Microprocessors and Microsystems 26 (2002) 363–371

MICROPROCESSORS AND MICROSYSTEMS

www.elsevier.com/locate/micpro

Particle swarm optimization for task assignment problem

Ayed Salman*, Imtiaz Ahmad, Sabah Al-Madani

Department of Computer Engineering, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait

Received 10 January 2002

Abstract

Task assignment is one of the core steps to effectively exploit the capabilities of distributed or parallel computing systems. The task assignment problem is an NP-complete problem. In this paper, we present a new task assignment algorithm that is based on the principles of particle swarm optimization (PSO). PSO follows a collaborative population-based search, which models over the social behavior of bird flocking and fish schooling. PSO system combines local search methods (through self experience) with global search methods (through neighboring experience), attempting to balance exploration and exploitation. We discuss the adaptation and implementation of the PSO search strategy to the task assignment problem. The effectiveness of the proposed PSO-based algorithm is demonstrated by comparing it with the genetic algorithm, which is well-known population-based probabilistic heuristic, on randomly generated task interaction graphs. Simulation results indicate that PSO-based algorithm is a viable approach for the task assignment problem. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Particle swarm optimization; Task assignment problem; Task interaction graph

1. Introduction

Distributed computing systems have emerged as a powerful platform for providing higher computational power required by a wide range of applications. The performance of a parallel application on distributed system is highly dependent on both the application characteristics (execution cost and communication cost between tasks etc.), and the platform features (computation capacity of the processors, number of processors, communication bandwidth, memory size etc.). To effectively exploit distributed systems, an important challenge is the mapping of tasks to processors in order to achieve some objective, such as load balancing, minimization of interprocessor communication, or some combination of the two for computationally demanding tasks with diverse computing needs. The widespread use of distributed computers in many computational-intensive applications makes the problem of mapping programs to distributed computers more crucial.

The task assignment (or mapping) problem is that of assigning tasks of a program among different processors of a distributed computer system in order to reduce the program turnaround time and to increase the system throughput [2–4]. This can be done by maximizing and balancing the utilization of resources while minimizing the communication between processors. However, there exist a conflict between the two criteria, as the load balancing calls for distributing the tasks over different processors, while minimizing of interprocessor communication drives the task assignment to assign all the tasks onto a single processor. The task assignment problem is known to be NP-complete [1]. Thus, a good exact algorithm for its optimal solution in polynomial time is unlikely to exist. Therefore, optimal solution strategies must be scarified in favor of fast heuristic techniques.

There are wide varieties of approaches that have been reported for solving the task assignment problem in distributed computing [2–17]. They can roughly be classified into four categories: graph theoretic [2,3,5–7,9, 10], integer linear programming [8], state-space search [11] and probabilistic approaches such as simulated annealing [12–14], meanfield annealing [15] and genetic algorithms [14,16,17]. A good overview of the task assignment problem can be found in Refs. [3,4]. Because of the intractable nature of the task assignment problem and its importance in distributed computing, it is desirable to explore other avenues for developing good heuristic algorithms for the problem. In this paper, we introduce an approach based on particle swarm optimization (PSO) for the static task assignment problem (where allocation does

* Corresponding author.

PII: S0141-9331(02)00053-4

not change during the lifetime of tasks) to effectively exploit the capabilities of distributed or parallel computing systems.

PSO is an algorithm that follows a collaborative population-based search model [18–23]. Each individual of the population, called a 'particle', flies around in a multidimensional search space looking for the optimal solution. Particles, then, may adjust their position according to their own and their neighboring-particles experience, moving toward their best position or their neighbor's best position. In order to achieve this, a particle keeps previously reached 'best' positions in a cognitive memory. PSO performance is measured according to a predefined fitness function (cost function of a problem). Balancing between global and local exploration abilities of the flying particles could be achieved through user-defined parameters. PSO has many advantages over other heuristic techniques such that it can be implemented in a few lines of computer code, it requires only primitive mathematical operators, and it has great capability of escaping local optima. We believe that to the best of our knowledge, ours is the first attempt in proposing a PSO algorithm for the task assignment problem.

The remainder of the paper is organized as follows: Section 2 defines the task assignment problem. Section 3 describes our PSO algorithm for the task assignment problem. Experimental results are reported in Section 4. Finally, Section 5 concludes the paper.

2. Formal problem definition

The task assignment problem is that of assigning tasks of a program among different processors of a distributed computer system in order to reduce the program turnaround time and to increase the system throughput. In our formulation, a distributed program is characterized by task interaction graph (TIG) $G(V, E_T)$, where $V = \{1, 2, \ldots, m\}$ represents the tasks of a program and E_T models the interaction between these tasks. The edge weight e_{ij} between node i and j denotes the information exchange between these pair of tasks. The node weight w_{ij} corresponds to the work to be performed by task i on processor j. In homogeneous system, processors have same computing power and each task takes same amount of time to execute on each processor. Many real-world problems can be modeled as TIG such as iterative solution of systems of equations, power system simulations, and VLSI simulation programs [2–4]. Fig. 1 shows an instance of the problem on a homogenous system.

