Optimization Techniques
Project
on
Particle Swarm Optimization

Sahaj Khandelwal
B.Tech. CSE(3rd Year)
Roll No.- 160001052

Rishabh Kumar Verma
B.Tech. CSE(3rd Year)
Roll No.- 160001048

Course Instructor : Dr. Kapil Ahuja

Overview

In computational science, particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. It solves a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle movement is influenced by its local best-known position but is also guided toward the best-known positions in the search space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

Goals

1. Applying Particle Swarm Optimization algorithm for NP-Hard combinatorial optimization problems.
2. Analysing the various implementations of PSO and discussing issues, proposing improvements.

Motivation

Particle swarm optimization (PSO) is an evolutionary computational technique used for optimization motivated by the social behaviour of individuals in large groups in nature. Different approaches have been used to understand how this algorithm works and trying to improve its convergence properties for different kind of problems. These approaches go from heuristic to mathematical analysis, passing through numerical experimentation. Although the scientific community has been able to solve a big variety of engineering problems, the tuning of the PSO parameters still remains one of its major drawbacks. When adapted efficiently to optimization problems, PSO is often characterized by fast convergence and

easy implementation. These characteristics motivate the choice of PSO for NP-Hard combinatorial optimization problems.

Particle Swarm Optimization Algorithm

PSO is an algorithm that follows a collaborativepopulation-based search model. Each individual of the population called a 'particle', fliecollaborative populations around in a multidimensional search space looking for the optimal solution. Particles, then, may adjust their position according to their own and their neighbouring-particles experience, moving toward their best position or their neighbour's best position. In order to achieve this, a particle keeps previously reached 'best' positions in a cognitive memory. PSO performance is measured according to a predefined fitnessfunction (cost function of a problem). Balancing between global and local exploration abilities of the flying particles could be achieved through user-defined parameters. PSO has many advantages over other heuristic techniques such that it can be implemented in a few lines of computer code, it requires only primitive mathematical operators, and it has a great capability of escaping local optima.

PSO performs searching via a swarm of particles that updates from iteration to iteration. To seek the optimal solution, each particle moves in the direction to its previous best (pbest) position and the global best (gbest) position in the swarm. One has-

$$pbest\,(i,t) = \underset{k=1,\ldots,t}{\arg\min}\left[f\left(P_i\left(k\right)\right)\right], \quad i \in \left\{1,2,\ldots,N_P\right\},$$

$$gbest\,(t) = \underset{\substack{i=1,\ldots,N_P \\ k=1,\ldots,t}}{\arg\min}\left[f\left(P_i\left(k\right)\right)\right],$$

Where i denotes the particle index,N_p the total number of particles,t the current iteration number, f the fitness function, and P the position. The velocity V and position P of particles are updated by the following equations:

$$V_i(t+1) = \omega V_i(t) + c_1 r_1 \left(pbest(i,t) - P_i(t)\right) + c_2 r_2 \left(gbest(t) - P_i(t)\right),$$

$$P_i(t+1) = P_i(t) + V_i(t+1),$$

where V denotes the velocity,w is the inertia weight used to balance the global exploration and local exploitation,r_1 and r_2 are uniformly distributed random variables within the range [0,1] and c_1 and c_2 are positive constant parameters called "acceleration coefficients."

The following depicts the basic PSO algorithm:-

Let P be the size of the PSO population.

Let $PSO[i]$ be the position of the i^{th} particle of the PSO population; this represents a candidate solution for the problem.

Let $fitness[i]$ be the cost function of the i^{th} particle.

Let $V[i]$ be the traveled distance (or velocity) of the i^{th} particle.

Let G_{best} be an index to global-best position.

Let $P_{best}[i]$ be the position of the local-best position of the i^{th} particle.

Let $P_{best}\_fitness[i]$ be the local-best fitness for the best position visited by the i^{th} particle.

Step1 (Initialization): For each particle i in the population:

Step1.1: Initialize $PSO[i]$ randomly.

Step1.2: Initialize $V[i]$ randomly.

Step1.3: Evaluate $fitness[i]$.

Step1.4: Initialize G_{best} with the index of the particle with the best fitness among the population.

Step1.5: Initialize $P_{best}[i]$ with a copy of $PSO[i]$ $\forall i \le P$.

Step2: Repeat until a stopping criterion is statisfied:

Step2.1: Find G_{best} such that $fitness[G_{best}] \ge fitness[i]$ $\forall i \le P$

Step2.2: For each particle i: $P_{best}[i] = PSO[i]$ *iff* $fitness[i] > P_{best}\_fitness[i]$ $\forall i \le P$.

