Using Bee Colony Optimization to Solve the Task Scheduling Problem in Homogenous Systems

Vahid Arabnejad[1], Ali Moeini[2] and Nasrollah Moghadam[3]

**[1] Department of Computer Engineering, Islamic Azad University, South branch
Tehran, Iran**

**[2] Computer Engineering Dept., University of Tehran
Tehran, Iran,**

**[3] Computer Engineering Dept., University of Tarbiat Modares
Tehran, Iran**

Abstract

Bee colony optimization (BCO) is one of the most recent algorithms in swarm intelligence that can be used in optimization problems this algorithm is based on the intelligent behavior of honey bees in foraging process. In this paper bee colony optimization is applied to solve the task scheduling problem which tasks have dependency with each other. Scheduling of tasks that represents by directed acyclic graph is a NP-complete problem. The main purpose of this problem is obtaining the minimum schedule length that is called make-span. To realize the performance of BCO in this problem, the obtained results are presented and compared with the most successful methods such as Ant colony system, Tabu search and simulate annealing. The comparison shows that BCO produces the solutions in a different way and it is still among the bests.

Keywords: *Bee Colony Optimization, Task Graph, Task Scheduling Problem, Homogenous Processors.*

1. Introduction

One of the most significant, vital, and complex problems of parallel execution is referred as Scheduling a set of either dependent or independent tasks on a set of processors . Parallel programs can be divided into a group of smaller tasks which are usually related to each other. Minimizing of the scheduling length (*make-span*) is known as the only purpose of task scheduling problem in order to allocate tasks to processors such that dependencies between tasks are satisfied.

Task scheduling problem is separated into two groups which are either with or without communication costs, in which each group could be individually proposed in heterogeneous or homogeneous systems.

The algorithms for finding the optimal result for the multiple-processor scheduling problem have been demonstrated to be NP-complete [1, 6].

Many metaheuristic have been proposed based on methods and approaches to the task scheduling [2-5].

Behaviors of Social insects such as ants and bees in the real world have been studied many years to solve many problems. Ant colony algorithm is an example of swarm intelligence algorithms for solving combinatorial optimization problems. Ants can find the shortest path from the food source to their nest by using pheromone [10].

In this paper Task Scheduling Problem has been solved by the bee colony optimization. The bee colony optimization algorithm is inspired by the behavior of a honey bee colony in nectar collection, is another example of swarm intelligence. BCO has been proposed by Lucic and Teodorovic [6-8]. Artificial bees in BCO cooperate to solve combinatorial optimization problem. Every bee during the search process makes some moves and constructs a solution [5]. Furthermore, we add a global memory for bees to compare their result with previous iteration results that will be explained in details later.

2. Definition of Task Scheduling Problem

The problem of task scheduling is indicated by a directed acyclic graph (DAG). This graph is shown by

G (V, E, w, c) which has four characters that are:

V is the set of *v* nodes, and each node $v_i \in V$ represents a task.

W is a *V* computation costs array in which each w_i gives the estimated time of task execution.

E is the set of communication edges. The directed edge e_{ij} joins nodes v_i and v_j , where node v_i is called the parent node and node v_j is called the child node.

C is the set of communication costs, and edge e_{ij} has a communication cost $c_{ij} \in C$.

The relationship of data-dependency from task t_i to t_j could be indicated via directed edge e_{ij} in the set E = $\{e_{ij} \mid i, j \in \{1, 2... |V|\}$. On the other words, task t_i transfers vital relevant information to task t_j after finishing its execution. The amount of data transferred from task t_i to task t_j , is measured by the weight of the edge e_{ij} , which is denoted D(t_i , t_j).

The task t_i is named the *predecessor* for the task t_j , and the task t_j is the *successor* for the task t_i. *Pred*(t_i) denotes a set of its predecessors, and *Succ*(t_i) denotes a set of its successors. In DAG, if a task t_i exists that could satisfy *Pred*(t_i)=φ, it is called the *entry task* and is denoted by t_{entry} . On the other hand, if there is a task t_j that could be able to satisfy the equation of *Succ*(t_j)=φ, this task is called the *exit task* and is denoted by t_{exit} .

Some virtual tasks under the following conditions are added into the DAG, in order to ensure the DAG has only one input and one output tasks. A virtual entry task with zero workload should be joined to the DAG while there are many entry tasks in a DAG. The directed edges from this virtual entry task to each entry task can be established, and the amount of transmission data of these directed edges is zero. On the other words, if there are many exit tasks in a DAG, and then a virtual exit task that has zero workload should be joined to the DAG. The directed edges from each exit task to this virtual exit task are established, and the amount of transmission data of these directed edges is zero, too. Therefore, a DAG that only has one t_{entry} and t_{exit} can be designed [11].

