Machine Scheduling Problem (MSP)

6. Local Search Methods (LSM)

Local search methods (LSM) form a very general class of heuristic to treat discrete optimization problems (DOP). Such problems are given by a finite set S of feasible solutions and an objective function $f:S \rightarrow R$. the goal is to find a solution with minimal objective value, i.e., we look for a solution $s^* \in S$ with

$$f(s^*) = \underset{s \in S}{\text{Min}} \{f(s)\}$$

Generally speaking, LSM move iteratively through the solution set S of a DOP. Based on the current and may be on the previous visited solutions, a new solution is chosen.

Basic structure of local search algorithm
Choose an initial solution;
REPEAT
Choose a solution from the neighborhood of the current solution and move to this solution;
UNTIL stopping criteria is met.

Evolutionary Algorithms (EAs) or **Local Search Methods** have been shown to be successful for a wide range of optimization problems. While these algorithms work well for many optimization problems in practice, a satisfying and rigorous mathematical understanding of their performance is an important challenge in the area of evolutionary computing [32].

6.1 Simulated Annealing (SA)

Simulated annealing (SA) is an algorithmic method that is able to escape from local minima. It is a randomized LSM for two reasons: First, from the neighborhood of a solution a neighbor is randomly selected. Second, in addition to better-cost neighbors, which are always accepted if they are selected, worse-cost neighbors are also accepted, although with a probability that is gradually decreased in the course of the algorithm's execution. The randomized nature enables asymptotic convergence to optimum solutions under certain mild conditions. Nevertheless, the energy landscape, which is determined by the objective function and the neighborhood structure, may admit many and/or "deep" local minimum. Therefore, avoiding local minima is a crucial part of the performance of the algorithm [46].

SA algorithm starts to work by generating random initial solution (s), then the difference $\Delta = F(s')-F(s)$ and neighbor (s') in the objective function is calculated. If $\Delta<0$, the neighbor (s') will be accepted to be the new solution in the next iteration since it has a better function value. If the objective function value does not decrease (i.e.$\Delta \geq 0$), the generated neighbor may also be accepted with a probability $\exp(-\Delta/T)$, where T is a control parameter called temperature. This temperature is always reduced by a cooling technique in every iteration. As a stopping criteria, one may use e.g. a given number of iteration, a time limit or a given number of iterations without an improvement of the best objective function value. In the first two cases, one must adjust the cooling scheme in such a way that SA stops with a small temperature. Let $p(\Delta,T) = e\exp(-\Delta Cost/T)$ is the probability depends on exponential function and FT be the final temperature [34].

Algorithm (1): Simulated Annealing (SA) Algorithm

Step 1: Input: T, FT, *cooling rate*, s;
Step 2: $s' = s$; *Cost* = Evaluate (s');
Step 3: while ($T > FT$) do
s_1 = Mutate (s');
NewCost = Evaluate (s_1);
$\Delta Cost = NewCost - Cost$;
if ($\Delta Cost \leq 0$) OR ($p(\Delta,T) >$ Rand) then
Cost = *NewCost*;
$s' = s_1$;
end
T = *cooling rate* × T;
end
Step 4: Output: the best s';

6.2 Genetic Algorithm (GA)

Genetic Algorithms (GA's) are search algorithms based on the mechanics of natural selection and natural genetics. GA is an iterative procedure, which maintains a constant size population of candidate solutions. During each iteration step (Generation) the structures in the current population are evaluated, and, on the basic of those evaluations, a new population of candidate solutions formed. The basic GA cycle shown in figure (2) [10].

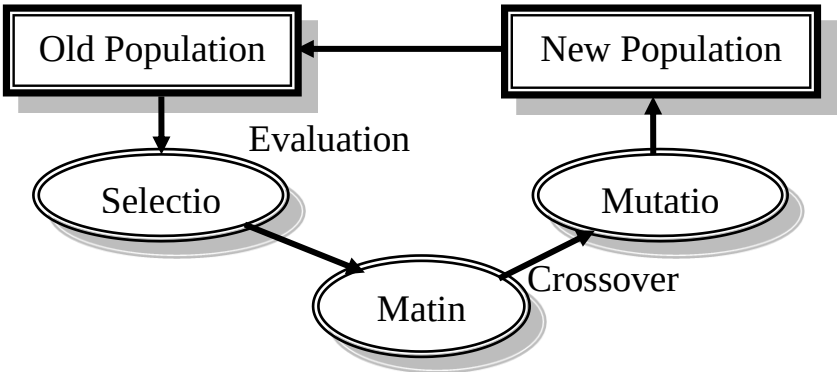

Figure (2) Basic cycle of GA.

Algorithm (2): Genetic Algorithm (GA)

Generation=0;
Initialize G(Pop); {G=Generation ; Pop=Population}
Evaluate G(Pop);
While (GA has not converged or terminated)
    Generation = Generation + 1;
    Select G(Pop) from G(Pop-1);
    Crossover G(Pop);
    Mutate G(Pop);
    Evaluate G(Pop);
End (While)

6.3 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) has found applications in a lot of areas. In general, all the application areas that the other evolutionary techniques are good at are good application areas for PSO [83].

