9

Nature Inspired Meta-heuristics for Grid Scheduling: Single and Multi-objective Optimization Approaches

Ajith Abraham[1], Hongbo Liu[2,3], Crina Grosan[4], and Fatos Xhafa[5]

[1] Centre for Quantifiable Quality of Service in Communication Systems, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
ajith.abraham@ieee.org
http://www.softcomputing.net
[2] School of Computer Science and Engineering, Dalian Maritime University, 116026 Dalian, China
[3] Department of Computer, Dalian University of Technology, 116023 Dalian, China
lhb@dlut.edu.cn
[4] Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş Bolyai University, Kogalniceanu 1, Cluj-Napoca, 3400, Romania
cgrosan@cs.ubbcluj.ro
[5] Dept. de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
C/Jordi Girona 1-3, 08034 Barcelona, Spain
fatos@lsi.upc.edu

Summary. In this chapter, we introduce several nature inspired meta-heuristics for scheduling jobs on computational grids. Our approach is to dynamically generate an optimal schedule so as to complete the tasks in a minimum period of time as well as utilizing the resources in an efficient way. We evaluate the performance of Genetic Algorithm (GA), Simulated Annealing (SA), Ant Colony optimization (ACO) and Particle Swarm Optimization (PSO) Algorithm. Finally, the usage of Multi-objective Evolutionary Algorithm (MOEA) for two scheduling problems are also illustrated.

Keywords: Nature Inspired Meta-heuristics, Multi-objective Optimization, Job Scheduling, Grid Computing, Genetic Algorithms, Simulated Annealing, Ant Colony, Particle Swarm Optimization.

9.1 Introduction

A computational grid is a large scale, heterogeneous collection of autonomous systems, geographically distributed and interconnected by low latency and high bandwidth networks [1]. The sharing of computational jobs is a major application of grids. Grid resource management provides functionality for discovery and publishing of resources as well as scheduling, submission and monitoring of jobs. However, computing resources are geographically distributed under different ownerships each having their own access policy, cost and various constraints. Every

F. Xhafa, A. Abraham (Eds.): Meta. for Sched. in Distri. Comp. Envi., SCI 146, pp. 247–272, 2008.
springerlink.com

resource owners will have a unique way of managing and scheduling resources and the grid schedulers are to ensure that they do not conflict with resource owner's policies. In the worst-case situation, the resource owners might charge different prices to different grid users for their resource usage and it might vary from time to time. The job schedule problem is known to be NP-complete [2]. Recently several metaheuristics were introduced to minimize the average completion time of jobs through optimal job allocation on each grid node in application-level scheduling [3], [4]. Because of the intractable nature of the problem and its importance in grid computing, it is desirable to explore other avenues for developing good heuristic algorithms for the problem.

Particularly, with its sound exploration ability both global and local, some new search techniques, nature inspired meta-heuristics, has become the new focus of research. In this chapter, we introduce several nature inspired meta-heuristics for scheduling jobs on computational grids. The nature inspired meta-heuristics involved are Genetic Algorithm (GA), Simulated Annealing (SA), Ant Colony optimization (ACO) and Particle Swarm Optimization (PSO) Algorithm. The PSO approach for scheduling jobs on computational grids is based on fuzzy matrices to represent the position and velocity of the particles in PSO [5], in which a new mapping between the job scheduling problem and the particle is constructed [13]. The approach is to dynamically generate an optimal schedule so as to complete the tasks in a minimum period of time as well as utilizing the resources in an efficient way. We also illustrate the use of Multi-objective evolutionary algorithms for job scheduling [7].

The Chapter is organized as follows. Section 2 deals with some theoretical foundations related to job scheduling. Various nature inspired heuristics are introduced in Section 3. In Section 4, experiment results and discussions are provided. Finally, we conclude our work.

9.2 Scheduling Problem Formulation

In the grid environment, there is usually a general framework focusing on the interaction between grid resource broker, domain resource manager and the grid information server [8]. Usually it is easy for the grid to get information about the speed of the available grid nodes but quite complicated to know the computational processing time requirements from the user. To conceptualize the problem as an algorithm, we need to dynamically estimate the job lengths from user application specifications or historical data. For clarity purposes, some key terminologies are defined as follows:

- Grid Node (computing unit)
  Grid node is a set of computational resources with limited capacities. It may be a simple personal machine, a workstation, a super-computer, or a cluster of workstations in the grid environment. The computational capacity of the node depends on its number of CPUs, amount of memory, basic storage space and other specializations. In other words, each node has its own processing speed, which can be expressed in number of Cycles Per Unit Time (CPUT).

- Jobs and Operations
  A job is considered as a single set of multiple atomic operations/tasks. Each operation will be typically allocated to execute on one single node without preemption. It has input and output data, and processing requirements in order to complete its task. The operation has the processing length expressed in number of cycles.
- Schedule and Scheduling Problem
  A schedule is the mapping of the tasks to specific time intervals of Grid nodes. A scheduling problem is specified by a set of machines, a set of jobs/operations, optimality criteria, environmental specifications, and by other constraints.

To formulate the problem, we consider J_j ($j \in \{1, 2, \cdots, n\}$) independent user jobs on G_i ($i \in \{1, 2, \cdots, m\}$) heterogeneous grid nodes with an objective of minimizing the completion time and utilizing the nodes effectively. The speed of each node is expressed in number of CPUT, and the length of each job in number of cycles. Each job J_j has its processing requirement (cycles) and the node G_i has its calculating speed (cycles/second). Any job J_j has to be processed in the one of grid nodes G_i, until completion. Since all nodes at each stage are identical and preemptions are not allowed, to define a schedule it suffices to specify the completion time for all tasks comprising each job.

To formulate our objective, define $C_{i,j}$ ($i \in \{1, 2, \cdots, m\}$, $j \in \{1, 2, \cdots, n\}$) as the completion time that the grid node G_i finishes the job J_j, $\sum C_i$ represents the time that the grid node G_i finishes all the jobs scheduling to itself. Define $C_{max} = max\{\sum C_i\}$ as the makespan, and $\sum_{i=1}^{m}(\sum C_i)$ as the flowtime.

An optimal schedule will be the one that optimizes the flowtime and makespan. The conceptually obvious rule to minimize $\sum_{i=1}^{m}(\sum C_i)$ is to schedule Shortest Job on the Fastest Node (SJFN). The simplest rule to minimize C_{max} is to schedule the Longest Job on the Fastest Node (LJFN). Minimizing $\sum_{i=1}^{m}(\sum C_i)$ asks the average job finishes quickly, at the expense of the largest job taking a long time, whereas minimizing C_{max}, asks that no job takes too long, at the expense of most jobs taking a long time. Minimization of C_{max} will result in maximization of $\sum_{i=1}^{m}(\sum C_i)$.

9.3 Nature Inspired Meta-heuristics

Combinatorial optimizationproblems are important in many real life applications and recently, the area has attracted much research with the advances in nature inspired heuristics and multi-agent systems. For scheduling problems, the dramatic increase in the size of the search space and the need for real-time solutions motivated research ideas into solving scheduling problems using nature inspired heuristic techniques. In this Chapter, we included evolutionary algorithms, simulated annealing, ant colony optimization and particle swarm optimization algorithm. The generic pseudo-code for the algorithms is illustrated in Algorithm 9.1.

Algorithm 9.1. General Description for Nature Inspired Algorithm

01. Initialize the solution vectors randomly and other parameters.
02. Evaluate the candidate solution(s);
03. Repeat
04. Generate new candidate solutions following the nature or social behaviors;
05. Evaluate the candidate solution;
06. Until terminating criteria.

The termination criteria are usually one of the following:

- Maximum number of iterations: the optimization process is terminated after a fixed number of iterations, for example, 1000 iterations.
- Number of iterations without improvement: the optimization process is terminated after some fixed number of iterations without any improvement.
- Minimum objective function error: the error between the obtained objective function value and the best fitness value is less than a pre-fixed anticipated threshold.
- Cost threshold: allocated budget (computation time/cost) reached.
- Manual inspection: the process is executed by human-computer interactively.
- Combinations of the above.

