Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computer Standards & Interfaces 28 (2006) 441–450

COMPUTER STANDARDS & INTERFACES

www.elsevier.com/locate/csi

A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems

Peng-Yeng Yin*, Shiuh-Sheng Yu, Pei-Pei Wang, Yi-Te Wang

Department of Information Management, National Chi Nan University, Nantou, Taiwan

Received 30 December 2004; received in revised form 17 March 2005; accepted 20 March 2005
Available online 10 May 2005

Abstract

In a distributed system, a number of application tasks may need to be assigned to different processors such that the system cost is minimized and the constraints with limited resource are satisfied. Most of the existing formulations for this problem have been found to be NP-complete, and thus finding the exact solutions is computationally intractable for large-scaled problems. This paper presents a hybrid particle swarm optimization algorithm for finding the near optimal task assignment with reasonable time. The experimental results manifest that the proposed method is more effective and efficient than a genetic algorithm. Also, our method converges at a fast rate and is suited to large-scaled task assignment problems.

Keywords: Task assignment problem; Distributed systems; Hybrid strategy; Particle swarm optimization; Genetic algorithm

1. Introduction

In a computation system with a number of distributed processors, it is desired to assign application tasks to these processors such that the resource demand of each task is satisfied and the system throughput is increased. However, the assignment of tasks will also incur some costs such as the execution cost and the communication cost. The task assignment problem (TAP) is to find an assignment of tasks which minimizes the incurred costs subject to the resource constraint. There exist polynomial time exact algorithms for solving the TAP in two-processor distributed systems [1]. However, the general n-processor TAP has been found to be NP-complete [2]. Therefore, finding exact solutions to large-scaled TAP is computationally prohibitive.

The existing approaches for tackling the TAP can be divided into three categories. (1) *Mathematical programming approaches* [3–5] using column generation or branch-and-bound techniques can solve the problem more efficiently. (2) *Customized algorithms* [6–10] have been developed for providing exact solutions in specific circumstances, such as the distributed systems with linear processor array, processor mesh, and partial k-tree communication

* Corresponding author. 303 University Rd., Puli, Nantou 545, Taiwan. Tel.: +886 49 2910960; fax: +886 49 2915205.
E-mail address: pyyin@ncnu.edu.tw (P.-Y. Yin).

0920-5489/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.csi.2005.03.005

graph. (3) *Meta-heuristic algorithms* involving genetic algorithms (GA) [11–13] and simulated annealing (SA) [14,15] have been used to derive approximate solutions with reasonable time.

The development of meta-heuristic optimization theory has been flourishing during the last decade. Many new search paradigms such as tabu search [16] and ant colony optimization [17] have shown their efficacy in solving computationally intensive problems. This suggests the exploration of the potentials for solving the TAP using new meta-heuristics. Recently, Kennedy and Eberhart [18] developed a new evolutionary algorithm called particle swarm optimization (PSO). The PSO optimizes an objective function by iteratively improving a swarm of solution vectors, called particles, based on special management of memory. Each particle is modified by referring to the memory of individual and swarm's best information. Due to the collective intelligence of these particles, the swarm is able to repeatedly improve its best observed solution and converges to an optimum. The PSO has shown successful applications in many domains, such as evolving weights and structure for artificial neural networks [19], manufacture end milling [20], reactive power and voltage control [21], state estimation for electric power distribution systems [22], just to name a few. This paper presents a PSO-based algorithm for conquering the TAP. A hill-climbing heuristic is embedded in the PSO iteration to expedite the convergence. The experimental results manifest that the proposed hybrid PSO algorithm outperforms a genetic algorithm on a large set of simulated instances, and the performance difference is more significant as the problem size is increased.

The remainder of this paper is organized as follows. Section 2 formulates the TAP that will be addressed in this paper. Section 3 presents the proposed hybrid PSO algorithm in details. Section 4 reports the comparative performances and convergence analysis. Finally, Section 5 concludes this work.

2. Problem formulation

In this paper, we consider the TAP with the following scenarios. The processors in the system are heterogeneous and they are capacitated with various units of memory and processing resources. Hence, a task will incur different *execution cost* if it is executed on different processors. On the other hand, all of the communication links are assumed to be identical and some *communication cost* between two tasks will be incurred if there is a communication need between them and they are executed on different processors. A task will consume some units of the resources from its execution processor. The notations that will be used in our problem formulation are listed in Table 1.

