Particle Swarm Optimisation

Particle swarm optimisation (PSO)

Generalities:

- First proposed in [Kennedy and Eberhart, 1995],
- inspired by the collective intelligence of birds:
- the social interaction of birds allows the solution of a problem.

Particle swarm optimisation

Metaphor and working principle:

- If taken one by one, birds in the swarm have no central control to allow a global interesting behaviour to emerge...
- but by interacting locally among themselves (sharing of local information), their are able to explore the sky looking for food.
- Each bird has a memory of the most successful place he has visited and share it with its neighbour;
- they tend to follow a leader (wisest bird in the swarm) who knows the best place to find food.
- Birds explore the area but keep in mind their previous success and the success of the leader.
- If a better place is found, the leader suddenly change and the swarm change direction.

PSO: representation

- In the most general case, a bird (or particle) $\mathbf{P_i}$ $(i = 1, 2, 3, \ldots, NP)$ in a swarm of NP particles in associated to:
  - its position $\mathbf{x_i} \in \mathbb{R}^n$ in the n-dimensional search space;
  - its velocity[1] vector $\mathbf{v_i} \in \mathbb{R}^n$;
- Also, every particle has a memory of:
  - its most successful position $\mathbf{x_i^{lb}}$ (in its neighbourhood: local best);
  - its most successful position $\mathbf{x_i^{lw}}$ (in its neighbourhood: local worst);

N.B. $\mathbf{x_i^{lw}}$ is rarely used, but some variants could employ it.

The best point (lower fitness value) amongst all $\mathbf{x_i^{lb}}$ $i = 1, 2, 3, \ldots, NP$ is obviously the best solution found for the problem $\mathbf{x^{gb}}$ (global best).

[1] It must not be confused with the physical meaning of this term, strictly speaking it is a displacement rather than a velocity.

Standard PSO working mechanism
Velocity and position update

- Positions and velocities are initialised uniformly within the search space, $\mathbf{x_i^{lb}}$ and $\mathbf{x^{gb}}$ assigned according to their fitness values;
- $\forall$ i-th particle in the swarm, velocities have to be updated

$$\mathbf{v_i} = \phi_1 \mathbf{v_i} + \phi_2 \left(\mathbf{x_i^{lb}} - \mathbf{x_i}\right) + \phi_3 \left(\mathbf{x^{gb}} - \mathbf{x_i}\right) \tag{1}$$

- and positions perturbed accordingly (iteration $k+1$):

$$\mathbf{x_i} = \mathbf{x_i} + \mathbf{v_i} \tag{2}$$

- if the new generated position has a lower fitness value than the local best position, then it replaces it.

Standard PSO: velocity update

randomised weights

inertia weight

$$\mathbf{v_i} = \phi_1 \mathbf{v_i} + \phi_2 \left(\mathbf{x_i^{lb}} - \mathbf{x_i}\right) + \phi_3 \left(\mathbf{x^{gb}} - \mathbf{x_i}\right)$$

"towards the local best" move

"towards the global best" move

- velocity is randomised and it is added to the position as in a stochastic search!
  - But can be "biased" by properly combining the 3 component on the right-hand side!
- Performance heavily depends on the choice of the weights (tuning is key)!

Standard PSO: velocity update
(external forces)

randomised weights

inertia weight

$\phi_2 = \mathcal{U}(0, c_1)$

$\phi_3 = \mathcal{U}(0, c_2)$

$$\mathbf{v_i} = \phi_1 \mathbf{v_i} + \phi_2 \left(\mathbf{x_i^{lb}} - \mathbf{x_i}\right) + \phi_3 \left(\mathbf{x^{gb}} - \mathbf{x_i}\right)$$

"towards the local best" move

"towards the global best" move

- $\phi_{2/3} = \mathcal{U}\left(0, c_{1/2}\right)$ is a vector uniformly sampled in $[0, c_{1/2}]^n$[2]
- The two moves can be seen as attractive forces produced by springs of random stiffness [Poli et al., 2007]:
  - by tuning c_1 and c_2 we make PSO more or less responsive!