9.3.1 Evolutionary Algorithms

In nature, evolution is mostly determined by natural selection, where individuals that are better are more likely to survive and propagate their genetic material. The encoding of genetic information (genome) is done in a way that admits asexual reproduction which results in offspring's that are genetically identical to the parent. Sexual reproduction allows some exchange and re-ordering of chromosomes, producing offspring that contain a combination of information from each parent. This is the recombination operation, which is often referred to as crossover because of the way strands of chromosomes crossover during the exchange. Diversity in the population is achieved by mutation. A typical evolutionary (genetic) algorithm procedure takes the following steps: A population of candidate solutions (for the optimization task to be solved) is initialized. New

Algorithm 9.2. Evolutionary Algorithm

01. Initialize the population randomly, and other parameters.
02. Evaluate the fitness of each individual in the population.
03. Repeat
04. Select best-ranking individuals to reproduce;
05. Breed new generation through crossover operator and give birth to offspring;
06. Breed new generation through mutation operator and give birth to offspring;
07. Evaluate the individual fitness of the offspring;
08. Replace worst ranked part of population with offspring;
09. Until terminating criteria.

solutions are created by applying genetic operators (mutation and/or crossover). The fitness (how good the solutions are) of the resulting solutions are evaluated and suitable selection strategy is then applied to determine which solutions will be maintained into the next generation. The procedure is then iterated [9]. A canonical version of the pseudo-code for the evolutionary algorithm is illustrated in Algorithm 9.2.

9.3.2 Evolutionary Multi-objective Optimization

Even though some real world problems can be reduced to a matter of single objective very often it is hard to define all the aspects in terms of a single objective. Defining multiple objectives often gives a better idea of the task. In single objective optimization, the search space is often well defined. As soon as there are several possibly contradicting objectives to be optimized simultaneously, there is no longer a single optimal solution but rather a whole set of possible solutions of equivalent quality. When we try to optimize several objectives at the same time the search space also becomes partially ordered. To obtain the optimal solution, there will be a set of optimal trade-offs between the conflicting objectives. A multiobjective optimization problem is defined by a function f which maps a set of constraint variables to a set of objective values.

A solution could be best, worst and also indifferent to other solutions (neither dominating or dominated) with respect to the objective values. Best solution means a solution not worst in any of the objectives and at least better in one objective than the other. An optimal solution is the solution that is not dominated by any other solution in the search space. Such an optimal solution is called Pareto optimal and the entire set of such optimal trade-offs solutions is called Pareto optimal set. As evident, in a real world situation a decision making (trade-off) process is required to obtain the optimal solution. Even though there are several ways to approach a multiobjective optimization problem, most work is concentrated on the approximation of the Pareto set.

Evolutionary algorithm is characterized by a population of solution candidates and the reproduction process enables to combine existing solutions to generate new solutions. Finally, natural selection determines which individuals of the current population participate in the new population. Multi-objective Evolutionary Algorithms (MOEA) can yield a whole set of potential solutions, which are all optimal in some sense. After the first pioneering work on multiobjective evolutionary optimization in the eighties [10], several different algorithms have been proposed and successfully applied to various problems. For comprehensive overviews and discussions, the reader is referred to [11].

9.3.3 Simulated Annealing

Simulated Annealing (SA) exploits an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure (the annealing process) and the search for a minimum in a more general system. SA's major advantage over other methods is an ability to avoid becoming trapped at local

minima [12]. The annealing schedule, i.e., the temperature-decreasing rate used in SA is an important factor, which affects SA's rate of convergence. The algorithm employs a random search, which not only accepts changes that decrease objective function "f", but also some changes that increase it. The latter are accepted with a probability $p = \exp\left(-\frac{\delta f}{T}\right)$, where δf is the increase in objective function, and "f" and T are control parameters. Several SAs have been developed with annealing schedule inversely linear in time (Fast SA), exponential function of time (Very Fast SA) etc. We explain a SA algorithm [13], which is exponentially faster than Very Fast SA whose annealing schedule is given by $T(k) = \frac{T_0}{\exp(e^k)}$, where $T_0$is the initial temperature, $T(k)$ is the temperature we wish to approach to zero for $k = 1, 2, \ldots$. If the generation function of the simulated annealing algorithm is represented as:

$$g_k(Z) = \prod_{i=1}^{D} g_k(z_i) = \prod_{i=1}^{D} \frac{1}{2(|z_i| + \frac{1}{\ln(1/T_i(k))})\ \ln\left(1 + \ln(1/T_i(k))\right)} \tag{9.1}$$

where $T_i(k)$is the temperature in dimension i at time k. The generation probability will be represented by

$$G_k(Z) = \int_{-1}^{z_1} \int_{-1}^{z_2} ..... \int_{-1}^{z_D} g_k(Z) dz_1 dz_2 .... dz_D = \prod_{i=1}^{D} G_{ki}(z_i) \tag{9.2}$$

where $G_{ki}(z_i) = \frac{1}{2} + \frac{sgn(z_i)\ln(1+|z_i|\ln(1/T_i(k)))}{2\ln(1+\ln(1/T_i(k)))}$

It is straightforward to prove that an annealing schedule for

$$T_i(k) = T_{0i} \exp(-\exp(b_i k^{1/D})) \tag{9.3}$$

A global minimum, statistically, can be obtained. That is,

$$\sum_{k=k_o}^{\infty} g_k = \infty \tag{9.4}$$

Algorithm 9.3. Simulated Annealing

01. Set initial temperature T_0, and other parameters.
02. Initialize the solution vectors randomly.
03. Repeat
04. Counter = 0;
05. Repeat
06. Evaluate the candidate solution;
07. Generate a neighbor and evaluate the cost of the neighbor solution;
08. Accept or reject the neighbor with a probability p;
09. Counter++;
10. Until (Counter = Number of Iterations at T_i);
11. $T_{i+1} = c * T_i$ (temperature reduction);
12. Until terminating criteria.

where $b_i > 0$ is a constant parameter and k_0 is a sufficiently large constant to satisfy Eq.(9.4), if the generation function in Eq.(9.1) is adopted. The pseudo-code for simulated annealing is illustrated in Algorithm 9.3.

9.3.4 Ant Colony Optimization

In nature, ants usually wander randomly, and upon finding food return to their nest while laying down pheromone trails. If other ants find such a path (pheromone trail), they are likely not to keep traveling at random, but to instead follow the trail, returning and reinforcing it if they eventually find food. However, as time passes, the pheromone starts to evaporate. The more time it takes for an ant to travel down the path and back again, the more time the pheromone has to evaporate (and the path to become less prominent). A shorter path, in comparison will be visited by more ants (can be described as a loop of positive feedback) and thus the pheromone density remains high for a longer time.

ACO is implemented as a team of intelligent agents which simulate the ants behavior, walking around the graph representing the problem to solve using mechanisms of cooperation and adaptation. ACO algorithm requires to define the following [14], [15]:

- The problem needs to be represented appropriately, which would allow the ants to incrementally update the solutions through the use of a probabilistic transition rules, based on the amount of pheromone in the trail and other problem specific knowledge. It is also important to enforce a strategy to construct only valid solutions corresponding to the problem definition.
- A problem-dependent heuristic function η that measures the quality of components that can be added to the current partial solution.
- A rule set for pheromone updating, which specifies how to modify the pheromone value τ.
- A probabilistic transition rule based on the value of the heuristic function η and the pheromone value τ that is used to iteratively construct a solution.