In order to find the finishing time of each node execution, its start time is added with its weight that is [12]:

$$t_f(i) = t_s(i) + w(i) \quad (1)$$

Two nodes could not be executed on just one processor simultaneously. The costs relationships between the nodes that are executed on a same processor are considered to be zero because these are some local relationships.

The time in which a communication arrives at the destination processor is mentioned as the edge finish time. For a graph G (V, E, w, c) with nodes n_i and n_j and the edge e_{ij} the amount of finishing time for that edge is equivalent with the sum of the completion time of node n_i execution and the weight of the edge e_{ij} [12].

$$t_f(e_{ij}, P_{src}, P_{dst}) = t_f(n_i, P_{src}) + \begin{cases} 0 & if P_{src} = P_{dst} \\ c(e_{ij}) & otherwise \end{cases} \quad (2)$$

As it could be seen from the equation above, there are two cases for the weight of edge e_{ij} that indicates the relationship's cost: if node n_j is executed on the same processor in which node n_i were processed, or in the other words nodes n_i and n_j have the same processor, the weight of that edge is considered to be zero. Otherwise, the written number on that edge shows its weight and n_j could not be executed as long as n_i that was executed completely. The node n_j will be started to be executed immediately after the completion of node n_i. This problem is known and defined as the problem of priority constraint (limitation). The nearest time that the execution of node n_i could be started is named Data Ready Time and it is indicated by DRT that could be computed through the equation below:

$$t_{dr}(n_j, P) = \max_{n_i \in Pred\, n_j} \{t_f(e_{ij}, proce(n_i), P)\} \quad (3)$$

If node n_j is an root node $t_{dr}(n_j, P) = 0$

Limitations on the start time of node n could be formulized via DRT:

$$t_s(n, P) \geq t_{dr}(n, P) \quad (4)$$

A scheduler duty is considered to be completed when the last node of our graph was scheduled and there were not any other nodes for scheduling. If we want to obtain the length of a scheduler it would be:

$$sl(S) = \max_{n \in V}\{t_f(n)\} - \min_{n \in V}\{t_s(n)\} \quad (5)$$

A target parallel system **P** consists of a set of identical connected processors which has the following properties:

1. All of the processors could execute only a task during its allocated period of time.
2. The amount of communication costs between tasks which are executed on the same processor should be as negligible as the case that it could be presume to zero.
3. The communication network is fully connected, in which every processor could communicate with other processors, directly.

3. **Bee Colony Optimization**

Each bee hive has a place which is called dance floor. Every Bee starts to dance after when it came back to its hive from a foraging. The main purpose of this kind of dancing is to convince the other bees to be accompanied by them. The procedure of finding a food source in the BCO algorithm is separated into 2 steps.

Forward pass: in this step bees leave their hive for finding a proper food source around their hive. A parameter which is called *NC (number of* solution components) is defined here. This parameter determines the number of tasks that must be visited by each bee in its forward pass. Then a partial solution is generated according to the tasks which are visited by each bee in every forward pass procedure.

The amount of NC is determined practically before the process of searching will be started.
Backward pass: all of bees come back to their home in this step and then they start to calculate and evaluate their answers. Afterwards, these answers are compared with each other in order to find the best answer. In other words, each bee should decide to be loyal to its path or not. Each bee could do one the three jobs below when it came back to the hive [13]:

1- It could advertise its own path in order to absorb the other bees.
2- It could leave its path and join to another bee.
3- It could decide not to advertise its own path; however it keeps on its path.

In these step, those bees which generate more appropriate and better results have more chances of success in order to advertise and absorb the other bees. They communicate their obtained information about the quality of partial solution and their results with other bees via their dances. The duration of every bee's dance is highly likely related to the quality of its obtained result.

These two steps are repeated consecutively in order to generate a complete result which is equivalent to execute all tasks in the task scheduling problem. At last, the most proper and best result is chosen.

Below, a pseudo code for the BCO algorithm is written [14]:

B: the number of bees involved in the search.
NC: the number of forward (backward) passes in a single iteration.
Do
1- Initializing
2- For (i = 0 ; i < NC ; i ++)
//forward pass
a) For (b = 0 ; b < B ; b ++)
(1) Evaluate all possible moves;
(2) Choose one move using the roulette wheel.
//backward pass
b) For (b = 0 ; b < B ; b ++)
Evaluate (partial/complete) solution for bee b;
c) For (b = 0 ; b < B ; b ++)
Loyalty decision using the roulette wheel for bee b;
d) For (b = 0 ; b < B ; b ++)
If (b is follower), choose a recruiter by the roulette wheel.

