Particle Swarm Optimization (PSO) for various Objective Functions

BHAVIK BHAGAT
St. Francis Xavier University
Department of Computer Science
Antigonish, Nova Scotia, Canada
x2020coq@stfx.ca

***Abstract*—I am proposing Particle Swarm Optimization (PSO)[9] algorithm to solve a number of Optimization problem. The Objective Function is the problem to be optimized. Some of the function that are used are given as the part of the library *pyswarms*[1]. PSO is ubiquitous algorithms amongst the optimization technique because it can be applied to any type of optimization problem it is not necessary for the problem to be a differentiable[3] problem as compared to other classical optimization techniques. Hence the technique is very versatile and suitable for various problem.**

The idea of PSO is to emulate the social behaviour of birds and fishes looking for the food. Particles (each bird/fish) is looking for the food in the given search space. Their movement are affected by two factors, 1) their desire to find the food individually 2) the collective action of the group. Their position is updated by communicating with one another constantly in order to find the food. The goal is to find global optimum (food). The group is technically called swarm here. Thus it is analogous to real world techniques used by those species, and one of the many techniques of Evolutionary Computation.

***Index Terms*—Evolutionary Computation; Particle Swarm Optimization; Exploitation vs Exploration; Optimization Problems; Swarm Intelligence**

I. Introduction

Particle Swarm Optimization (PSO) is a method for solving problems by iteratively trying to improve a candidate solution against a fitness function. We move each particle in the search space according to their position and velocity equations. The particle's movement is influenced by its local best known position, as well as the global best known position in the search-space. The global best position is determined by communicating the best known local position of number of particles. Eventually swarm is expected to move towards this global best solution. This will be discussed in details later in the methodII section. This video provides nice explaination of how it works.

The PSO is originally attributed to James Kennedy[7], Russell C. Eberhart[2] and Yuhiu Shi[11], who used it for simulation of social behavior of bird flock or fish school as mentioned in abstract. On simplification, it was observed to perform optimization task underneath. Their book describes the philosophical aspects of PSO and swarm intelligence.

PSO is a metaheuristic[8], as it makes few, if any, assumptions about the problem to be solved and can search a large space of candidate solutions. It doesn't use gradient[4] of the function, so the objective function to be optimized need not to be differentiable, hence it can work with any kind of problems compared to classical techniques like gradient_descent[5] or quasi_newton[10] methods. However, metaheuristics algorithm such as PSO doesn't guarantee an optimal solution is ever found.

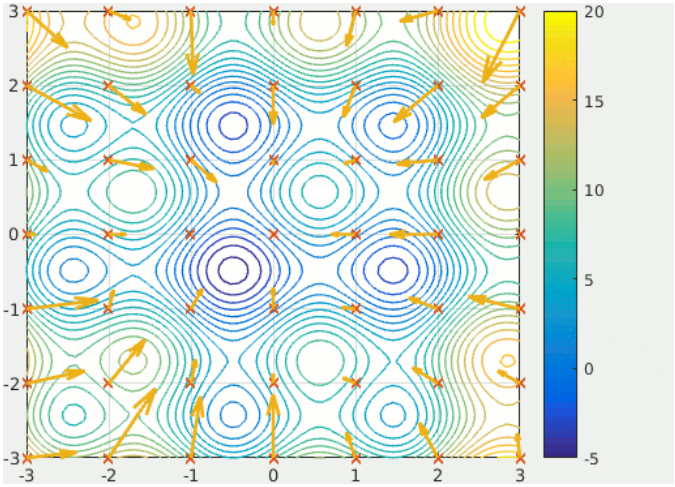

Fig. 1: animation of particle exploring the search space

II. Methods

We have to form a mathematical model, to make the swarm find the global optima using the above principles of swarm intelligence.

Mathematical Model:

- Each particle in the swarm has a position, velocity vector and a fitness value
- Each particle records its own best position and the best fitness value. Check figure 2
- By communication with other, the record of global best position and global best fitness value is recorded. Check figure 3
- Starting at random, with each iteration, all the particles in the swarm try to move towards the global optima, this improving by each iteration towards the solution.

