Improved Particle Swarm Optimization for Solving Multiprocessor Scheduling Problem: Enhancements and Hybrid Methods

F. CHOONG, S. PHON-AMNUAISUK, M.Y. ALIAS
School of Engineering
Taylor's University
No. 1, Jalan Taylor's, 47500 Subang Jaya, Selangor
MALAYSIA
florence.choong@gmail.com http://www.taylors.edu.my

Abstract: - Memetic algorithms (MAs) are hybrid evolutionary algorithms (EAs) that combine global and local search by using an EA to perform exploration while the local search method performs exploitation. Combining global and local search is a strategy used by many successful global optimization approaches, and MAs have in fact been recognized as a powerful algorithmic paradigm for evolutionary computing. This paper presents a hybrid heuristic model that combines particle swarm optimization (PSO) and simulated annealing (SA). This PSO/SA hybrid was applied on the multiprocessor scheduling problem to perform static allocation of tasks in a heterogeneous distributed computing system in a manner that is designed to minimize the cost. Additionally, this paper also focuses on the design and implementation of several enhancements to PSO based on diversity and efficient initialization using different distributions. The results show the effectiveness and superiority of the hybrid algorithms.

Key-Words: - Memetic Algorithms, Particle Swarm Optimization, Simulated Annealing, Hybrid, Multiprocessor Scheduling, Optimization

1 Introduction

Memetic algorithms (MAs) are evolutionary algorithms (EAs) that apply a separate local search process to refine individuals. These methods are inspired by models of adaptation in natural systems that combine evolutionary adaptation of populations of individuals with individual learning within a lifetime [1]. MAs include a broad class of metaheuristics. This method is based on a population of agents and proved to be of practical success in a variety of problem domains. Unlike traditional Evolutionary Computation approaches, MAs are concerned with exploiting all available knowledge about the problem under study. This is not as an optional mechanism, but as a fundamental feature.

From an optimization point of view, MAs are hybrid EAs that combine global and local search by using an EA to perform exploration while the local search method performs exploitation. Combining global and local search is a strategy used by many successful global optimization approaches, and MAs have in fact been recognized as a powerful algorithmic paradigm for evolutionary computing. In this paper, the proposed hybrid heuristic model involves a Particle Swarm Optimization (PSO) [2] and Simulated Annealing (SA) [3] algorithm resulting in a fast and easily implemented hybrid algorithm. This PSO/SA hybrid performs static allocation of tasks in a heterogeneous distributed computing system in a manner that is designed to minimize the cost. The problem of scheduling a set of dependent or independent tasks in a distributed computing system is a well-studied area. In this section, a static task allocation [4] in the heterogeneous computing system is examined which provides a variety of architectural capabilities, orchestrated to perform on application problems whose tasks have diverse execution requirements. The proposed method assigns the tasks to processors and avoids becoming trapped in local optimum and also leads to faster convergence towards the targeted solution.

Several research works have been carried out in Task Assignment Problem (TAP). The traditional methods such as branch and bound, divide and conquer, and dynamic programming give the global optimum, but are often too time consuming or do not apply for solving typical real-world problems. The researchers [5-7] had derived optimal task assignments to minimize the sum of task execution and communication costs with the branch-and-bound method and evaluated the computational complexity of this method using simulation. Many of the heuristic algorithms use a graphical representation of the task-processor system such that a Max Flow/Min Cut Algorithm can be utilized to find assignments of tasks to processors which minimize total execution and communication costs [8-9] and conclude that a measure of degree to which an algorithm achieves load balancing [10] can yield fairly unbalanced assignments. Traditional methods used in optimization are deterministic, fast, and give exact answers but often tends to get stuck on local optima. Also the time complexity from exponential to polynomial for traditional search algorithms on NP-hard problems cannot be changed. Consequently, another approach is needed when the traditional methods cannot be applied. The modern heuristic approach helps in such situation. Modern heuristics are general purpose optimization algorithms. Their efficiency or applicability is not tied to any specific problem-domain. Available heuristics include SA, Genetic Algorithm (GA) [11] and Ant Colony algorithm [12]. Peng-Yeng et.al (2006) had proposed a hybrid strategy using Hill Climbing algorithm as a local search method along with PSO [13]. However, hill climbing heuristic has a major problem of getting trapped in local optima.

2 Problem Formulation

This paper considers the TAP with the following scenario. The system consists of a set of heterogeneous processors (n) having different memory and processing resources, which implies that tasks (r), executed on different processor encounters different execution cost. The communication links are assumed to be identical, however communication cost between two tasks will be encountered when executed on different processors. A task will make use of the resources from its execution processor [9]. The objective is to minimize the total execution and communication cost encountered by the task assignment subject to the resource constraints. To achieve minimum cost for the TAP, the function is formulated as shown in Eqs. (1)-(5):

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

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

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

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

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

x_{ik} is set to 1 if task i is assigned to processor k. n denotes the number of processors, r denotes the number of tasks, e_{ik} denotes the incurred execution cost if tasks i is executed on processor k. c_{ij} is the incurred communication cost if tasks i and j are executed on different processors. m_i and p_i represents the memory requirements and processing requirements of task i respectively. M_k and P_k are the memory and processing capacity of processor k. Q(X) is the objective function which combines the total execution cost and total communication cost as specified in 1st and 2nd terms of Eq. (1). The first constraint mentioned in Eq. (2) says that each task should be assigned to exactly one processor. Eq. (3) and Eq. (4) are the 2nd and 3rd constraints respectively and they assure that the resource demand should never exceed the resource capacity. The final constraint as mentioned in Eq. (5) specifies that the x_{ik} is a binary decision variable.