3. Evaluate all solutions and find the best one
While stopping criteria is not satisfied

In task scheduling problem, two factors should be considered by each bee in every forward pass:

1) Which task should be selected to execute
2) Which CPU should be chosen to execute the task

First of all, each bee calculates the number of executable tasks, which could be executed when all of their dependent precedence tasks were completed. Then, each bee could peek one of these tasks by consideration of some factors such as duration of task execution, the number of other tasks which are related to a significant task and etc. after choosing a desired task, a proper processor should be chosen by that bee. The probability of choosing a proper processor could be calculated via the formula below:

$$p_j = \frac{1 - T_j}{\sum_{i=0}^{n} T_i} \qquad \text{When j} = 1,2, \dots \text{k} \qquad (6)$$

Where, T_j is the quickest duration of task execution on the jth CPU, and also k is the number of CPUs.
In this algorithm a global memory is defined and considered for all of bees. When each stage of forward pass completes, after the determination of all the bees which have the permission of dance, their results are averaged and saved in the memory. The bees use these saved information after the next iteration wants to be started. While a partial solution is generated, if the average of posterior results is not acceptable in comparison with the prior results, the mentioned way will be forgotten and leaved by bees and they will come back to their hive. The speed of execution would be increased clearly via this method.
After the first step of task scheduling is completed and all the bees come back to their hive, they will start to share their information to the other bees. In this stage the amount of each bee's loyalty to its path could be calculated by the formula below [14, 15]:

$$P_b^{u+1} = e^{-(O_{max} - O_b)/u} \qquad ; \text{b} = 1, 2 \dots \text{B} \qquad (7)$$

Where
u - The forward pass counter (taking values 1, 2… NC)

O_b is calculated by

$$O_b = \frac{C_{max} - C_b}{C_{max} - C_{min}} \qquad (8)$$

C_b is the result of partial solution for the bth bee. Partial solution result means the latest time point of finishing the last task at any processors.
C_{max} and C_{min} are respectively the largest and smallest partial solution results producing by all bees.

3-Result

In this paper the random graph generator is used to test the proposed algorithm. Graphs based on parameters that will describe below, have been produced.

1. N: Number of nodes (tasks) in the DAG
2. Width or Fat: This parameter represents the maximum number of tasks that can be executed simultaneously. It means that the higher value of this parameter will result a higher degree of parallelism.

3. Density: this parameter indicates the numbers of edges between tasks of two levels of the DAG.
4. Regularity: It is the uniformity of the number of tasks in each level;
5. Jump: this factor indicates the maximum number of levels that an edge could go. For example, every edge can connect with other nodes in 4 levels below with jump=4in the DAG.
6. CCR: it is the ratio of the communication cost to computation cost.

These parameters can have different values. We generated about 648 different graphs with combination of these different values that indicate in table [1].

Table 1: PARAMETERS AND THEIR VALUES USED FOR GENERATING DAGS

N	10 - 20 - 30 - 40
Jump	1 - 2 - 4
Width	0.1 - 0.2 - 0.8
Density	0.2 - 0.8
Regularity	0.2 - 0.8
CCR	0.1 - 0.5 - 0.8
Number of processor	4 - 8 - 16

We selected 5 random graphs among all the generated graphs that their details presented in the table below:

Table 2: DAG properties

	With	Density	Regularity	Jump	CCR
DAG1	0.1	0.2	0.2	1	0.5
DAG2	0.1	0.2	0.8	4	0.1
DAG3	0.1	0.8	0.2	4	0.1
DAG4	0.8	0.8	0.8	2	0.8
DAG5	0.2	0.2	0.8	4	0.1

We compared make-span of these selected graphs in our propose algorithm with Ant colony system (ACS), Simulate Annealing (SA), and Tabu Search (TS). Each of these DAGs is executed on 4, 8 and 16 processors.

Table 3: DAG1 result

	Number of task = 30				Number of task = 40			
	TS	SA	ACS	BCO	TS	SA	ACS	BCO
Number of processor =4	497	497	497	497	673	666	666	666
Number of processor =8	442	442	442	442	585	579	579	579
Number of processor =16	493	493	493	493	663	663	663	663

Table 4: DAG2 result

	Number of task = 30				Number of task = 40			
	TS	SA	ACS	BCO	TS	SA	ACS	BCO
Number of processor =4	201	200	201	201	290	290	290	290
Number of processor =8	253	253	258	253	317	317	320	317
Number of processor =16	267	267	269	269	335	335	337	337

Table 5: DAG3 result

	Number of task = 30				Number of task = 40			
	TS	SA	ACS	BCO	TS	SA	ACS	BCO
Number of processor =4	241	241	242	241	288	287	297	292
Number of processor =8	267	267	272	267	332	332	339	332
Number of processor =16	248	250	253	253	285	285	291	291

Table 6: DAG4 result

	Number of task = 30				Number of task = 40			
	TS	SA	ACS	BCO	TS	SA	ACS	BCO
Number of processor =4	165	166	164	165	213	211	212	212
Number of processor =8	143	141	144	143	152	150	153	152
Number of processor =16	98	98	98	98	106	106	108	108

