-time of the task n_i on the processor p_j should be computed using

$$EST(n_i, p_j) = \begin{cases} 0, & if \quad n_i = entery - node \\ \max\limits_{n_k \in Parent(n_i)} \begin{cases} (AFT(n_k)), & if \quad processor(n_k) = p_j \\ (AFT(n_k) + c(n_k, n_i)), & else \end{cases} \end{cases}, \quad (1)$$

where $AFT(n_k) = AST(n_k) + w_k$ is the actual finish-time of the task n_k, and $Parents(n_i)$ is the set of all the parents of n_i, $AST(n_k)$ is the actual start-time of the task n_k computed using (2).

$$AST(n_k) = \min_{j=1}^{m} \left(\max \left(Avail(p_j), EST(n_k, p_j) \right) \right) \quad (2)$$

Finally, the total finish-time of the inputted parallel program can be calculated using (3).

$$makespan = \max_{i=1}^{n}\left(AFT(n_i)\right) \tag{3}$$

For a given task-graph with n tasks inside using its adjacency matrix, an efficient implementation of the EST method for assigning all the tasks in the task-graph to a given m identical processors has a time-complexity belonging to $O(mn^2)$ [2].

The efficiency of each related approach introduced in the literature is often originated from how they exploit the background knowledge about the problem, also called task's priority measurements. Some of these measurements frequently used to assign priority to the tasks are *TLevel* (Top-Level), *BLevel* (Bottom-Level), *SLevel* (Static-Level), *ALAP* (As-Late-As-Possible), and the new proposed *NOO* (The-Number-Of-Offspring) [19]. The *TLevel* or *ASAP* (As-Soon-As-Possible) of a node n_i is the length of the longest path from an entry-node to the n_i excluding n_i itself, where the length of a path is the sum of all the nodes and edges weights along the path. The *TLevel* of each node in the task graph can be computed by traversing the graph in the topological order using (4).

$$TLevel(n_i) = \max_{j\in Parents(n_i)}\left(TLevel(n_j) + c(n_j, n_i) + w_j\right) \tag{4}$$

The *BLevel* of a node n_i is the length of the longest path from n_i to an exit-node. It can be computed for each task by traversing the graph in the reversed topological order, as follows:

$$BLevel(n_i) = \max_{j\in Children(n_i)}\left(BLevel(n_j) + c(n_i, n_j)\right) + w_i, \tag{5}$$

where $Children(n_i)$ is the set of all the children of n_i.

If the edges weights are not considered in the computation of *BLevel*, a new attribute called Static-Level or simply *SLevel* can be generated using (6).

$$SLevel(n_i) = \max_{j\in Children(n_i)}\left(SLevel(n_j)\right) + w_i \tag{6}$$

The *ALAP* start-time of a node is a measure of how far the node's start-time can be delayed without increasing the overall schedule-length. It can be drawn for each node using

$$ALAP(n_i) = \min_{j\in Children(n_i)}\left(CPL, ALAP(n_j) - c(n_i, n_j)\right) - w_i, \tag{7}$$

where *CPL* is the Critical-Path-Length, that is, the length of the longest path in the given task graph.

Finally, the *NOO* of n_i is simply the number of all its descendants (or offspring) computed for each task in the task graph using (8).

$$NOO(n_i) = 1 + NOO(n_j) \quad \forall n_j \in Children(n_i) \tag{8}$$

Table 1 lists the above-mentioned measures for each node in the task graph of **Fig. 1**. In addition, a comprehensive list of the notations applied in this Section is reviewed in **Table 2**. To open up how these measures can be utilized in order to schedule the tasks of a task-graph, five well-known traditional list-scheduling algorithms will be surveyed as follows.

Table 1. *TLevel*, *BLevel*, *SLevel*, *ALAP*, and *NOO* of Each Node in the Task Graph of **Fig. 1**.

Node	*TLevel*	*BLevel*	*SLevel*	*ALAP*	*NOO*
n_1	0	37	12	0	8
n_2	6	23	8	14	4
n_3	3	23	8	14	3
n_4	3	20	9	17	2
n_5	3	30	10	7	2
n_6	10	15	5	22	1
n_7	22	15	5	22	1
n_8	18	15	5	22	1
n_9	36	1	1	36	0

A. The HLFET Algorithm

The HLFET (Highest Level First with Estimated Times) [3] first calculates the *SLevel* of each node in the task-graph. Then, make a ready-list in the descending order of *SLevel*. At each instant, it schedules the first node in the ready-list to the processor that allows the earliest-execution-time (using the non-insertion approach) and then, updates the ready-list by inserting the new nodes ready now to execute, until all the nodes are scheduled. For this algorithm, the time-complexity of the sequencing subproblem for a task-graph with n tasks is $O(n^2)$, where assigning tasks to the m given processor using EST belongs to $O(mn^2)$. **Fig. 2 (a)** shows the scheduling Gantt chart of the graph in **Fig. 1** using HLFET algorithm on two processor elements (the gaps between the tasks are because of the latencies inspired from the communication costs).

B. The MCP Algorithm

The MCP (Modified Critical Path) algorithm [9] uses the *ALAP* of the nodes as the priority. It first computes the *ALAP* times of all the nodes, and then constructs a ready-list in the ascending order of *ALAP*s. Ties are broken by considering the *ALAP* times of the children of the nodes. The MCP algorithm then schedules the nodes in the list one by one to the processor that allows the earliest-start-time using the insertion approach. For this algorithm, the time-complexity of the sequencing subproblem for a task-graph with n tasks is $O(n^2 \log n)$, where assigning tasks to the m given processor using EST belongs to $O(mn^2)$. The scheduling of the task graph in **Fig. 1** using MCP algorithm on two processor elements is shown by **Fig. 2 (b).**

C. The DLS Algorithm

The DLS (Dynamic Level Scheduling) algorithm [8] uses an attribute called dynamic-level (or *DL*) that is the difference between the *SLevel* of a node and its earliest-start-time on a processor. At each scheduling step, the DLS algorithm computes the *DL* for every node in the ready-list for all processors. The node-processor pair that gives the largest *DL* is selected to schedule, until all the nodes are scheduled. The algorithm tends to schedule nodes in a descending order of *SLevel* at the beginning, but nodes in an ascending order of their *TLevel* near the end of the scheduling process. The overall time-complexity of algorithm belongs to

$O(mn^3)$. **Fig. 2 (c)** shows scheduling of the task graph in **Fig. 1** using DLS algorithm on two processor elements.

D. The ETF Algorithm

The ETF (Earliest Time First) algorithm [7] computes the earliest-start-times for all the nodes in the ready-list by investigating the start-time of a node on all processors exhaustively. Then, it selects the node that has the smallest start-time for scheduling; ties are broken by selecting the node with the higher *SLevel* priority. The overall time-complexity of algorithm belongs to $O(mn^3)$. The scheduling of the task graph in **Fig. 1** using EST algorithm on two processor elements is shown by **Fig. 2 (d)**.

E. The ISH Algorithm

The ISH (Insertion Scheduling Heuristic) algorithm [4] uses the schedule-holes, the idle time slots, in the partial schedules. The algorithm tries to fill the holes by scheduling other nodes into them, and use *SLevel* as the priority measurement of a node. The overall time-complexity of the algorithm is $O(mn^2)$, and **Fig. 2 (e)** shows scheduling Gantt chart for the task graph in **Fig. 1** using ISH algorithm on two processor elements.

