A Cooperative Approach to Particle Swarm Optimization

Authors: Frans van den Bergh, Andries P. Engelbrecht

Journal: Transactions on Evolutionary Computation, vol 8, No. 3, June 2004

Presentation: Jose Manuel Lopez Guede

Introduction

- “Curse of dimensionality”

- PSO
- CPSO
- CPSO-S_k
- CPSO-H_k
- GA comparation

- Results

Particle Swarm Optimizers I

- **PSO:**
  - Stochastic optimization technique
  - Swarm: a population
  - During each iteration each particle accelerates influenced by:
    - Its own personal best position
    - Global best position

Particle Swarm Optimizers II

- s denote the swarm size
- Each individual $1 \leq i \leq s$
  - space $\mathbf{x}_i$
  - current velocity $\mathbf{v}_i$
  - personal best position in the search space $\mathbf{y}_i$

Particle Swarm Optimizers III

– During each iteration, each particle is updated:

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,i}(t)\left[y_{i,j}(t) - x_{i,j}(t)\right] + c_2 r_{2,i}(t)\left[\hat{y}_j(t) - x_{i,j}(t)\right] \quad (1)$$

for all $j \in 1 \ldots n$, thus, $v_{i,j}$ is the velocity of the jth dimension of the ith particle, and c_1 and c_2 denote the *acceleration coefficients*. The new position of a particle is calculated using

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1). \quad (2)$$

$r_1 \sim U(0,1), r_2 \sim U(0,1)$

w in (1) is called the *inertia weight*;

Acceleration coefficients c_1 and c_2

Particle Swarm Optimizers III

– During each iteration, each particle is updated:

The personal best position of each particle is updated using

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t), & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1), & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i)(t) \end{cases} \quad (3)$$

– The global best position is updated:

$$\hat{\mathbf{y}}(t+1) = \arg\min_{\mathbf{y}_i} f(\mathbf{y}_i(t+1)), \quad 1 \leq i \leq s.$$

Cooperative Learning I

- **PSO:**

position

Best position of the particle

Best position of the swarm

```
Create and initialise an n-dimensional PSO : P
repeat:
    for each particle i ∈ [1..s] :
        if f(P.x_i) < f(P.y_i)
            then P.y_i = P.x_i
        if f(P.y_i) < f(P.ŷ)
            then P.ŷ = P.y_i
    endfor
    Perform PSO updates on P using eqns. (1–2)
until stopping criterion is met
```

  – Each particle represents an n-dim vector that can be used as a potential solution.

Cooperative Learning II

- Drawback:
  - Authors show a numerical example where PSO goes to a worst value in an iteration.
  - Cause: error function is computed only after all the components of the vector have been updated to their new values.
- Solution:
  - Evaluate the error function more frecuently.
  - For every time a component in the vector has been updated.
- New problem:
  - The evaluation is only possible with a compete vector.

Cooperative Learning III

- **CPSO-S:**
  - n-dim vectors are partitioned into n swarms of 1-D
  - Each swarm represents 1 dimension of the problem
  - “Context vector”:
    - f requires an n-dim vector
    - To calculate the context vector for the particles of swarm j, the remainig components are the best values of the remaining swarms.

Cooperative Learning IV

context vector

define
$\mathbf{b}(j,z) \equiv (P_1.\hat{\mathbf{y}}, P_2.\hat{\mathbf{y}}, \ldots, P_{j-1}.\hat{\mathbf{y}}, z, P_{j+1}.\hat{\mathbf{y}}, \ldots, P_n.\hat{\mathbf{y}})$
Create and initialise n one-dimensional PSOs : $P_j,\ j \in [1..n]$
repeat:
  for each swarm $j \in [1..n]$ **:**
    for each particle $i \in [1..s]$ **:**
      if $f(\mathbf{b}(j, P_j.\mathbf{x}_i)) < f(\mathbf{b}(j, P_j.\mathbf{y}_i))$
        then $P_j.\mathbf{y}_i = P_j.\mathbf{x}_i$
      if $f(\mathbf{b}(j, P_j.\mathbf{y}_i)) < f(\mathbf{b}(j, P_j.\hat{\mathbf{y}}))$
        then $P_j.\hat{\mathbf{y}} = P_j.\mathbf{y}_i$
    endfor
    Perform PSO updates on P_j using equations (1–2)
  endfor
until stopping condition is true