ACO was first introduced using the Traveling Salesman Problem (TSP). Starting from its start node, an ant iteratively moves from one node to another. When being at a node, an ant chooses to go to a unvisited node at time t with a probability given by

$$p_{i,j}^k(t) = \frac{[\tau_{i,j}(t)]^\alpha [\eta_{i,j}(t)]^\beta}{\sum_{l \in N_i^k} [\tau_{i,j}(t)]^\alpha [\eta_{i,j}(t)]^\beta} \qquad j \in N_i^k \tag{9.5}$$

where N_i^k is the feasible neighborhood of the ant_k, that is, the set of cities which ant_k has not yet visited; $\tau_{i,j}(t)$ is the pheromone value on the edge (i,j) at the time t, α is the weight of pheromone; $\eta_{i,j}(t)$ is a priori available heuristic information on the edge (i,j) at the time t, β is the weight of heuristic information. Two parameters α and β determine the relative influence of pheromone trail and heuristic information. $\tau_{i,j}(t)$ is determined by

$$\tau_{i,j}(t) = \rho\tau_{i,j}(t-1) + \sum_{k=1}^{n} \Delta\tau_{i,j}^{k}(t) \qquad \forall(i,j) \tag{9.6}$$

$$\Delta\tau_{i,j}^{k}(t) = \begin{cases} \frac{Q}{L_k(t)} & \text{if the edge } (i,j) \text{ chosen by the } ant_k \\ 0 & \text{otherwise} \end{cases} \tag{9.7}$$

where ρ is the pheromone trail evaporation rate ($0 < \rho < 1$), n is the number of ants, Q is a constant for pheromone updating.

Reader is advised to consult [16], [17], [15] for more technical details and other applications of ACO. A generalized version of the pseudo-code for the ACO algorithm with reference to the TSP is illustrated in Algorithm 9.4.

Algorithm 9.4. Ant Colony Optimization Algorithm

```
01. Initialize the number of ants n, and other parameters.
02. Repeat
03.   t++;
04.   For k= 1 to n
05.       ant_k is positioned on a starting node;
06.       For m= 2 to problem_size
07.           Choose the state to move into
07.               according to the probabilistic transition rules;
08.           Append the chosen move into tabu_k(t) for the ant_k;
09.       Next m
10.       Compute the length L_k(t) of the tour T_k(t) chosen by the ant_k;
11.       Compute Δτ_{i,j}(t) for every edge (i,j) in T_k(t) according to Eq.(9.7);
12.   Next k
13.   Update the trail pheromone intensity for every edge (i,j) according to Eq.(9.6);
14.   Compare and update the best solution;
15. Until terminating criteria.
```

9.3.5 Particle Swarm Optimization

Particle swarm algorithm is inspired by social behavior patterns of organisms that live and interact within large groups. In particular, it incorporates swarming behaviors observed in flocks of birds, schools of fish, or swarms of bees, and even human social behavior, from which the Swarm Intelligence (SI) paradigm has emerged [11], [12]. It could be implemented and applied easily to solve various function optimization problems, or the problems that can be transformed to function optimization problems.

As an algorithm, its main strength is its fast convergence, which compares favorably with many global optimization algorithms [9], [12]. The canonical PSO model consists of a swarm of particles, which are initialized with a population of random candidate solutions. They move iteratively through the d-dimension problem space to search the new solutions, where the fitness, f, can be calculated as the certain qualities measure.

Each particle has a position represented by a position-vector $\mathbf{x}_i$ (i is the index of the particle), and a velocity represented by a velocity-vector $\mathbf{v}_i$. Each particle remembers its own best position so far in a vector $\mathbf{x}_i^{\#}$, and its j-th dimensional value is $x_{ij}^{\#}$. The best position-vector among the swarm so far is then stored in a vector $\mathbf{x}^*$, and its j-th dimensional value is x_j^*. During the iteration time t, the update of the velocity from the previous velocity to the new velocity is determined by Eq.(9.8). The new position is then determined by the sum of the previous position and the new velocity by Eq.(9.9).

$$v_{ij}(t+1) = wv_{ij}(t) + c_1 r_1(x_{ij}^{\#}(t) - x_{ij}(t)) + c_2 r_2(x_j^*(t) - x_{ij}(t)). \tag{9.8}$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1). \tag{9.9}$$

where w is called as the inertia factor, r_1 and r_2 are the random numbers, which are used to maintain the diversity of the population, and are uniformly distributed in the interval [0,1] for the j-th dimension of the i-th particle. c_1 is a positive constant, called as coefficient of the self-recognition component, c_2 is a positive constant, called as coefficient of the social component.

From Eq.(9.8), a particle decides where to move next, considering its own experience, which is the memory of its best past position, and the experience of its most successful particle in the swarm. In the particle swarm model, the particle searches the solutions in the problem space with a range $[-s, s]$ (If the range is not symmetrical, it can be translated to the corresponding symmetrical range.) In order to guide the particles effectively in the search space, the maximum moving distance during one iteration must be clamped in between the maximum velocity $[-v_{max}, v_{max}]$ given in Eq.(9.10):

$$v_{ij} = sign(v_{ij}) min(|v_{ij}|, v_{max}). \tag{9.10}$$

$$x_{i,j} = sign(x_{i,j}) min(|x_{i,j}|, x_{max}). \tag{9.11}$$

The value of v_{max} is $p \times s$, with $0.1 \leq p \leq 1.0$ and is usually chosen to be s, i.e. $p = 1$. The pseudo-code for particle swarm optimization algorithm is illustrated in Algorithm 9.5.

The role of inertia weight w, in Eq.(9.8), is considered critical for the convergence behavior of PSO. The inertia weight is employed to control the impact of the previous history of velocities on the current one. Accordingly, the parameter w regulates the trade-off between the global (wide-ranging) and local (nearby) exploration abilities of the swarm. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine-tuning the current search area. A suitable value for the inertia weight w usually provides balance between global and local exploration abilities and consequently results in a reduction of the number of iterations required to locate the optimum solution. Initially, the inertia weight is set as a constant. However, some experiment results indicates that it is better to initially set the inertia to a large value, in order to promote global exploration of the search space, and gradually decrease it to get more refined solutions [20], [21]. Thus, an

Algorithm 9.5. Particle Swarm Optimization Algorithm

01. Initialize the size of the particle swarm n, and other parameters.
02. Initialize the positions and the velocities for all the particles randomly.
03. Repeat
04. $t++$;
05. Calculate the fitness value of each particle;
06. $\mathbf{x}^* = argmin_{i=1}^{n}(f(\mathbf{x}^*(t-1)), f(\mathbf{x}_1(t)), f(\mathbf{x}_2(t)), \cdots, f(\mathbf{x}_i(t)), \cdots, f(\mathbf{x}_n(t)))$;
07. For i= 1 to n
08. $\mathbf{x}_i^{\#}(t) = argmin_{i=1}^{n}(f(\mathbf{x}_i^{\#}(t-1)), f(\mathbf{x}_i(t))$;
09. For $j = 1$ to $Dimension$
10. Update the j-th dimension value of $\mathbf{x}_i$ and $\mathbf{v}_i$
10. according to Eqs.(9.8), (9.9), (9.10), (9.11);
12. Next j
13. Next i
14. Until terminating criteria.

initial value around 1.2 and gradually reducing towards 0 can be considered as a good choice for w. A better method is to use some adaptive approaches (example: fuzzy controller), in which the parameters can be adaptively fine tuned according to the problem under consideration [22], [16].

The parameters c_1 and c_2, in Eq.(9.8), are not critical for the convergence of PSO. However, proper fine-tuning may result in faster convergence and alleviation of local minima. As default values, usually, $c_1 = c_2 = 2$ are used, but some experiment results indicate that $c_1 = c_2 = 1.49$ might provide even better results. Recent work reports that it might be even better to choose a larger cognitive parameter, c_1, than a social parameter, c_2, but with $c_1 + c_2 \leq 4$ [23].

The particle swarm algorithm can be described generally as a population of vectors whose trajectories oscillate around a region which is defined by each individual's previous best success and the success of some other particle. Various methods have been used to identify some other particle to influence the individual. Eberhart and Kennedy called the two basic methods as "gbest model" and "lbest model" [11]. In the lbest model, particles have information only of their own and their nearest array neighbors' best (lbest), rather than that of the entire group.

In the gbest model, the trajectory for each particle's search is influenced by the best point found by any member of the entire population. The best particle acts as an attractor, pulling all the particles towards it. Eventually all particles will converge to this position. The lbest model allows each individual to be influenced by some smaller number of adjacent members of the population array. The particles selected to be in one subset of the swarm have no direct relationship to the other particles in the other neighborhood.

