Project 3 in Artificial Intelligence Programming - IT3105

Knut Andreas Hasund

November 25, 2013

Abstract

An implementation of a Particle Swarm Optimization algorithm.

The Circle Problem

The circle problem is defined as

$$f(x_1, \ldots, x_n) = x_1^2 \cdot \ldots \cdot x_n^2 \tag{1}$$

where n is the number of spatial dimensions. Given some closed subspace $\mathbb{D} \subset \mathbb{R}^n$, the optimization problem on this subspace then becomes

$$\arg\min_{\mathbf{x} \in \mathbb{D}} f(\mathbf{x}). \tag{2}$$

It is easy to see that this problem is convex with only one optimal solution in $\mathbf{x} = [0, \ldots, 0]$.

Task 1

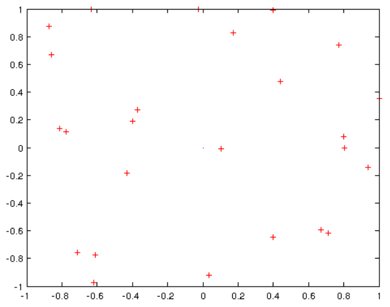
An **PSO** (*Particle Swarm Optimization algorithm*) has been implemented using the instructions provided by the assignment. For each iteration two steps is carried out

$$v_k^{t+1} = w \cdot v_t^k + \left[c_1 \cdot \mathrm{U}_{(0,1)} \cdot \left(p_k^t - x_k^t\right)\right] + \left[c_2 \cdot \mathrm{U}_{(0,1)} \cdot \left(g_k^t - x_k^t\right)\right] \tag{3}$$

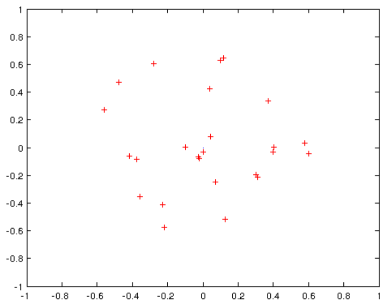
$$x_k^{t+1} = x_k^t + v_k^{t+1} \tag{4}$$

where $k \in \{0, \ldots, n\}$, t is the iteration number and $\mathrm{U}_{(0,1)}$ is a random scalar on the interval $(0, 1)$. The rest of the notation is equal to the one provided by the assignment.

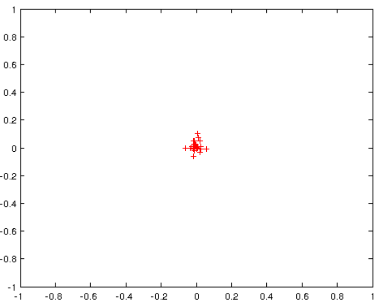
I have chosen to interpret the **BGF** (*best global fitness*) as the best known state among all particles, across all previous iterations. Thus it becomes the best state yet encountered by the algorithm. An alternative would be to let the **BGF** be the sum or mean of each particles induvidually encountered best state. When using **BGF** as a stopping criterion for the algorithm, this would make it less probable that the algorthm stops in a local minima and instead lets the particles «settle». An illustration of how the algorithm would behave with the last case **BGF** as a stopping criterion can be seen in figure 1, as an contrast to figure 2 where the former definition has been used. Since (2) is a convex problem, this will not be an issue and it is therefore safe to use the first definition.



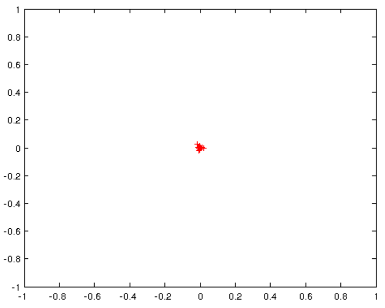
(a) Iteration 0



(b) Iteration 4

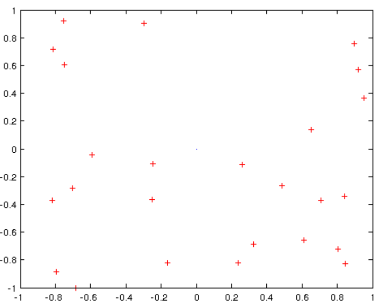


(c) Iteration 11

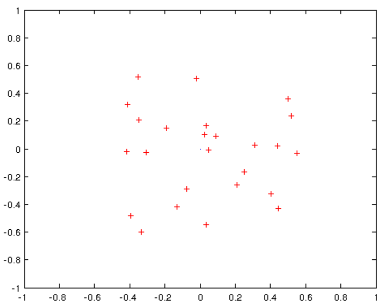


(d) Iteration 15, **BGF** criterion reached

Figure 1: (2) solved using **PSO**, $n = 2$, global sum **BGF**



(a) Iteration 0



(b) Iteration 4, **BGF** criterion reached

Figure 2: (2) solved using **PSO**, $n = 2$, single node **BGF**

a)