Table 2. A Comprehensive List of the Notations Applied to Formulate the Multiprocessor Task Graph-Scheduling Problem

Symbol	Description
$\mathbf{G} = (\mathbf{N}, \mathbf{E}, \mathbf{W}, \mathbf{C})$	A given task graph
$\mathbf{N} = \{n_1, n_2, \ldots, n_n\}$	Set of tasks in the task graph
$\mathbf{E} = \{(n_i, n_j) \mid n_i, n_j \in \mathbf{N}\}$	Set of edges (precedence constraints) among tasks in the task graph
$\mathbf{W} = \{w_1, w_2, \ldots, w_n\}$	Set of the required execution times of the tasks
$\mathbf{C} = \{c(n_i, n_j) \mid (n_i, n_j) \in \mathbf{E}\}$	Set of the communication costs (delays) among tasks in the task graph
n	The number of tasks in the task graph
entry-node	A node without any parents
exit-node	A node without any children
$\mathbf{P} = \{p_1, p_2, \ldots, p_m\}$	Set of processor elements
m	The number of available processors
Ready-List [ ]	Current set of the tasks ready to be scheduled considering precedence constraints among tasks
$Avail(p_j)$	The earliest time when p_j is ready to execute the next task
$AFT(n_k)$	The actual finish-time of task n_k
$AST(n_k)$	The actual start-time of task n_k
$EST(n_i, p_j)$	The earliest start-time of task n_i on processor p_j
Parents**(n_i)**	Set of all the parents of n_i
Children**(n_i)**	Set of all the children of n_i
$Processor(n_k)$	The processor on which task n_k is executed
makespan	The total finish time of a parallel program, or scheduling length

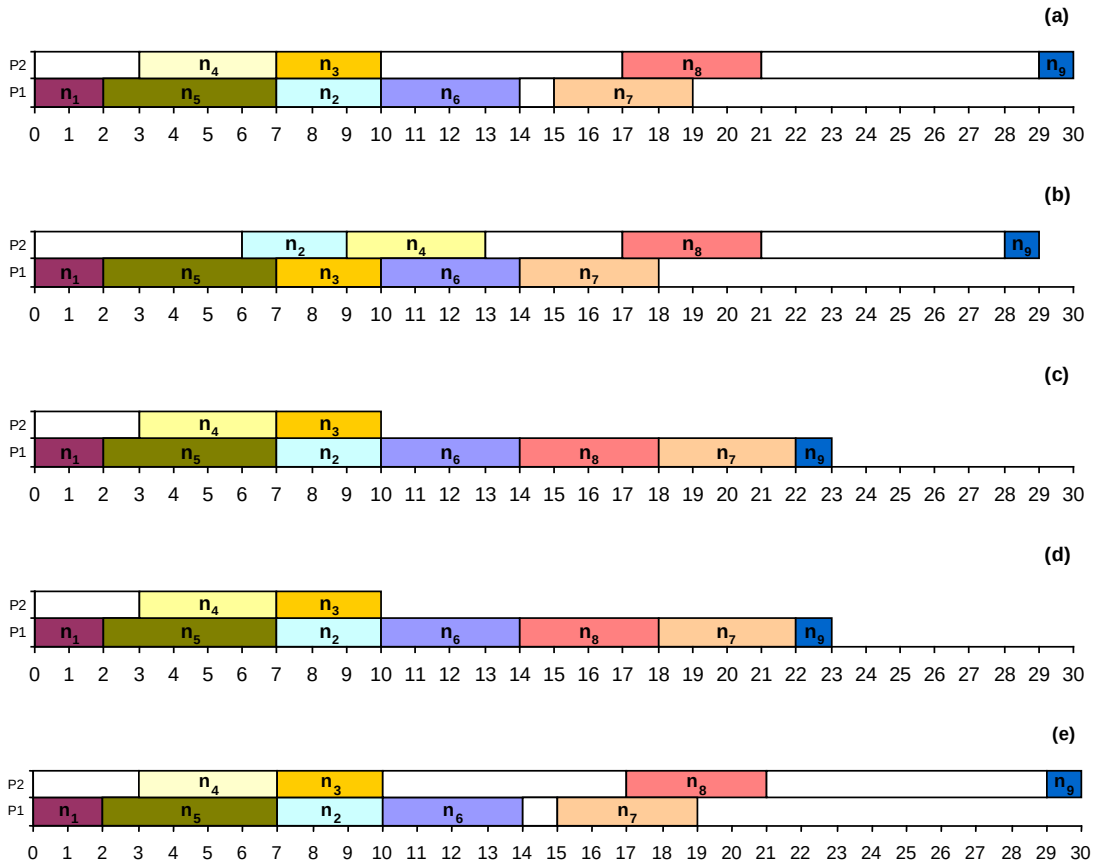

Fig. 2. The scheduling of the task graph of **Fig. 1** achieved by the four introduced traditional heuristics. (a) The HLFET algorithm. (b) The MCP algorithm. (c) The DLS algorithm. (d) The ETF algorithm. (e) The ISH algorithm.

3. ANT COLONY OPTIMIZATION

Ant colony metaheuristic is a concurrent algorithm in which a colony of artificial ants cooperates to find optimized solutions of a given problem. Ant algorithm was first proposed by Dorigo et al. as a multi-agent approach to solve traveling salesman problem (TSP) [12], and since then, it has been successfully applied to a wide range of difficult discrete optimization problems such as quadratic assignment problem, job-shop scheduling, vehicle routing, graph coloring, sequential ordering, network routing, to mention a few [13].

Leaving the nest, ants have a completely random behavior. As soon as they find a food, while walking from the food to the nest, they deposit on the ground a chemical substance called pheromone, forming in this manner a pheromone trail. Ants smell pheromone. Other ants are attracted by environment pheromone, and subsequently they will find the food source too. More pheromone is deposited, more ants are attracted, and more ants will find the food. It is a kind of autocatalytic behavior. In this way (by pheromone trails), ants have an indirect communication which are locally accessible by the ants so-called Stigmergy, a powerful tool

enabling them to be very fast and efficient. Pheromone is evaporated by sunshine and environment heat time by time destroying undesirable pheromone paths.

If an obstacle of which one side is longer than the other side cuts the pheromone trail. At first, ants have random motions to circle round the obstacle. Nevertheless, the pheromone of the longer side is evaporated faster, and little by little, ants will convergence to the shorter side, and hereby, they always find the shortest path from food to the nest vice versa.

Ant colony optimization tries to simulate this foraging behavior. In the beginning, each state of the problem takes a numerical variable named pheromone-trail or simply pheromone. Initially these variables have an identical and very small value. Ant colony optimization is an iterative algorithm. In each iteration, one or more ants are generated. In fact, each artificial ant is just a list (or Tabu-list) keeping the visited states by the ant. Ant is placed on the start state, and then selects next state using a probabilistic decision based on the value of pheromone trails of the adjacent states. Ant repeats this operation, until it reaches to the final state. In this time, the values of the pheromone variables of the visited states are increased based on the desirability of the achieved solution (depositing pheromone). Finally, all the variables are decreased simulating pheromone evaporation. By mean of this mechanism ants convergence to the more optimal solutions [13].

4. The Proposed Approach

The proposed approach is a hybridization of two different ACO-based subapproaches, each of which has its own strategies. The first subapproach use an artificial ant colony optimization to find the best possible sequence of tasks in the given task-graph (to solve the sequencing subproblem) [10], and the second subapproach try its bests to map the sequence obtained from the first subapproach to the existing processors using an incremental ACO-based method (to solve the assigning subproblem) [11].

A. The Sequencing Subapproach

At first, an $n \times n$ matrix named τ is considered as pheromone variables, where n is the number of tasks in the given task graph. Actually, τ_{ij} is the desirability of selecting task n_j, when the immediate previous selected task to be scheduled was the task n_i. All the elements of this matrix are initialized by a same and very small value according to the classical ACO. Then, the iterative ant colony algorithm is executed. Each iteration has the following steps:

1. Generate ant (or ants).
2. Loop for each ant (until a complete scheduling, that is the scheduling of all tasks in the task-graph).
   - Select the next task according to the pheromone variables and background knowledge of the ready-tasks using a probabilistic decision-making (roulette-wheel based selection).
3. Deposit pheromone on the visited states.
4. Daemon activities (to boost the algorithm)
5. Evaporate pheromone.