The underling computing system is represented by a processors interaction graph (PIG) $G(P, E_p)$, where $P = \{1, 2, \ldots, n\}$ represents the set of processors in the system and E_p represents the set of communication links between these processors. An edge weight d_{kj} between any two nodes k and j represents the length of the shortest path between the corresponding processors. The assignment problem can be formally represented as follows:

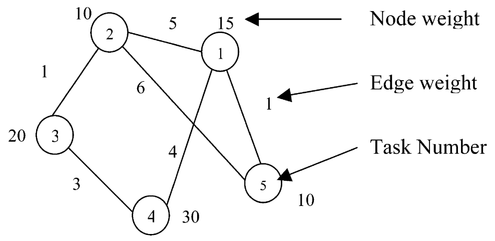

Fig. 1. A TIG instance on a homogeneous system.

$$A : V \rightarrow P \tag{1}$$

where A is a function that maps the set of tasks V to the set of processors P. $A(i) = j$ if task i is assigned to processor j. Let m be the total number of tasks and n be the total number of processors, where $1 \le i \le m$, $1 \le j \le n$, then the objective function (a minimax cost model) is as follows:

'Find a mapping instance A, such that when estimating the total time required by each processor, the largest time among all processors is minimized'.

Let $C_{\text{exe}}(A)_k$ be the total execution time of all tasks assigned to processor k, such that

$$C_{\text{exe}}(A)_k = \sum_i w_{ik} \qquad \forall A(i) = k \tag{2}$$

and let $C_{\text{com}}(A)_k$ be the total interaction time between tasks assigned to processor k and those that are not assigned to that processor in an assignment A, i.e.

$$C_{\text{com}}(A)_k = \sum_i \sum_j d_{A(i),A(j)} e_{ij} \qquad \forall A(i) = k \text{ and } A(j) \ne k \tag{3}$$

Then, for a given assignment A, $C_{\text{total}}(A)_k$, the total time of a processor k, will equal to the sum of the execution time and the interaction time for processor k, i.e.

$$C_{\text{total}}(A)_k = C_{\text{exe}}(A)_k + C_{\text{com}}(A)_k \tag{4}$$

Then, the final cost of an assignment will be dominated by the processor with the highest $C_{\text{total}}(A)$, i.e.

$$\text{Cost}(A) = \max(C_{\text{total}}(A)_k \qquad \text{for } 1 \le k \le n) \tag{5}$$

The goal of the task assignment problem is to find an assignment A which has the minimum cost for a given TIG on a given PIG, i.e.

$$\text{Minimize}(\text{Cost}(A) \qquad \forall A) \tag{6}$$

3. Particle swarm optimization and task assignment problem

In this paper, we propose a new population-based heuristic method called PSO to solve the task assignment problem. Kennedy and Eberhart initially pointed out PSO algorithm through scientific simulation of simplified social life models [18]. It was applied successfully since then to a

```
Let P be the size of the PSO population.
Let $PSO[i]$ be the position of the $i^{th}$ particle of the PSO population; this represents a candidate solution
for the problem.
Let $fitness[i]$ be the cost function of the $i^{th}$ particle.
Let $V[i]$ be the traveled distance (or velocity) of the $i^{th}$ particle.
Let $G_{best}$ be an index to global-best position.
Let $P_{best}[i]$ be the position of the local-best position of the $i^{th}$ particle.
Let $P_{best\_}fitness[i]$ be the local-best fitness for the best position visited by the $i^{th}$ particle.
Step1 (Initialization): For each particle $i$ in the population:
    Step1.1: Initialize $PSO[i]$ randomly.
    Step1.2: Initialize $V[i]$ randomly.
    Step1.3: Evaluate $fitness[i]$.
    Step1.4: Initialize $G_{best}$ with the index of the particle with the best fitness among the population.
    Step1.5: Initialize $P_{best}[i]$ with a copy of $PSO[i]$ $\forall i \le P$.
Step2: Repeat until a stopping criterion is statisfied:
    Step2.1: Find $G_{best}$ such that $fitness[G_{best}] \ge fitness[i]$ $\forall i \le P$
    Step2.2: For each particle i: $P_{best}[i] = PSO[i]$ iff $fitness[i] > P_{best\_}fitness[i]$ $\forall i \le P$.
    Step2.3: For each particle i: update $V[i]$ and $PSO[i]$ according to equations 7 and 8.
    Step2.4: Evaluate $fitness[i]$ $\forall i \le P$.
```

Fig. 2. Simple PSO algorithm.

handful of computer science and engineering problems [18–23]. It has the advantage of having a cognitive memory, maintaining a population of solution, and having an elegant productive cooperation between its populations. PSO, as a general optimization tool, is described next, and then a PSO algorithm to solve the task assignment problem will be described in Section 3.1.