Table 1
Notations used in our problem formulation

x_{ik}	Decision variable: $x_{ik}=1$ if task i is assigned to processor k, and $x_{ik}=0$ otherwise
y_{ijkl}	Decision variable: $y_{ijkl}=1$ if task i is assigned to processor k and task j is assigned to processor l, and $y_{ijkl}=0$ otherwise
n	Number of processors
r	Number of tasks
e_{ik}	Incurred execution cost if tasks i is executed on processor k
c_{ij}	Incurred communication cost between tasks i and j if they are executed on different processors
m_i	Memory requirements of task i from its execution processor
M_k	Memory capacity of processor k
p_i	Processing requirements of task i from its execution processor
P_k	Processing capacity of processor k

Our objective is to minimize the total execution and communication costs incurred by the task assignment subject to all of the resource constraints. Hence, the considered TAP can be formulated as the following 0–1 quadratic integer programming problem.

$$\text{Min} \quad Q(\boldsymbol{X}) = \sum_{i=1}^{r}\sum_{k=1}^{n} e_{ik}x_{ik} + \sum_{i=1}^{r-1}\sum_{j=i+1}^{r} \times c_{ij}\left(1 - \sum_{k=1}^{n} x_{ik}x_{jk}\right), \tag{1}$$

$$\text{subject to} \quad \sum_{k=1}^{n} x_{ik} = 1, \qquad \forall i = 1,2,\ldots,r \tag{2}$$

$$\sum_{i=1}^{r} m_i x_{ik} \le M_k, \qquad \forall k = 1,2,\ldots,n \tag{3}$$

$$\sum_{i=1}^{r} p_i x_{ik} \le P_k, \qquad \forall k = 1,2,\ldots,n \tag{4}$$

$$x_{ik} \in \{0,1\}, \qquad \forall i,k \tag{5}$$

The first and second terms in the objective function (1) represent the total execution cost and communication cost, respectively, incurred by the assignment $\boldsymbol{X}=\{x_{ik}\}_{1\le i\le r,\le k\le n}$. Constraint (2) states that each task should be assigned to exactly one processor. Constraints (3) and (4) ensure that the memory and processing resource capacity of each processor is no less than the total amount of resource demands of all of its assigned tasks. The last constraint (5) guarantees that x_{ik} are binary decision variables.

Since this formulation is an integer program with a quadratic objective function and is computationally prohibitive due to enormous computation efforts, its transformations to linear programs have been proposed [3,4]. Let y_{ijkl} be a binary variable and $y_{ijkl}=1$ if and only if task i is assigned to processor k and task j is assigned to processor l. The formulation $Q(\boldsymbol{X})$ can be transformed to the following 0–1 integer linear program.

$$\text{Min} \quad L(X)=\sum_{i=1}^{r}\sum_{k=1}^{n} e_{ik}x_{ik}+\sum_{i=1}^{r-1}\sum_{j=i+1}^{r}\sum_{k=1}^{n}\sum_{l\neq k} c_{ij}y_{ijkl}, \tag{6}$$

$$\text{subject to} \quad \sum_{k=1}^{n}\sum_{l=1}^{n} y_{ijkl}=1, \qquad \forall i,j=1,2,\ldots,r \tag{7}$$

$$\sum_{k=1}^{n} y_{ijkl}=x_{jl}, \qquad \forall i,j=1,2,\ldots,r;\ l=1,2,\ldots,n \tag{8}$$

$$\sum_{l=1}^{n} y_{ijkl}=x_{jk}, \qquad \forall i,j=1,2,\ldots,r;\ k=1,2,\ldots,n \tag{9}$$

$$\sum_{i=1}^{r} m_i x_{ik}\le M_k, \qquad \forall k=1,2,\ldots,n \tag{10}$$

$$\sum_{i=1}^{r} p_i x_{ik}\le P_k, \qquad \forall k=1,2,\ldots,n \tag{11}$$

$$x_{ik}, y_{ijkl}\in\{0,1\}, \qquad \forall i,j,k,l. \tag{12}$$

Exact solutions can be found using the new formulation $L(\boldsymbol{X})$ by mathematical programming techniques such as branch and bound; however, it is still time-consuming for deriving optimal solutions to large-scaled problems. On the other hand, an alternative for solving TAP efficiently is to find approximate solutions based on meta-heuristics. PSO has been shown to successfully optimize a wide range of nonlinear objective functions [19–22], the attractiveness of using PSO is due to the following features: natural metaphor, simplicity, stochastic move, adaptivity, positive feedback, and high quality solutions. Thus, we intend to further extend the application of PSO and propose a PSO-based algorithm for tackling the TAP.