Typically lbest neighborhoods comprise exactly two neighbors. When the number of neighbors increases to all but itself in the lbest model, the case is equivalent to the gbest model. Some experiment results testified that gbest model converges quickly on problem solutions but has a weakness for becoming trapped

in local optima, while lbest model converges slowly on problem solutions but is able to "flow around" local optima, as the individuals explore different regions. The gbest model has faster convergence. But very often for multi-modal problems involving high dimensions it tends to suffer from premature convergence.

9.3.6 A Fuzzy Scheme Based on Particle Swarm Optimization

In this section, we design a fuzzy scheme based on discrete particle swarm optimization to solve the job scheduling problem on computational grids. The vectors to fuzzy matrices are extended to represent the position and velocity of the particles for computational grid job scheduling.

Suppose $G = \{G_1, G_2, \cdots, G_m\}$, $J = \{J_1, J_2, \cdots, J_n\}$, then the fuzzy scheduling relation from G to J can be expressed as follows:

$$S = \begin{bmatrix} s_{11} & s_{12} & \cdots & s_{1n} \\ s_{21} & s_{22} & \cdots & s_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ s_{m1} & s_{m2} & \cdots & s_{mn} \end{bmatrix}$$

Here s_{ij} represents the degree of membership of the i-th element G_i in domain G and the j-th element J_j in domain J to relation S. The fuzzy relation S between G and J has the following meaning: for each element in the matrix S, the element

$$s_{ij} = \mu_R(G_i, J_j), i \in \{1, 2, \cdots, m\}, j \in \{1, 2, \cdots, n\}. \tag{9.12}$$

μ_R is the membership function, the value of s_{ij} means the degree of membership that the grid node G_j would process the job J_i in the feasible schedule solution. In the grid job scheduling problem, the elements of the solution must satisfy the following conditions:

$$s_{ij} \in [0, 1], i \in \{1, 2, \cdots, m\}, j \in \{1, 2, \cdots, n\}. \tag{9.13}$$

$$\sum_{i=1}^{m} s_{ij} = 1, i \in \{1, 2, \cdots, m\}, j \in \{1, 2, \cdots, n\}. \tag{9.14}$$

For applying PSO successfully, one of the key issues is finding how to map the the problem solution into the PSO particle, which directly affects its feasibility and performance. We assume that the jobs and grid nodes are arranged in an ascending order according to the job lengths and the node processing speeds. The information related job lengths may be derived from historical data, some kind of strategy defined by the user or through load profiling.

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}$$

Accordingly, the elements of the matrix X must satisfy the following conditions:

$$x_{ij} \in [0,1], i \in \{1,2,\cdots,m\}, j \in \{1,2,\cdots,n\}. \tag{9.15}$$

$$\sum_{i=1}^{m} x_{ij} = 1, i \in \{1,2,\cdots,m\}, j \in \{1,2,\cdots,n\}. \tag{9.16}$$

We define similarly the velocity of the particle as:

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{bmatrix}$$

The symbol "$\otimes$" is used to denote the modified multiplication. Let α be a real number, $\alpha \otimes V$ or $\alpha \otimes X$ means all the elements in the matrix V or X are multiplied by α. The symbol "$\oplus$" and symbol "$\ominus$" denote the addition and subtraction between matrices respectively. Suppose A and B are two matrices which denote position or velocity, then $A \oplus B$ and $A \ominus B$ are regular addition and subtraction operation between matrices.

Then we get the equations (9.8) and (9.9) for updating the positions and velocities of the particles in the fuzzy discrete PSO:

$$V(t+1) = w \otimes V(t) \oplus (c_1 * r_1) \otimes X^{\#}(t) \ominus X(t)) \oplus (c_2 * r_2) \otimes (X^*(t) \ominus X(t)). \tag{9.17}$$

$$X(t+1) = X(t) \oplus V(t+1)). \tag{9.18}$$

The position matrix may violate the constraints (9.15) and (9.16) after some iterations, so it is necessary to normalize the position matrix. First we make all the negative elements in the matrix become zero. If all elements in a column of the matrix are zero, they need be re-evaluated using a series of random numbers with the interval [0,1]. Then the matrix undergoes the following transformation without violating the constraints:

$$Xnormal = \begin{bmatrix} x_{11}/\sum_{i=1}^{m} x_{i1} & x_{12}/\sum_{i=1}^{m} x_{i2} & \cdots & x_{1n}/\sum_{i=1}^{m} x_{in} \\ x_{21}/\sum_{i=1}^{m} x_{i1} & x_{22}/\sum_{i=1}^{m} x_{i2} & \cdots & x_{2n}/\sum_{i=1}^{m} x_{in} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1}/\sum_{i=1}^{m} x_{i1} & x_{m2}/\sum_{i=1}^{m} x_{i2} & \cdots & x_{mn}/\sum_{i=1}^{m} x_{in} \end{bmatrix}$$

Since the position matrix indicates the potential scheduling solution, we should "decode" the fuzzy matrix and get the feasible solution. We use a flag array to record whether we have selected the columns of the matrix and a scheduling array to record the scheduling solution. First all the columns are not selected, then for each columns of the matrix, we choose the element which has the max value, then mark the column of the max element "selected", and the column number are recorded to the

scheduling array. After all the columns have been processed, we get the scheduling solution from the scheduling array and the makespan of the scheduling solution.

To optimize the makespan and flowtime we propose to swap the usage of LJFN and SJFN heuristic alternatively every time the new jobs are allocated to the grid nodes. If the number of jobs is less than the number of grid nodes, we propose to allocate the jobs based on a First-Come-First-Serve basis and LJFN heuristic (if possible). In a grid environment, a scheduler might have to make a multi-criteria decision analysis (access policy, access cost, resource requirements, processing speed, etc.) for selecting an optimal solution. To formulate the algorithm, we propose the following job lists and grid node lists. $JList_1$ and $GList_1$ are to be dynamically updated through load profiling, grid node health status, and forecasted load status, etc. along with grid information services. The entire job and the grid node lists are to be arranged in the ascending order of the job lengths and processing speeds/access-cost (based on multi-criteria decision analysis). Frequency of updating the lists will very much depend on the grid condition, availability of grid nodes and jobs.

- $JList_1$ = Job list maintaining the list of all the jobs to be processed.
- $JList_2$ = Job list maintaining only the list of jobs being scheduled.
- $JList_3$ = Job list maintaining only the list of jobs already allocated ($JList_3 = JList_1 - JList_2$).
- $GList_1$ = List of available grid nodes (including time frame).
- $GList_2$ = List of grid nodes already allocated to jobs.
- $GList_3$ = List of free grid nodes ($GList_3 = GList_1 - GList_2$).

A scheme based on fuzzy discrete PSO for job scheduling is depicted in Algorithm 9.6.

9.4 Experimental Illustrations

The scheduling problem is to determine both an assignment and a sequence of the operations on all machines that minimize some criteria. The following optimality criteria are to be minimized:

1. the maximum completion time (makespan): C_{max};
2. the sum of the completion times (flowtime): C_{sum}.