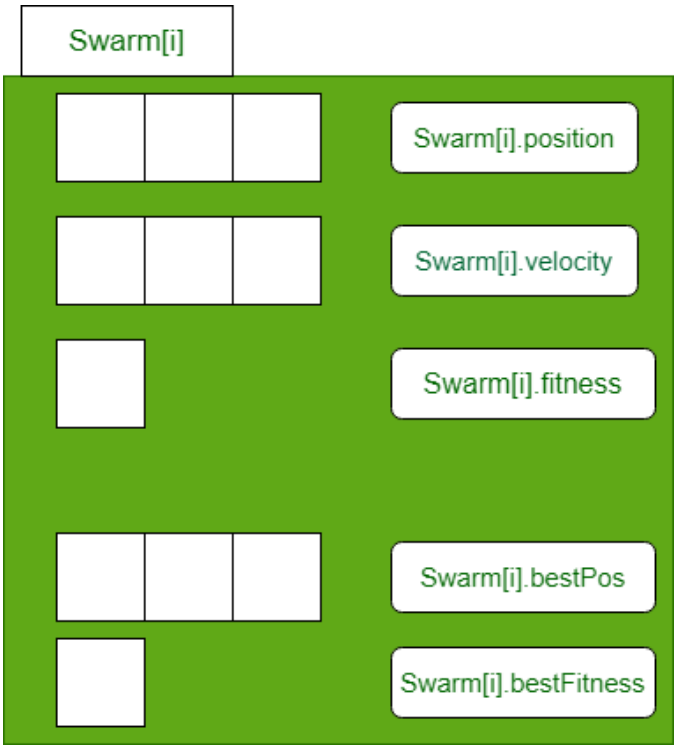

Fig. 2: particle records

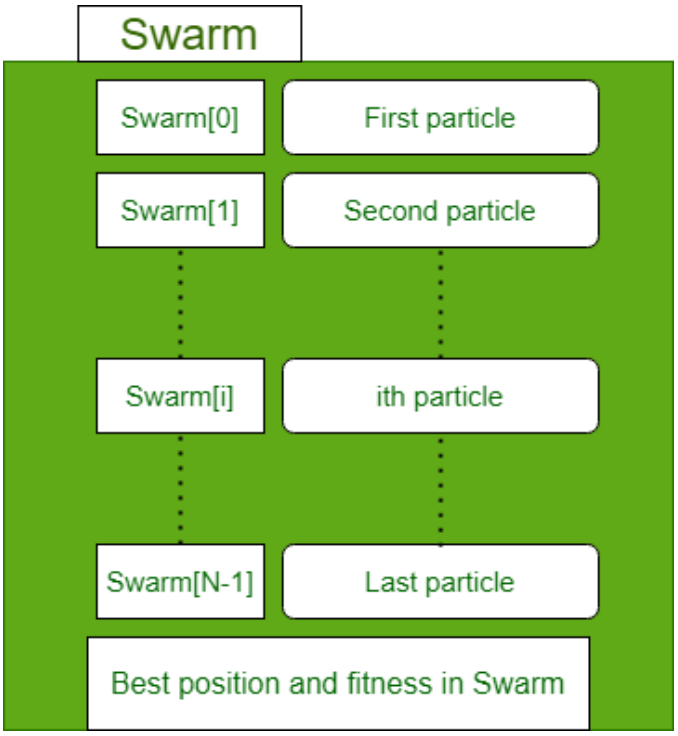

Fig. 3: swarm records

```
Step1: Randomly initialize Swarm population of N particles Xi ( i=1, 2, …, n)
Step2: Select hyperparameter values
          w, c1 and c2
Step 3: For Iter in range(max_iter):  # loop max_iter times
           For i in range(N):  # for each particle:
              a. Compute new velocity of ith particle
                   swarm[i].velocity =
                        w*swarm[i].velocity +
                        r1*c1*(swarm[i].bestPos - swarm[i].position) +
                        r2*c2*( best_pos_swarm - swarm[i].position)
              b. If velocity is not in range [minx, max] then clip it
                   if swarm[i].velocity < minx:
                       swarm[i].velocity = minx
                   elif swarm[i].velocity[k] > maxx:
                       swarm[i].velocity[k] = maxx
              c. Compute new position of ith particle using its new velocity
                   swarm[i].position += swarm[i].velocity
              d. Update new best of this particle and new best of Swarm
                    if swarm[i].fitness < swarm[i].bestFitness:
                       swarm[i].bestFitness = swarm[i].fitness
                       swarm[i].bestPos = swarm[i].position

                    if swarm[i].fitness < best_fitness_swarm
                       best_fitness_swarm = swarm[i].fitness
                       best_pos_swarm = swarm[i].position
            End-for
        End -for
Step 4: Return best particle of Swarm
```

Fig. 4: PSO Algorithm

The algorithm is described in the figure4 taken from this link: PSO from Geeks-for-Geeks.