3. Hybrid particle swarm optimization

The PSO algorithm was proposed by Kennedy and Eberhart [18] in 1995. It is inspired by the behavior of bird flocking and fish schooling. A large number of birds/fishes flock synchronously, change direction suddenly, and scatter and regroup together. Each individual, called a particle, benefits from the experience of its own and that of the other members of the swarm during the search for food. The PSO models the social dynamics of flocks of birds and serves as an optimizer for both of continuous and discrete functions. The convergence and parameterization aspects of the PSO have been discussed thoroughly [23–25]. It has also been shown that a hybrid strategy which embeds a local optimizer such as hill-climbing in between the iterations of a meta-heuristic algorithm can improve the performance significantly [26]. In light of this, we present a hybrid PSO for solving TAP hereafter.

3.1. Particle representation and initial swarm generation

In PSO, each particle corresponds to a candidate solution of the underlying problem. Thus, we let each particle represent a decision for task assignment using a vector of r elements, and each element is an integer value between 1 to n. Fig. 1 shows an illustrative example for the ith particle which corresponds to a task assignment that assigns five tasks to three processors, and $\text{particle}_{i,5}=2$ means that task 5 is assigned to processor 2.

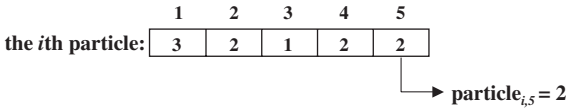


Fig. 1. An example for the ith particle.

The PSO randomly generates an initial swarm of K particles, where K is the swarm size. These particle vectors will be iteratively modified based on collective experiences in order to improve their solution quality.

3.2. Fitness evaluation

Each particle vector in the swarm is assigned a fitness value indicating the merit of this particle vector such that the swarm evolution is navigated by best particles. Clearly, the objective value of $Q(\boldsymbol{X})$ (see Eq. (1)) can be used to measure the quality of each particle vector. However, this value is discredited if the particle vector violates at least one of the constraints (2)–(5). In modern heuristics [26], infeasible solutions also provide valuable clue to targeting the optimal solution. The degree of infeasibility for trial solutions are measured and transformed to a penalty which grows in proportional to the infeasibility level, thus guiding the search toward feasible space. Consequently, we devise a penalty function to estimate the infeasibility level of a particle solution. Since constraints (2) and (5) are always satisfied if our particle representation scheme is adopted, the penalty function is only related to constraints (3) and (4), and it is given by

$$\text{Penalty}(\boldsymbol{X}) = \max\left(0, \sum_{i=1}^{r} m_i x_{ik} - M_k\right) + \max\left(0, \sum_{i=1}^{r} p_i x_{ik} - P_k\right) \quad (13)$$

Thus, the penalty function gives some penalties on the assignment solution whose requested resource exceeds the system capacity. The more the exceeding resource demand is, the more the penalty is. The fitness function of the particle vector can finally be defined as

$$\text{Fitness}(\boldsymbol{X}) = (Q(\boldsymbol{X}) + \text{Penalty}(\boldsymbol{X}))^{-1} \quad (14)$$

Hence, the higher the fitness value is, the better the quality of the particle vector is.

3.3. Particle vector modification

To simulate the bird flocking for food foraging, the particle vectors are iteratively modified during the PSO evolution. According to the fitness values of these particle vectors, each particle remembers the best vector it experienced so far, referred to as *pbest*, and the best vector experienced by its *neighbors*. The particle's neighbors are defined as the particles within its topological neighborhood in the solution space. There are two versions for keeping the neighbors' best vector, namely *lbest* and *gbest*. In the local version, each particle keeps track of the best vector lbest attained by its local topological neighborhood of particles. In many applications, the neighborhood size is set to about 15% of the swarm size. For the global version, the best vector gbest is determined by any particles in the entire swarm. Hence, the gbest model is a special case of the lbest model. During each PSO iteration, particle i adjusts its velocity v_{ij} and position vector particle$_{ij}$ through each dimension j by referring to, with random multipliers, the personal best vector (pbest$_{ij}$) and the swarm's best vector (gbest$_j$, if the global version is adopted) using Eqs. (15) and (16) as follows.

$$v_{ij} \leftarrow v_{ij} + c_1 \text{rand}_1\left(\text{pbest}_{ij} - \text{particle}_{ij}\right) + c_2 \text{rand}_2\left(\text{gbest}_j - \text{particle}_{ij}\right) \quad (15)$$

and

$$\text{particle}_{ij} \leftarrow \text{particle}_{ij} + v_{ij} \quad (16)$$

where c_1 and c_2 are the cognitive coefficients and rand_1 and rand_2 are random real numbers drawn from $U(0,1)$. Thus, the particle flies through potential solutions toward pbest$_i$ and gbest in a navigated way while still exploring new areas by the stochastic mechanism to escape from local optima. The cognitive coefficients c_1 and c_2 represent the weightings that pull each particle toward pbest$_i$ and gbest. Low values let particles wander around their local neighborhood, while high values cause particles to fly toward, or pass, optimal solutions [18]. Many applications set c_1 and c_2 each equal to 2.0. The

particle's velocity on each dimension is set restricted by a maximum velocity $v_{\max}$, which controls the maximum travel distance during each iteration to avoid this particle flying past good solutions. The PSO algorithm is terminated with a maximal number of iterations or the best particle vector of the entire swarm cannot be improved further after a sufficiently large number of iterations.