Table 7: DAG5 result

	Number of task = 30				Number of task = 40			
	TS	SA	ACS	BCO	TS	SA	ACS	BCO
Number of processor =4	200	200	207	201	225	225	229	225
Number of processor =8	169	169	170	169	263	263	269	263
Number of processor =16	203	203	209	209	200	200	203	203

Table8: PAIR-WISE COMPARISON OF THE SCHEDULING AGORITHM

		BCO	SA	TS	ACS
BCO	better equal worse	*	30% 55% 15%	45% 35% 20%	60% 30% 10%
SA	better equal worse	15% 55% 30%	*	25% 60% 15%	40% 45% 15%
TS	better equal worse	20% 35% 45%	15% 60% 25%	*	20% 65% 15%
ACS	better equal worse	10% 30% 60%	15% 45% 40%	15% 65% 20%	*

4. Conclusions

Since the Swarm intelligence become one of the interesting methods in Parallel Computing field, a modified version of the BCO algorithm (which is one of the most recent nature inspired algorithms) has been applied for solving task scheduling problem in this paper. It simulates the intelligent behavior of bees when they are faced with a source. The Task scheduling problem is a kind of NP hard problems which cannot be solved with linear algorithms. Thus metaheuristic algorithms become so interesting to employ for solving such problems. BCO has been rarely applied in this field and this application is a new area for it.

There are some novelties in the presented algorithm, and the most important innovation is considering a general memory for all bees, to compare their obtained results with the acceptable results which are obtained previously. Like other metaheuristic methods BCO has demonstrated solid solutions on this problem, and the obtained results has been presented and compared with some other powerful and well known metaheuristic algorithms such as ACS, SA and TS. The BCO solutions are considerably close to SA that is the best scheme, however this mentioned algorithm has better results in comparison with the ACS algorithm. The results of these algorithms are shown and compared in table8. Consequently, BCO could be considered as a suitable solving method in order to face NP hard problems.

References

[1] M. R. Garey, and D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness", W. H. Freeman and Company, 1979.

[2] P. Shroff , "Genetic Simulated Annealing for Scheduling Data-dependent tasks in Heterogeneous Environments" Proceedings of Heterogeneous Computing Workshop, Apr 1996, pp.98-117.

[3] F.A. Omara and M.M. Arafa , " Genetic algorithms for task scheduling problem" , Journal of Parallel and Distributed Computing 70 (2010) pp 13_22

[4] N. Nissanke, A. Leulseged and S. Chillara, "Probabilistic performance analysis in multiprocessor scheduling", Journal of Computing and Control Engineering, 2002, Vol. 13, No. 4, pp.171–179.

[5] M . Rapaic, Z. Kanovic and Z. Jelicic, "A theoretical and empirical analysis of convergence related particle swarm optimization ", WSEAS Transactions on Systems and Control, Nov 2009, Vol. 4, Issue 11, pp. 541-550

[6] P. Lucic, D. Teodorovic, "Bee system :modeling combinatorial optimization transportation engineering problems by swarm intelligence", in preprints of the TRISTAN IV triennial symposium on transportation analysis, Sao Miguel, Azores Islands;2001.

[7] P. Lucic, D. Teodorovic, "Transportation modeling: an artificial life approach", in: Proceedings of the 14th IEEE international conference on tools with artificial intelligence, Washington,DC;2002.

[8] P. Lucic, D. Teodorovic, "Computing with bees: attacking complex transportation engineering problems", International Journal on Artificial Intelligence Tools 2003.

[9] E.G. Co_man, Computer and job-shop scheduling theory. In Wiley, 1976.

[10] M. Dorigo, G. Di Caro and L.M. Gambardella, "Ant algorithms for discrete optimization", Artificial Life, 5:137-172, 1999.

[11] C.chaing , Y.Lee , C.Lee and T.chou , "Ant colony optimization for task matching and scheduling" , Computers and Digital Techniques, IEE Proceedings - Nov. 2006 Volume: 153 Issue: 6.

[12] O. Sinnen , Task Scheduling for parallel system , Wiley,2007.

[13] D.Teodorovic, M. Dell'Orco, "Bee colony optimization: a cooperative learning approach to complex transportation problems", in Advanced OR and Al. Methods in Transportation, 2005, pp. 51-60.

[14] T.Davidovic , D.Ramljak, M.Selmic and D.Teodorovic , "Bee colony optimization for the p-center problem" , Computers & Operations Research 38 (2011) 1367–1376.

[15] T.Davidovic, M.Selmic, D.Teodorovic, "Scheduling Independent Tasks: Bee Colony Optimization Approach", 17th Mediterranean Conference on Control & Automation Makedonia Palace, Thessaloniki, Greece June 24 - 26, 2009.

.