Step2.3: For each particle i: update $V[i]$ and $PSO[i]$ according to equations 7 and 8.

Step2.4: Evaluate $fitness[i]$ $\forall i \le P$.

Task Assignment Problem

The task assignment (or mapping) problem is that of assigning tasks of a program among different processors of a distributed computer system in order to reduce the program turnaround time and to increase the system throughput. This can be done by maximizing andbalancing the utilization of resources while minimizing the communication between processors.

Consider a system consisting of *N* number of processors, to which *N* number of tasks have to be assigned individually, under the constraint that a single task can be assigned to a single processor and a single processor can hold a single task only. Given, is the cost incurred by each processor to complete each task respectively.

Mapping of TAP (Task Assignment Problem) into PSO (Particle Swarm Optimization)

We set up a search space of *N* dimension for an *N* task assignment problem. Each dimension has a discrete set of possible values limited to *s* = { P_i : 1<=i<=N}; such that N is the number of processors in the distributed system under consideration.

Using such particle representation, the PSO population is represented as an *NxN* two-dimensional array consisting of *N* particles, each represented as a vector of *N* tasks. Thus, a particle flies in an *N-dimensional* search space. A task is internally represented as an integer value indicating the processor number to which this task is assigned to during the course of PSO.In our PSO algorithm, we map an *N* -task assignment instance into corresponding *N*-coordinate particle position.The algorithm starts by generating randomly as many potential assignments for the problem as the size of the initial population of the PSO. It then measures particles'fitness.

Fitness function :

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

where $C_{total}(A)_k$ represents the total cost of doing performing a task. And, Cost(A) represents the minimum cost of doing a task for the kth processor.

The goal of the task assignment problem is to find an assignment *A* which has the minimum cost for a given TIG, minimise (Cost(A) for all A).

Mapping and Working of the algorithm

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

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

The following is a task assignment problem-based PSO algorithm description:

Let N be the number of processors given by the problem.

Let M be the number of tasks in a TIG instance.

Let P be the size of the PSO population.

Let $PSO[i]$ be the position of the i^{th} particle of the PSO population represented as an M-dimensional vector, whose entries' values belong to the set $\{1, \ldots, N\}$;

Then $PSO[i][j]$ be the processor number to which the j^{th} task in the i^{th} particle is assigned.

Let $fitness[i]$ be the cost function of the i^{th} particle according to Equation 5.

Let $V[i]$ be the traveled distance (or velocity) of a i^{th} particle represented as an M-dimension real-coded vector.

Let G_{best} be an index to global-best position.

Let $P_{best}[i]$ be the position of the local-best position of the i^{th} particle.

Let $P_{best}\_fitness[i]$ be the local-best fitness for the best position visited by the i^{th} particle.

Step1 (Initialization): For each particle i in the population:

- For each task j:
  - Initialize $PSO[i][j]$ randomly from the set $\{1, \ldots, N\}$.
- Initialize $V[i]$ randomly.
- Evaluate $fitness[i]$.
- Initialize G_{best} with the index of the particle with the best fitness (lowest cost) among the population.
- Initialize $P_{best}[i]$ with a copy of $PSO[i]$ $\forall i \le P$.

Step2: Repeat until a number of generation, equal to twice the total number of task, is passed:

- Step2.1: Find G_{best} such that $fitness[G_{best}] \ge fitness[i]$ $\forall i \le P$.
- Step2.2: For each particle i: $P_{best}[i] = PSO[i]$ *iff* $fitness[i] > P_{best}\_fitness\ [i]$ $\forall i \le P$.
- Step2.3: For each particle i: update $V[i]$ and $PSO[i]$ according to equations 7 and 8.
- Step2.4: Evaluate $fitness[i]$ $\forall i \le P$.

Conclusion and Future Scope

In distributed computing systems, qualified assignment of tasks among processors is an important step for efficient utilization of resources. And, the Particle Swarm Optimization Algorithm provides a new heuristic approach for solving the Task Assignment Problem. We concluded that real-life problems can similarly be mapped to common optimization problems, and that, algorithms like Particle Swarm Optimization are a good option in providing heuristic solutions to them.

A natural extension of this project could be to apply a similar approach to solving heterogeneous variations of the Task Assignment Problem.

The Github link to show the implementation in C++ and a visualisation of PSO in python: Github Link to Code.

References: Research Paper