The weighted aggregation is the most common approach to the problems. According to this approach, the objectives, $F_1 = min\{C_{max}\}$ and $F_2 = min\{C_{sum}\}$, are aggregated as a weighted combination:

$$F = w_1 min\{F_1\} + w_2 min\{F_2\} \tag{9.19}$$

where w_1 and w_2 are non-negative weights, and $w_1 + w_2 = 1$. These weights can be either fixed or adapt dynamically during the optimization. The fixed weights, $w_1 = w_2 = 0.5$, are used in this article. In fact, the dynamic weighted aggregation mainly takes C_{max} into account [25] because C_{sum} is commonly much larger

Algorithm 9.6. A scheduling scheme based on fuzzy discrete PSO

0 If the grid is active and ($JList_1 = 0$) and no new jobs have been submitted, wait for new jobs to be submitted. Otherwise, update $GList_1$ and $JList_1$.
1 If ($GList_1 = 0$), wait until grid nodes are available. If $JList_1 > 0$, update $JList_2$. If $JList_2 < GList_1$ allocate the jobs on a first-come-first-serve basis and if possible allocate the longest job on the fastest grid node according to the LJFN heuristic. If $JList_1 > GList_1$, job allocation is to be made by following the fuzzy discrete PSO algorithm detailed below. Take jobs and available grid nodes from $JList_2$ and $GList_3$. If $m * n$ (m is the number of the grid nodes, n is the number of the jobs) is larger than the dimension threshold D_T, the jobs and the grid nodes are grouped into the fuzzy discrete PSO algorithm loop, and the single node flowtime is accumulated. The LJFN-SJFN heuristic is applied alternatively after a batch of jobs and nodes are allocated.
2 At $t = 0$, represent the jobs and the nodes using fuzzy matrix.
3 Begin fuzzy discrete PSO Loop
3.0 Initialize the size of the particle swarm n and other parameters.
3.1 Initialize a random position matrix and a random velocity matrix for each particle, and then normalize the matrices.
3.2 Repeat
3.2.0 $t++$;
3.2.1 Defuzzify the position, and calculate the makespan and total flowtime for each particle (the feasible solution);
3.2.2 $X^* = argmin_{i=1}^{n}(f(X^*(t-1)), f(X_1(t)), f(X_2(t)), \cdots, f(X_i(t)), \cdots, f(X_n(t)))$;
3.2.3 For each particle, $X_i^{\#}(t) = argmin_{i=1}^{n}(f(X_i^{\#}(t-1)), f(X_i(t))$
3.2.4 For each particle, update each element in its position matrix and its velocity matrix according to equations (9.17, 9.11, 9.18 and 9.10);
3.2.5 Normalize the position matrix for each particle;
3.3 Until terminating criteria.
4 End of the fuzzy discrete PSO Loop.
5 Check the feasibility of the generated schedule with respect to grid node availability and user specified requirements. Then allocate the jobs to the grid nodes and update $JList_2$, $JList_3$, $GList_2$ and $GList_3$. Un-allocated jobs (infeasible schedules or grid node non-availability) shall be transferred to $JList_1$ for re-scheduling or dealt with separately.
6 Repeat steps 0-5 as long as the grid is active.

than C_{max} and the solution has a large weight on C_{sum} during minimization of the objective. Alternatively, the weights can be changed gradually according to the Eqs. (9.20) and (9.21). The changes in the dynamic weights ($R = 200$) are illustrated in Fig. 9.1.

$$w_1(t) = |sin(2\pi t/R)| \tag{9.20}$$

$$w_2(t) = 1 - w_1(t) \tag{9.21}$$

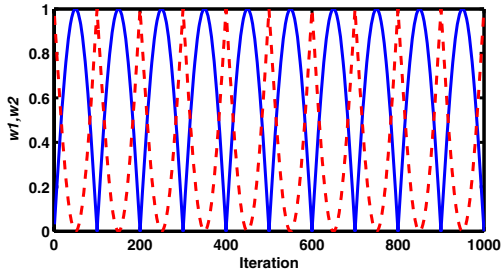

Fig. 9.1. Dynamic weight adaptation

9.4.1 Scheduling Using Fuzzy Particle Swarm Optimization Algorithm

Since the position matrix indicates the potential scheduling solution, we choose the element which has the max value, then tag it as "1", and other numbers in the column are set as "0" in the scheduling array. After all the columns have been processed, we get the scheduling solution from the scheduling array and the makespan (solution). In the experiments, genetic algorithm and simulated annealing were used to compare the performance with PSO. Specific parameter settings of all the considered algorithms are described in Table 9.1.

Each experiment (for each algorithm) was repeated 10 times with different random seeds. Each trial had a fixed number of $50 * m * n$ iterations (m is the number of the grid nodes, n is the number of the jobs). The makespan values of the best solutions throughout the optimization run were recorded and the averages and the standard deviations were calculated from the 10 different trials. In a grid environment, the main emphasis is to generate the schedules as fast as possible. So the completion time for 10 trials were used as one of the criteria to improve their performance.

To illustrate, we start with a small scale job scheduling problem involving 3 nodes and 13 jobs represented as (3, 13). The node speeds are 4, 3, 2 CPUT, and the job lengths of 13 jobs are 6, 12, 16, 20, 24, 28, 30, 36, 40, 42, 48, 52, 60 cycles, respectively.

Fig. 9.2 illustrates the performance of GA, SA and PSO algorithms. The empirical results (makespan) for 10 GA runs were {47, 46, 47, 47.3333, 46, 47, 47, 47, 47.3333, 49}, with an average value of 47.1167. The results of 10 SA runs were {46.5, 46.5, 46, 46,46, 46.6667, 47, 47.3333, 47, 47}with an average value of 46.6. The results of 10 PSO runs were {46, 46, 46, 46, 46.5, 46.5, 46.5, 46, 46.5, 46.6667}, with an average value of 46.2667. The optimal result is supposed to be 46. While GA provided the best results twice, SA and PSO provided the best results three and five times respectively. Table 9.2 depicts one of the best job scheduling results for (3,13), in which "1" means the job is scheduled to the respective grid node.

Table 9.1. Parameter settings for the algorithms

Algorithm	Parameter name	Parameter value
GA	Size of the population	20
	Probability of crossover	0.8
	Probability of mutation	0.02
	Scale for mutations	0.1
SA	Number operations before temperature adjustment	20
	Number of cycles	10
	Temperature reduction factor	0.85
	Vector for control step of length adjustment	2
	Initial temperature	50
PSO	Swarm size	20
	Self-recognition coefficient c_1	1.49
	Social coefficient c_2	1.49
	Inertia weight w	$0.9 \rightarrow 0.1$

Table 9.2. An optimal schedule for (3,13)

Grid Node	Job												
	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}	J_{11}	J_{12}	J_{13}
G_1	0	0	1	0	0	0	1	1	0	1	0	0	1
G_2	1	0	0	1	1	0	0	0	1	0	1	0	0
G_3	0	1	0	0	0	1	0	0	0	0	0	1	0

Table 9.3. Performance comparison between GA, PSO and SA

Algorithm	Item	Instance			
		(3,13)	(5,100)	(8,60)	(10,50)
GA	Average makespan	47.1167	85.7431	42.9270	38.0428
	Standard Deviation	±0.7700	±0.6217	±0.4150	±0.6613
	Time	302.9210	2415.9	2263.0	2628.1
SA	Average makespan	46.6000	90.7338	55.4594	41.7889
	Standard Deviation	±0.4856	±6.3833	±2.0605	±8.0773
	Time	332.5000	6567.8	6094.9	6926.4
PSO	Average makespan	46.2667	84.0544	41.9489	37.6668
	Standard Deviation	±0.2854	±0.5030	±0.6944	±0.6068
	Time	106.2030	1485.6	1521.0	1585.7

Table 9.4. Run time performance comparison for large dimension problems

(G,J)	PSO	GA
(60,100)	1721.1	1880.6
100,1000)	3970.80	5249.80

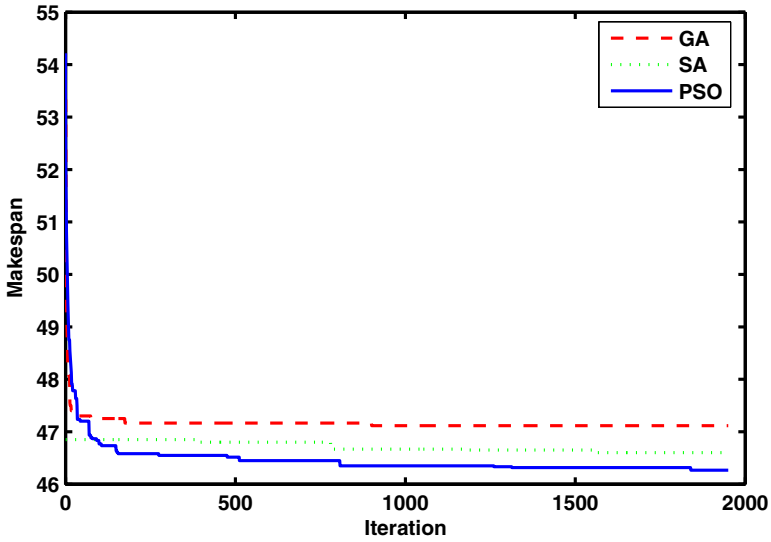

Fig. 9.2. Performance for job scheduling (3,13)