3.1. Particle swarm optimization

PSO is an algorithm proposed by Kennedy and Eberhart in 1995 [18]. Social behavior of organisms such as bird flocking and fish schooling motivated them to look into the effect of collaboration of species onto achieving their goals as a group. Years of study for the dynamics of bird flocking resulted in the possibilities of utilizing this behavior as an optimization tool. They named it PSO. In a PSO system, multiple candidate solutions coexist and collaborate simultaneously. Each solution candidate, called a 'particle', flies in the problem search space (similar to the search process for food of a bird swarm) looking for the optimal position to land. A particle, as time passes through his quest, adjusts its position according to its own 'experience', as well as according to the experience of neighboring particles. Tracking and memorizing the best position encountered build particle's *experience*. For that reason, the PSO algorithm possesses a memory (i.e. every particle remembers the best position it reached during the past). PSO system combines local search methods (through self experience) with global search methods (through neighboring experience), attempting to balance exploration and exploitation. Two factors characterize a particle status on the search space: its position and velocity. Kennedy and Eberhart explore several models to manipulates these factors to accurately resembles the dynamic of the social behavior of birds, before reaching to the following equations which amazingly achieve good performance on optimization problems [18]:

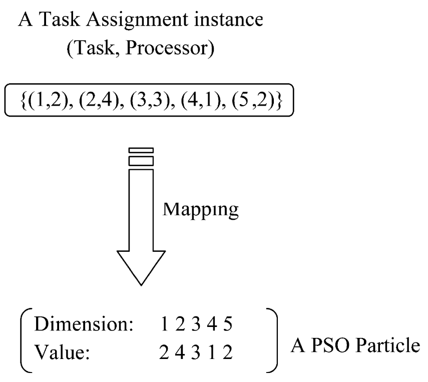

Fig. 3. Task assignment to PSO particle mapping.

$$V_{id} = WV_{id} + c1\text{rand}(\,)(P_{id} - X_{id}) + c2\text{Rand}(\,)(P_{gd} - X_{id}) \tag{7}$$

$$X_{id} = X_{id} + V_{id} \tag{8}$$

where V_{id}, called the velocity for particle i, represents the distance to be traveled by this particle from its current position, X_{id} represents the particle position, P_{id} represents its best previous position (i.e. its experience), and P_{gd} represents the best position among all particles in the population. Rand and rand are two random functions with a range [0,1]. $c1$ and $c2$ are positive constant parameters

```
Let $N$ be the number of processors given by the problem.
Let $M$ be the number of tasks in a TIG instance.
Let $P$ be the size of the PSO population.
Let $PSO[i]$ be the position of the $i^{th}$ particle of the PSO population represented as an $M$-
dimensional vector, whose entries' values belong to the set $\{1, \dots, N\}$;
Then $PSO[i][j]$ be the processor number to which the $j^{th}$ task in the $i^{th}$ particle is assigned.
Let $fitness[i]$ be the cost function of the $i^{th}$ particle according to Equation 5.
Let $V[i]$ be the traveled distance (or velocity) of a $i^{th}$ particle represented as an $M$-dimension real-
coded vector.
Let $G_{best}$ be an index to global-best position.
Let $P_{best}[i]$ be the position of the local-best position of the $i^{th}$ particle.
Let $P_{best\_}fitness[i]$ be the local-best fitness for the best position visited by the $i^{th}$ particle.
Step1 (Initialization): For each particle $i$ in the population:
    o For each task $j$:
        • Initialize $PSO[i][j]$ randomly from the set $\{1, \dots, N\}$.
    o Initialize $V[i]$ randomly.
    o Evaluate $fitness[i]$.
    o Initialize $G_{best}$ with the index of the particle with the best fitness (lowest cost) among the
      population.
    o Initialize $P_{best}[i]$ with a copy of $PSO[i]$ $\forall i \le P$.

    Step2: Repeat until a number of generation, equal to twice the total number of task, is
passed:
• Step2.1: Find $G_{best}$ such that $fitness[G_{best}] \ge fitness[i]$ $\forall i \le P$.
• Step2.2: For each particle i: $P_{best}[i] = PSO[i]$ iff $fitness[i] > P_{best\_}fitness[i]$ $\forall i \le P$.
• Step2.3: For each particle i: update $V[i]$ and $PSO[i]$ according to equations 7 and 8.
• Step2.4: Evaluate $fitness[i]$ $\forall i \le P$.
```

Fig. 4. PSO algorithm for task assignment problem.

called *acceleration coefficients* (which control the maximum step size the particle can do). The inertia weight, W is a user-specified parameter that controls, with $c1$ and $c2$, the impact of previous historical values of particle velocities on its current one. A larger inertia weight pressures towards global exploration (searching new area) while a smaller inertia weight pressures toward fine-tuning the current search area. Suitable selection of the inertia weight and acceleration coefficients can provide a balance between the global and the local search. Eq. (7) is used to calculate the particle's new velocity according to its previous velocity and to the distances of its current position from both its own best historical position and its neighbors' best position. Then the particle flies toward a new position according to Eq. (8). The performance of each particle is measured according to a pre-defined fitness function, which is usually proportional to the cost function associated with the problem [20]. This process is repeated until user-defined stopping criteria are satisfied. Two versions of PSO exists, *gbest* and *lbest*. The difference is in the neighborhood topology used to exchange experience among particles. In the *gbest* model, the neighborhood of the particle is the whole swarm (i.e. the best particle is determined from the entire swarm). In the *lbest* model, a swarm is divided into overlapping neighborhoods of particles. This best particle is called the *neighborhood best* particle. Fig. 2 shows a simple PSO algorithm. Interested readers could refer to Kennedy and Eberhart [18,19] for more details. Next, we present a *gbest*-model PSO algorithm for the problem at-hand: task assignment problem.