The algorithm has been implemented with a fully connected social topology and $w = 1$.

b)

The algorithm has been run 10 times using the parameters in table 1. The results can be seen in table 2. One can observe that the difference in iterations used is quite high. This is probably due to how good the initial configuration was, since it is generated randomly. The effect is expected to fall off as n increases, as the probability of generating a particle close to the origo gets much lower.

x_k	v_k	c_1	c_2	n	#particles
$(-1000, 1000)$	$(-100, 100)$	1	1	1	25

Table 1: Parameters for **PSO**

Iterations used	**BGF**
6	0.000025760
11	0.000000000
31	0.000934065
60	0.000012673
65	0.000278281
74	0.000055350
80	0.000164895
142	0.000230701
155	0.000196483
200	0.000117011

Table 2: PSO with fully connected topology run 10 times on (2), sorted by iterations used.

c)

With the same configuration as **b)** but with $n = 2$ the **PSO** finished in 1000 iterations with a **BGF** of 0.0643135.

Task 2

a)

A nearest neighbour topology has been implemented using the Euclidean distance

$$||\mathbf{z}|| = \sqrt{\mathbf{z}^*\mathbf{z}} = \sqrt{z_0^2 + \ldots + z_n^2}, \quad \mathbf{z} = \mathbf{x} - \mathbf{y} \tag{5}$$

as a distance measurement. Generating a particles neighbours is done by searching through all other particles and finding the 3 clostest. This is done for each particle each turn thus increasing the complexity of the algorithm by the square of the number of particles each iteration.

The best idea I have had so far on how to reduce complexity, is to divide the domain into subdomains, where each subdomain only contains a up to a certain amount of particles. Each subdomain would have to be able to split up into two or more new domains if the amount of particles should become too high in that area. Then it should be possible to search only in the closest subdomains for neighbours, thus with proper data structures, one could (hopefully) reduce the complexity to some degree. Unfortunately I did not have enough time to implement this to see if it would work. An illustration is provided in figure 3.

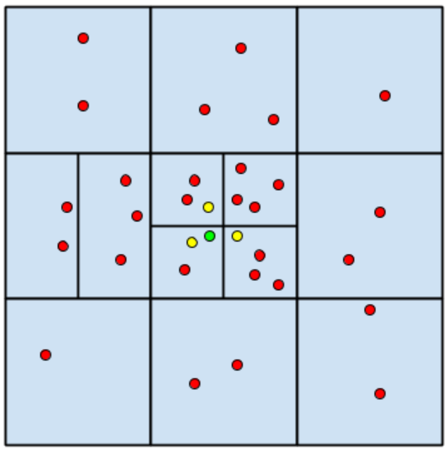


Figure 3: Fictive neighbourhood generation utopia

A plot of a run with the **PSO** utilizing the new topology can be seen in figure 4. The **PSO** configuration is equal to that of table 1 with $n = 1$ and $n = 2$. Since we are solving a convex problem, it is reasonable to expect a poorer convergence using the neighbourhood topology (since the **BGF** always is closer to the actual solution than any other point). Another interesting thing to point out is that when running the **PSO** with $n = 2$, the last $\sim 90\%$ of the iterations iterations provides no improvement to the **BGF**. This is probably due to the velocity v being too high, thus not having high enough accuracy to converge further.

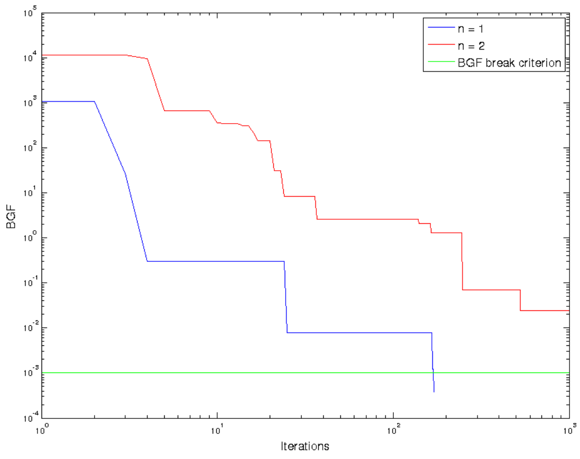


Figure 4: Neighbour topology

b)

I have chosen to change w based on the relative change in the **BGF**

$$w^t = 0.4 + (1.0 - 0.4)\frac{\mathbf{BGF}^t}{\mathbf{BGF}^0}, \tag{6}$$

where $\mathbf{BGF}^0$ is the initial **BGF**. This should provide a linear relation between the **PSO** convergence and the reduction of w. The same configuration as **a)**, but with w as defined in (6) has been used in figure 5.