3 Implementation

This section discusses the simple PSO and the proposed hybrid PSO/SA. In PSO, each particle corresponds to a candidate solution of the underlying problem. In the proposed method each particle represents a feasible solution 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 where each row represents the particles which correspond to a task assignment that assigns five tasks to three processors, and Particleparticle$_{3,T4}$ = P1 means that in particle 3 the Task 4 is assigned to Processor 1.

Particle number	T1	T2	T3	T4	T5
particle 1	P3	P2	P1	P2	P2
particle 2	P1	P2	P3	P1	P1
particle 3	P1	P3	P2	P1	P2
particle 4	P2	P1	P2	P3	P1
particle 5	P2	P2	P1	P3	P1

Fig. 1 Representation of particles

In the hybrid version, hybridization is done by performing SA at the end of an iteration of simple PSO [14]. Generally the TAP is to assign the n tasks to m processors so that the load is shared and also balanced. Here the proposed system considers n=20 and r=5 i.e. 20 tasks should be shared among 5 processors. The system calculates the fitness value of each assignment and selects the optimal assignment from the set of solutions. The system compares the memory and processing capacity of the processor with the memory and processing requirements of the tasks respectively. If capacity is enough then the task is assigned, else a penalty is added to the calculated fitness value.

3.1 Fitness Evaluation

The initial population is generated randomly and checked for the consistency [15]. Then each particle must be assigned with the velocities obtained randomly and it lies in the interval [0, 1]. Each solution vector in the solution space is evaluated by calculating the fitness value for each vector. The objective value of Q(X) in Eq. (1) can be used to measure the quality of each solution vector. In modern heuristics the infeasible solutions are also considered since they may provide a valuable clue to targeting optimal solution [16]. A penalty function as shown in Eq. (6) is devised to estimate the infeasibility level of a solution. The penalty function is only related to constraints (3) and (4), and it is given by Eq. (6):

$$Penalty(X) = \max\left(0, \sum_{i=1}^{r} m_i x_{ik} - M_k\right) + \max\left(0, \sum_{i=1}^{r} p_i x_{ik} - P_k\right) \quad (6)$$

This penalty is added to the objective function whenever the resource requirement exceeds the capacity. Hence the fitness function of the particle vector can finally be defined as in Eq. (7),

$$Fitness(X) = (Q(X) + Penalty(X))^{-1} \quad (7)$$

Hence, as the fitness value increases the total cost is minimized which is the objective of the problem.

3.2 Simple PSO

In implementing the simple PSO algorithm, two versions for keeping the neighbors' best vector, namely lbest and gbest are considered. The position of a particle is influenced by the best position visited by itself i.e. its own experience and the position of the best particle in its neighborhood i.e. the experience of neighboring particles. When the neighborhood of a particle is the entire swarm, the best position in the neighborhood is referred to as the global best position of the particle, and the resulting algorithm is referred to as the *gbest* PSO. When smaller neighborhoods are used, the algorithm is generally referred to as the *lbest* PSO. The performance of each particle is measured using a fitness function that varies depending on the optimization problem [12].

The global neighborhood 'gbest' is the most intuitive neighborhood. In the local neighborhood 'lbest', a particle is just connected to a fragmentary number of processes [2]. The best particle is obtained from, the best particle in each fragment. In the global version, every particle has access to the fitness and best value so far of all other particles in the swarm. Each particle compares its fitness value with all other particles. This method implements the star topology. It has exploitation of solution spaces but exploration is weak [17]. In the local version, each particle keeps track of the best vector lbest attained by its local topological neighborhood of particles. Each particle compares with its neighbors decided based on the size of the neighborhood. The groups exchange information about local optima. Here the exploitation of solution space is weakened and exploration becomes stronger.

3.3 Hybrid PSO

Modern metaheuristics 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 employing a hybrid strategy, which embeds a local optimizer in between the iterations of the meta-heuristics [18].

In SA, the fundamental idea is to allow moves resulting in solutions of worse quality than the current solution (uphill moves) in order to escape from local minima. The probability of doing such a move is decreased during the search. The high level algorithm is described in Fig. 2. The algorithm starts by generating an initial solution (either randomly or heuristically constructed) and by initializing the so-called temperature parameter T. Then, at each iteration, a solution is randomly sampled and it is accepted as new current solution with a certain probability. The temperature T is decreased during the search process, thus at the beginning of the search the probability of accepting uphill moves is high and it gradually decreases, converging to a simple iterative improvement algorithm. This process is analogous to the annealing process of metals and glass, which assume a low energy configuration when cooled with an appropriate cooling schedule. Regarding the search process, this means that the algorithm is the result of two combined strategies; random walk and iterative improvement. In the first phase of the search, the bias toward improvements is low and it permits the exploration of the search space; this erratic component is slowly decreased thus leading the search to converge to a local minimum. The probability of accepting uphill moves is controlled by the difference of the objective functions and the temperature.

```
s ← GenerateInitialSolution()
T ← T₀
while termination conditions not met do
    s' ← PickAtRandom(N(s))
    if (f(s') < f(s)) then
        s ← s'
    else
        Accept s' as new solution with probability p(T,s',s)
    endif
    Update(T)
endwhile
```

Fig. 2 Algorithm: Simulated Annealing (SA).

In the application of the TAP, the embedded SA 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. A neighbor of the new particle is selected. The fitness values for the new particle and its neighbor are found. They are compared and the minimum value is selected. This minimum value is assigned to the personal best of this particle. The heuristic is terminated if all the elements of the particle have been examined for updating and all the particles are examined. The computation for the fitness value due to the element updating can be maximized. 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.