The parameters of the algorithm are as follows:

- Number of dimensions (**d**)
- Lower bound (**minx**)
- Upper bound (**maxx**)

With that algorithms, there are some parameters we can tweak to get the best result (i.e. optimum solution). This technique is called *hyper parameter tuning*[6]. This is a painful task to tune the parameters. Professor Dr. James Hughes mentioned several times in the classes not to try to tweak so much so that you can geek out the best result which the employer wouldn't understand. In practice, often the fairly acceptable result is the one that works. Hence, I didn't try to tweak so many settings this times like the other problems Genetic Algorithm for TSP and Genetic Programming for Symbolic Regression. Instead, I am trying to make the PSO work for a number of different objective functions.

The list of hyper-parameters is as follows:

1) Number of particles (**N**)
2) Maximum number of iterations (**max_iter**)
3) Inertia component (**w**)
4) Cognition of particle component - local (**C1**)
5) Social influence of swarm component - global (**C2**)

We have **N** particles. Let's say we denote position of particle i at time t as $X^i(t) = (x^i(t), y^i(t))$ where $(x(t), y(t))$ are the *position coordinates* of particle i on the Cartesian coordinate plane.

Then we can compute the position at time $t+1$ using the equation:

$$X^i(t+1) = X^i(t) + V^i(t+1)$$

Where $V^i(t+1)$ is the Velocity of the particle i at time $t+1$, which is computed using the equation:

TABLE I: This is a Table

function	w	C_1	C_2	cost	final position
Sphere	0.6	0.3	0.2	0.0004	$(0.063, -0.0058)$
Beale	0.8	0.5	0.5	0.00	$(2.9999, 0.5)$
Booth	0.7	0.3	0.4	0.1096	$(1.2379, 2.7704)$
Matyas	0.9	0.4	0.3	0.0000	$(0.0002, 0.0001)$
Levi	0.5	0.2	0.2	0.0000	$(1.0, 0.9983)$
Eggholder	0.8	0.2	0.3	-62.513	$(6.0788, 10.8024)$
Schaffer2	0.9	0.2	0.4	0.0000	$(0.0002, -0.0002)$
Rastrigin	0.7	0.2	0.5	0.0000	$(0.0000, -0.0000)$
Threehump	0.9	0.4	0.4	0.0000	$(0.0001, 0.0005)$
Rosenbrock	0.8	0.2	0.4	0.0001	$(1.0085, 1.7071)$

$$V^i(t+1) = wV^i(t) + C_1 r_1 (P^i_{best} - X^i(t)) + C_2 r_2 (G_{best} - X^i(t)$$

The P^i_{best} is the personal best position of particle i and G_{best} is the global known best position.

This velocity part has 3 components. These are the same components I mentioned at the beginning of this section.

1) **Inertia Component:** $wV^i(t)$ The particle i wants to maintain its velocity and keep moving as it was doing already, it keeps some portion of the current velocity **w** for the next time stamp.
2) **Cognitive Component:** $C_1 r_1 (P^i_{best} - X^i(t))$ Influence of personal best known position on the velocity. If C_1 is high, it exploit the local parts of the search space, behaving like an introvert particle.
3) **Social Component:** $C_2 r_2 (G_{best} - X^i(t))$ Influence of global best known position on the velocity. If C_2 is high, it exploit the global parts of the search space, behaving like an extroverted particle.

III. Results and Discussion

Generated result with random initialization of all 3 velocity components with 100 iterations and 10 particles. The cost and the final position are calculated. The results for the different objective functions are shown in the table below. Plots are plotted for the cost function. The animation are generated for all those functions. I don't know how to put gif files here, so I am saving all the results in one folder.

Table I.

A. Functions

Several Functions Equations I used are as follows:

- Sphere: $f(x) = \sum_{i=1}^{n} x_i^2$
- Beale: $f(x, y) = (1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$
- Booth: $f(x, y) = (x + 2y - 7)^2 + (2x + y - 5)^2$
- Matyas: $f(x, y) = 0.26(x^2 + y^2) - 0.48xy$
- Levi: $f(x, y) = sin^2 3\pi x + (x-1)^2(1 + sin^2 3\pi y) + (y-1)^2(1 + sin^2 2\pi y)$

B. Plots

Several Functions Cost Plots are as Follows. Almost all cost plots are looking the same, hence I am not putting all the figures. But the table shows the overall result of the functions. All the plots can be found in the results folder along with submission and their respected gif file that shows the visualization part.