PSO was originally developed by a social-psychologist J. Kennedy and an electrical engineer R. Eberhart in 1995 [58] and emerged from earlier experiments with algorithms that modeled the "flocking behavior" seen in many species of birds. It is yet another optimization algorithm that falls under the soft computing umbrella that covers genetic and evolutionary computing algorithms as well.

PSO is an extremely simple concept, and can be implemented without complex data structure. No complex or costly mathematical functions are used, and it doesn't require a great amount of memory [79]. The facts of PSO has fast convergence, only a small number of control parameters, very simple computations, good performance, and the lack of derivative computations made it an attractive option for solving the problems.

The **PSO algorithm** depends in its implementation in the following two relations:

$$v_{id} = w^* v_{id} + c_1 * r_1 * (p_{id} - x_{id}) + c_2 * r_2 * (p_{gd} - x_{id}) \quad \ldots(1)$$

$x_{id} = x_{id} + v_{id}$ …(2)

where *w* is the inertia weight for convergence, c_1 and c_2 are positive constants, r_1 and r_2 are random functions in the range [0,1], $x_i=(x_{i1},x_{i2},\ldots,x_{id})$ represents the i^{th} particle; $pa_i=(p_{i1},p_{i2},\ldots,p_{id})$ represents the (pbest) best previous position (the position giving the best fitness value) of the i^{th} particle; the symbol g represents the index of the best particle among all the particles in the population, $v_i=(v_{i1},v_{i2},\ldots,v_{id})$ represents the rate of the position change (velocity) for particle i [83].

Algorithm (3): **Particle Swarm Optimization (PSO) algorithm**

1. Initialize a population of particles with random positions and velocities on d-dimensions in the problem space.
2. PSO operation includes:
   a. For each particle, evaluate the desired optimization fitness function in d variables.
   b. Compare particle's fitness evaluation with its pbest. If current value is better than pbest, then set pbest equal to the current value, and pa_i equals to the current location x_i.
   c. Identify the particle in the neighborhood with the best success so far, and assign it index to the variable g.
   d. Change the velocity and position of the particle according to equations (1) and (2).
3. Loop to step (2) until a criterion is met.

Like the other evolutionary algorithms, a PSO algorithm is a population based on search algorithm with random initialization, and there is an interaction among population members. Unlike the other evolutionary algorithms, in PSO, each particle flies through the solution space, and has the ability to remember its previous best position, survives from generation to another.

A number of factors will affect the performance of the PSO. These factors are called **PSO parameters**, these parameters are [58]:

1. Number of particles in the swarm affects the run-time significantly, thus a balance between variety (more particles) and speed (less particles) must be sought.
2. Maximum velocity (v_{max}) parameter. This parameter limits the maximum jump that a particle can make in one step.
3. The role of the inertia weight w, in equation (2.1a), is considered critical for the PSO's convergence behavior. The inertia weight is employed to control the impact of the previous history of velocities on the current one.
4. The parameters c_1 and c_2, in equation (2.1a), are not critical for PSO's convergence. However, proper fine-tuning may result in faster

convergence and alleviation of local minima, it better to choose a larger cognitive parameter c_1 than a social parameter c_2 but with $c_1 + c_2 = 4$.

5. The parameters r_1 and r_2 are used to maintain the diversity of the population, and they are uniformly distributed in the range [0,1].

Flowchart of PSO algorithm is depicted in figure (3) [92].

6.4 Bee Algorithm (BA)

Honey Bees Optimization (MBO) is a new development which is based on the haploid-diploid genetic breeding of honey bees and is used for a special group of propositional satisfiability problems. The main processes in MBO are: the mating flight of the queen bee with drones, the creation of new broods by the queen bee, the improvement of the broods' fitness by workers, the adaptation of the workers' fitness, and the replacement of the least fit queen with the fittest brood [12].

The challenge is to adapt the self-organization behavior of the colony for solving the problems. The **Bees Algorithm (BA)** is an optimization algorithm inspired by the natural foraging behavior of honey bees to find the optimal solution. The pseudo code for the BA in its simplest form is as follows [76].

The algorithm requires a number of parameters to be set, namely:

n	:	Number of scout bees.
ss	:	Number of sites selected out of n visited sites.
E	:	Number of best sites out of ss selected sites.
nep	:	Number of bees recruited for best e sites.
nsp	:	Number of bees recruited for the other (ss-e) selected sites.
ngh	:	Initial size of patches which includes site and its neighborhood and stopping criterion.

Algorithm (4): Bees Algorithm (BA)

INPUT: n, ss, e, nep, nsp, Maximum of iterations.

Step1. Initialize population with random solutions**.**

Step2. Evaluate fitness of the population**.**

Step3. REPEAT

Step4. Select sites for neighborhood search**.**

Step5. Recruit bees for selected sites (more bees for best e sites) and evaluate fitness's**.**

Step6. Select the fittest bee from each patch**.**

Step7. Assign remaining bees to search randomly and evaluate their fitness's**.**

Step8. UNTIL stopping criterion is met.

OUTPUT: Optimal or near optimal solutions**.**

END.

The advantages of Bees algorithm [12]:

- ❖ BA is more efficient when finding and collecting food that is it takes less number of steps.
- ❖ BA is more scalable, it requires less computation time to complete the task.