A flowchart of these operations with more details and an implementation in pseudo-code are also shown in **Fig. 3** and **Fig. 4**, respectively. In the first stage, just a list with the length of n, is created as ant. At first, this list is empty, and will be completed during the next iterative stage. In the second stage, there is a loop for each ant; in each iteration, the generated ant should select a task from the ready-list using a probabilistic decision-making based on the

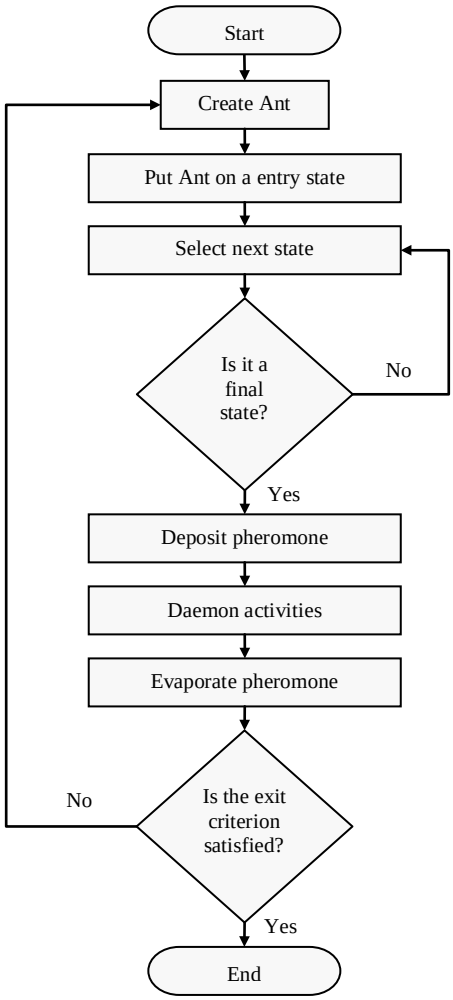

Fig. 3. The flowchart of the sequencing subapproach [10].

values of the pheromone variables and heuristic values (or priorities) of the ready tasks. Priority measurements introduced in Section II are used extensively as background knowledge about the problem, and efficiency of each related approach is highly correlated with how the approach is exploiting this background knowledge.

Therefore, in each iteration such as t, the desirability of selecting task n_j for scheduling, when the immediate previous selected task was the task n_i is obtained using the composition of the local-pheromone-trail values with the local-heuristic values as in

$$a_{ij}(t) = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_j]^{\beta}}{\sum_{l \in \mathrm{N}(t)}[\tau_{il}(t)]^{\alpha}[\eta_l]^{\beta}} \qquad \forall j \in \mathrm{N}(t), \tag{9}$$

where $\tau_{ij}(t)$ is the amount of pheromone on the edge (n_i, n_j) at time instant t, η_j is the heuristic value (priority measurement) of task n_j, N(t) is the current set of ready-tasks (ready-list), and α and β are two parameters that control the relative weight of pheromone-trail and heuristic-

```
00: int n ← the_number_of_tasks_in_the_task_graph;
01: int m ← the_number_of_rows and columns_of_cluster_computing_environment;
02: int Ready-List [1..n] ← 0;   {"Current set of the tasks ready to be scheduled considering precedence constraints"}
03: int rear ← 0;   {"The number of ready-tasks in the Ready-List [ ] at each iteration"}
04: int Parents [1..n] ← 0;   {"The number of yet unscheduled parents for each task"}
05: int w [1..n] ← Required execution times of the tasks
06: float τ [1..n, 1..n] ← ε;   {"Initiating the global pheromone matrix by a uniform very small value"}
07: float a [1..rear] ← 0;   {"The desirability of selecting each task among all the ready-tasks in the Ready-List [ ] "}
08: float p [1..rear] ← 0;   {"The probability of assigning each task among all the ready-tasks in the Ready-List [ ] "}
09: int FT [1..n] ← 0;   {"The actual finish-time for each task"}
10: int Ant^[1..x] [1..n];   {"Where x is the_total_number_of_the_ants"}
11: int Ant^min [1..n] ← ∞;
12: int makespan;
13: for k = 1 to the_total_number_of_the_ants
14:    Ant^k [1..n] ← 0;   {"Initiating Ant^k and Parents"}
15:    Parents [1..n] ← The total number of  parents for each task;
16:    rear ← 0;   {"Initializing the number of ready-tasks in the Ready-List [ ]"}
17:    for t = 1 to n   {"For all the tasks in the task-graph"}
18:       for i = 1 to n   {"Regeneration of the Ready-list [ ]"}
19:          if Parents [i] = 0 then
20:             AddQueue (n_i, Ready-List [ ]);   {"Insert n_i in to the rear of Ready-List [ ]"}
21:             rear ← rear + 1;
22:          Endif
23:       next i
24:       for j = 1 to rear   {"For all the ready-tasks in the Ready-List [ ]"}
25:          compute_the_desirability_vector (a^t [j]);   {"Using Eq. (9)"}
26:          compute_the_probability_vector (p^t [j]);   {"Using Eq. (10)"}
27:       next j
28:       r ← randomized_number (between [0, 1));
29:       Ant^k [t] ← for iteration t, select one of the ready-tasks roulette wheel based and according to the generated r and p^t [1..n];
30:       DeleteQueue (Ready-List [ ], Ant^k [t]);   {"Delete the selected task from the Ready-List [ ]"}
31:       for i = 1 to n   {"For each child of the selected task i.e. Ant^k [t]"}
32:          if Ant^k [t] ∈ Parents (n_i) then Parents [i] = Parents [i] - 1;
33:       next i
34:    next t

35:    --- {"Start of task-mapping using EST method, which should be replaced by the proposed ACO-based method"} ---

36:    for i = 1 to n   {"For the task-order generated by Ant^k"}
37:       AFT [i] = AST (n_i) + w [i];   {"Calculating actual finish-time for each task using Eq. (1) and (2)"}
38:    next i
39:    makespan ← MAX (AFT [1..n]);   {"The maximum finish-time among all the tasks"}

40:    --- {"End of task-mapping using EST method"} ---

41:    for i = 1 to n - 1   {"Depositing pheromone on the visited states by Ant^k"}
42:       update τ [Ant^k [i], Ant^k [i + 1]] based on the achieved makespan;   {"Using Eq. (11)"}
43:    next i
44:    if Ant^k < Ant^min then Ant^min = Ant^k;   {"Starting daemon activities"}
45:    for i = 1 to n – 1   {"Depositing pheromone on the visited states in Ant^min"}
46:       update τ [Ant^min [i], Ant^min [i + 1]] based on the makespan of Ant^min;   {"Using Eq. (12)"}
47:    next i
48:    for i = 1 to n
49:       for j = 1 to n
50:          τ [i, j] ← τ [i, j] × (1 – ρ);   {"Pheromone evaporation using Eq. (13)"}
51:    next i, j
52: next k
53: print Ant^min;
```

Fig. 4. The sequencing subapproach in pseudo-code [10].

value. It should be noted that different priority measurements such as *TLevel*, *BLevel*, *SLevel*, *ALAP*, and *NOO* can be used as heuristic values, and the best one should be selected experimentally. Accordingly, for ant k at time instant t, the probability of selecting task n_j just after selecting task n_i is computed using (10).

$$p_{ij}^{k}(t) = \frac{a_{ij}(t)}{\sum_{l \in \mathrm{N}(t)} a_{il}} \tag{10}$$

Then a random number is generated, and the next task will be selected according to the generated number using a roulette-wheel based selection; of course for each ready task in the ready-list, the higher pheromone-value and the higher priority, the bigger chance to be selected. Then, the selected task is appended to the ant's list, removed from the ready-list, and its children ready-to-execute-now will be augmented to the ready-list. These operations are repeated, until a complete scheduling of all the tasks is generated, which means the completion of the ant's list.