4 Results

This section describes the results of simulations conducted to gain insight into the performance of the PSO-SA hybrid implementation. Various versions of PSO algorithm like the simple PSO, the global PSO and Hybrid PSO were implemented. The experimental results clearly demonstrate the effectiveness of the Hybrid PSO. The value of (r, n) is set to (20, 5). The values of the other parameters are generated randomly as specified in [13]. The results of this experiment are obtained by varying the number of particles, number of iterations and topology of neighbourhood particles.

4.1 Cost Evaluation

The task incurs the execution cost and communication cost when executed on different machines. Our objective is to minimize this total cost. This section discusses the evaluation of cost in various versions of PSO using two methods: Cost vs. Number of iterations and Cost vs. Population Size. For each version the number of iterations was increased up to 100 and the results were recorded.

Cost evaluation in done for the global best PSO (gBest), local best PSO (lBest) and the hybrid PSO. Two methods were carried out. In the first method the cost is compared with an increase in the number of iterations. For the gBest PSO, the cost obtained initially was 1011. As illustrated in Fig. 3, the cost reduces as we increase the number of iterations and gBest PSO converges to the minimum cost of 857 at the 28th iteration and remains the same till the last iteration.

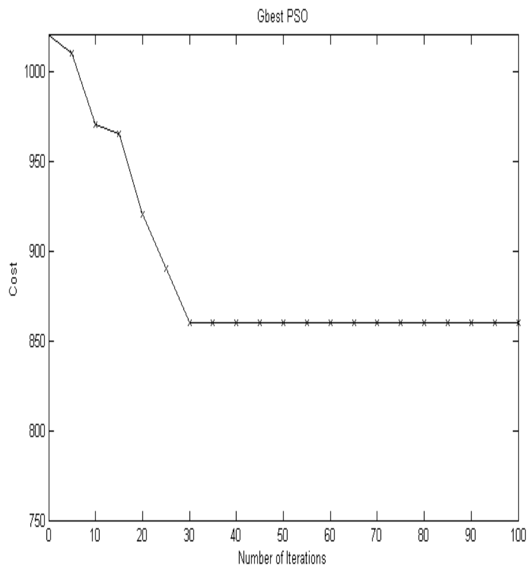


Fig. 3 Global best PSO with varying number of iterations

Fig. 4 depicts the cost obtained for the lBest PSO. The initial cost is 1035 and then reduces as we increase the number of iterations and lBest PSO converges to the minimum cost of 857 at the 100th iteration only. This is because it gets the information only from its neighbours.

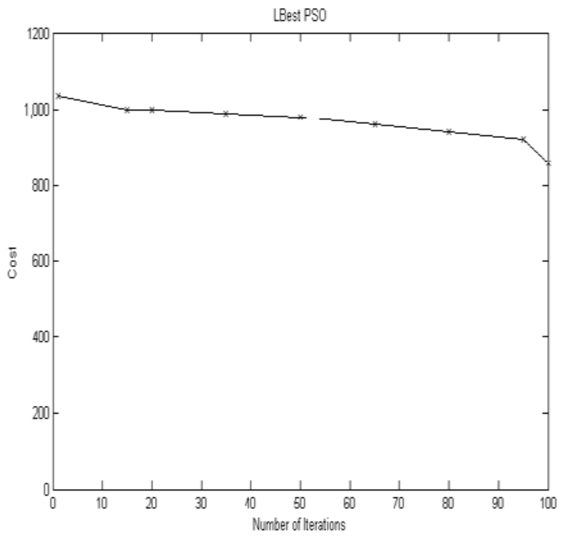


Fig. 4 Local best PSO with varying number of iterations

Running the same test on the hybrid PSO yields an initial cost of 940 as shown in Fig. 5. The cost reduces as we increase the number of iterations and Hybrid PSO converges to the minimum cost of 857 at the 21st iteration and remains the same till the last iteration. The faster convergence is due to the hybridization with simulated annealing.

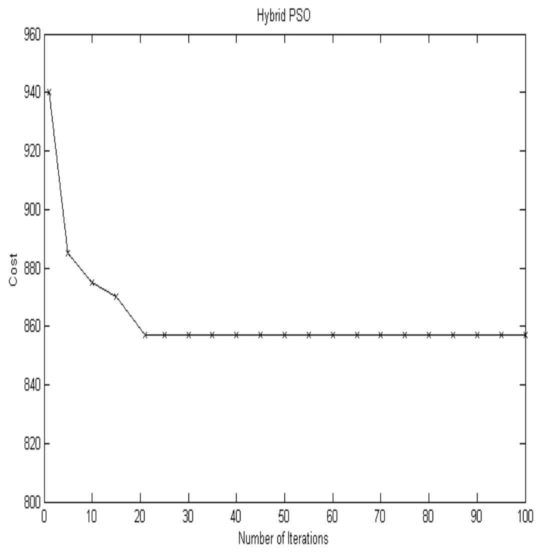


Fig. 5 Hybrid PSO with varying number of iterations

In the second method the cost obtained is compared with an increase in the population size. For the gBest PSO, the cost initially was 883 as illustrated in Fig. 6. The cost then reduces as we increase the population size and it finally converges to the minimum cost of 783 for the population size of 500 and remains the same albeit the increase in the population size.