3.2. Particle swarm optimization for task assignment problem

In this section, we describe the formulations of a PSO algorithm for the task assignment problem. In this paper, we considered homogeneous systems where processor computing powers are the same. In addition, we assume that processors directly connected with each other, hence distance will not be a factor in Eq. (3).

One of the key issues in designing a successful PSO algorithm is the representation step, i.e. finding a suitable mapping between problem solution and PSO particle. In this paper, we setup a search space of M dimension for an M task

Table 1
An execution trace of PSO algorithm

Particle velocity					Particle position					Fitness
For generation [1]:										
3.000	3.000	0.000	3.000	0.000	1	0	0	2	0	43.0
−3.000	−1.000	3.000	−3.000	−1.000	0	3	0	3	3	51.0
−1.000	0.000	1.000	−2.000	−3.000	1	2	1	0	0	48.0
−2.000	2.000	0.000	1.000	0.000	3	2	2	2	0	66.0
3.000	−2.000	−3.000	2.000	3.000	3	1	1	2	2	40.0
−3.000	−2.000	1.000	−1.000	2.000	0	0	2	1	1	40.0
0.000	1.000	0.000	2.000	−2.000	3	0	3	1	0	48.0
−2.000	0.000	−1.000	1.000	3.000	2	3	3	1	0	39.0
−2.000	3.000	−3.000	−3.000	0.000	0	0	2	0	3	63.0
0.000	0.000	0.000	−3.000	−3.000	0	2	1	2	1	47.0
For generation [2]:										
−0.927	2.943	0.732	−1.226	0.000	0	2	0	0	0	80.0
−1.132	−2.250	3.378	−0.472	−2.316	2	0	3	2	0	51.0
1.042	−1.783	−0.322	−0.058	−0.750	0	0	0	0	0	85.0
−2.755	−1.885	−1.950	2.221	0.000	0	0	0	0	0	85.0
−1.146	−0.482	0.098	−1.055	−1.980	1	0	1	0	0	51.0
−0.610	2.467	−1.696	−0.250	−0.337	0	2	0	0	0	80.0
1.467	2.470	1.000	−0.500	0.520	0	2	0	0	0	80.0
−1.500	1.000	1.250	−0.250	0.750	0	0	0	0	0	85.0
0.116	1.872	−1.359	−0.179	−1.281	0	1	0	0	1	71.0
2.870	−1.972	−0.866	−0.322	0.340	2	0	0	1	1	40.0
For generation [3]:										
1.756	1.412	2.295	1.808	0.000	1	3	2	1	0	51.0
−0.657	3.549	−1.082	−1.300	2.873	1	3	1	0	2	48.0
2.029	2.353	1.171	0.414	−0.188	2	2	1	0	0	40.0
3.277	1.156	1.016	0.388	0.000	3	1	1	0	0	40.0
1.212	2.547	0.519	2.812	0.409	2	2	1	2	0	63.0
1.727	0.428	2.597	1.340	0.910	1	2	2	1	0	51.0
2.938	−0.228	3.693	0.818	−0.125	2	1	3	0	0	40.0
2.009	3.630	2.118	1.638	0.188	2	3	2	1	0	48.0
1.387	1.017	1.413	−0.011	0.697	1	2	1	0	1	57.0
0.468	2.949	1.189	−0.331	−0.972	2	2	1	0	0	40.0
For generation [24]:										
0.000	−0.440	−0.094	0.071	0.196	1	1	0	1	0	63.0
0.000	0.000	0.000	0.000	0.000	1	3	2	0	1	34.0
0.921	0.759	0.341	0.000	0.278	1	2	1	0	0	48.0
1.108	−0.049	0.927	0.000	0.750	2	1	1	0	0	40.0
−0.502	0.940	0.277	0.000	0.000	1	2	1	0	1	57.0
1.138	0.600	0.000	0.706	0.000	1	1	2	0	1	40.0
0.759	0.487	0.920	0.596	0.990	1	2	2	0	0	40.0
0.000	0.000	0.000	0.000	0.000	1	3	2	0	1	34.0
0.000	0.000	0.000	0.000	0.183	1	3	2	0	0	40.0
0.298	2.190	0.364	0.236	0.000	1	3	1	0	1	57.0

assignment problem. Each dimension has discrete set of possible values limited to $s = \{P_i : 1 \le i \le N\}$; such that N is the number of processors in the distributed system under consideration. For example, consider the problem to map the tasks in Fig. 1 to a 4-processor distributed computing system. Fig. 3 shows a mapping between a one possible assignment instance to a particle position coordinates in the PSO domain. We proceed by using the terms ‘particle’, ‘assignment’ and ‘solution’ interchangeably throughout the course of the paper. Using such particle representation, the PSO population is represented as a *PxM* two-dimensional array consisting of N particles, each represented as a vector of M tasks. Thus, a particle flies in an M-dimensional search space. A task is internally represented as an integer value indicating the processor number to which this task is assigned to during the course of PSO.