[2] The element-wise product is performed with $\left(\mathbf{x_i^{gb}} - \mathbf{x_i}\right)$ and $\left(\mathbf{x_i^{lb}} - \mathbf{x_i}\right)$.

Standard PSO: velocity update
(particles' inertia)

randomised weights

inertia weight

$$\mathbf{v_i} = \phi_1 \, \mathbf{v_i} + \phi_2 \left(\mathbf{x_i^{lb}} - \mathbf{x_i}\right) + \phi_3 \left(\mathbf{x^{gb}} - \mathbf{x_i}\right)$$

"towards the local best" move

"towards the global best" move

- ϕ_1 tuning is key: high inertia $\Rightarrow$ big steps, low $\Rightarrow$ small steps:
  - ideally, it is high (e.g. 0.9) at the beg beginning (*exploration*),
  - and low towards the end of the optimisation (*exploitation*)!
- This can be achieved by tuning it on-the-fly!
- Many methods were proposed (see HERE), a popular one is:

$$\phi_1 = \phi_{max} - \frac{(\phi_{min} - \phi_{max})\, k}{k_{max}}$$ ("Linear Decreasing Inertia Weight").

Standard PSO

memory structures:

- two memory structures are necessary, the first one to keep track of local best positions, the second for the current positions, i.e. the trials;
- with respect to Evolutionary Algorithms we may think we have two populations instead of one (we perturb the first one, we make converge the other the second one!);
- on the other hand, the set of current positions can be seen as an offspring population.

PSO as EA

- PSO can be seen as population-based algorithm where the particle positions evolve over time;
- each local best is a candidate solution while a current position is a trial solution;
- in a way, the metaphor is not so important: PSO can possibly be seen as an EA.

Main difference with EAs

- The main difference is not in the metaphor but in the logic related to the selection:
  - In EAs the selection (either parent or survivor) must be fitness based and take into account the entire population (e.g. arranging a ranking or a tournament);
  - in PSO (and all Swarm Intelligence Algorithms) the selection is done by considering the solution before and after the perturbation.

Standard PSO: implementation details

Standard PSO pseudo-code

```
initialise NP                                          ▷ Swarm size
initialise V_max                                       ▷ usually, V_max ← x^U − x^L
V_min = −V_max
for i = 1 : NP do
    V_i ∼ U(x^L, x^U)                                  ▷ also suggested V_i ∼ U(V_min/3, V_max/3)
    x_i ∼ U(x^L, x^U)
    x_i^lb ← x_i
    if f(x_i) ≤ f(x^gb) || i = 1 then
        x^gb ← x_i
    end if
end for
while Condition on budget do
    for i = 1 : NP do
        Update V_i and perturb x_i                     ▷ Formula 1 and 2
        if f(x_i) ≤ f(x_i^lb) then                     ▷ 1-to-1 spawning
            x_i^lb ← x_i                               ▷ local best update
            if f(x_i) ≤ f(x^gb) then
                x^gb ← x_i                             ▷ global best update
            end if
            Update swarm
        end if
    end for
end while
Output x^gb
```

PSO variants

- In the past 20 years a multitude of variants has been proposed,
- these variants are mainly based on modifications of the velocity update strategy.
- Discrete and hybrid (e.g. with LS algorithms and EAs) implementations have been proposed too.
- We will briefly consider two important variants: CLPSO and CCPSO2.

Comprehensive learning Particle swarm optimisation (CLPSO) *[Liang et al., 2006]*

- Same structure and selection of a standard PSO.
- The perturbation logic is drastically changed:
  - the following learning strategy is employed in updating $\mathbf{V_i}$, whereby all the local best positions help form the perturbation vector

$$\mathbf{v_i} = \phi_1 \mathbf{v_i} + \phi_2 \mathbf{U} \times \left(\mathbf{x_i^{rlb}} - \mathbf{x_i} \right) \tag{3}$$

- The proposed modification has shown to be performing in handling problems in up to $50D$.