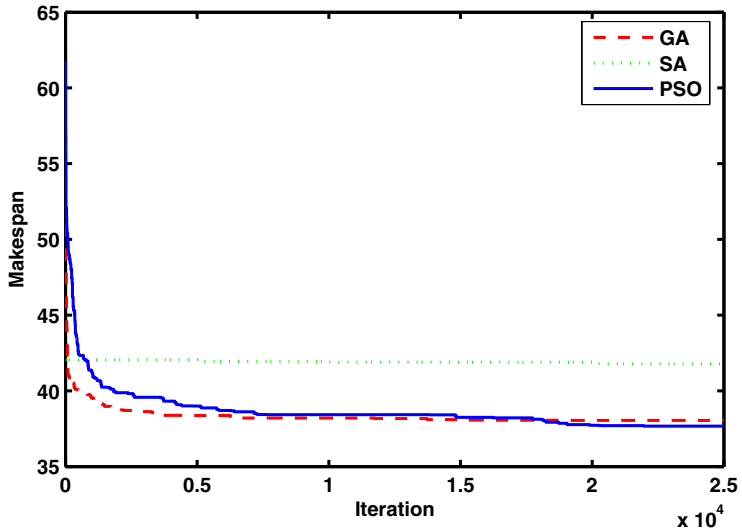

Fig. 9.3. Performance for job scheduling (5,100)

Further, we considered the three algorithms for other three (G, J) pairs, i.e. (5,100), (8,60) and (10,50). All the jobs and the nodes were submitted at one time. Figs. 9.2, 9.3 illustrate the performance for GA, SA and PSO algorithms during the search process for (3, 13), (5,100) respectively. The average makespan

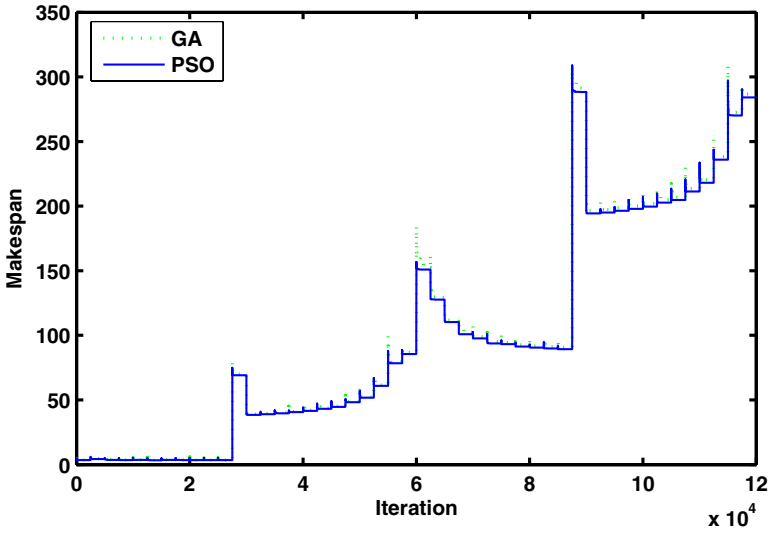

Fig. 9.4. Performance for job scheduling (60,500)

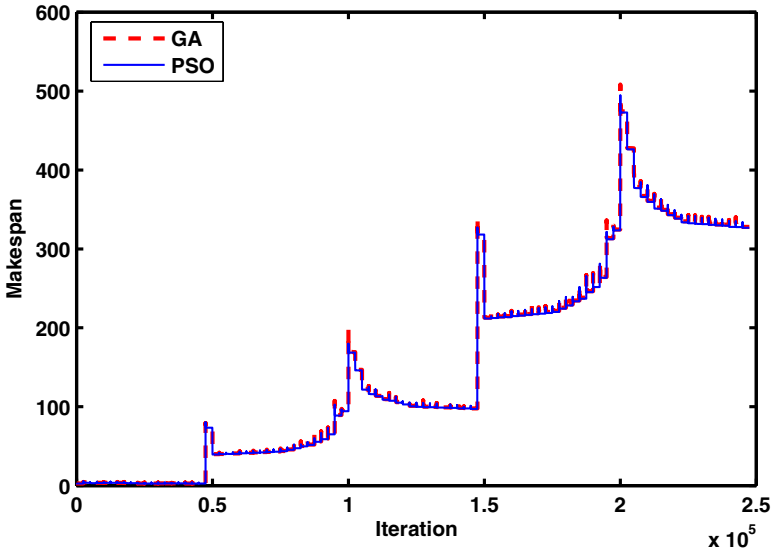

Fig. 9.5. Performance for job scheduling (100,1000)

values, the standard deviations and the time for 10 trials are illustrated in Table 9.3. Although the average makespan value of SA was better than that of GA for (3,13), the case was reversed for bigger problem sizes. PSO usually had better average makespan values than the other two algorithms. The makespan

results of SA seemed to depend on the initial solutions extremely. Although the best values in the ten trials for SA were not worse than other algorithms, it had larger standard deviations. For SA, there were some "bad" results in the ten trials, so the averages were the largest. In general, for larger (G, J) pairs, the time was much longer. PSO usually spent the least time to allocate all the jobs on the grid node, GA was the second, and SA had to spent more time to complete the scheduling. It is to be noted that PSO usually spent the shortest time to accomplish the various job scheduling tasks and had the best results among all the considered three algorithms.

It is possible that (G, J) is larger than the dimension threshold D_T. We considered two large-dimensions of (G, J), (60, 500) and (100, 1000) by submitting the jobs and the nodes in multi-stages consecutively. In each stage, 10 jobs were allocated to 5 nodes, and the single node flowtime was accumulated. The LJFN-SJFN heuristic was applied alternatively after a batch of jobs and nodes were allocated. Figs. 9.4, 9.5 and Table 9.4 illustrate the performance of GA and PSO during the search process for the considered (G, J) pairs. As evident, even though the performance were close enough, PSO generated the schedules much faster than GA as illustrated in Table 9.4.

9.4.2 Job Scheduling Using ACO

For illustration, we considered two problem instances: (3,13) and (5,100) [26]. The parameters used for GA, SA and PSO were the same as depicted in Table 9.1 and the ACO algorithm parameters are as follows:

Number of ants = 5
Weight of pheromone trail $\alpha = 1$
Weight of heuristic information $\beta = 5$
Pheromone evaporation parameter $\rho = 0.8$
Constant for pheromone updating $Q = 10$

Each experiment (for each algorithm) was repeated 10 times with different random seeds. Each trial had a fixed number of $50 * m * n$ iterations (m is the number of the grid nodes, n is the number of the jobs). The makespan values of the best solutions throughout the optimization run were recorded and the averages and the standard deviations were calculated from the 10 different trials.

Fig. 9.6 illustrates the performance of GA, SA, PSO and ACO algorithms for (3,13). The empirical results for 10 ACO runs were {46, 46, 46, 46, 46.5, 46.5, 46.5, 46, 46, 46.5}, with an average value of 46.2667. The optimal result is supposed to be 46. While GA provided the best results twice, SA, PSO, ACO provided the best results three, five and six times respectively. Empirical results are summarized in Table 9.5 for (3,13) and (5,100). As evident, ACO algorithm seems to work well but as the problem dimensions got bigger, the computational time also increased drastically.