3.4. Hybrid strategy

Modern meta-heuristics manage to combine *exploration* and *exploitation* search. The exploration search seeks for new regions, and once it finds a good region, the exploitation search kicks in. However, since the two strategies are usually inter-wound, the search may be conducted to other regions before it reaches the local optima. As a result, many researchers suggest to employ a hybrid strategy which embeds a local optimizer such as hill-climbing heuristic in between the iterations of the meta-heuristics [26]. In light of this, we embed a hill-climbing heuristic in between the iterations of the PSO.

The embedded hill-climbing heuristic proceeds as follows. Given a particle vector, its r elements are sequentially examined for updating. The value of the examined element is replaced, in turn, by each integer value from 1 to n, and retains the best one that attains the highest fitness value among them. While an element is examined, the values of the remaining $r-1$ elements remain unchanged. The heuristic is terminated if all the elements of the particle have been examined for updating. The computation for the fitness value due to the element updating can be minimized. Since a value change in one element affects the assignment of exactly one task, we can save the fitness computation by only recalculating the system costs and constraint conditions related to the reassigned task.

3.5. The HPSO algorithm

The details of the proposed hybrid PSO (referred to as HPSO) algorithm are presented in Fig. 2. The algorithm starts with an initial swarm of K particles. Each particle vector corresponds to a candidate solution of the underlying problem. Then, all of the particles repeatedly move until a maximal number of iterations have been passed. During each iteration, the particle individual best and swarm's best positions are determined. The particle adjusts its position based on the individual experience (pbest_i) and the swarm's intelligence (gbest) as described in Eqs. (15) and (16). To expedite the convergence speed, all of the particles are further updated using the hill-climbing heuristic before entering the next iteration. When the algorithm is terminated, the incumbent gbest and the corresponding fitness value are output and considered as the optimal task assignment and the minimum cost.

4. Experimental results

We render the inter-task communication by a task interaction graph (TIG), $G(V,E)$, where V is a set of r nodes indicating the r tasks to be executed and E is a

1. Initialize.
   1.1 Generate K particles at random.
   1.2 Generate velocities v_{ij}, $1 \le i \le K$ and $1 \le j \le r$, where v_{ij} is randomly drawn from [0.0, 1.0].
2. Repeat until a given maximal number of iterations is achieved.
   2.1 Evaluate the fitness of each particle.
   2.2 Determine the best vector *pbest* visited so far by each particle.
   2.3 Determine the best vector *gbest* visited so far by the whole swarm.
   2.4 Update velocities v_{ij} using (15) restricted by a maximum threshold v_{max}.
   2.5 Update particles' vectors using (16).
   2.6 Improve the solution quality of each particle using the embedded hill-climbing heuristic.

Fig. 2. The HPSO algorithm for the task assignment problem.

set of edges specifying the communication requirements among these tasks. Fig. 3 gives an example of a TIG where the communication requirements among four tasks are specified. Here, we define the task interaction density d of $G(V,E)$ as

$$d = \frac{|E|}{r(r-1)/2}, \tag{17}$$

where $|E|$ calculates the number of existing communication demands in the TIG, and $r(r-1)/2$ indicates the maximal number of communication demands among r tasks. Therefore, the task interaction density quantifies the ratio of the inter-task communication demands for a TIG and can serve as one of the key factors that affect the problem complexity. The other key factors are the number of tasks (r) and the number of processors (n).

The testing data set is generated according to different problem characteristics. We set the value of (r, n) to (5, 3), (10, 6), (15, 9), and (20, 12), respectively, in order to testify the algorithm with different problem scales. For each pair of (r, n), we generate three different TIGs at random with density d equivalent to 0.3, 0.5, and 0.8. Furthermore, 10 problem instances are generated at random for each specification of r, n, and d. The values of the other parameters are generated randomly: the execution cost is between 1 and 200, the communication cost is between 1 and 50, the memory and processing capacity of each processor varies from 50 to 250, and the memory and processing requirement of each task ranges from 1 to 50. As such, we obtain a testing data set of 120 problem instances for evaluating the comparative performances of the competing methods. In the following sections, all of the experiments are conducted on a 2.4 GHz PC with 256 MB RAM.