In our PSO algorithm, we map an M-task assignment instance into corresponding M-coordinate particle position. The algorithm starts by generating randomly as many potential assignments for the problem as the size of the initial population of the PSO. It then measures particles’ fitness. We used Eq. (5) as our fitness function.

The algorithm keeps an updated version of two special variables through out the course of its execution:

'global-best' position and 'local-best' position. It does that by conducting two ongoing comparisons: First, it compares the fitness of each particle being in its 'current' position with fitness of other particles in the population in order to determine the global-best position each generation. Then, it compares different visited positions of a particle with its current position, in order to determine a local-best position for every particle. These two positions affect the new velocity of every particle in the population according to Eq. (7). As shown in this equation, two random parameters control amount of effect the two positions (i.e. global and local best positions) impose over the new particle velocity. The purpose of these parameters is to introduce a randomize and unbiased affect from either positions; hence, introduces a bit of exploration at some times and a bit of exploitation at another time randomly. The algorithm uses the new velocity to update the particle current position to a new position according to Eq. (8). Once all particles adjust their positions, they will constitute the new status of the PSO population. Then, the algorithm evaluates the fitness of these particles according to their new positions. Finally, the algorithm repeats this whole process of determining the global and the local best positions, updating particle position and evaluating new particle position until a user-determined criterion is satisfied. In our case, we mapped this criterion to a maximum number of generations. The proposed PSO algorithm for task assignment problem is summarized in Fig. 4.

3.3. Trace of PSO algorithm

In this section, we will show an execution trace of PSO algorithm with the help of an example. In this example, we consider the problem TIG shown in Fig. 1 and a homogenous distributed system of four processors. As shown in Fig. 4, the PSO algorithm begins by initializing randomly a population of particles based on the given TIG. As discussed earlier, each entry in a particle array represents a task in the TIG, and each value for these entries is a processor number to which the corresponding task is assigned. Initially, particle velocities are initialized randomly. For the sake of simplicity, assume a population of 10 particles initialized as shown in Table 1. For example, in the first generation particle 1's position $\{1, 0, 0, 2, 0\}$ (for the sake of programming simplicity, processor numbering in the code starts from 0 and ends at $N-1$) suggests a solution to the problem assigning task 1 to processor 1, tasks 2, 3 and 5 to processor 0, and task 4 to processor 2. Particle fitness, which is the cost of assignment (calculated according to Eq. (5), is shown adjacent to each particle. In this population, the best particle (*Gbest*) has a fitness value of 39. A copy of *Gbest* position is updated throughout generations.

Since it is the initial population, historical local best position for a PSO particle is its initial position. PSO then updates each particle velocity using Eq. (7). Then the updated velocities are used in Eq. (8) to get the new position for each particle. For example, as shown in Table 1, velocities vector $\{1.756, 1.412, 2.295, 1.808, 0.00\}$ in the third generation is added to position vector $\{0, 2, 0, 0, 0\}$ in second generation to give a new vector equals to $\{1.756, 3.412, 2.295, 1.808, 0.00\}$. Since, values in a particle are processor numbers, a real value such as 1.756 is meaningless. Therefore, in the algorithm we usually round these numbers to the closest processor number by dropping the sign and the fractional part; thus, we yield the particle position as $\{1, 3, 2, 1, 0\}$ in the third generation. The effects of converting from continuous domain to discrete domain need to be determined further, which mapping will yield the better results. The new particle position is a new solution to the mapping problem. This adjustment is done for all particles and the cycle of finding the *Gbest* and local bests and updating the particle positions continues until satisfactory results or termination criteria are met such as population converge onto one solution. The trace of few generations of PSO algorithm is shown in Table 1.

4. Experimental results

The proposed PSO algorithm for the task assignment problem was implemented in C programming language on an AMD 450MHZ, 128MB machine running under windows environment. To measure the effectiveness and viability of PSO algorithm, results are compared with other probabilistic GA based technique. We implemented a generational GA model with roulette wheel selection, 2-point crossover operator, a mutation operator and an elitism mechanism for the comparison purpose. We used suites of randomly generated TIGs to measure the performance of both the algorithms. The criteria of performances considered were the quality of the solutions (cost of assignment) and the amount of CPU time used for the benchmarks. The percentage improvement in cost is computed as:

$$\left(1-\frac{\mathrm{Cost}_{\mathrm{PSO}}}{\mathrm{Cost}_{\mathrm{GA}}}\right)\times 100$$

and the normalized running time is computed by dividing the total GA algorithm time with PSO algorithm running time for the benchmark under consideration. For each algorithm, the results were averaged over 20 trials. In addition, we used statistical significance measure (T-test) to insure the reliability of our results compared to those obtained from GA.

