Particle Swarm Optimization (PSO)

For a basic PSO, the velocity of a particle at time (t+1) is calculated as follows:

$$\vec{v}(t+1) = \vec{v}(t) + (c_1 * r_1 * (\vec{p}(t) - \vec{x}(t)) + (c_2 * r_2 * (\vec{g}(t) - \vec{x}(t))$$

p is the best position the particle has seen this far.
g is the best position all the particles has seen this far (if fully connected).

After finding the new velocity, a particles position is updated as follows:

$$\vec{x}(t+1) = \vec{x}(t) + \vec{v}(t+1)$$

r1 and r2 are random values in the range $[0, 1)$.
c1 is a weight for the local attraction in the range $[0, 2)$.
c2 is a weight for the global attraction in the range $[0, 2)$.

General

You will have to demonstrate several parts of the code. In addition you should write a report containing all tables, figures and plots. The code must be delivered together with the report when demonstrating. Ask the student assistants about booking time for your demonstration.

Task 1: 30%

The circle problem is defined as: f(u1,…,ux) = ((u1 * u1) + … + (ux * ux))

a) 20%
Implement your own basic PSO with a fully connected social topology (all particles are connected/aware of all other particles). Se lecture notes for details. Use your PSO to solve the 1D circle problem.

Demonstrate your implementation by running until the best global fitness is less then 0.001, or 1000 iterations have executed (print out the best global fitness).

b) 5%
Run your implementation 10 times. Make a table to report the fitness or the iteration number when the algorithm terminates.

c) 5%
Extend your implementation to solve the 2D circle problem.
Demonstrate your implementation for the 2D problem, by running until the best global fitness is less then 0.001, or 1000 iterations have executed (print out the best global fitness).

Task 2: 20%

a) 10%
Add support to the basic PSO for a nearest neighbour topology. A particle should use only the 3

nearest neighbours when computing the social part of velocity update. Plot the global best fitness of your implementation solving the 1D and 2D circle problem in the same figure.

b) 10%
Add support for inertia weight w to the PSO. Velocity update is then:

$$\vec{v}(t+1) = (w * \vec{v}(t)) + (c_1 * r_1 * \left(\vec{p}(t) - \vec{x}(t)\right) + (c_2 * r_2 * (\vec{g}(t) - \vec{x}(t))$$

Let the inertia weight decrease from 1.0 toward 0.4 during the execution of the PSO. Plot the global best fitness of your implementation solving the 1D and 2D circle problem in the same figure.

Task 3: 30%

a) 20%
Implement a basic, fully connected PSO, to solve the following problem:

> A small container has a weight limit of 1000kg.
> A package has a weight and a value.
> You have a sett of 2000 packages to select from, as defined in the file packages.txt
> The file packages.txt has two parameters per line, first parameter is value, second parameter is weight
> Select packages to get fill your container with the most value while still being under the weight limit.

Demonstrate a run of your algorithm that terminates after 500 iterations.

b) 5%
Run the PSO 3 times using different c1,c2 values.
Plot the best global fitness for each run in the same figure.

c) 5%
Implement support for inertia weight. Let the inertia weight decrease from 1.0, toward 0.4 during the execution of the PSO. Run the PSO 3 times using different c1 and c2 values.
Plot the best global fitness for each run in the same figure.

Note! This is known as the 0-1 knapsack problem.

Task 4: 20%

a) 20%
Extend and use your PSO to solve the 0-1 knapsack problem where a package has weight, value and volume. Assume your container can contain 1000kg and 1000m3. Use values from the file packages.txt, and for each line/package, add a random volume between 1m3 and 5m3.

Demonstrate a run of your implementation terminating after 1000 iterations.