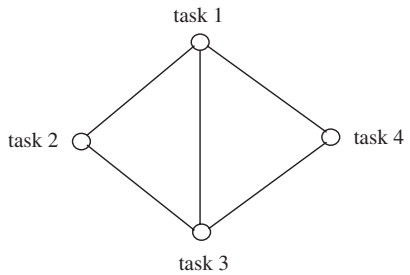


Fig. 3. An example of TIG.

Table 2
The minimum costs obtained using the Lingo package and the used CPU time

r	n	d	Minimum cost	CPU time
5	3	0.3	180.97	1 s
		0.5	202.40	1 s
		0.8	215.63	1 s
10	6	0.3	374.68	91 s
		0.5	403.15	202 s
		0.8	715.61	1746 s
	Others		Infeasible or unknown	>90 h

4.1. Exact solutions

To derive the exact solutions of the testing problems, we develop a program using the commercial Lingo package to solve the 0–1 linear program formulation $L(\boldsymbol{X})$ (see Eqs. (6)–(12)). The maximum allowed CPU time for solving an instance by the Lingo package is set to 90 h. The numerical results are shown in Table 2. It is observed that the used CPU time by Lingo to derive the exact solutions is dependent upon two key factors, the value of (r, n) and the value of d. As the numbers of tasks and processors increase, the problem is more difficult. Also, for a particular instance of (r, n), a TIG with denser inter-task communications (i.e., with a higher value of d) will incur more computations. When Lingo fails to solve the linear program within 90 h, it is terminated with an infeasible solution or an unknown status and we discard such cases for further comparison.

4.2. Comparative performances

In this section, we present the comparative performances between the proposed HPSO and a genetic algorithm (GA) because both of them are population-based, evolutionary computation algorithms. The GA uses the same coding scheme (see Fig. 1) and fitness function (see Eq. (14)) as used by the HPSO. The single-point crossover and bit-flipping mutation are performed during the evolution. The parameter values used in both of HPSO and GA are optimally tuned by intensive preliminary experiments to let the competing algorithms perform at the best level. To be specific, the parameter setting used by HPSO is (number of particles = 80, $c_1 = c_2 = 2$) and GA (population size = 80, crossover rate = 0.7, mutation

rate=0.1). Since the evaluation of fitness function is the main source providing the guidance for targeting the optimal solution and is also the most time-consuming component for both algorithms, the performance is evaluated as the minimum cost obtained when the testing algorithms have run for a specified number of fitness evaluations (here, we set this number to 80,000). Moreover, both HPSO and GA are stochastic-based algorithms and each independent run of the same algorithm on a particular testing problem may yield a different result, we thus calculate the average cost over 10 independent runs of each algorithm for every problem instance.

Table 3 tabulates the average derived costs obtained using the two algorithms. It is observed that for the whole problem category (r, n)=(5, 3), both algorithms can derive exact solutions as reported by Lingo (see Table 2), but GA needs more computation time. For the other categories, the costs obtained by GA are larger than those produced by HPSO. The harder the problem is, the larger the cost difference between the two algorithms, which means the proposed HPSO algorithm is more scalable against problem complexity. To clearly realize the superiority of HPSO over GA, we conduct the matched pair t-test on the cost differences and obtain t value as 3.967. Since the confidence coefficient is 1.796 for the 95% confidence interval over 12 cases, we observe that the cost difference is statistically significant.

Table 3
The average derived costs and the used CPU times (in s) by GA and HPSO over 120 problem instances which are classified into 12 categories

r	n	d	GA		PSO		Cost difference
			Cost	CPU time	Cost	CPU time	
5	3	0.3	180.97	8.64	180.97	2.36	0
		0.5	202.40	9.00	202.40	2.56	0
		0.8	215.63	8.16	215.63	2.32	0
10	6	0.3	436.32	10.78	376.11	4.62	60.21
		0.5	501.05	10.82	403.15	4.50	97.90
		0.8	806.21	11.04	719.55	4.32	86.66
15	9	0.3	889.37	18.08	705.20	9.34	184.17
		0.5	1408.24	17.62	1221.02	9.20	187.22
		0.8	1975.20	19.74	1740.22	9.54	234.98
20	12	0.3	1744.44	26.56	1431.39	14.30	313.05
		0.5	2645.37	27.78	2269.32	15.44	376.05
		0.8	3744.47	27.36	3400.52	15.40	343.95

The cost differences between the two algorithms are also reported.