In the third stage, tasks are extracted one by one from the ant's list, and mapped to the processors that supply the earliest-start-time. Then, the maximum finish-time is calculated as *makespan* that is also the desirability of the obtained scheduling for this ant. According to this desirability, the quantity of pheromone which should be deposited on the visited states by this ant is calculated using

$$\Delta\tau_{ij}^{k} = \frac{1}{L^{k}} \qquad if\,(n_i, n_j) \in T^{k}, \tag{11}$$

where L^k is the overall finish-time or *makespan* obtained by the ant k and T^k is the executed tour of this ant. Accordingly, $\Delta\tau_{ij}^{k}$ should be deposited on every τ_{ij} if and only if the (n_i, n_j) exists in the T^k (task n_j has been selectedby this ant just after selecting task n_i); otherwise, τ_{ij} will remain unchanged.

In the forth stage (daemon activity), to intensify and to avoid removing good solutions, the best-ant-until-now ($\mathrm{Ant}^{\min}$), is selected, and some extra pheromone is deposited on the states visited by this ant using (12).

$$\Delta\tau_{ij}^{\min} = \frac{1}{L^{\min}} \quad if\,(n_i, n_j) \in T^{\min} \tag{12}$$

Last by not least, using (13), pheromone variables are decreased simulating pheromone evaporation in the real environments. It should be taken into account to avoid premature convergence and stagnation because of the local minima.

$$\tau_{ij} = (1-\rho)\tau_{ij}, \tag{13}$$

where, ρ is the evaporation rate in the range of [0, 1) should be determined experimentally.

B. The Assigning Subapproach

At first, a $n \times m$ matrix named τ is considered to represent the pheromone variables, where n is the number of tasks in the given task graph, and m is the number of existing processors. Actually, τ_{ij} is the desirability of assigning task n_i to processor p_j. All the matrix elements are initiated by the same and very small value as in the classical ACO. Actually, each ant is a list with the length n and has a novel encoding as follows. Each element in the ant's list, such as ant [6] = 2, demonstrates that the task n_6 will be executed on the processor p_2. To clarify the issue, a typical ant's list filled by the proposed approach, along with its task-order generated

by ants in the previous sequencing subapproach, and the corresponding scheduling on two processors, demonstrated by a Gantt chart, is shown in **Fig. 5**.

Afterwards, the iterative ant colony algorithm is executed; each individual iteration has the following steps:

1. Generate ant (or ants).
2. Loop for each ant (until the complete scheduling of all tasks in the selected task order).
   - Assign the next task to the processors using a probabilistic decision-making based on the pheromone variables.
3. Deposit pheromone on the visited states.
4. Evaporate pheromone.
5. Daemon activities (to improve the results)

The flowchart of these operations with more details and an implementation in pseudo-code are provided in **Figs. 6** and **Fig. 7**, respectively. In the first stage, just a list with length n is constructed as an ant, and filled by the task order generated by the previous scheduling subapproach.

In the second stage, there is a loop for each ant. In each iteration such as t, the active ant assigns the next node in the selected task order into the supposed suitable processor using a probabilistic decision making based on the current values of the pheromone variables. The desirability of assigning task n_i to processor p_j at time instant t is obtained using

$$a_{ij}(t) = \frac{\tau_{ij}(t)}{\sum_{l \in \mathrm{N}(t)} \tau_{il}(t)} \qquad \forall j \in \mathrm{N}(t), \tag{14}$$

where $\tau_{ij}(t)$ is the amount of pheromone on the edge (n_i, p_j) at time instant t and $\mathrm{N}(t)$ is the set of m existing processors. For ant k at time instant t, we need to compute the probability of assigning task n_i to each processor such as p_j using (15).

$$p_{ij}^{k}(t) = \frac{a_{ij}(t)}{\sum_{l \in \mathrm{N}(t)} a_{il}} \qquad \forall j \in \mathrm{N}(t) \tag{15}$$

Actually, $p_{ij}^{k}(t)$ for all the existing processors should be computed. Then, a random number in the range of [0, 1) is generated, and a processor will be selected according to the generated number using roulette-wheel based selection; of course for each processor, the higher the pheromone value, the bigger the chance to be selected. These operations are repeated until a

A typical task-order generated by ants in the sequencing subapproach	n_1	n_5	n_4	n_2	n_3	n_6	n_7	n_8	n_9

A typical Ant's list in the assigning subapproach	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9
	1	1	2	2	1	1	1	2	2

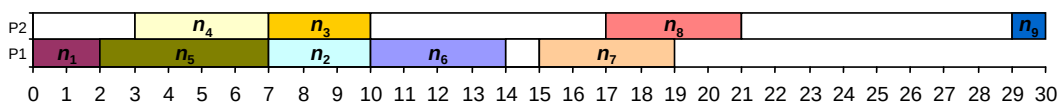

Fig. 5. A typical task-order generated by ants in the sequencing subapproach, and a typical ant's list filled by the assigning subapproach, along with the corresponding scheduling on two processors, demonstrated by a Gantt chart [11].

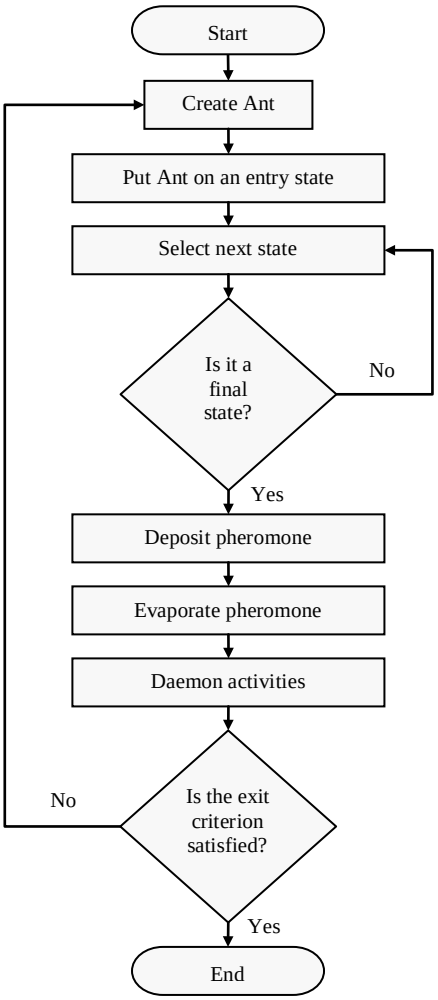

Fig. 6. The flowchart of the assigning subapproach [11].

complete assigning of all the tasks happens, which means the completion of the ant's list.

In the third stage, based on each selected pair of (n_i, p_j), using (3), the maximum finish-time is calculated as *makespan*, which is also the desirability of the assigning obtained by this ant, and according to this desirability, the quantity of pheromone that should be deposited on the states visited by this ant is calculated using

$$\Delta\tau_{ij}^{k} = \frac{1}{L^{k}} \qquad if\,(n_i, p_j) \in T^{k}, \tag{16}$$

where L^k is the overall finish-time or *makespan* obtained by ant k and T^k is the set of all (n_i, p_j) selected by this ant. Accordingly, $\Delta\tau_{ij}^{k}$ should be deposited on every τ_{ij} if and only if (n_i, p_j) exists in T^k (task n_j has been executed on the processor p_j); otherwise, τ_{ij} will remain unchanged.