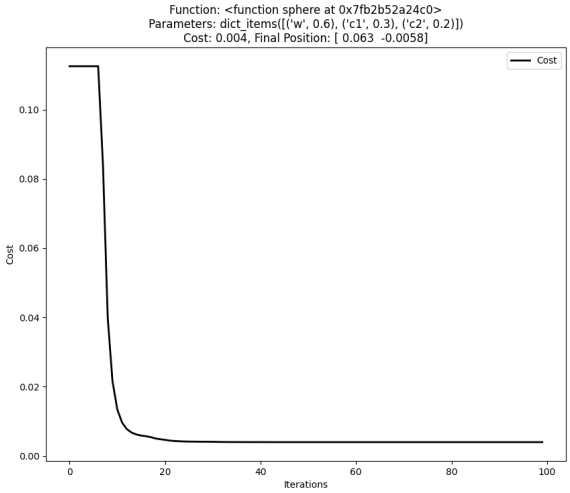

Fig. 5: Sphere

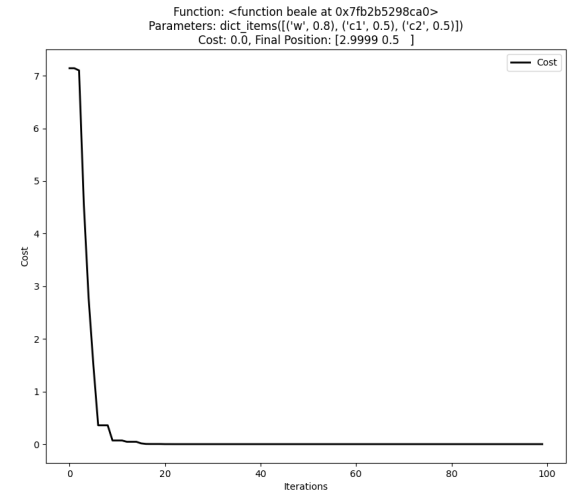

Fig. 6: Beale

IV. Modifications

I have tried 2 modifications here.

1) Neighborhood
2) Velocity Clamp

For local optimization I tried with different number of neighbors. Neighbors are ranging from 3 to 20. The results are plotted and save in the results folder. I could not attach the gif file here.

Table II.

TABLE II: This is a Table

function	w	C_1	C_2	K	P	cost	final position
Sphere	0.9	0.5	0.3	7	2	0.179	$(0.423, 0.0093)$
Beale	0.7	0.3	0.2	4	2	4.5039	$(1.0788, 0.2639)$
Booth	0.8	0.3	0.4	10	2	0.8775	$(1.5646, 2.3019)$