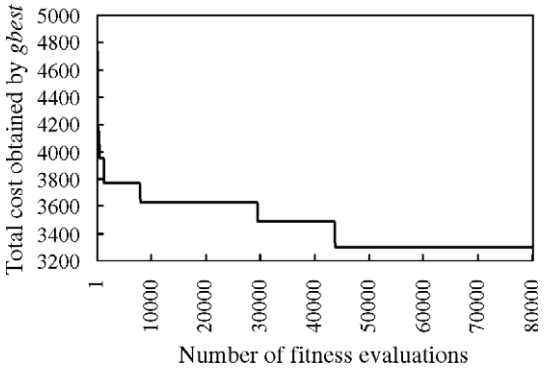


Fig. 4. The swarms best solution in terms of the total cost obtained by gbest versus the number of fitness evaluations.

4.3. Convergence analysis

To analyze the convergence behavior of the proposed algorithm, we monitor the variations of experiences learned by the entire swarm (as in gbest) and individual particles (as in pbest). Fig. 4 displays a typical run of the HPSO for solving a problem instance with (r, n, d)=(20, 12, 0.8). The swarm's incumbent best solution in terms of the total cost incurred by the task assignment obtained by gbest decreases as the number of fitness evaluations increases. This validates the correctness of the proposed fitness function (Eq. (14)) which navigates the evolution of the particle swarm toward quality solutions.

We further testify whether the entire swarm evolves to the same optimization goal and the final high-quality solution is a consequence of the collective intelligence instead of the movement of a lucky particle. We propose the information *entropy* for measuring the similarity convergence among the individual pbest experiences as follows. Let $\text{prob}_j(s)$ be the probability of the individual's decision that assigns the jth task to the sth processor, viz.

$$\text{prob}_j(s) = \Pr\left\{\text{pbest}_{ij} = s | i = 1,2,\ldots,K\right\}. \tag{18}$$

The information entropy on the decision with respect to the assignment of the jth task is given by

$$\text{entropy}_j = -\sum_{s=1}^{n} \text{prob}_j(s)\log_2\left(\text{prob}_j(s)\right). \tag{19}$$

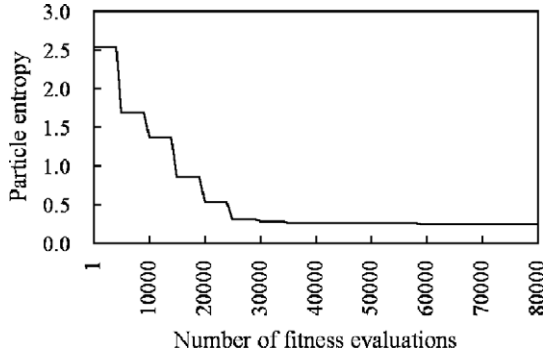


Fig. 5. The average particle entropy over all pbest versus the number of fitness evaluations.

The smaller the value of entropy_j, the larger the number of particles that make a consensus decision by assigning the jth task to the same processor. To provide an overall measure, we compute the average particle entropy value as

$$\text{entropy} = \sum_{j=1}^{r} \text{entropy}_j / r. \tag{20}$$

Fig. 5 shows the variations of the average particle entropy value for a typical run on the problem instance with (r, n, d)=(20, 12, 0.8). It is observed that the entropy value drops as the number of fitness evaluations increases, meaning that the particle individual experience is resorting to the same high-quality solution as the swarm converges.

4.4. Worst-case analysis

Since the HPSO is a stochastic algorithm, that is, each separate run of the program could result in a different result. It is desired to know the worst performance one may obtain if the HPSO is adopted. We conduct the worst-case analysis which is set up as the worst optimum solution we could get after a specific number of repetitive runs of the HPSO. Fig. 6 shows the worst-case analysis where the HPSO is executed on a problem instance with (r, n, d)=(20, 12, 0.8) for 1000 times. The curve is intuitive since the optimum solutions obtained from the repetitive runs of the HPSO algorithm follow a normal distribution (see Fig. 7). The result from the worst-case analysis is useful when the user requests a guarantee for the quality of the optimum solution. For example, if the user requests a guarantee for a solution with cost less than 3250, he/she can obtain such a solution by executing the HPSO for less than 100 repetitive runs since in the worst case, as shown in Fig. 6, the solution with cost less than 3250 can be obtained after 100 runs of HPSO, but in general case, most trial runs in 100 repetitions will obtain a better solution.