We used in experiments the following values for parameters that control each algorithm:

(1) PSO:
   (a) The size of the population equals to twice the number of tasks.
   (b) The inertia weight W is set to 0.9 in Eq. (7).
   (c) $C1 = C2 = 1.0$.
(2) GA (parameters values where the normal ones usually selected by GA practitioners):

Table 2
Comparison of results for tree type graph structure

Tasks	PSO		GA	
	Time (s)	Cost	Time (s)	Cost
10	0.006868	272.6592	0.0087	255.0182
20	0.070055	518.9565	0.080128	551.5127
30	0.24359	643.2102	0.27793	642.7952
40	0.632326	809.5605	0.686814	852.3117
50	1.164377	1029.841	1.330586	1073.758
60	1.942308	995.8665	2.354395	1046.949
70	3.061356	1372.893	3.735806	1543.903
80	4.71566	1267.116	5.864469	1361.481
100	9.722069	1560.504	12.34386	1564.47
Total	28.429488	9941.9239	35.299904	10 366.7308
Improvement in cost (%)			4.099	
Normalized running time			1.241	

(a) Crossover probability = 0.9 (on average, 10% of the parent population will survive the crossover operation).
(b) Mutation probability = 1/(total number of tasks).
(c) Population size = twice the number of tasks.
(d) Elitism (preserving the best solution through out different generations) is used.
(e) Roulette wheel selection.

4.1. Test suite 1

In this test suite, we generated TIG representing real-life problems such as Tree, Fork-Join, Laplace Equation Solver, Gaussian Elimination and LU decomposition type of problems. The structure of these TIGs is fixed and each of them required different number of vertices. We varied the size of TIG. The cost of each node was randomly selected from a normal distribution with mean equal to the specified average computation cost. The cost of each edge was also randomly selected from a normal distribution with mean equal to the product of the average computation cost and the communication-to-computation ratio (CCR). Five different values of CCR were selected: 0.1, 0.5, 1.0, 2.0, and 10.0. Thus, the suite consists of five graphs for each size.

Tables 2–6 show the results for different graph structures on a 10-processor homogeneous system. Each result is an average of five test cases. Summary of the results is reported in Table 7. Algorithm performances vary depending upon the structure of TIGs. For tree, Gaussian and Fork–Join type graph structures, the PSO technique solution quality is better as compared with GA. For example, for Fork–Join type graph structure, PSO solution quality is 22.49% better than that of GA. While for Laplace and LU decomposition type of graph structures, GA solution quality is marginally better as compared with PSO solution quality. However, GA on the average runs 1.3 times slower than PSO technique. Thus, the PSO technique possesses the strength to explore good solutions for different types of graph structures with different CCR.

Table 3
Comparison of results for Laplace type graph structure

Tasks	PSO		GA	
	Time (s)	Cost	Time (s)	Cost
16	0.030678	742.035	0.048535	737.6933
25	0.145147	1428.599	0.170788	1376.966
36	0.444139	2415.008	0.54533	2252.045
49	1.107601	3491.969	1.395605	3463.448
64	2.487637	5219.71	3.184524	5056.859
81	5.103938	7153.59	6.713829	6961.95
100	9.596156	9991.452	12.86447	9686.616
121	17.23077	13 544.1	23.73169	12 538.71
144	29.20787	17 739.47	41.0847	16 456.78
169	47.45422	22 469.5	67.9913	20 642.87
Total	112.808156	84 195.433	157.730771	79 173.9409
Improvement in cost (%)			−6.34	
Normalized running time			1.398	

Table 4
Comparison of results for Gaussian elimination type graph structure

Tasks	PSO		GA	
	Time (s)	Cost	Time (s)	Cost
18	0.057234	902.0797	0.06685	903.2298
25	0.151099	1424.92	0.181777	1384.546
33	0.351191	2089.284	0.423993	1941.878
42	0.723443	2852.528	0.903388	2736.764
52	1.372252	3710.799	1.726648	3924.065
63	2.403388	4997.596	3.055861	5389.485
75	4.018315	7018.544	5.197344	6933.335
88	6.551282	8439.837	8.667582	8767.15
102	10.39652	11 051.69	14.13233	11 124.64
117	15.54212	13 188.43	21.65018	13 289.01
Total	41.566844	55 675.7077	56.0059	56 394.1028
Improvement in cost (%)			1.27	
Normalized running time			1.347	

Table 5
Comparison of results for Fork–Join type graph structure

Tasks	PSO		GA	
	Time (s)	Cost	Time (s)	Cost
10	0.009158	310.571	0.009616	298.7352
20	0.076923	595.1565	0.08837	557.5495
30	0.255037	843.3817	0.303572	819.1293
40	0.616758	1168.828	0.7587	1303.026
50	16.09478	550.8643	16.26603	573.2727
60	2.089744	1438.425	2.679029	1692.423
70	3.298078	2012.21	4.243131	2099.977
80	5.017399	2393.497	6.624999	2345.468
90	7.206502	2263.488	9.568681	2520.966
100	9.820055	2437.536	13.2184	2950.474
Total	29.999132	11 750.4605	39.121101	15 161.0207
Improvement in cost (%)			22.49	
Normalized running time			1.3	