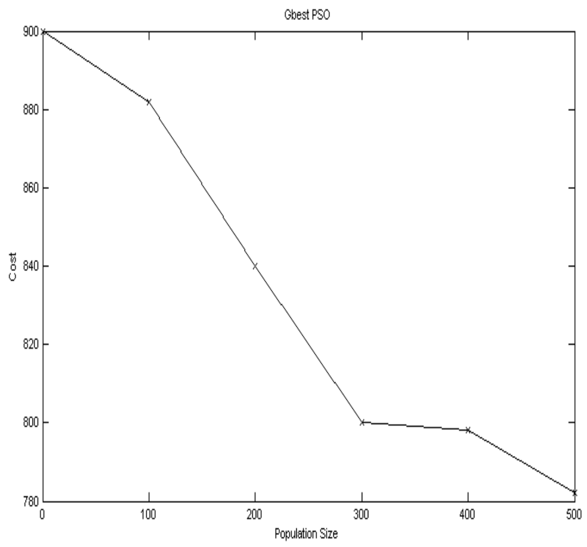


Fig. 6 Decreasing Cost in Gbest PSO with varying population size

Fig. 7 depicts the cost obtained for the lBest PSO. Initially the cost obtained is 914 and then reduces as we increase the population size. It then converges to the minimum cost of 818 for the population size of 500. The increase in cost is due to the fact that each particle gets the information only from its neighbors.

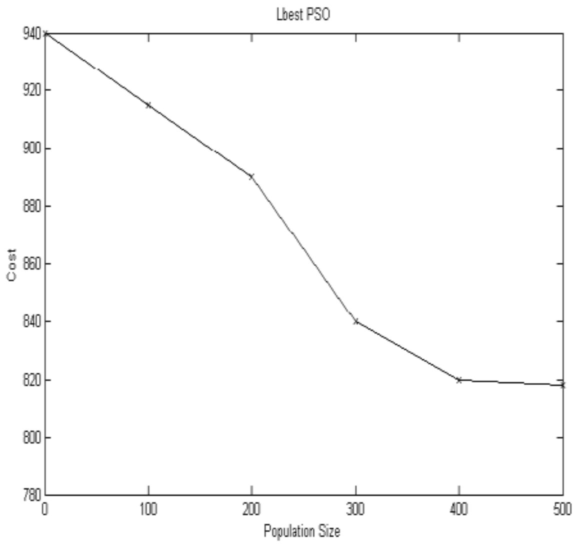


Fig. 7 Decreasing Cost in Lbest PSO with varying population size

The hybrid PSO managed to obtain an initial cost of 885 as shown in Fig. 8. The cost reduces as we increase the population size and hybrid PSO finally converges to the minimum cost of 783 for a population size of 500 and remains the same for any increase in the population size. The faster convergence is contributed by the hybridization with simulated annealing.

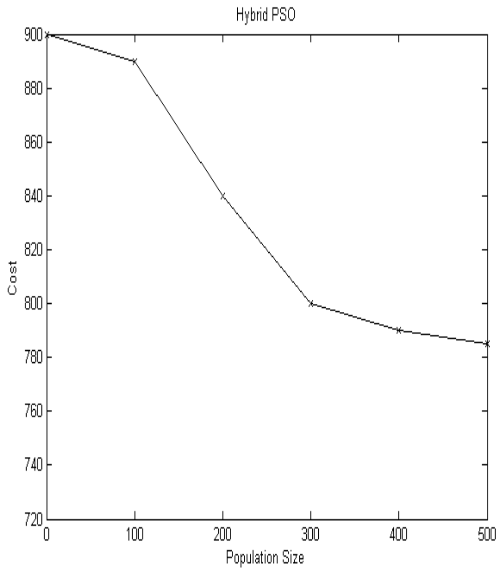


Fig. 8 Hybrid PSO with varying population size

The gBest topology compares each particle with every other particle, therefore the convergence is faster. The lBest topology compares each particle with its immediate neighborhood, thus resulting in a slower convergence as compared to gBest. On the other hand, in the hybrid PSO, the inclusion of simulated annealing at the end of iteration makes the search refined only towards the feasible solution which leads to faster convergence compared to the other two methods. For all three methods, increasing the population size results in a further minimization of the cost because the exploration is high.

4.2 Time Taken for Convergence

Next, the time taken for the convergence of the particles is compared. As expected, the hybrid PSO converged faster than all other versions. This is checked for varying population. As the population increases the gBest and lBest versions takes longer time for the convergence. However, the hybrid PSO performs better than the other two methods because the search in the negative direction is prevented. This can be inferred from Fig. 9.

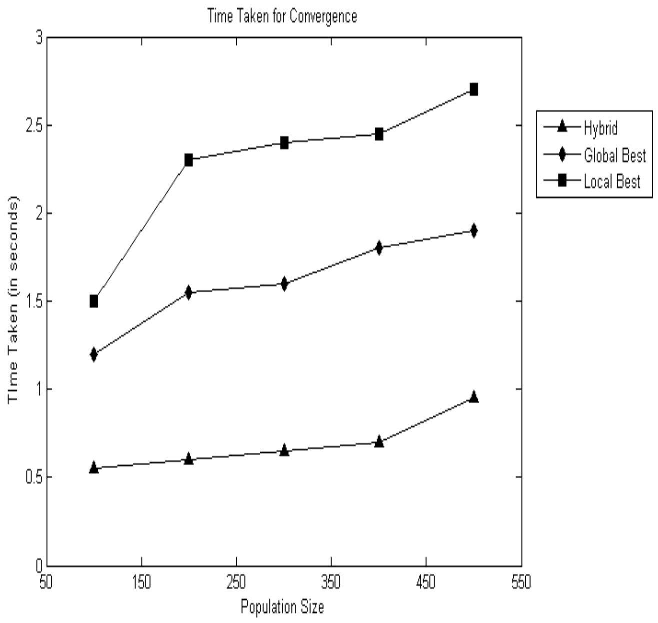


Fig. 9 Time Taken for convergence

5 PSO Enhancements

Although PSO has shown a very good performance in solving many test as well as real life optimization problems, it suffers from the problem of premature convergence like most of the stochastic search techniques, particularly in case of multimodal optimization problems. The *curse* of premature convergence greatly affects the performance of PSO and many times lead to a sub optimal solution [19]. Empirical studies have shown that the basic PSO has a tendency of premature convergence [20] and the main reason for this behavior is due to the loss of diversity in successive iterations. It has been observed that the presence of a suitable operator may help in improving the performance of PSO quite significantly. This section presents two enhancements done on the basic PSO; the first is on the use of diversity to guide the swarm and second is the efficient generation of population using different initialization schemes.