CLPSO
velocity update:

inertia weight (constant) weight

$$\mathbf{v_i} = \phi_1 \mathbf{v_i} + \phi_2 \mathbf{U} \times \left(\mathbf{x_i^{rlb}} - \mathbf{x_i} \right)$$

random $n \times n$ matrix, $[u_{i,j}] = \mathcal{U}(0,1)$ "randomised" local best

- $\mathbf{x_i^{rlb}}$ borrows components from all the local bests of the swarm:
- $\forall$ design variable j ($j = 1, 2, 3, \ldots, n$)
  - if $\mathcal{U}(0,1) > P_{c,i} \Rightarrow \mathbf{x_i^{rlb}}[j] = \mathbf{x_i^{lb}}[j]$;
  - otherwise, two particles are randomly selected from the swarm
    - the fittest of the the two gives the j^{th} design variable to $\mathbf{x_i^{rlb}}$.

CLPSO

The threshold $P_{c,j}$

- $P_{c,j}$ is generated for each design variable such that it gets less likely to "borrow" variables:

$$P_{c,j} = 0.05 + 0.45 \cdot \frac{e^{\frac{10(j-1)}{NP-1}} - 1}{e^{10} - 1}$$

- it can be easily seen that we have a 95% probability for borrowing the first design variable ($j = 1$), and only 50% for the last one ($j = NP$).

Cooperatively Co-evolving PSO
for large scale optimization (CCPSO2)

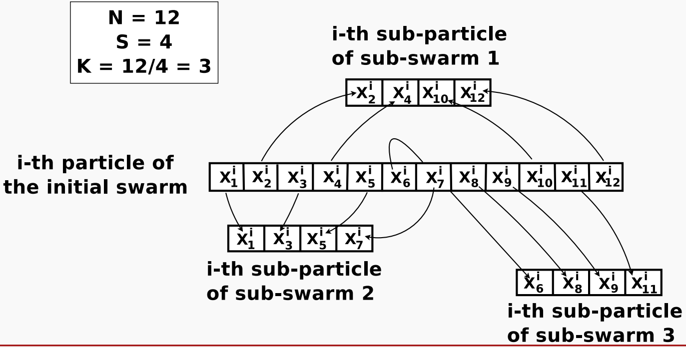
- Simple and efficient scheme to address large scale problems, but often very efficient also in low dimensions!.
- We describe the main idea to understand how the "coevolving" approach can be beneficial in large scale optimisation;
- however, this variant, namely CCPSO2, does not employ the classic PSO velocity based perturbation strategy.
- For more details see the original paper [Li and Yao, 2012].

CCPSO2: main idea

- If the problem is characterized by many variables, it is worthy to attempt to decompose the problem and start multiple sub-PSO instances on the sub-problems;
- This division is arbitrary and it is unknown whether allows an improvement.
- If improvements occur, we keep evolving the sub-swarms; conversely, we restart the algorithm with a different grouping.

CCPSO2: domain decomposition

- In order to create sub-swarms we randomly pick up a divisor S from a set of potential options. e.g. $S \in \{2, 5, 50, 100, 200\}$;
- then, $k = \frac{n}{S}$ sub-swarms in S dimension values are considered.
- N.B. The fitness functional call must be done by using an n-dimensional vector! (e.g. injection of s new components into the global best).



CCPSO2: position update

- In CCPSO2 there is no velocity update and the new position is generated by using a Normal (Gaussian) or Cauchy distribution as follow:

$$\mathbf{x}_i = \begin{cases} \mathbf{x^{gb}}[j] + \mathcal{C}(0,1)\,|\mathbf{x_i^{lb}}[j] - \mathbf{x^{gb}}[j]| & \text{if } \mathcal{U}(0,1) < p \\ \mathbf{x^{gb}}[j] + \mathcal{N}(0,1)\,|\mathbf{x_i^{lb}}[j] - \mathbf{x^{gb}}[j]| & \text{otherwise} \end{cases} \qquad j = 1, 2, \ldots, S$$

- It must be remarked that in this case $\mathbf{x^{gb}}$ is the fittest amongst the current i^{th} particle in the sub-swarm, its its immediate left and its immediate right neighbors.
- $p = 0.5$ is suggested.
- Both the distribution are centred in $\mathbf{x^{gb}}$, and scaled in $\mathbf{x_i^{lb}}[j] - \mathbf{x^{gb}}[j]$, but Cauchy performs longer steps than Gaussian distribution.