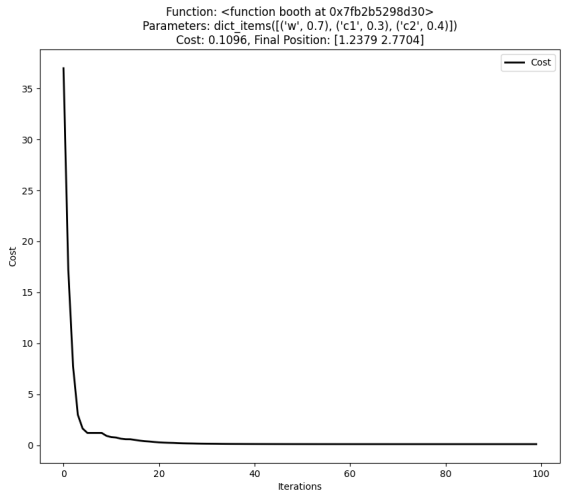

Fig. 7: Booth

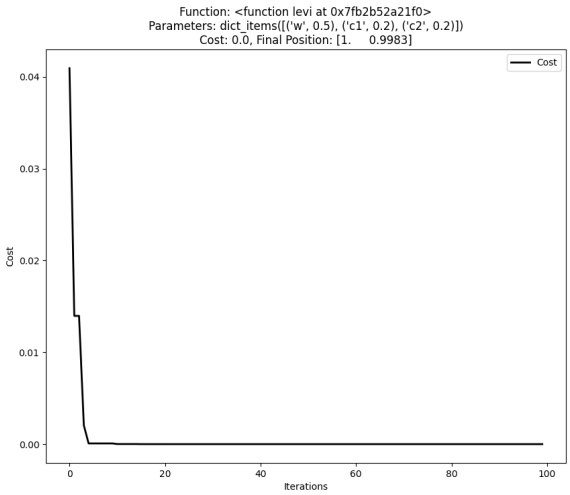

Fig. 9: Levi

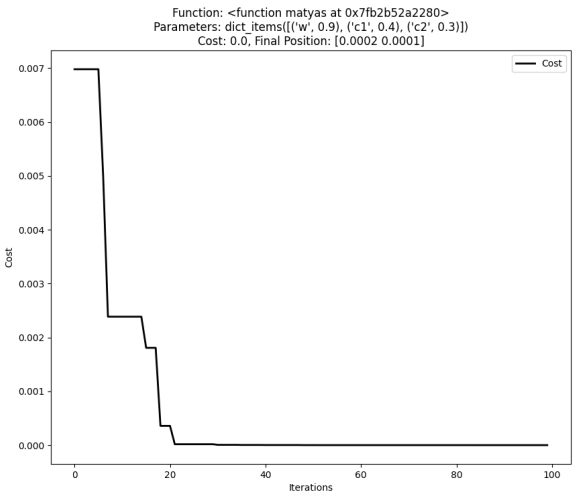

Fig. 8: Matyas

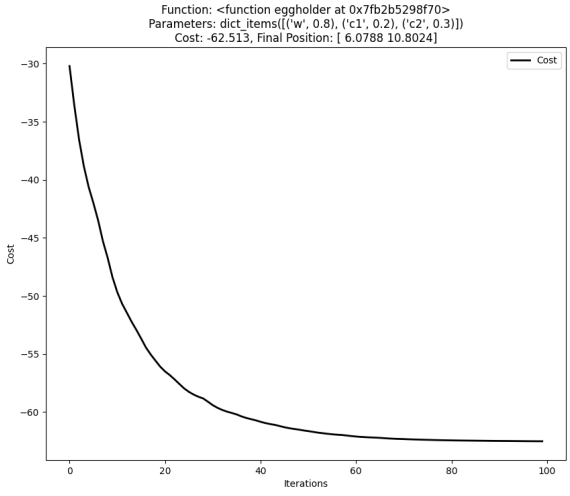

Fig. 10: Eggholder

V. Conclusions and Future Work

With more number of particles, algorithms converges rapidly. This can be shown using the animation files I have created. I couldn't attach those files here, but they are stored in the results folder to check the visualization part of the convergence.

With parameters $w = 0.9$, $C_1 = 0.5$, $C_2 = 0.4$, PSO converges very nicely for most of the functions I have used above.

With Velocity Clamping in effect, $V_{min} = 0.1$ & $V_{max} = 0.3$ gives the better results for most of the functions.

For Local Optimization task, with Neighborhood Modification, number of neighbors increase gives the better results locally.

References

[1] Lester James V. Miranda. Pyswarms, 2021. [Onine; accessed 22-Nobember-2021].

[2] Wikipedia contributors. Russell c. eberhart — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Russell_C._Eberhart&oldid=985511968, 2020. [Online; accessed 22-November-2021].

[3] Wikipedia contributors. Differentiable function — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Differentiable_function&oldid=1048851162, 2021. [Online; accessed 22-November-2021].

[4] Wikipedia contributors. Gradient — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Gradient&oldid=1048090185, 2021. [Online; accessed 22-November-2021].

[5] Wikipedia contributors. Gradient descent — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=1054928479, 2021. [Online; accessed 22-November-2021].