$f(\mathbf{b}(1, P_1.\hat{\mathbf{y}}))$ is a strictly nonincreasing function

Cooperative Learning V

- Advantage:
  - The error function f is evaluated after each component in the vector is updated.
- However:
  - Some components in the vector could be correlated.
  - These components should be in the same swarm, since the independent changes made by the different swarms have a detrimental effect on correlated variables.
  - Swarms of 1-D, and swarms of c-D, taken blindly.

Cooperative Learning VI

- **CPSO-S_k:**
  - Swarms of 1-D, and swarms of c-D, taken blindly, hoping that some correlated variables end up in the same swarm.
  - Split factor: The vector is split in K parts (swarms)
  - It is a particular CPSO-S case, where n=K.

Cooperative Learning VII

define

$\mathbf{b}(j,\mathbf{z}) \equiv (P_1.\hat{\mathbf{y}}, \ldots, P_{j-1}.\hat{\mathbf{y}}, \mathbf{z}, P_{j+1}.\hat{\mathbf{y}}, \ldots, P_K.\hat{\mathbf{y}})$

$K_1 = n \bmod K$

$K_2 = K - (n \bmod K)$

Initialise K_1 $\lceil n/K \rceil$-dimensional PSOs:

$P_j,\ \ j \in [1..K_1]$

Initialise K_2 $\lfloor n/K \rfloor$-dimensional PSOs:

$P_j,\ \ j \in [(K_1+1)..K]$

repeat:

  for each swarm $j \in [1..K]$ **:**

    for each particle $i \in [1..s]$ **:**

      if $f(\mathbf{b}(j, P_j.\mathbf{x}_i)) < f(\mathbf{b}(j, P_j.\mathbf{y}_i))$

        then $P_j.\mathbf{y}_i = P_j.\mathbf{x}_i$

      if $f(\mathbf{b}(j, P_j.\mathbf{y}_i)) < f(\mathbf{b}(j, P_j.\hat{\mathbf{y}}))$

        then $P_j.\hat{\mathbf{y}} = P_j.\mathbf{y}_i$

    endfor

    Perform PSO updates on P_j using (1–2)

  endfor

until stopping condition is true

CPSO-S

Cooperative Learning VII

– Drawback:
  - It is possible that the algorithm become trapped in a state where all the swarms are unable to discover better solutions: stagnation.
  - Authors show an example.

Hybrid CPSOs – CPSO-H_k I

- Motivation:
  - CPSO-S_k can become trapped.
  - PSO has the hability to scape from pseudominimizers.
  - CPSO-S_k has faster convergence.
- Solution:
  - Interleave the two algorithms.
    - Execute CPSO-S_k for one iteration, followeb by one iteration of PSO.
    - Information interchange is a form of cooperation.

define

CPSO-Sk

$\mathbf{b}(j,\mathbf{z}) \equiv (P_1.\hat{\mathbf{y}}, \ldots, P_{j-1}.\hat{\mathbf{y}}, \mathbf{z}, P_{j+1}.\hat{\mathbf{y}}, \ldots, P_K.\hat{\mathbf{y}})$
$K_1 = n \bmod K$
$K_2 = K - (n \bmod K)$
Initialise K_1 $\lceil n/K \rceil$-dimensional PSOs:
$P_j,\ j \in [1..K_1]$
Initialise K_2 $\lfloor n/K \rfloor$-dimensional PSOs:
$P_j,\ j \in [(K_1+1)..K]$

Initialise an n-dimensional PSO : Q

repeat:

CPSO-Sk

for each swarm $j \in [1..K]$ **:**
  for each particle $i \in [1..s]$ **:**
    if $f(\mathbf{b}(j, P_j.\mathbf{x}_i)) < f(\mathbf{b}(j, P_j.\mathbf{y}_i))$
      then $P_j.\mathbf{y}_i = P_j.\mathbf{x}_i$
    if $f(\mathbf{b}(j, P_j.\mathbf{y}_i)) < f(\mathbf{b}(j, P_j.\hat{\mathbf{y}}))$
      then $P_j.\hat{\mathbf{y}} = P_j.\mathbf{y}_i$
  endfor
  Perform PSO updates on P_j using (1–2)
endfor