5.1 Diversity-Guided PSO

As an extension to the PSO algorithm and to avoid premature convergence, maintaining a certain degree of diversity is crucial. Diversity may be defined as the dispersion of potential candidate solutions in the search space. Interested readers may please refer to [21] for different formulae used for calculating diversity. One of the drawbacks of most of the population based search techniques is that they work on the principle of contracting the search domain towards the global optima. Due to this reason after a certain number of iterations all the points get accumulated to a region which may not even be a region of local optima, thereby giving suboptimal solutions [19]. Thus without a suitable diversity enhancing mechanism it is very difficult for an optimization algorithm to reach towards the true solution.

A diversity measure control, the attraction-repulsion PSO (ARPSO) is introduced in the PSO algorithm to control the swarm in phases of attraction and repulsion. The attraction phase is basically the same as the basic PSO algorithm. In the basic PSO algorithm particles tend to attract to each other due to the level of communication between particles, this is exactly what is required in the attraction phase so it stays the same. Therefore, a positive feedback leads to contraction of the swarm and lower diversity. In the repulsion phase we want particles to move away from those positions that are seen as best and explore new sections of the search space. This is done by reversing the formula used to update the velocity and making the particle move away from its personal best and the swarms' global best. Due to the clustering of the swarm in the attraction phase, diversity tends to be compromised and decrease. It is when the diversity has dropped below an acceptable level that we want the repulsion stage to kick in and offer more variety to where particles are searching. The negative feedback leads to an explosion of the swarm and higher diversity. We do not however want to constantly stay in this repulsion stage as instead of finding new global best and searching these new global bests further we may just keep moving away from these global bests. Therefore when the diversity hits a specified high we want to switch back to the attraction stage so as to explore the new found areas of interest further. The ARPSO uses diversity threshold values d_{low} and d_{high} to guide the movement of the swarm. The threshold values are predefined by the user. The main idea is to alternate between phases of repulsion when diversity drops below d_{low} to encourage further exploration of the search space and attraction when diversity exceeds d_{high} to encourage exploitation of the solutions. The diversity is measured based on Eq. 8:

$$diversity(S) = \frac{1}{|S| \cdot |L|} \sum_{i=1}^{|S|} \sqrt{\sum_{j=1}^{N} \left(p_{ij} - avg_j\right)^2} \tag{8}$$

where:

- S is the swarm and |S| is the swarmsize,
- L is the length of the longest diagonal in the search space,
- N is the dimensionality of the search space,
- pij is the j'th value of the i'th particle, and
- avgj is the j'th value of the swarms average point.

Threshold values: $d_{high} = 0.25$, $d_{low} = 5.0*10^{-6}$

In making the comparison between GA, PSO and ARPSO, the swarm / population size is fixed to 20 for all algorithms, and the generation number is varied. The GA used here can be found in the previous work [11]. The convergence rate with the number of generations for PSO, GA and ARPSO for benchmarks EWF and FIR [11] and is shown in Fig. 10 and Fig. 11 respectively. It is clear that, ARPSO produces a better solution quality as compared to that for GA and PSO. The ARPSO continues to search for new areas during its whole search-process. It continues to "pump" diversity into the swarm during optimization.

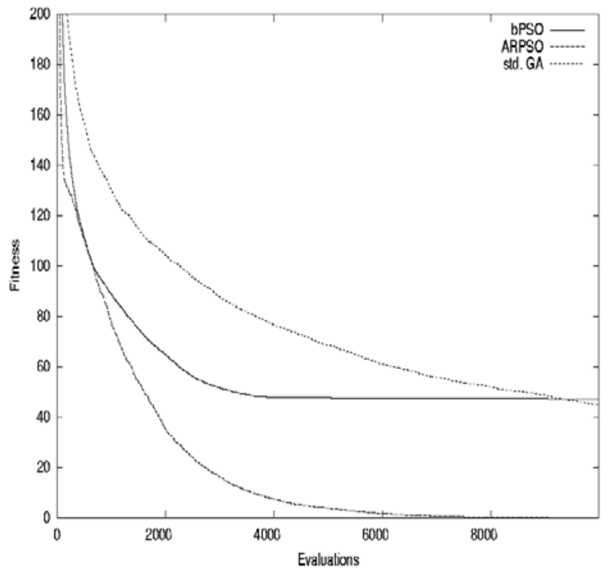


Fig. 10 The convergence rate with the number of generations for PSO, GA and ARPSO for EWF benchmark.

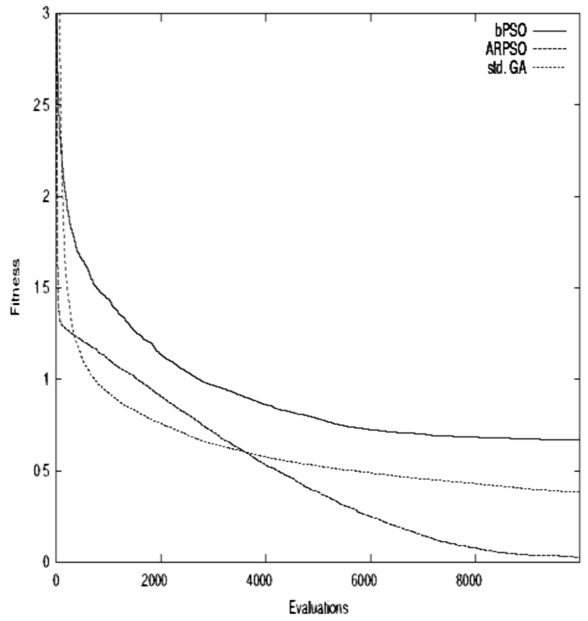


Fig. 11 The convergence rate with the number of generations for PSO, GA and ARPSO for FIR benchmark.