More SI algorithms

- There are many other optimisers based on SI logic
  - sometimes they look quite similar!
- PSO is probably the most used, but many are currently being used, such as:
  - Ant Colony Optimisation (ACO) [Dorigo et al., 2006];
  - Bacterial Forging Optimisation (BFO) [Passino, 2012]
  - Migrating Birds Optimisation (MBO) [Duman et al., 2012]
  - Artificial Bee Colony (ABC) algorithm [Karaboga, 2005].
  - etc.
- All the mentioned variants have the 1-to-1 spawning strategy in common (as in PSO), and can adopt combinations of popular operators... e.g. let us briefly consider ABC...

Artificial Bee Colony (ABC)
([Karaboga, 2005])

- Inspired by the behavior of the bees;
- from a different metaphor, but still employs commonly used working principles (1-to-1 spawning, mutations, fitness proportionate selection etc.).
- The main idea is to explore different locations (flowers) by combining two perturbation strategies instead of one:
  - one of these strategies is explorative while the other is exploitative (conceptually not so different from mutation and crossover in EAs).

ABC: brief description

a complete iteration consist of:

1. We perturb the position ($\mathbf{x_i}$ $i = 1, 2, 3, \ldots, NB$) of each (*employed*) bee in the swarm via:

$$\mathbf{v_i} \leftarrow \mathbf{x_i} + \mathcal{U}\left(-1, 1\right)\left(\mathbf{x_i} - \mathbf{x_j}\right) \quad (j \neq i,) \tag{4}$$

2. We perform replacement via 1-to-1 spawning.
3. By means of roulette wheel with fitness proportionate selection, we select NB (*onlooker*) bees to perform step 1 and 2 again.
4. Those i^{th} positions that are not improved by this procedure are re-sampled (*scout* bees):

$$\mathbf{x_i} \leftarrow \mathcal{U}\left(\mathbf{x^L}, \mathbf{x^U}\right) \tag{5}$$

ABC
some considerations

- ACO can be considered as an SI algorithm:
  - 1-to-1 spawning
  - the information form the best "empoyed" bees are then used by the "onlokeer" bees (as the local best is used to perform the perturbation in PSO).
- However, this framework borrows may operators from EAs:
  - Equation 4 is basically a mutation;
  - Equation 5 is the initialisation process.

Laboratory and participation work:

- Implement a standard PSO (toroidal bounds), for the usual four problems (De Jong, Rastrigin, Schwefel and Michalewicz) in 10D and 50D.
- Implement CLPSO and run on the same problems.
- Compare the results and write one paragraph to explain what makes CLPSO a successful framework (feel free to use the literature to get some inspiration and cite the studies you found helpful to make a conjecture).

References I

Dorigo, M., Birattari, M., and Stützle, T. (2006).
Ant colony optimization.
Computational Intelligence Magazine, IEEE, 1(4):28–39.

Duman, E., Uysal, M., and Alkaya, A. F. (2012).
Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem.
Information Sciences, 217:65–77.

Karaboga, D. (2005).
An idea based on honey bee swarm for numerical optimization.
Technical report, Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

Kennedy, J. and Eberhart, R. C. (1995).
Particle swarm optimization.
In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948.

References II

Li, X. and Yao, X. (2012).
Cooperatively coevolving particle swarms for large scale optimization.
Evolutionary Computation, IEEE Transactions on, 16(2):210–224.

Liang, J. J., Qin, A. K., Suganthan, P. N., and Baskar, S. (2006).
Comprehensive learning particle swarm optimizer for global optimization of multimodal functions.
IEEE Transactions on Evolutionary Computation, 10(3):281–295.

Passino, K. M. (2012).
Bacterial foraging optimization.
Innovations and Developments of Swarm Intelligence Applications, page 219.

Poli, R., Kennedy, J., and Blackwell, T. (2007).
Particle swarm optimization.
Swarm intelligence, 1(1):33–57.