Overwite 1 particle k of the swarm Q

Select random $k \sim U(1, s/2) \mid Q.\mathbf{y}_k \neq Q.\hat{\mathbf{y}}$
$Q.\mathbf{x}_k = \mathbf{b}(1, P_1.\hat{\mathbf{y}})$

PSO

for each particle $j \in [1..s]$ **:**
  if $f(Q.\mathbf{x}_j) < f(Q.\mathbf{y}_j)$
    then $Q.\mathbf{y}_j = Q.\mathbf{x}_j$
  if $f(Q.\mathbf{y}_j) < f(Q.\hat{\mathbf{y}})$
    then $Q.\hat{\mathbf{y}} = Q.\mathbf{y}_j$
endfor
Perform PSO updates on Q using (1–2)

Swarms of the CPSO-Sk

for swarm $j \in [1..K]$ **:**
  Select random $k \sim U(1, s/2) \mid P_j.\mathbf{y}_k \neq P_j.\hat{\mathbf{y}}$
  $P_j.\mathbf{x}_k = Q.\hat{\mathbf{y}}_j$
endfor

until stopping condition is true

Experimental Setup I

- Compare the PSO, CSPO-S_k, CSPO-H_k algorithms.
- Measure: #function evaluations.
- Several popular functions in the PSO comunity were selected for testing.

Experimental Setup II

The Rosenbrock (or banana-valley) function (unimodal)

$$f_0(\mathbf{x}) = \sum_{i=1}^{\frac{n}{2}} \left(100\left(x_{2i} - x_{2i-1}^2\right)^2 + (1 - x_{2i-1})^2\right).$$

The Quadric function (unimodal)

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j\right)^2 .$$

Ackley's function (multimodal)

$$f_2(\mathbf{x}) = -20 \exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e.$$

The generalized Rastrigin function (multimodal)

$$f_3(\mathbf{x}) = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right).$$

The generalized Griewank function (multimodal)

$$f_4(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1.$$

"all the functions where tested under coordinate rotation using Salomon's algorithm"

Experimental Setup III

- PSO configuration:
  - All experiments were run 50 times
  - 10, 15, 20 particles per swarm.
  - Results reported are averages os the best value in the swarm.

PARAMETERS USED FOR EXPERIMENTS

Function	n	domain	threshold
f_0	30	2.048	100
f_1	30	100	0.01
f_2	30	30	5.00
f_3	30	5.12	100
f_4	30	600	0.1

Domain: "magnitude to which the initial random particles are scaled"

Experimental Setup IV

- PSO: "plain" swarm using $c_1 = 1.49$, $c_2 = 1.49$, $w = 0.72$, and $v_{\max}$ is clamped to the domain, following Eberhart and Shi [17].
- CPSO-S: A maximally "split" swarm using $c_1 = 1.49$, $c_2 = 1.49$, w decreases linearly over time, and $v_{\max}$ is clamped to the domain (refer to Table I).
- CPSO-S_6: A "split" swarm using $c_1 = 1.49$, $c_2 = 1.49$, w decreases linearly over time, and $v_{\max}$ is clamped to the domain (refer to Table I). The difference between this swarm type and the split CPSO (above) is that the search-space vector for CPSO-S_6 is split into only six parts (of five components each), instead of 30 parts.
- CPSO-H: A hybrid swarm, consisting of a maximally split swarm, coupled with a plain swarm, described in Section III-A. Both components use the values $c_1 = 1.49$, $c_2 = 1.49$, w decreasing linearly over time, and $v_{\max}$ clamped to the domain (refer to Table I).
- CPSO-H_6: A hybrid swarm, consisting of a CPSO-S_6 swarm, coupled with a plain swarm, described in Section IV. Both components use the values $c_1 = 1.49$, $c_2 = 1.49$, w decreasing linearly over time, and $v_{\max}$ clamped to the domain (refer to Table I).