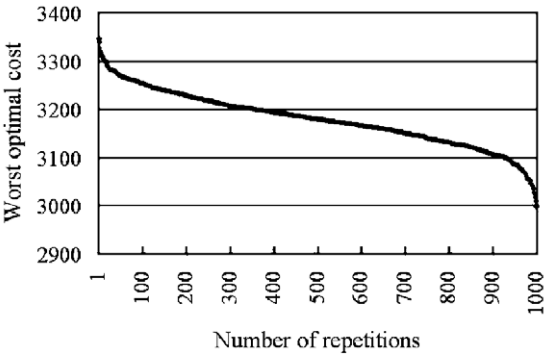


Fig. 6. The worst-case analysis versus the number of repetitive runs of HPSO.

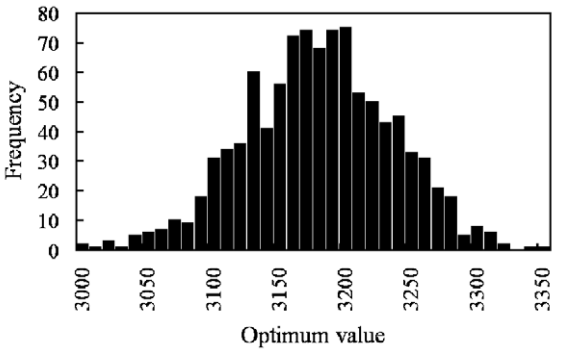


Fig. 7. The histogram of the optimum solutions obtained from 1000 repetitive runs of HPSO.

5. Conclusions

In many problem domains, we are required to assign the tasks of an application to a set of distributed processors such that the incurred cost is minimized and the system throughput is maximized. Several versions of the task assignment problem (TAP) have been formally defined but, unfortunately, most of them are NP-complete. In this paper, we have proposed a hybrid particle swarm optimization (HPSO) algorithm which finds a near-optimal task assignment with reasonable

time. The computational experience manifests that the HPSO reports exact optimal solutions verified by a commercial linear programming software for a number of TAP instances. For large scale TAPs, in which case the commercial software fails to derive exact optimal solutions, the performance of the HPSO is assessed by competing with a genetic algorithm (GA). The experimental result shows that the HPSO outperforms the GA and the cost difference between the two approaches is statistically significant. Also, the convergence and the worst-case analyses of the HPSO have been empirically conducted.

We are currently conducting our research for using HPSO to solve another version of the TAP where each processor and each communication link has a failure ratio and the problem objective is to maximize the reliability for accomplishing the task execution.

References

[1] H.S. Stone, Critical load factors in two-processor distributed systems, IEEE Transactions on Software Engineering, SE-4 (1978) 254–258.

[2] V.M. Lo, Task assignment in distributed systems, PhD dissertation, Dep. Comput. Sci., Univ. Illinois, Oct. 1983.

[3] A. Ernst, H. Hiang, M. Krishnamoorthy, Mathematical programming approaches for solving task allocation problems, Proc. of the 16th National Conf. Of Australian Society of Operations Research, 2001.

[4] A. Billionnet, M.C. Costa, A. Sutter, An efficient algorithm for a task allocation problem, Journal of ACM 39 (1992) 502–518.

[5] G.H. Chen, J.S. Yur, A branch-and-bound-with-underestimates algorithm for the task assignment problem with precedence constraint, Proc. of the 10th International Conf. on Distributed Computing Systems, 1990, pp. 494–501.

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

[7] D.M. Nicol, D.R. O'Hallaron, Improved algorithm for TAP pipelined and parallel computations, IEEE Transactions on Computers 40 (1991) 295–306.

[8] D. Fernandez-Baca, A. Medepalli, Parametric task allocation on partial *k*-trees, IEEE Transactions on Computers 42 (1993) 738–742.

[9] C.H. Lee, K.G. Shin, Optimal task assignment in homogeneous networks, IEEE Transactions on Parallel and Distributed Systems 8 (1997) 119–129.

[10] M. Kafil, I. Ahmad, Optimal task assignment in heterogeneous distributed computing systems, IEEE Concurrency 6 (1998) 42–50.

[11] E.S.H. Hou, N. Ansari, H. Ren, A genetic algorithm for multiprocessor scheduling, IEEE Transactions on Parallel and Distributed Systems 5 (1994) 113–120.

[12] A.B. Hadj-Alouane, J.C. Bean, K.G. Murty, A hybrid genetic/optimization algorithm for a task allocation problem, Journal of Scheduling 2 (1999) 189–201.

[13] A.K. Tripathi, B.K. Sarker, N. Kumar, A GA based multiple task allocation considering load, International Journal of High Speed Computing (2000) 203–214.