5.2 Effects of Swarm Size on ARPSO

As shown in the previous section, diversity is essential in PSO to continue finding better solutions. By introducing the idea of certain particles being competitive or reversing the concept of particles flowing towards the current best solution once diversity reaches a certain low, we can ensure that other areas of the solution space get looked at and clustering around these local best is reduced. This is where the concept of competitive PSO comes in [22]. The ARPSO introduced in the previous section is a form of competitive PSO that utilizes some sort of difference between groups of particles to ensure diversity is maintained.

Another factor in ensuring that diversity is maintained is to have an adequate sized swarm. The swarm should be of a necessary size to ensure that the optimum solution can be found whilst not being so large that it adversely affects the performance of the algorithm. This section aims to test two different functions with varying swarm sizes to see if a pattern emerges of what is the best swarm size for competitive PSO. For the experiment, two different benchmarks were used, EWF and FIR. Each benchmark was tested with a dimension of 20 and was evaluated 20,000 times. The swarm size is varied with each test, beginning with 20, then 50, then 80 and finally 100. After this we took larger jumps in the swarm size from 200 to 500 and finally to 800 and 1,000 to see whether or not exceptionally large swarms improved the result considerably.

The results from the EWF are compiled in Table 1. They show a clear improvement in the best result as the swarm increases in size and also a clear increase in the time taken for the 50 runs to complete as the swarm size becomes larger. If we break the table of results in two parts, the first part being from 20 to 100 and the second from 200 to 1,000, notice how the improvement found for each part is greatly different. Each halves swarm size increases by the same factor yet in the first half it is found that a 15% improvement in the result given at the expense of just a 0.7% increase in time spent obtaining the solution. In the second half of the table there is a similar improvement in the results obtained yet it is still less than that of the first half at only 14%. Yet this improvement is at the expense of a 19% increase in the time spent to reach those results. This shows that while a larger swarm size does improve the results we get, it begins to start costing too much in time spent for the function to complete once the size of the swarm reaches a certain point.

Table 1: Results from the EWF benchmark carried out over 50 runs with dimension 20 and 20,000 evaluations per run.

Swarm size	Best average result	Time (s)
20	141.368297	13.4
50	129.883767	13.3
80	123.778254	13.5
100	122.581813	13.5
200	119.946907	13.8
500	110.415802	14.9
800	105.406012	16.0
1000	104.351577	16.5

The results of the FIR are presented in Table 2. Again we see a clear trend of an improvement in the results shown as the swarm size increases. The time taken to get these results also increases with FIR as it did with EWF. Again if we break the table up into two halves as we did with the EWF it is noticed that the first half shows a much better return. From a swarm size of 20 to a swarm size of 100 there is a 7.7% improvement in the results obtained at an expense of 1.5% more time taken. The second half shows an 8% improvement at the expense of 20% more time taken. This again shows that eventually the size of the swarm becomes a negative factor on the time taken to produce a result.

In conclusion we can establish that increased swarm size means that there are more particles searching for solutions and thus the solutions provided improve as the amount of particles increases. This however leads to the function taking longer to finish a run and has a negative effect on the performance of the algorithm. Obviously various different problems will require different swarm sizes. From our results we can determine that once the size of the swarm goes over 100 there does appear to be a detrimental effect on the time taken to produce results.

Table 2: Results from the FIR benchmark carried out over 50 runs with dimension 20 and 20,000 evaluations per run.

Swarm size	Best average result	Time (s)
20	1.952897	12.8
50	1.838701	12.7
80	1.824146	12.9
100	1.813096	12.7
200	1.771412	12.8
500	1.684667	14.0
800	1.641820	15.4
1000	1.520966	15.4

5.3 Efficient Initialization in PSO

PSO (and other search techniques, which depend on the generation of random numbers) works very well for problems having a small search area (i.e. a search area having low dimension), but as the dimension of search space is increased, the performance deteriorates and many times converge prematurely giving a suboptimal result. This problem becomes more persistent in case of multimodal functions having several local and global optima. One of the reasons for the poor performance of a PSO may be attributed to the dispersion of initial population points in the search space i.e. to say, if the swarm population does not cover the search area efficiently, it may not be able to locate the potent solution points, thereby missing the global optimum [23]. This difficulty may be minimized to a great extent by selecting a well-organized distribution of random numbers.

This section analyses the behaviour of some variations of PSO where only the initial distribution of random numbers is changed. Initially in the algorithms the initial uniform distribution is replaced by other probability distributions like exponential, lognormal and Gaussian distributions. It is interesting to see that even a small change in the initial distribution produces a visible change in the numerical results. After that more specialized algorithms are designed which use low discrepancy sequences for the generation of random numbers. A brief description of the algorithms is given in the subsequent sections.

Different Probability Distributions like Exponential and Gaussian have already been used for the fine tuning of PSO parameters [24-25]. However, for initializing the swarm most of the approaches use uniformly distributed random numbers. Here, the possibility of having a different probability distribution (Gaussian, Exponential, Lognormal) for the generation of random number other than the uniform distribution is investigated. Empirical results showed that distributions other than uniform distribution are equally competent and in most of the cases are better than uniform distribution. The algorithms GPSO, EPSO and LNPSO use Gaussian, exponential and lognormal distributions respectively. The algorithms follow the steps of the basic PSO given in Section except for the fact that they use mentioned distributions in place of uniform distributions. The probability distributions for initializing the swarm using Gaussian, Exponential and Log-normal distributions are shown in Eqs 9-11 respectively:

Gaussian distribution:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \tag{9}$$

with mean 0 and standard deviation 1, i.e. N (0,1).