In the fourth stage, using (17), all the pheromone variables are decreased to simulate pheromone evaporation in real environments. Of course, this stage is very important to prevent premature convergence and stagnation caused by the local minima in the search space.

```
01: int n ← the_number_of_nodes_in_the_task_order;
02: int m ← the_number_of_existing_processor_elements;
03: int task-order [1..n] ← a selected topological task-order to assign to the processors, extracted by the sequencing subapproach;
04: float τ[1..n, 1..m] ← ε;   {"Initiating the global pheromone matrix"}
05: float a [1..m] ← 0;   {"The desirability of assigning a task to each of the m existing processors"}
06: float p [1..m] ← 0;   {"The probability of assigning a task to each of the m existing processors"}
07: int Avail [1..m] ← 0;   {"The earliest time for each processor to be available for running the next task"}
08: int Ant^{1..x}[1..n];   {"Where x is the_total_number_of_the_ants"}
09: int Ant^min [1..n] ← ∞;
10: int makespan;
11: for k = 1 to the_total_number_of_the_ants
12:    Ant^k [1..n] ← 0;   {"Initiating the Ant^k"}
13:    Avail [1..m] ← 0;
14:    for t = 1 to n   {"For each task in the task graph"}
15:       for i = 1 to m
16:          compute_the_desirability_vector (a^t [i]);   {"Using Eq. (14)"}
17:          compute_the_probability_vector (p^t [i]);   {"Using Eq. (15)"}
18:       next i
19:       r ← randomized_number (between [0, 1));
20:       Ant^k [t] ← for the task n_t in Ant^k, select one of the m processors roulette-wheel based and according to the r and p^t [m];
21:       for j = 1 to n   {"For each parent of the task n_t e.g. n_j∈Parents(n_t)"}
22:          update EST (n_t, Ant^k [t]);   {"Using Eq. (1)"}
23:       next j
24:       Avail [Ant^k [t]] ← EST (n_t, Ant^k [t]);
25:    next t
26:    makespan ← MAX (Avail [1..m]);
27:    for i = 1 to n   {"Each task–processor pair in Ant^k"}
28:       update τ[i, Ant^k [i]] based on the makespan;   {"Using Eq. (16)"}
29:    next i
30:    for i = 1 to n
31:       for j = 1 to m
32:          τ[i, j] ← τ[i, j] × (1 – ρ);   {"Pheromone evaporation using Eq. (17)"}
33:    next i, j
34:    if Ant^k < Ant^min, then Ant^min = Ant^k;   {"Starting daemon activities"}
35:    for i = 1 to n   {"Each task–processor pair in Ant^min"}
36:       update τ[i, Ant^min [i]] based on the makespan of Ant^min;   {"Using Eq. (18)"}
37:    next i
38: next k
39: makespan ← Ant^min;
```

Fig. 7. The assigning subapproach in pseudo-code [11].

$$\tau_{ij} = (1-\rho)\tau_{ij}, \tag{17}$$

where again ρ is the evaporation rate in the range of [0, 1) and should be determined experimentally.

In the last stage, we have daemon activity (any other activities than real ant colonies such as local search, extra pheromone deposition, looking ahead, backtracking, and so on to boost the ACO). In this stage, to enhance the performance of the proposed approach and to especially avoid removing good solutions, the best-ant-until-now (Ant^{min}) is selected (as the best solution), and some extra pheromone is deposited on the states visited by this ant using (18).

$$\Delta\tau_{ij}^{\min} = \frac{1}{3\times L^{\min}} \qquad if\,(n_i, p_j) \in T^{\min} \tag{18}$$

5. Implementation and Experimental Results

The proposed approach was implemented on a Pentium IV (8-core 3.9 GHz i7–3770K processor) desktop computer with Microsoft Windows 7 (X64) platform using Microsoft Visual Basic 6.0 programming language. In the both subapproaches, all the initial values of the pheromone variables were identically set to 0.1. The evaporation rate was considered as 0.998, and the parameters α and β were elected 1 and 0.5 respectively obtained experimentally. The algorithm was terminated after 2500 iterations, that is after generating 2500 ants as explained and used in [10] and [11].

The implementation of the first subapproach approach in pseudo-code in **Fig. 4** reveals that there are two nested-iterations in the sequencing subproblem (lines 13, 17, and 18) and (lines 13, 48, and 49) with time-complexity $\in \theta(the\_total\_number\_of\_the\_ants \times n^2)$ where n is the number of tasks in the task graph. Since *the_total_number_of_the_ants* is a constant initiated to 2500, for the big-enough numbers of n, we can assume that the overall time-complexity of the proposed approach for sequencing subproblem belongs to the $O(n^2)$, which is equal or better than the traditional preintroduced heuristic methods. Also, the time-complexity of assigning the generated task-order to the existing m processors using EST method (lines 36 and 37) is $O(mn^2)$ as usual.

On the other hand, the implementation of the second subapproach in pseudo-code in **Fig. 7** suggests that there are a main nested-iteration in the algorithm (lines 11, 14, 15, and 21); hence, the overall time-complexity of the proposed approach is $\in \theta\,(the\_total\_number\_of\_the\_ants \times (mn + n^2))$, where n is the number of tasks in the task graph, and m is the total number of the available processors. Because *the_total_number_of_the_ants* is a constant limited to 2,500, for sufficiently large numbers of n and m, we can assume that overall time complexity of the proposed assigning subapproach belongs to $O(mn + n^2)$, which is slightly better than the traditional EST method whose time complexity is $O(mn^2)$. That is, for large-scale samples, we expect that the actual performance of the proposed approach will be slightly better than the results presented in the following experiments.

A. The Utilized Dataset

Table 3 lists six task graphs of the real-world applications (and their comments) considered to evaluate the proposed approach. These six graphs are the standard ones in the literature, and utilized to evaluate a number of related works; hence, they let us to compare the proposed approach against its traditional counterparts. All these six graphs are used to compare the proposed approach with the traditional heuristics not only the list-scheduling algorithms but also the other scheduling methods. Also, the proposed approach will be evaluated in comparison with the best genetic algorithm introduced in the problem's literature [16] using the last two graphs (that are G5 and G6).

In addition, a set of 125 random task graphs are used for a rational judgment and better evaluation of the proposed hybrid approach. These random task graphs have different shapes on three following parameters.

Table 3. Selected Task Graphs for Evaluating the Proposed Approach

Graph	Comments	Nodes	Communication Costs
G1	Kwok and Ahmad [2]	9	Variable
G2	Al-Mouhamed [14]	17	Variable
G3	Wu and Gajski [9]	18	60 and 40
G4	Al-Maasarani [15]	16	Variable
G5	**Fig. 1** [16]	9	Variable
G6	Hwang *et al.* [16]	18	120 and 80

- Size (n): that is the number of tasks in the task graph. Five different values were considered {32, 64, 128, 256, and 512}.
- Communication-to-Computation Ratio (*CCR*): demonstrate how much a graph is communication or computation base. The weight of each node was randomly selected from uniform distribution with mean equal to the specified average computation cost that was 50 time-instance. The weight of each edge was also randomly selected from uniform distribution with mean equal to average-computation-cost × *CCR*. Three different values of *CCR* were selected {0.1, 0.5, 1.0, 5.0, and 10.0}. Selecting 0.1 makes computation intensive task-graphs. In contrast, selecting 10.0 makes communication intensive ones.
- Parallelism: the parameter which determine the average number of children for each node in the task-graph. Increase in this parameter makes the graph more connected. Three different values of parallelism were chosen {3, 5, 10, 15, and 20}.

Because the archived makespan of these random graphs are in the wide range regarding their various parameters, *NSL* (normalized schedule length), which is a normalized measure, is used. It can be calculated for every given task-graph by dividing the achieved *makespan* to the lower-bound defined as the sum of weights of the nodes on the original critical path (CP) using

$$NSL = \frac{Schedule - Length}{\sum_{n_i \in CP} w_i}, \tag{19}$$

where *CP* is the set of nodes on the critical path (the longest path) of the given graph.