Table 6
Comparison of results for LU–decomposition type graph structure

Tasks	PSO		GA	
	Time (s)	Cost	Time (s)	Cost
9	0.006182	450.1878	0.007555	410.0268
14	0.025412	860.0773	0.033654	809.7198
20	0.071429	1429.921	0.085165	1372.417
27	0.184066	2070.631	0.222528	2099.44
35	0.398352	2926.093	0.478709	2840.742
44	0.801511	3860.564	0.991072	3852.341
54	1.489011	5064.765	1.872253	4995.823
465	2.59272	6420.099	3.300137	6484.433
77	4.362638	8362.886	5.71978	8400.818
90	7.016483	10 259.6	9.258242	10 272.25
Total	16.947804	41 704.8241	21.96995	41 538.0106
Improvement in cost (%)			−0.401	
Normalized running time			1.296	

Table 7
Summary of results for test suite 1

Graph type	Normalized running time	Improvement in cost (%)
Tree	1.241	4.099
Laplace	1.398	−6.34
Gaussian elimination	1.347	1.27
Fork–Join	1.3	22.49
LU-decomposition	1.296	−0.401
Average	1.316	4.223

4.2. Test suite 2

To demonstrate the effectiveness of our PSO technique, in this test suite we consider a large suite of 150 randomly generated TIG. The size of TIG was varied from 50 to 300 nodes with increments of 50. The cost of each node was randomly selected from a normal distribution with mean equal to the specified average computation cost. The cost of each edge was also randomly selected from a normal distribution with mean equal to the product of the average computation cost and the CCR. Five different values of CCR were selected: 0.1, 0.5, 1.0, 2.0, and 10.0. For generating the random task graphs, we used another parameter called parallelism (P). This parameter determined the average number of children nodes for each node. Five different values of parallelism were chosen: 1, 2, 3, 4, and 5. Thus, the suite consists of 25 graphs for each size. We compare our results with the GA for 10 fully connected homogeneous processors. Two types of comparisons are carried out based on the results obtained by running each algorithm on this suite of 150 graphs. First, we compare the assignment cost for all the random graphs generated by the technique and the average running times of these algorithms. The assignment cost generated by GA technique and PSO technique for randomly generated task graphs is shown in Fig. 5. Each point in the figure is the average of 25 test cases. PSO solution quality is on the average 18.10% better than GA. On the average, the GA algorithm is 1.62 times slower than PSO technique. These results indicate that the proposed PSO algorithm is a viable alternative for solving the task assignment problem.

5. Conclusions

In distributed computing systems, qualified assignment of tasks among processors is an important step for efficient utilization of resources. In this paper, a new heuristic algorithm called PSO algorithm is proposed for the task assignment problem for homogeneous distributed computing systems. The performance of PSO algorithm is evaluated in comparison with well-known GA algorithm for a number of randomly generated mapping problem instances. The results showed that the PSO algorithm solution quality is better than that of GA in most of the test cases. Moreover, the PSO algorithm runs faster as compared with GA. These results indicate that the proposed PSO algorithm is an attractive alternative for solving the task assignment problem. A natural extension to this work would be applying PSO to the heterogeneous version of this problem. We are currently studying the discrete representation for particles PSO algorithm for discrete optimization problems. Moreover, PSO application to other combinatorial optimization problems needs further investigation.

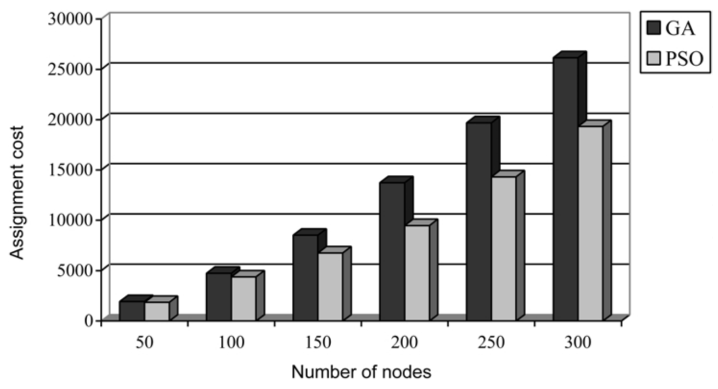

Fig. 5. Comparison of solution quality of PSO against GA for randomly generated TIGs.

Acknowledgments

The author acknowledges the financial support of Kuwait University for funding this project through research grant EO 01/00.

References

[1] M.R. Garey, D.S. Johnson, Computers and Intractability: a Guide to the Theory of NP-Completeness, Freeman, New York, 1979.

[2] S.H. Bokhari, Assignment Problems in Parallel and Distributed Computing, Kluwer Academic Publishers, Boston, MA, 1987.