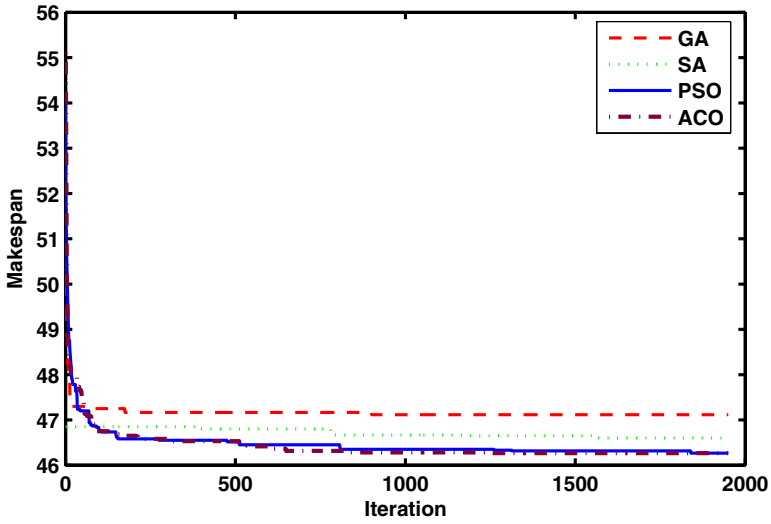

Fig. 9.6. ACO algorithm performance for (3,13)

Table 9.5. Comparing the performance of the considered algorithms

Algorithm	Item	Instance	
		(3,13)	(5,100)
GA	Average makespan	47.1167	85.7431
	Standard Deviation	±0.7700	±0.6217
	Time	302.9210	2415.9
SA	Average makespan	46.6000	90.7338
	Standard Deviation	±0.4856	±6.3833
	Time	332.5000	6567.8
PSO	Average makespan	46.2667	84.0544
	Standard Deviation	±0.2854	±0.5030
	Time	106.2030	1485.6
ACO	Average makespan	46.2667	88.1575
	Standard Deviation	±0.2854	±0.6423
	Time	340.3750	6758.3

9.4.3 Scheduling Using Evolutionary Multi-objective Optimization Approach

Instead of considering the objectives involved by using techniques which combines objectives and reduce the problem to a single objective one (as illustrated in the previous Experiment sections), in this Section, we illustrate the use of Pareto dominance concept and all the objectives are considered as independent.

Even though several optimization criteria can be considered, we considered a bi-objective minimization problem with the task of minimization of *makespan*

and *flowtime*. The most common approaches of a multiobjective optimization problem use the concept of Pareto dominance as defined below:

Pareto Dominance Concept

Consider a maximization problem. Let x, $y$be two decision vectors (solutions) from the definition domain. Solution x *dominate* y (also written as $x \succ y$), if and only if the following conditions are fulfilled:

(i) $f_i(x) \geq f_i(y); \forall i = 1,2,\ldots, n;$

(ii) $\exists j \in \{1, 2,\ldots,n\} : f_j(x) > f_j(y).$

That is, a feasible vector x is Pareto optimal if no feasible vector y can increase some criterion without causing a simultaneous decrease in at least one other criterion.

Multi-objective Evolutionary Algorithms (MOEA) can yield a whole set of potential solutions, which are all optimal in some sense. The main challenge in a multiobjective optimization environment is to minimize the distance of the generated solutions to the Pareto set and to maximize the diversity of the developed Pareto set. A good Pareto set may be obtained by appropriate guiding of the search process through careful design of reproduction operators and fitness assignment strategies. To obtain diversification special care has to be taken in the selection process. Special care is also to be taken care to prevent non-dominated solutions from being lost.

Solution Representation and Genetic Operators

The solution is represented as a string of length equal to the number of jobs. The value corresponding to each position i in the string represent the machine to which job i was allocated. Consider we have 10 jobs and 3 machines. Then a chromosome and the job allocation is represented as follows:

1	**2**	**3**	**2**	**1**	**1**	**3**	**2**	**1**	**3**

Machine 1: Job1, Job 5, Job 6, Job 9
Machine 2: Job 2, Job 4, Job 8
Machine 3: Job 3, Job 7, Job 10

Mutation and crossover are used as operators and binary tournament selection was used in the implementation. The Pareto dominance concept is used in order to compare 2 solutions. The one which dominates is preferred. In case of nondominance, the solution whose jobs allocation between machines is uniform is preferred. This means, there will not be idle machines as well as overloaded machines. The evolution process is similar to the evolution scheme of a standard evolutionary algorithm for multiobjective optimization. Reader is advised to consult [11] more details about MOEA approach.

Experiment Illustrations Using MOEA

We considered two scheduling instances (3,13) and (10,50). Specific parameter settings for MOEA, SA, PSO and GA are depicted in Table 9.6. Each experiment was repeated 10 times with different random seeds. Each trial (except for MOEA) had a fixed number of $50 * m * n$ iterations (m is the number of the grid nodes, n is the number of the jobs). The makespan values of the best solutions throughout the optimization run were recorded. First we tested a small scale job scheduling problem involving 3 nodes and 13 jobs represented as (3,13). The node speeds of the 3 nodes are 4, 3, 2 CPUT, and the job length of 13 jobs are 6, 12, 16, 20, 24, 28, 30, 36, 40, 42, 48, 52, 60 cycles, respectively. The results (makespan) for 10 runs are as follows:

Genetic Algorithm: {47, 46, 47, 47.3333, 46, 47, 47, 47, 47.3333, 49}, average value = 47.1167

Table 9.6. Parameter settings for the different algorithms

Algorithm	Parameter name	Parameter value
GA	Population size	20
	Probability of crossover	0.8
	Probability of mutation	0.02
	Scale for mutations	0.1
SA	Number operations before temperature adjustment	20
	Number of cycles	10
	Temperature reduction factor	0.85
	Vector for control step of length adjustment	2
	Initial temperature	50
PSO	Swarm size	20
	Self-recognition coefficient c_1	1.49
	Social coefficient c_2	1.49
	Inertia weight w	$0.9 \rightarrow 0.1$
MOEA	Population size	100 (500 for the second experiment)
	Number of generations	200 (1000 for the second experiment)
	Mutation probability	1 (0.9 for the second experiment)
	Crossover probability	1 (0.9 for the second experiment)

Table 9.7. Performance comparison for (10, 50)

Algorithm	Average makespan
GA	38.04
SA	41.78
PSO	37.66
MOEA	36.68

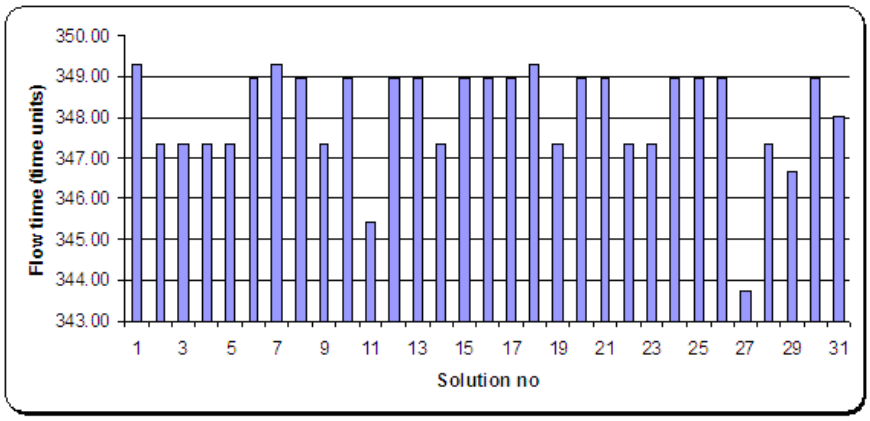

Fig. 9.7. Makespan from 31 non-dominated solutions in the final population for (10, 50)