B. The Expriments and Results (Sequencing Subapproach)

The first set of experiments has been conducted to select a proper priority measurement as heuristic values for using in Eq. (9). **Table 4** lists the results of mean of 10 times of algorithm execution on all the given six graphs using various priority measurements that are *TLevel*, *BLevel*, *SLevel*, *ALAP*, and *NOO* introduced in Section II. Since the algorithm using *ALAP* was statistically more successful in average *NSL*, this priority measurement will be used in all the subsequent experiments.

Table 4. Results of Mean of 10 Times of Execution of Algorithm Using Different Priority Measurement as Heuristic Values

Graph	*CPL*	*TLevel*	*BLevel*	*SLevel*	*ALAP*	*NOO*
G1	11	16.9	16.2	16.1	16.200	16.2
	NSL:	1.536	1.473	1.464	1.473	1.473
G2	28	38	38	38	38	38
	NSL:	1.357	1.357	1.357	1.357	1.357
G3	300	390	390	390	390	390
	NSL:	1.300	1.300	1.300	1.300	1.300
G4	34	47	47	47	47	47
	NSL:	1.382	1.382	1.382	1.382	1.382
G5	12	23.9	23.5	23.4	23.7	23.2
	NSL:	1.992	1.958	1.950	1.975	1.933
G6	300	470	470	459	443	470
	NSL:	1.567	1.567	1.530	1.477	1.567
Average *NSL:*		**1.522**	**1.506**	**1.497**	**1.494**	**1.502**

Table 5. The Best Achieved Scheduling of the Proposed Approach (ACO) and the Four Heuristics Using Only Two Processor Elements

Graph	HLFET	ISH	MCP	DLS	ETF	ACO
G1	23	23	19	21	21	17
G2	44	44	43	46	44	42
G3	410	410	420	410	400	390
G4	63	59	62	60	60	52
G5	30	30	29	23	23	21
G6	540	520	550	520	520	440

The second set of experiments evaluates the proposed approach against all the five traditional heuristics introduced in Section II. All the six given graphs in **Table 3** have been used. **Table 5** shows the best scheduling achieved by the proposed approach along with the others using only two processor elements. Restricting the number of processors is a key experiment which reveals how much a method is capable to produce compact scheduling and to operate correctly in the lack of resource. As it can be seen, the proposed approach outperforms the other heuristics in all the cases.

In the next set of experiments, the number of processor elements is large enough for each algorithm to show its best performance. This is another key experiment revealing how efficient an approach is in terms of existing sufficient resources. This experiment makes it possible to compare the proposed approach versus a wide range of the traditional heuristics introduced in the literature. The results of these experiments is listed in **Table 6**. Again, the proposed approach has a better performance versus the others, and this is another strong evidence to verify the efficiency of the proposed sequencing subapproach.

Finally, the last two graphs (G5 and G6) evaluate the proposed subapproach compared to the one of the best genetic algorithms (GA) introduced for homogeneous multiprocessor task-graph scheduling (without task duplication as our proposed method) [16]. **Table 7** lists the achieved results of not only these two algorithms but also four other traditional ones (for a better justification). The results show the proposed approach as well as the genetic algorithm outperforms the other methods, yet the proposed approach has a better performance on the first graph. While, in this genetic algorithm, each generation has 100 chromosomes, and the maximum number of generations is 1000; that is, it achieves its best scheduling by generating 100,000 solutions while the proposed approach examines only 2500 complete scheduling (2500 ants) to find its best answer. In other words, the proposed approach finds its solution so faster than the genetic algorithm. It is logical here because in contrast to the GA in which most of the operations are random, and most of the experiences are also dropped away in each selection phase, the ant colony optimization has an indirect communication by the pheromone variables, called stigmergy, so that each new decision is based on the experiences of all the previously active ants. Moreover, most of the efficiency of the proposed ACO-based approach here is for the properly utilization of background knowledge about the problem (the priority measurements of tasks stated and emphasized in the Section II).

Table 6. THE BEST ACHIEVED RESULTS OF THE PROPOSED APPROACH (ACO) AND SOME TRADITIONAL HEURISTIC METHODS [2] AND [16].

Graph	LC	EZ	MD	DSC	DCP	HLFET	ISH	ETF	LAST	MCP	DLS	ACO
G1	19	18	17	-	16	19	19	19	19	20	19	16
G2	39	40	38	38	38	41	38	41	43	40	41	38
G3	420	540	420	390	390	390	390	390	470	390	390	390
G4	-	-	-	-	-	48	48	48	-	48	47	47
G5	-	-	32	27	23	29	29	29	-	29	29	21
G6	-	-	460	460	440	520	520	520	-	520	520	440

Table 7. The Best Achieved Results of ACO, Genetic Algorithm [16], and Four Other Traditional Heuristics [10]

Graph	MCP	DSC	MD	DCP	GA [16]	ACO
G5	29	27	32	32	23	21
G6	520	460	460	440	440	440

C. The Expriments and Results (Assignng Subapproach)

One of the most challenging issues in utilization of the meta-heuristic approaches is their poor performance, encountering problems with huge dimensions, that is, when each state of the problem has a large number of neighborhoods. In such a condition, each decision goes almost scholastic, and the convergence cannot be achieved but with a large number of iterations. Here in the assigning subproblem, increasing the number of processors makes the dimensions of the problem to grow. As a result, the proposed ACO-based assigning subapproach will be caught in the situation in which most of the decisions are taken stochastically, with no convergence to a good solution.

In order to tackle the issue, we revised the proposed approach to an incremental one, that is, we break the whole colony into m–1 different subcolonies, each of which uses a different number of available processors in an incremental manner, where m is the total number of existing processors. They sequentially explore the problem space. For example, if m = 6, then there will be five different subcolonies. The first subcolony explores the problem using only two processors to schedule the tasks. The second subcolony uses three processors to work, the next one uses four, and so on. All the ants' experiences such as pheromone trails, Ant^{min} etc. will be retained and sent forward to the next subcolonies. By utilizing this incremental policy, the ACO, if needed, tries to use fewer processors, meaning a compact scheduling that leads to finding better solutions with faster convergence in a number of cases. **Fig. 8** shows the superiority of the proposed approach in comparison with the basic EST method and a more sophisticated CLA-based approach introduced in [17] and [18] for task mapping in multiprocessor environments.

D. Final Expriments and Results (The Hybrid Approach)

Now, the sequencing and assigning subapproaches are gathered together making a hybrid approach (ACO-ACO or simply ACO) to be tested as a whole system. All the final experiments are conducted using all the aforementioned 125 random task graphs as input samples. The first set of the experiments is regarding different graph sizes in order to investigate the impact of this parameter on the final results achieved by the proposed approach and the others as well. **Fig. 9** illustrates the achieved results (in *NSL*) for the final hybrid proposed approach besides its traditional counterparts. Obviously, the lesser *NSL*, the better performance. The entire 125 random task graphs are used, and the results are favored the proposed approach. Generally speaking, the achieved *NSL*s grows proportionately according to the increase in the input graph sizes, but the relation in not linear. In average *NSL*, the performance ranking of the approaches is {ACO, ETF, MCP≈DLS, ISH, HLFET}. It should be note that each ranking starts with the best approach and ends with the worst one with respect to the given comparison metric; that is, the ACO was the best (especially with the large-scale inputs), ETF was a little bit worse, MCP and DLS were moderate and about identical (MCP was slightly better than DLS with the small-scale graphs), and the ISH and HLFET were the worst methods. The number of processors was large enough for each approach to produce its best scheduling; however, all the presented experiments were done again using only two processor elements, and the achieved ranking were exactly identical with the results, and certified the final conclusions.