Exponential distribution:

$$f(x) = \frac{1}{2b}\exp(-|x-a|/b), \qquad -\infty \le x < \infty, \tag{10}$$

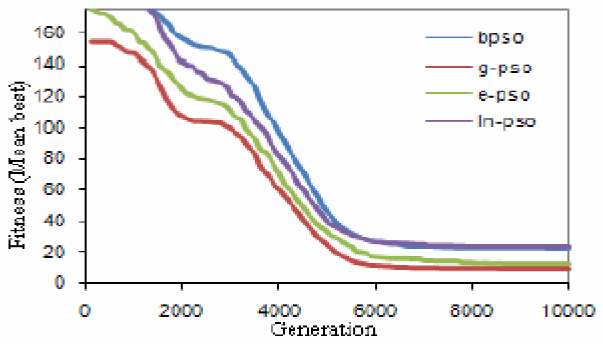
with a, b > 0.It is evident that one can control the variance by changing the parameters a and b.

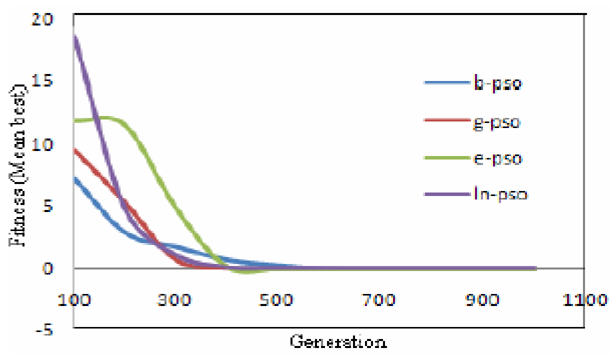
Log-normal distribution:

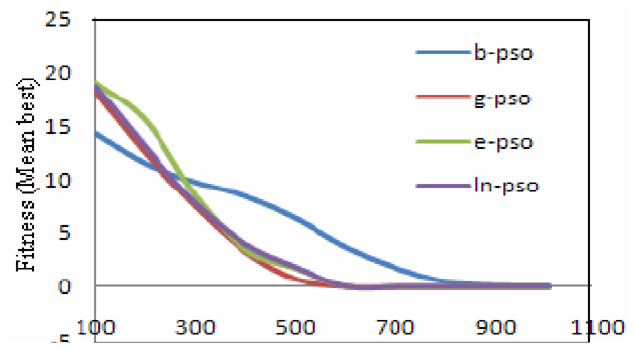
$$f(x) = \frac{e^{-(\ln x)^2/2}}{x\sqrt{2\pi}} \tag{11}$$

with mean 0 and standard deviation 1.

Fig. 12 gives the numerical results of PSO versions initialized with Gaussian, exponential and lognormal probability distributions. The tests were done on ten, three, two and five tasks respectively. From the numerical results it can be seen that the PSO using Gaussian mutation, GPSO, gave the best performance in comparison to other versions, followed by EPSO and LNPSO. For the first test case consisting of ten tasks, GPSO gave the best function value of approximately 10.00 which is much better than the values obtained by the other algorithms. For the second test case consisting of three tasks, all the algorithms gave more or less similar results. However GPSO and LNPSO gave a slightly better performance. For the third test consisting of two tasks, once again GMPSO outperformed the other algorithms. For the last test consisting of five tasks, both GMPSO and EPSO gave same result, which is better than the other two algorithms. In all the test cases the results were better than basic PSO using uniform distribution.







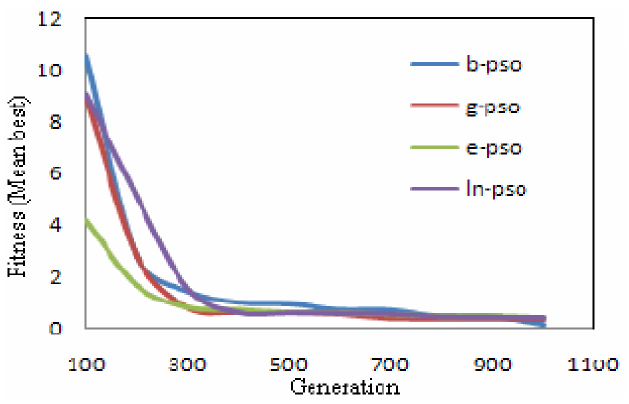


Fig. 12 Performance for BPSO, GPSO, EPSO and LNPSO for ten, three, two and five tasks respectively.

6 Conclusion

This paper tackles the problem of assignment of 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 have been formally defined but, unfortunately, most of them are NP-complete. Here, we have proposed a PSO/SA hybrid algorithm which finds a near-optimal task assignment with reasonable time. It is found that the hybrid PSO performed better than the lBest and the gBest PSO models.

Next, the ARPSO is introduced to the basic PSO algorithm for two main reasons: low diversity is needed for fine-tuning of a solution found in a given area and high diversity is needed to discover other areas with entirely different solutions. It is a good tradeoff between more frequently avoiding premature convergence and on the other hand relaxing the convergence rate a “bit”. From the experimental results, it is clear that the proposed algorithm produced better solutions compared to BPSO and GA.

Finally, the three algorithms namely GPSO, EPSO and LNPSO presented in this paper used different initialization schemes for generating the swarm population. These schemes include Gaussian, exponential and lognormal probability distribution to initialize the swarm. As expected, PSO algorithms initialized with quasi random sequences performed much better than the PSO initiated with the usual computer generated random numbers having uniform distribution. However the interesting part of the study is that PSO initiated with Gaussian, exponential and lognormal distribution improved its performance quite significantly.