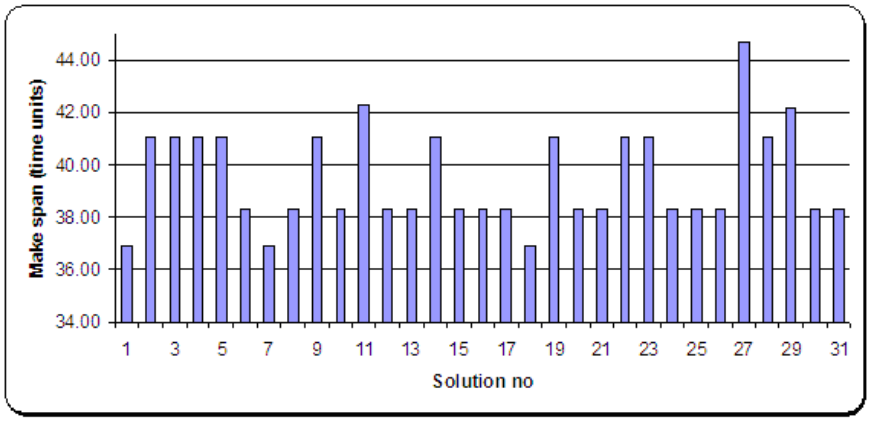

Fig. 9.8. Flowtime from 31 non-dominated solutions in the final population for (10, 50)

Simulated Annealing: {46.5, 46.5, 46, 46, 46, 46.6667, 47, 47.3333, 47, 47} average value = 46.6

Particle Swarm Optimization Algorithm: {46, 46, 46, 46, 46.5, 46.5, 46.5, 46, 46.5, 46.6667}, average value = 46.2667

Multi-objective Optimization Algorithm: 46, 46, 46, 46, 46, 46, 46, 46, 46, 46, average value = 46

The optimal result for (3,13) makespan is supposed to be 46 and the MOEA approach gave 46. It is to be noted that the MOEA approach obtained the best results in each of the considered runs.

Further, we tested the MOEA approach for (10, 50). The average makespan values for 10 trials are illustrated in Table 9.7. Although the average makespan value of SA was better than that of GA for (3,13), the case was reversed for this second case. Using the MOEA approach, the total average flow time obtained is = 348.07. Figs. 9.7 and 9.4.3 illustrate the makespan and flow time given by 31 non-dominated solutions from the final population. The user would have the option to go for a better flow time solution at the expense of a non-optimal makespan. As evident from the figure, the lowest flow time was 343.72 with the makespan of 44.75 for solution no. 27.

As evident from the empirical results, MOEA have given excellent results when compared to other techniques modeled using a single objective approach. Figs.9.7 and 9.4.3 illustrate the makespan and flow time given by 31 non-dominated solutions from the final population. The user would have the option to go for a better flow time solution at the expense of a non-optimal makespan. As evident from the Figs. 9.7 and 9.4.3, the lowest flow time was 343.72 with the makespan of 44.75 for solution no. 27. By seeing the population of solutions as illustrated in Figs. 9.7 and 9.4.3, the user will have the option to choose a particular schedule depending on the importance of the objectives. For example, the user can give more preference to a schedule which could offer a minimal flowtime but not an optimal makespan, etc.

9.5 Conclusions

In this Chapter, we illustrated the usage of several nature inspired meta-heuristics for scheduling jobs. Our approach was to dynamically generate an optimal schedule so as to complete the tasks in a minimum period of time as well as utilizing the resources in an efficient way. We evaluated the performance of the heuristic approaches using a single and multi-objective optimization approaches.

Empirical results clearly illustrate the success of nature inspired heuristics in providing real-time good solutions especially when the search space is very huge. Our experiments also illustrate the importance and benefits of considering the objectives separately (multi-objective optimization approach) rather than combining them for the sake of simplicity.

Acknowledgments

F. Xhafa acknowledges partial support by Projects ASCE TIN2005-09198-C02-02, FP6-2004-ISO-FETPI (AEOLUS) and MEC TIN2005-25859-E and FORMALISM TIN2007-66523.

References

1. Foster, I., Kesselman, C.: The Grid: Blueprint For A New Computing Infrastructure. Morgan Kaufmann, USA (2004)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, CA (1979)
3. Martino, V.D., Mililotti, M.: Sub optimal scheduling in a grid using genetic algorithms. Parallel Computing 30, 553–565 (2004)
4. Gao, Y., Rong, H.Q., Huang, J.Z.: Adaptive Grid Job Scheduling With Genetic Algorithms. Future Generation Computer Systems 21, 151–161 (2005)
5. Pang, W., Wang, K.P., Zhou, C.G., et al.: Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In: Proceedings of the 4th International Conference on Computer and Information Technology. IEEE CS Press, Los Alamitos (2004)
6. Abraham, A., Liu, H., Zhang, W., Chang, T.G.: Job Scheduling on Computational Grids Using Fuzzy Particle Swarm Algorithm. In: Gabrys, B., Howlett, R.J., Jain, L.C. (eds.) KES 2006. LNCS (LNAI), vol. 4252, pp. 500–507. Springer, Heidelberg (2006)
7. Grosan, C., Abraham, A., Helvik, B.: Multi-objective Evolutionary Algorithms for Scheduling Jobs on Computational Grids. In: Guimaraes, N., Isaias, P. (eds.) International Conference on Applied Computing 2007, Salamanca, Spain, pp. 459–463 (2007) ISBN 978-972-8924-30-0
8. Abraham, A., Buyya, R., Nath, B.: Nature's Heuristics For Scheduling Jobs on Computational Grids. In: Proceedings of the 8th International Conference on Advanced Computing and Communications, pp. 45–52. Tata McGraw-Hill, India (2000)
9. Goldberg, D.E.: Genetic Algorithms in search, optimization, and machine learning. Addison-Wesley Publishing Corporation, Inc., Reading (1989)
10. Schaffer, J.D.: Multiple Objective Optimization with Vector Evaluated Genetic Algorithms, Ph. D. Thesis, Vanderbilt University, Nashville, TN (1984)
11. Abraham, A., Jain, L., Goldberg, R. (eds.): Evolutionary Multi-objective Optimization: Theoretical Advances and Applications, ch. 12, p. 315. Springer, London (2005)
12. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by Simulated Annealing. Science 220(4598), 671–680 (1983)
13. Yao, X.: A New Simulated Annealing Algorithm. International Journal of Computer Mathematics 56, 161–168 (1995)
14. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York (1999)
15. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
16. Gambardella, L.M., Dorigo, M.: Ant-Q: A reinforcement learning approach to the traveling salesman problem. In: Proceedings of the 11th International Conference on Machine Learning, pp. 252–260 (1995)
17. Stützle, T., Hoo, H.H.: MAX-MIN ant system. Future Generation Computer Systems 16, 889–914 (2000)
18. Kennedy, J., Eberhart, R.: Swarm Intelligence. Morgan Kaufmann, San Francisco (2001)
19. Clerc, M.: Particle Swarm Optimization. ISTE Publishing Company, London (2006)

20. Kennedy, J., Mendes, R.: Population structure and particle swarm performance. In: Proceeding of IEEE conference on Evolutionary Computation, pp. 1671–1676 (2002)
21. Abraham, A., Liu, H., Chang, T.G.: Variable neighborhood particle swarm optimization algorithm. In: Genetic and Evolutionary Computation Conference (GECCO 2006), Seattle, USA (2006)
22. Shi, Y.H., Eberhart, R.C.: Fuzzy adaptive particle swarm optimization. In: Proceedings of IEEE International Conference on Evolutionary Computation, pp. 101–106 (2001)
23. Liu, H., Abraham, A.: Fuzzy Adaptive Turbulent Particle Swarm Optimization. In: Proceedings of the Fifth International conference on Hybrid Intelligent Systems, pp. 445–450 (2005)
24. Clerc, M., Kennedy, J.: The Particle Swarm-explosion, Stability, and Convergence in A Multidimensional Complex Space. IEEE Transactions on Evolutionary Computation 6, 58–73 (2002)
25. Parsopoulos, K.E., Vrahatis, M.N.: Recent Approaches to Global Optimization Problems through Particle Swarm Optimization. Natural Computing 1, 235–306 (2002)
26. Abraham, A., Guo, H., Liu, H.: Swarm Intelligence: Foundations, Perspectives and Applications. In: Nedjah, N., Mourelle, L. (eds.) Swarm Intelligent Systems. Studies in Computational Intelligence, pp. 3–25. Springer, Germany (2006)