[14] F.T. Lin, C.C. Hsu, Task assignment scheduling by simulated annealing, Proceeding of Conference on Computer and Communication Systems, 1990, pp. 279–283.

[15] Y. Hamam, K.S. Hindi, Assignment of program tasks to processors: a simulated annealing approach, European Journal of Operational Research 122 (2000) 509–513.

[16] F. Glover, Tabu search: Part I, ORSA Journal on Computing 1 (1989) 190–206.

[17] M. Dorigo, L. Gambardella, Ant colony system: a cooperative learning approach to the traveling salesman problem, IEEE Transactions on Evolutionary Computation 1 (1997) 53–66.

[18] J. Kennedy, R.C. Eberhart, Particle swarm optimization, Proceedings IEEE Int'l. Conf. on Neural Networks, vol. IV, 1995, pp. 1942–1948.

[19] R.C. Eberhart, Y. Shi, Evolving artificial neural networks, Proceedings Int'l. Conf. on Neural Networks and Brain, 1998, pp. PL5–PL13.

[20] V. Tandon, Closing the gap between CAD/CAM and optimized CNC end milling, Master thesis, Purdue School of Engineering and Technology, Indiana University Purdue University Indianapolis, 2000.

[21] H. Yoshida, K. Kawata, Y. Fukuyama, Y. Nakanishi, A particle swarm optimization for reactive power and voltage control considering voltage stability, Proceedings Int'l. Conf. on Intelligent System Application to Power Systems, 1999, pp. 117–121.

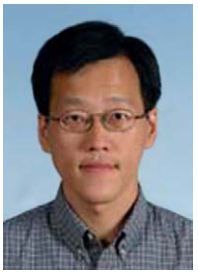
[22] N. Shigenori, G. Takamu, Y. Toshiku, F. Yoshikazu, A hybrid particle swarm optimization for distribution state estimation, IEEE Transactions on Power Systems 18 (2003) 60–68.

[23] K.E. Parsopoulos, M.N. Vrahatis, Recent approaches to global optimization problems through particle swarm optimization, Natural Computing 1 (2002) 235–306.

[24] M. Clerc, J. Kennedy, The particle swarm explosion, stability, and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (2002) 58–73.

[25] I.C. Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection, Information Processing Letters 85 (2003) 317–325.

[26] Z. Michalewicz, D.B. Fogel, How to Solve It: Modern Heuristics, Springer-Verlag, 2002.



Peng-Yeng Yin received his BS, MS and PhD degrees in Computer Science from the National Chiao Tung University, Hsinchu, Taiwan. From 1993 to 1994, he was a visiting scholar at the Department of Electrical Engineering, University of Maryland, College Park, and the Department of Radiology, Georgetown University, Washington DC. In 2000, he was a visiting Professor in the Visualization and Intelligent Systems Laboratory (VISLab) at the Department of Electrical Engineering, University of California, Riverside (UCR). From 2001 to 2003, he was a Professor at the Department of Computer Science and Information Engineering, Ming Chuan University, Taoyuan, Taiwan. Since 2003, he has been a Professor of the Department of Information Management, National Chi Nan University, Nantou, Taiwan, and is currently the Chairman of the Department there. Dr. Yin received the Overseas Research Fellowship from Ministry of Education in 1993, Overseas Research Fellowship from National Science Council in 2000. He has received the best paper award from the Image Processing and Pattern Recognition Society of Taiwan. He is a member of the Phi Tau Phi Scholastic Honor Society and listed in *Who's Who in the World*. His current research interests include pattern recognition, content-based image retrieval, relevance feedback, machine learning, computational intelligence, and computational biology.

Shiuh-Sheng Yu received his BS and PhD degrees in Computer Science and Information Engineering from National Taiwan University, and his MS degree in Computer Science from State University of New York at Stony Brook. From 1996 to 1999, he was a Lecturer at the Department of Information Management, National Chi Nan University, Nantou, Taiwan. Since 1999, he has been an Associate Professor of the same department. His current research interests include health-care information systems, digital museums, bioinformatics, and XML databases.

Pei-Pei Wang received her BS degree in Information Management from Ming Chuan University, Taoyuan, Taiwan, in 2003, and is currently pursuing the MBA degree in Information Management at National Chi Nan University, Nantou, Taiwan. Her research interests include computational intelligence, machine learning, and distributed systems.

Yi-Te Wang received her BSW degree in Social Work from Fu Jen Catholic University, Taipei, Taiwan, in 2003, and is currently pursuing an MBA degree in Information Management at National Chi Nan University, Nantou, Taiwan. Her research interests include web programming, content management, and system security.