References:

[1] Pablo, M., *NP Optimization Problems, Approximability and Evolutionary Computation: From Practice to Theory*, Ph.D. Dissertation, Universidade Estadual de Campinas, Brazil, 2001.

[2] Kennedy, J., and R. Eberhart, *Swarm Intelligence*, Academic Press, 1st ed., San Diego, CA. 2001.

[3] Chen D.J., Lee C.Y., Park C.H., Mendes P., Parallelizing simulated annealing algorithms based on high-performance computer, *J. Glob. Optim.*, Vol. *39*, 2007, pp. 261–289.

[4] Abdelmageed, E.A, and B. Earl Wells, A Heuristic model for task allocation in heterogeneous distributed computing systems, *The International Journal of Computers and Their Applications*, Vol. 6, No. 1, 1999, pp. 746-753.

[5] Annie, S., W. Shiyun Jin, Kuo-Chi Lin and Guy Schiavone, Incremental Genetic Algorithm Approach to Multiprocessor Scheduling, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 2, No. 5, 2004, pp. 135-142.

[6] Dar-Tzen Peng, G. S. Kang and F. Tarek Abdelzaher, Assignment and Scheduling Communicating Periodic Tasks in Distributed Real-Time Systems, *IEEE Transactions on Software Engineering*. Vol. 23, No. 12, 1997.

[7] Ruey-Maw, C., and H. Yueh-Ming, Multiprocessor Task Assignment with Fuzzy Hopfield Neural Network Clustering Techniques, *Journal of Neural Computing and Applications*, Vol.10, No.1, 2001.

[8] Virginia, M.L., Heuristic algorithms for task assignment in distributed systems, *IEEE Transactions on Computers*, Vol. 37, No. 11, 1998, pp. 1384– 1397.

[9] Ioan, C. T., The particle swarm optimization algorithm: convergence analysis and parameter selection, *Information Processing Letters*, Vol. 85, 2003, pp. 317–325.

[10] Batainah, S., and M. AI-Ibrahim, Load management in loosely coupled multiprocessor systems, *Journal of Dynamics and Control*, Vol.8, No.1, 1998, pp. 107-116.

[11] Choong, F., S. Phon-Amnuaisuk, M.Y. Alias and W.L. Pang, Adaptive Genetic Algorithm: An Essential Ingredient in High-level Synthesis, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, Vol. 5, No. 6, 2008, pp. 3837-3844.

[12] Graham, R., *Static, Multi-processor scheduling with Ant Colony Optimization and Local search*, Master of Science thesis , University of Edinburgh, 2003.

[13] Peng-Yeng, Y., Y. Shiuh-Sheng, W. Pei-Pei and W. Yi-Te, A hybrid particle swarm optimization algorithm for optimal task assignment in distributed systems, *Computer Standards & Interfaces,* Vol.28, 2006, pp. 441-450.

[14] Yskandar, H., and S. Khalil Hindi, Assignment of program modules to processors: A simulated annealing approach, *European Journal of Operational Research*, Vol. 1, No. 22, 2000, pp.509-513.

[15] Tzu-Chiang, C., C. Po-Yin, and H. Yueh-Ming, Multi-Processor Tasks with Resource and Timing Constraints Using Particle Swarm Optimization, *IJCSNS International Journal of Computer Science and Network Security*. Vol.6, No.4, 2006.

[16] Shi, Y., and R. Eberhart, Parameter Selection in Particle Swarm Optimization, Evolutionary Programming VII, *Proceedings of Evolutionary Programming*, 1998, pp. 591-600.

[17] Maurice, C., and J. Kennedy, The Particle Swarm—Explosion, Stability, and Convergence in a Multidimensional Complex Space, *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 1, 2002.

[18] Zhang, Y., R. Kamalian, A.M. Agogino and C.H. Séquin, Hierarchical MEMS Synthesis and Optimization, *Smart Structures and Materials, Smart Electronics, MEMS, BioMEMS, and Nanotechnology, Proceedings of SPIE*, Vol. 5763, 2005, pp. 96-106.

[19] Liu, H., A. Abraham and W. Zhang, A Fuzzy Adaptive Turbulent Particle Swarm Optimization, *International Journal of Innovative Computing and Applications*, Vol 1, No. 1, 2007, pp. 39–47.

[20] Grosan, C., A. Abraham and M. Nicoara, Search Optimization Using Hybrid Particle Sub-Swarms and Evolutionary Algorithms, *International Journal of Simulation Systems, Science & Technology*, Vol. 6, No. 10&11, 2005, pp. 60–79.

[21] Engelbrecht, A.P., *Fundamentals of Computational Swarm Intelligence*, John Wiley & Sons Ltd., Chichester, 2005.

[22] Blackwell, T., and J. Branke, Multi-swarm optimization in dynamic environments, *Applications of Evolutionary Computing*, Vol. 3, No. 5, 2008, pp. 45-54.

[23] Grosan, C., A. Abraham and M. Nicoara, Search Optimization Using Hybrid Particle Sub-Swarms and Evolutionary Algorithms, *International Journal of Simulation Systems, Science & Technology*, Vol. 6, No. 10&11, 2005, pp. 60–79.

[24] Krohling, R.A., A Novel Particle Swarm Optimization Algorithm, *In: Proc. of the 2004 IEEE Conference on Cybernetics and Intelligent Systems*, 2004, pp. 372–376.

[25] Krohling, R.A., and L.S. Coelho, PSO-E: Particle Swarm with Exponential Distribution, *In: IEEE Congress on Evolutionary Computation*, 2006, pp. 1428–1433.