Experimental Setup V

- GA configuration:
  - GA: A standard genetic algorithm, with parameters specified below.
  - CCGA: A cooperative genetic algorithm [4], where the search-space vector is maximally split so that each component belongs to its own swarm. For the functions tested here, this implies that 30 populations were employed in a cooperative fashion.

Experimental Setup VI

The parameters for both types of GA are as follows.

- Chromosome type: binary coded.
- Chromosome length: 48 bits per function variable.
- Crossover probability: 0.6.
- Crossover strategy: Two-point.
- Mutation probability: $1/(48 \times 30)$, assuming 30 variables per function.
- Fitness scaling: Scaling window of length 5.
- Reproduction strategy: Fitness-proportionate with a 1-element elitist strategy.
- Population size: 100.

Results I

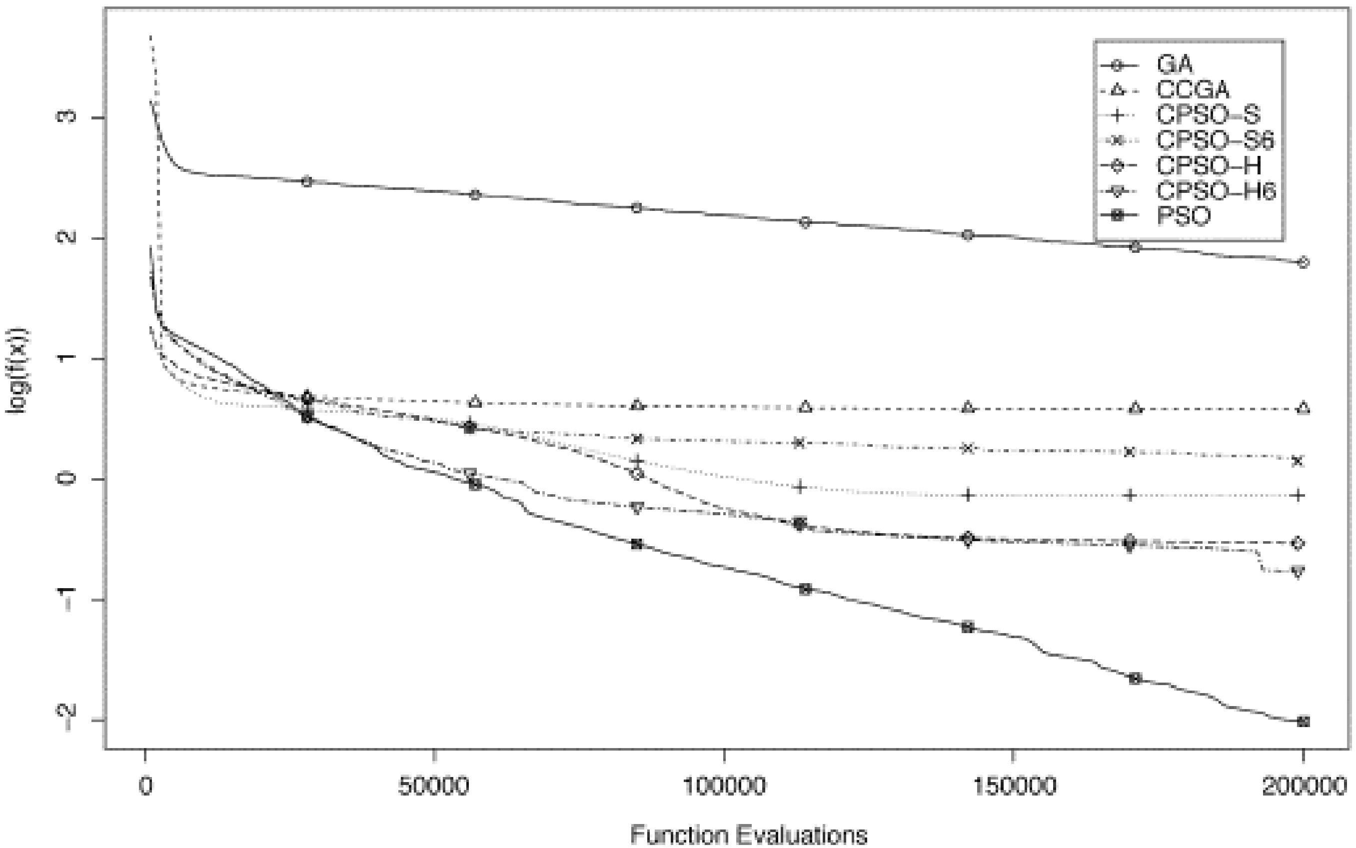
- Fixed-Iteration Results I
  - 2.10^5 function evaluations.