[6] Wikipedia contributors. Hyperparameter optimization — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=

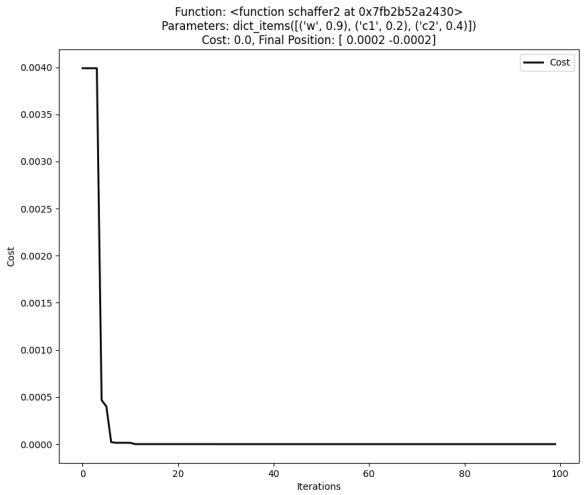

Fig. 11: Schaffer2

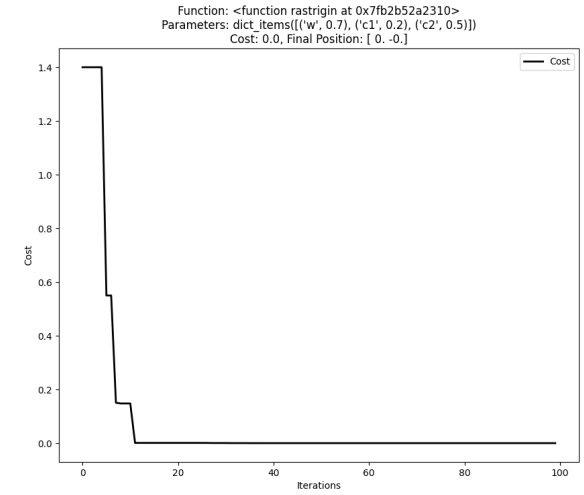

Fig. 12: Rastrigin

Hyperparameter_optimization&oldid=1037728107, 2021. [Online; accessed 22-November-2021].

[7] Wikipedia contributors. James kennedy (social psychologist) — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=James_Kennedy_(social_psychologist)&oldid=1055346190, 2021. [Online; accessed 22-November-2021].

[8] Wikipedia contributors. Metaheuristic — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Metaheuristic&oldid=1050320840, 2021. [Online; accessed 22-November-2021].

[9] Wikipedia contributors. Particle swarm optimization — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Particle_swarm_optimization&oldid=1050772675, 2021. [Online; accessed 22-November-2021].

[10] Wikipedia contributors. Quasi-newton method — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Quasi-Newton_method&oldid=1054625144, 2021. [Online; accessed 22-November-2021].

[11] Wikipedia contributors. Yuhui shi — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Yuhui_Shi&oldid=1041679284, 2021. [Online; accessed 22-November-2021].

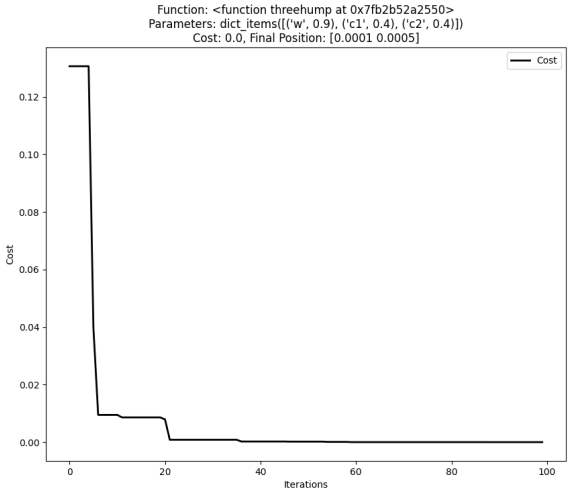

Fig. 13: Threehump

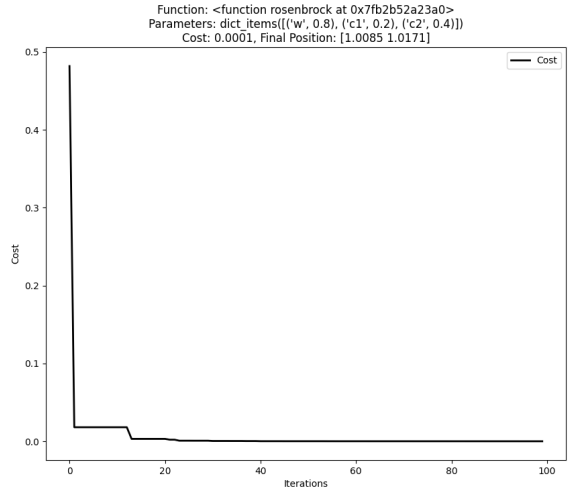

Fig. 14: Rosenbrock