In addition, for investigating the effect of the number of processor used, another set of experiments is conducted. **Fig. 10** shows the diagram for the achieved *NSL*s of the entire 125 random task graphs regarding utilization of the different number of processors ranging from 2 processor up to the 64 ones. Again, the performance ranking of the approaches is {ACO, ETF, MCP≈DLS, ISH, HLFET}, and the proposed approach is the winner of the race versus other methods in all the cases. The performance difference between the proposed approach and their counterparts is increasing regarding the increase in the number of processor used; it is because the ACO-based mapper is able to dynamically accommodate itself with growing in the problem dimensions using the incremental policy, while other methods use EST to map the selected sequence on the processors which has a static method and cannot exploit the full potentials of increase in the number of resources (processors).

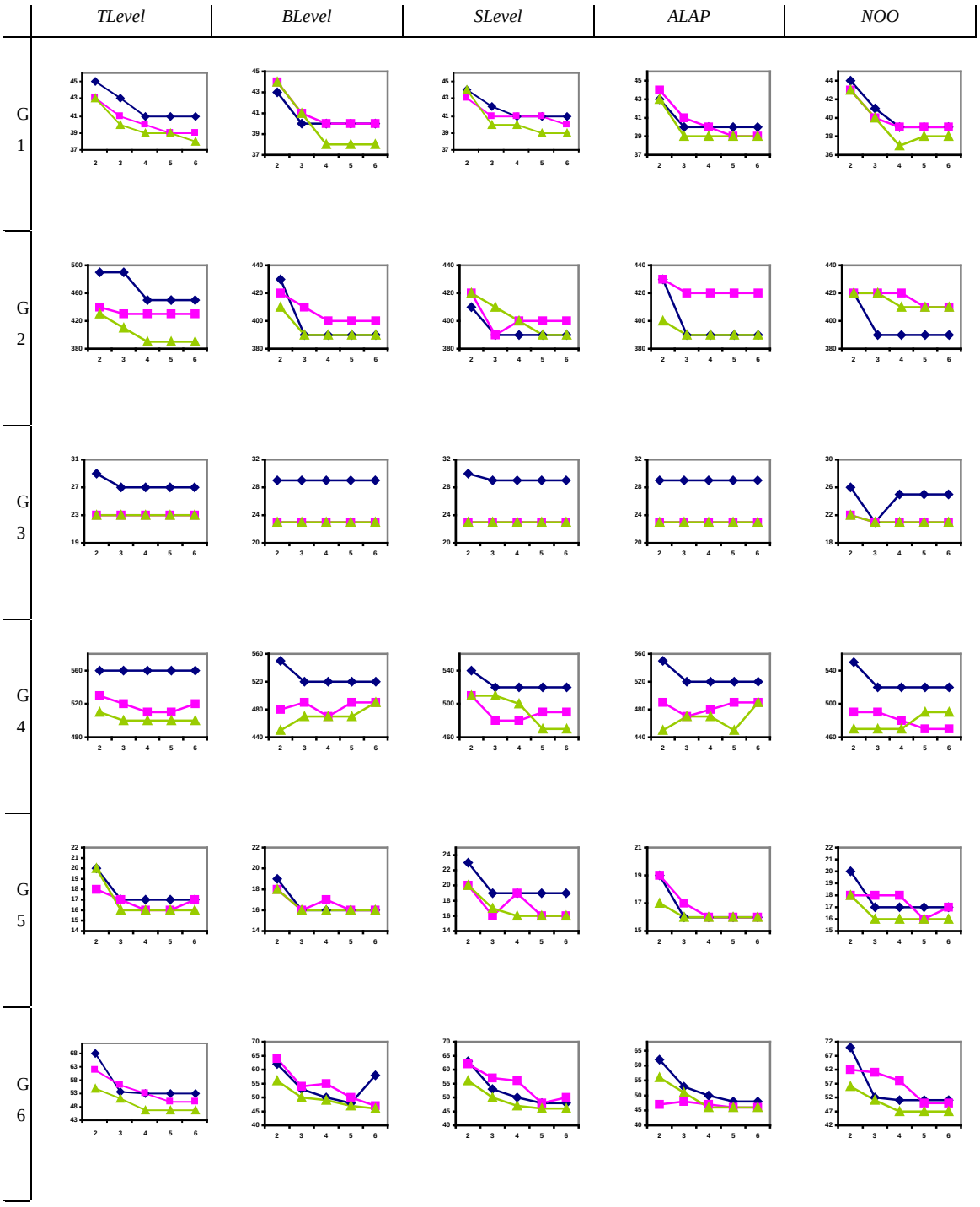

Fig. 8. The results obtained by the proposed ACO-based assigning subapproach versus the traditional EST and a more sophisticated CLA-based method, for all the different task orders of each of the six task graphs shown in **Table 2**, using different numbers of processor ranging from 2 to 6: EST CLA ACO [11].

Fig. 11 shows the average *NSL*s achieved by all the experiments conducted on entire 125 random task-graphs. As a rule, the final performance ranking is {ACO, ETF, MCP≈DLS, ISH,

HLFET}, that is the ACO is the best (especially with the large-scale inputs), ETF is a little bit worse, MCP and DLS were moderate and about identical (their rankings change alternately in the different experiments), and the ISH and HLFET were the worst methods from the performance point of view.

6. CONCLUSION

In this paper, a new hybrid approach based on the ant colony optimization for static homogeneous multiprocessor task-graph scheduling problem was introduced. In the proposed approach, two different artificial ant colonies cooperate to make a complete solution; first colony finds the best possible sequence of the tasks in the given task-graph (to solve the sequencing subproblem), and the second assigns the sequence obtained from the first subapproach to the existing processors in the best way using an incremental ACO-based method (to solve assigning subproblem). The first set of experiments was conducted to specify the qualified priority measurement as heuristic values; *TLevel*, *BLevel*, *SLevel*, *ALAP*, and *NOO* were considered, and finally the results showed that *ALAP* is more appropriate. Six task-graphs from real-world programs were selected to evaluate the proposed approach against the traditional methods. Two sets of experiments were done to compare the proposed approach with traditional heuristics, one on the restricted number of processors (restricted by two processor elements), and another on an unbounded number of ones. The proposed method outperformed the others in all the cases. In addition, a comparison with one of the best introduced GA-based approaches in the literature shows the superiority of the proposed approach not only in terms of performance but also in terms of time-complexity. The next set of experiments was conducted to evaluate the ACO-based task assigner which was proposed to tackle the assigning subproblem. The obtained results show the superiority of the proposed approach in comparison with the basic EST method and a more sophisticated CLA-based approach introduced in the literature. Eventually, the proposed approach was the winner of the race on a comprehensive set of 125 random task-graphs with different shape parameters such as size, *CCR* and the parallelism. All of these are of the strong evidences to demonstrate the capability and superiority of the proposed approach in static homogeneous multiprocessor task-graph scheduling.

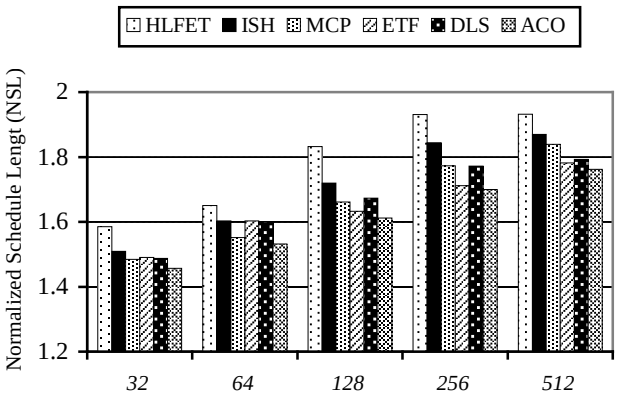

Fig. 9. The achieved results (in *NSL*) of the proposed approach besides its traditional counterparts on the entire 125 random task-graphs with respect to the different graph sizes.