ROSENBROCK (f_0) AFTER 2×10^5 FUNCTION EVALUATIONS

Algorithm	s	Mean(Unrotated)	Mean(Rotated)
PSO	10	1.30e−01 ± 1.45e−01	3.32e−01 ± 9.50e−02
	15	5.53e−03 ± 6.19e−03	2.84e−01 ± 5.17e−02
	20	9.65e−03 ± 7.28e−03	3.16e−01 ± 3.41e−02
CPSO-S	10	7.58e−01 ± 1.16e−01	3.23e+00 ± 7.78e−01
	15	7.36e−01 ± 3.04e−02	2.58e+00 ± 5.36e−01
	20	9.06e−01 ± 3.56e−02	4.37e+00 ± 8.51e−01
CPSO-H	10	2.92e−01 ± 2.19e−02	4.26e−01 ± 3.83e−02
	15	3.14e−01 ± 1.74e−02	4.96e−01 ± 4.53e−02
	20	4.35e−01 ± 2.48e−02	1.06e+00 ± 2.96e−01
CPSO-S_6	10	1.41e+00 ± 4.73e−01	2.65e+00 ± 6.69e−01
	15	2.47e+00 ± 7.00e−01	3.84e+00 ± 9.81e−01
	20	1.59e+00 ± 5.03e−01	4.27e+00 ± 7.73e−01
CPSO-H_6	10	1.94e−01 ± 2.63e−01	1.77e−01 ± 3.62e−02
	15	2.59e−01 ± 2.47e−01	3.73e−01 ± 2.07e−01
	20	4.21e−01 ± 3.21e−01	4.73e−01 ± 1.35e−01
GA	100	6.32e+01 ± 1.19e+01	6.15e+01 ± 1.42e+01
CCGA	100	3.80e+00 ± 1.93e−01	1.32e+01 ± 2.19e+00

Results II

- Fixed-Iteration Results II
  - 2.10^5 function evaluations.



Results III

- Fixed-Iteration Results III
  – PSO-based algs. performed better that GA algs. in general.
  – Cooperative algorithms collectivelly performed better than the standard PSO in 80% of the cases.

Results IV

- Robustness and speed Results I
  - “**Robustness**”: the algorithm succeed in reducing the the f below a specified threshold using fewer that than a number of evaluations.
  - “**A robust algorithm**”: one that manages to reach the threshold consistentle (during all runs).

Results V

- Robustness and speed Results II

QUADRIC (f_1) ROBUSTNESS ANALYSIS

Algorithm	s	Unrotated		Rotated	
		Succeeded	Fn Evals.	Succeeded	Fn Evals.
PSO	10	38	34838	0	N/A
	15	50	16735	1	26161
	20	50	14574	2	175788
CPSO-S	10	50	70215	0	N/A
	15	50	77265	0	N/A
	20	50	83168	0	N/A
CPSO-H	10	50	40056	0	N/A
	15	50	53341	0	N/A
	20	50	61430	0	N/A
CPSO-S_6	10	50	77818	0	N/A
	15	50	101565	0	N/A
	20	50	115687	0	N/A
CPSO-H_6	10	50	22200	1	126271
	15	50	31503	0	N/A
	20	50	43918	0	N/A
GA	100	0	N/A	0	N/A
CCGA	100	0	N/A	0	N/A

Results VI

- Robustness and speed Results III
  - CPSO-H_6 appears to be the winner because it achieved a perfect score in 7 of 10 cases.
  - There is a tradeoff between the convergence speed and the robustness of the algorithm.