[3] V. Chaudhary, J.K. Aggarwal, A generalized scheme for mapping parallel algorithms, IEEE Transactions on Parallel and Distributed Systems 4 (March) (1993) 328–346.

[4] M.G. Norman, P. Thanisch, Models of machines and computation for mapping in multicomputers, ACM Computing Surveys 25 (September) (1993) 263–302.

[5] P.R. Ma, E.Y. Lee, M. Tsuchiya, A task allocation model for distributed computing systems, IEEE Transactions on Computers 31 (January) (1982) 41–47.

[6] C.C. Hui, S.T. Chanson, Allocating task interaction graphs to processors in heterogeneous networks, IEEE Transactions on Parallel and Distributed Systems 8 (September) (1997) 908–925.

[7] H.S. Stone, Multiprocessor scheduling with the aid of network flow algorithms, IEEE Transactions on Software Engineering SE-3 (January) (1977) 85–93.

[8] W.W. Chu, L.Y. Holloway, M.T. Lan, K. Efe, Task allocation in distributed data processing, IEEE Computer 13 (November) (1980) 57–69.

[9] C. Woodside, G. Monforton, Fast allocation of processes in distributed and parallel systems, IEEE Transactions on Parallel and Distributed Systems 4 (February) (1993) 164–173.

[10] V. Lo, Heuristic algorithms for task assignment in distributed systems, IEEE Transactions on Computers 37 (November) (1988) 1384–1397.

[11] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurrency 6 (July–September) (1998) 42–51.

[12] S. Kirkpatrik, C.D. Gelatt Jr., M.O. Vecchi, Optimization by simulated annealing, Science 220 (May) (1983) 671–680.

[13] F.T. Lin, C.C. Hsu, Task assignment scheduling by simulated annealing, IEEE Region 10 Conference on Computer and Communication Systems (1990) 279–283.

[14] E.G. Tabi, T. Muntean, Hill-climbing, Simulated Annealing and Genetic Algorithms: a Comparative Study and Application to the Mapping Problem, IEEE 26th Hawaii International Conference System Sciences, 1993, pp. 565–573.

[15] T. Bultan, C. Aykanat, A new mapping heuristic based on mean field annealing, Journal of Parallel and Distributed Computing 16 (December) (1992) 292–305.

[16] I. Ahmad, M.K. Dhodhi, Task assignment using problem-space genetic algorithm, Concurrency: Practice and Experience 7 (August) (1995) 411–428.

[17] T. Chockalingam, S. Arunkumar, A randomized heuristics for the mapping problem: the genetic approach, Parallel Computing 18 (1992) 1157–1165.

[18] J. Kennedy, R.C. Eberhart, Particle swarm optimization, Proceedings of the IEEE International Conference on Neural Networks (December) (1995) 1942–1948.

[19] J. Kennedy, R.C. Eberhart, Y. Shi, Swarm Intelligence, Morgan Kaufmann, San Mateo, CA, 2001.

[20] Y. Shi, R. Eberhart, Empirical Study of Particle Swarm Optimization, Proceedings of the Congress on Evolutionary Computation (CEC99) (July), 1999, pp. 1945–1950.

[21] P.N. Sugantha, Particle Swarm Optimization with Neighborhood Operator, Proceedings of the Congress on Evolutionary Computation (CEC99) (July), 1999, pp. 1958–1962.

[22] E. Ozcan, C.K. Mohan, Particle Swarm Optimization: Surfing the Waves, Proceedings of the Congress on Evolutionary Computation (CEC99) (July), 1999, pp. 1939–1944.

[23] M. Clerc, The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization, Proceedings of the Congress on Evolutionary Computation (CEC99) (July), 1999, pp. 1951–1957.

Ayed A. Salman received his BSc in Computer Engineering from Kuwait University Kuwait, and both MSc and PhD in Computer from Syracuse University, Syracuse, New York, in 1993, 1997, and 1999, respectively. Since September 1999, he had been with the Department of Computer Engineering at Kuwait University, Kuwait, where he is currently an assistant professor. His research interests include application of evolutionary computations (e.g. Genetic Algorithms, Genetic Programming, Particle Swarm Optimization) in data mining, multiobjective optimization, and Machine learning.

Imtiaz Ahmad received his BSc in Electrical Engineering from University of Engineering and Technology, Lahore, Pakistan, MSc in Electrical Engineering from King Fahd University of Petroleum and Minerals, Dahran, Saudia Arabia, and PhD in Computer Engineering from Syracuse University, Syracuse, New York, in 1984, 1988, and 1992, respectively. Since September 1992, he had been with the Department of Computer Engineering at Kuwait University, Kuwait, where he is currently a professor. His research interests include design automation of digital systems, high-level synthesis, and parallel and distributed computing.

Sabah Al-Madani received both her BSc and MSc in Computer Engineering from Kuwait University Kuwait, in 1996 and 2001, respectively. She is currently a computer lecturer in the Ministry of Education, Kuwait.