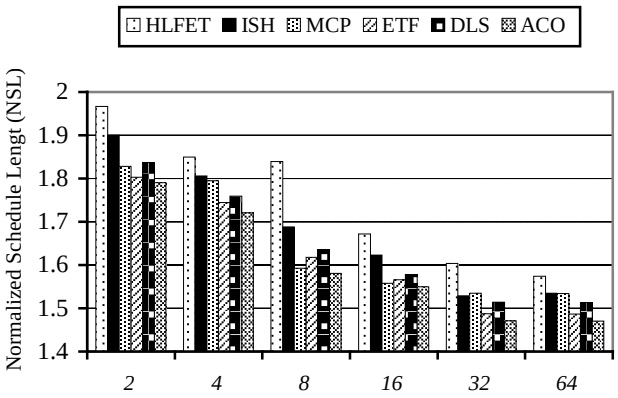

Fig. 10. The diagram of the achieved *NSL*s of the entire 125 random task graphs regarding the different number of utilized processors.

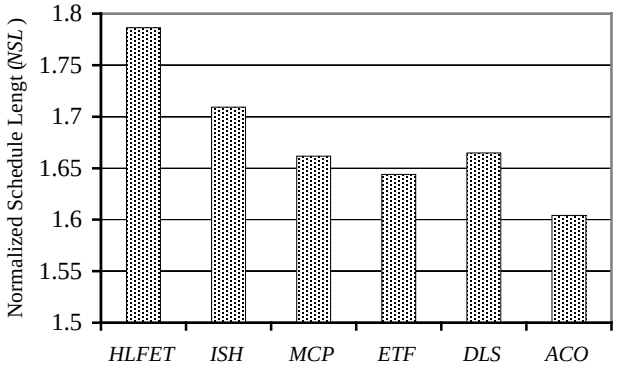

Fig. 11. The average *NSL* of all the experiments conducted on entire 125 random task-graphs.

ACKNOWLEDGEMENTS

Thanks to the Sama Technical and Vocational Training College, Islamic Azad University, Shoushtar Branch, Shoushtar, Iran, for supporting this study as a research project.

REFERENCES

[1] P. Chretienne *et al*, *Scheduling Theory and Its Application*, New York, Wiley, 1995.
[2] Kwok Yu-Kwong, and Ishfaq Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406-471, 1999. Article (CrossRef Link)
[3] Thomas L. Adam, K. Mani Chandy and J. R. Dickson, "A comparison of list schedules for parallel processing systems," *Communications of the ACM*, vol. 17, no. 12, pp. 685-690, 1974. Article (CrossRef Link)
[4] B. Kruatrachue and TG. Lewis, "Duplication Scheduling Heuristics (DSH): A New Precedence Task Scheduler for Parallel Processor Systems," *Technical report: Oregon State University*, Corvallis, Report No.: OR 97331, 1987.

[5] McCreary Carolyn, and Helen Gill, "Automatic determination of grain size for efficient parallel processing," *Communications of the ACM*, vol. 32, no. 9, pp. 1073-1078, 1989. Article (CrossRef Link)
[6] J. Baxter and JH. Patel, "The LAST Algorithm: A Heuristic-Based Static Task Allocation Algorithm," in *Proc. of the 1989 Int'l Conf. Parallel Processing*, pp. 217-222, Aug. 1989.
[7] JJ. Hwang, YC. Chow, FD. Anger and CY. Lee, "Scheduling Precedence Graphs in Systems with Interprocessor Communication Times," *SIAM J. Computing*, vol. 18, no. 2, pp. 244-257, Apr. 1989. Article (CrossRef Link)
[8] GC. Sih and EA. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. Parallel and Distributed Systems*, vol. 4, no. 2, pp. 75-87, Feb. 1993. Article (CrossRef Link)
[9] MY. Wu and DD. Gajski, "Hypertool: A Programming Aid for Message-Passing Systems," *IEEE Trans. Parallel and Distributed Systems*, vol. 1, no. 3, pp. 330-343, Jul. 1990. Article (CrossRef Link)
[10] H. R. Boveiri, "A Novel ACO-Based Static Task Scheduling Approach for Multiprocessor Environments," *International Journal of Computational Intelligence Systems*, vol. 9, no. 5, pp. 800-811, 2016. Article (CrossRef Link)
[11] H. R. Boveiri, "An incremental ACO-based approach to task assignment to processors for multiprocessor scheduling," *Frontiers of Information Technology & Electronic Engineering*, 2016, In the Press. Article (CrossRef Link)
[12] M. Dorigo, V. Maniezzo and A Colorni, "Positive feedback as a search strategy," *Technical report: Politecnico di Milano*, Milan, Report No.: 91-016, 1991.
[13] M. Dorigo, G. DiCaro and L. Gambardella, "Ant Algorithm for Discrete Optimization," *Artificial Life*, vol. 5, no. 2, pp. 137-172, 1999. Article (CrossRef Link)
[14] MA. Al-Mouhamed, "Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs with Communication Costs," *IEEE Trans. Software Engineering*, vol. 16, no. 12, pp. 1390-1401, Dec. 1990. Article (CrossRef Link)
[15] A. Al-Maasarani, "Priority-Based Scheduling and Evaluation of Precedence Graphs with Communication Times," *M.S. Thesis: King Fahd University of Petroleum and Minerals*, Saudi Arabia, 1993.
[16] R. Hwang, M. Gen and H. Katayama, "A comparison of multiprocessor task scheduling algorithms with communication costs," *Computer & Operations Research*, vol. 35, no. 3, pp. 976-993, 2008. Article (CrossRef Link)
[17] H. R. Boveiri, "Assigning Tasks to the Processors for Task-Graph Scheduling in Parallel Systems Using Learning and Cellular Learning Automata," in *Proc. of the 1st National Conf. on Comp. Eng. and Info. Tech*, pp. 1-8, Shoushtar, Iran, Feb. 2014 (in Farsi).
[18] H. R. Boveiri, "Multiprocessor Task Graph Scheduling Using a Novel Graph-Like Learning Automata," *International Journal of Grid and Distributed Computing*, vol. 8, no. 1, pp. 41-54, Feb. 2015. Article (CrossRef Link)
[19] H. R. Boveiri, "An Efficient Task Priority Measurement for List-Scheduling in Multiprocessor Environments," *International Journal of Software Engineering and Its Applications (IJSEIA)*, vol. 9, no. 5, pp. 233-246, May 2015. Article (CrossRef Link)

Hamid Reza Boveiri received his B.Sc. degree from Birjand University, Birjand, Iran, in 2005, M.Sc. degree from IAU, Science and Research Branch, Ahvaz, Iran, in 2009, both in software engineering, and Ph.D. candidate in Shiraz University of Technology, Shiraz, Iran, in information technology (*IT*), communications networks. He is currently a faculty member of Computer Engineering Department of Sama College, IAU, Shoushtar Branch, Shoushtar, Iran. He had been serving as the Dean of IAU, Gotvand Branch, Iran, from 2014 up to 2016. He was also a member of Young Researchers & Elites Club of IAU, Shoushtar Branch, and there, he was the head advisor of research workgroup from 2010 up to 2012. He has already published more than 40 research articles, surveys and technical reports in prestigious national and international conferences and journals. His research interests include scheduling & optimization, machine learning & meta-heuristics, signal processing, and pattern recognition, and parallel & distributed systems. He is a member of *ISI* (Information Society of Iran) and *IEEE*.

Raouf Khayami received his BS degree in computer engineering (hardware systems) in 1993, the MS degree in artificial intelligence and robotics in 1996, and the Ph.D. in software systems in 2009, all from Shiraz University, Shiraz, Iran. He is currently an assistant professor in the Computer Engineering and Information Technology Department, Shiraz University of Technology, Shiraz, Iran, and there, he is the Head of the Department. His research interests include data mining, business intelligence, and enterprise architecture, on which he has published a number of refereed articles, surveys and technical reports in prestigious national and international conferences and journals. He is also active in consulting and industrial projects.