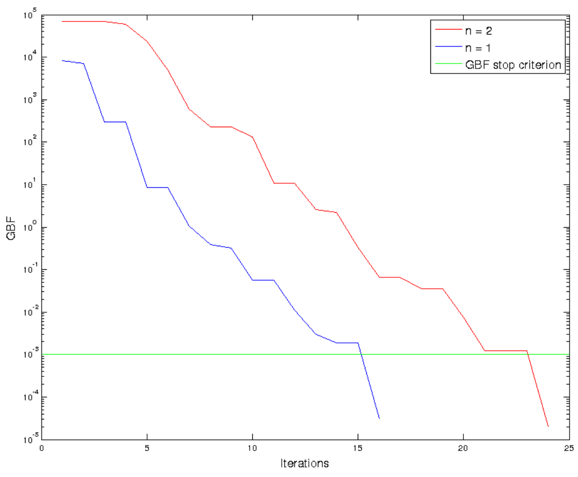


Figure 5: Decreasing velocity weight

Task 3

a)

The 1-0 knapsack problem has been implemented using the same framework as the implementation in task 1 with some slight changes. The optimization problem is now

$$\arg\max_{\mathbf{x}\in\{0,1\}^n} value(\mathbf{x}) \iff \arg\min_{\mathbf{x}\in\{0,1\}^n} \left[-value(\mathbf{x})\right], \tag{7}$$

where $value(\mathbf{x})$ is the total value of the packages selected. In addition we have the constraint

$$weight(\mathbf{x}) < 1000 \tag{8}$$

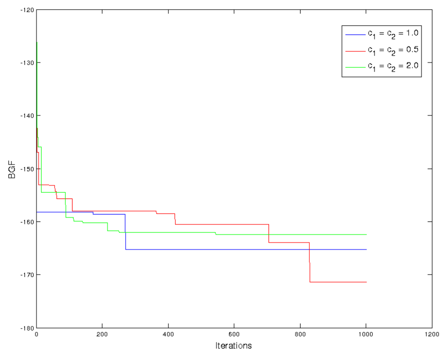
where $weight(\mathbf{x})$ is the total weight of all the packages selected. Both the values and the weights of each package is stored in a text file.

When initializing a particle a random bounded number of packages is randomly selected. The number of packages selected has an upper boundary decided the package average weight plus some number related to the package maximum weight.

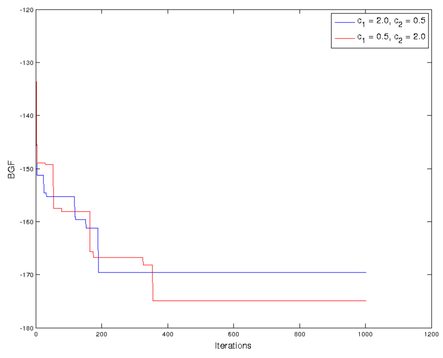
In order to make sure (8) is satisfied, each state particle that breaks the constraint is reinitialized. I first tried giving the invalid states a negative total value, but that was not as effective.

b)

Plots with different configurations of c_1 and c_2 can be seen in figure 6.



(a) $c_1 = c_2$



(b) $c_1 \neq c_2$

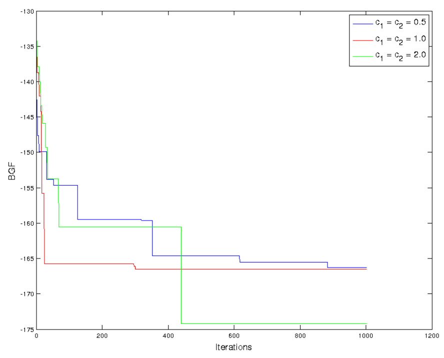
Figure 6: 1-0 knapsack over 1000 iterations using **PSO**, negative **BGF** is better. $\#particles = 100, n = 2001, v \in [-4.25, 4.25], w = 1.0$.

c)

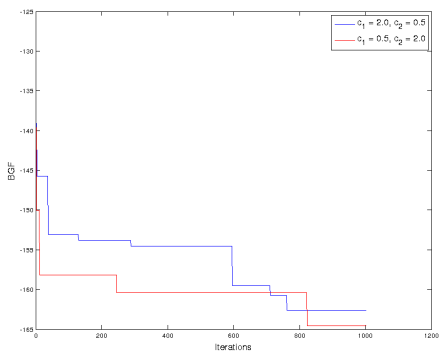
Since this problem is far off from convex (unless the package data is constructed to be that way) we are not guaranteed convergence. Therefore it is more reasonable to govern the velocity weight using the current and maximum number of iterations.

$$w(t) = 0.4 + (1.0 - 0.4)\frac{t_{max}}{t_{max} - t}. \tag{9}$$

Plots with different configurations of c_1 and c_2 can be seen in figure 7.



(a) $c_1 = c_2$



(b) $c_1 \neq c_2$

Figure 7: 1-0 knapsack over 1000 iterations using **PSO**, negative **BGF** is better. $\#particles = 100, n = 2001, v \in [-4.25, 4.25]$, $w(t)$.

Task 4

The implementation of the volume constraint was analogous to the weight constraint in every aspect. Each particle now needs to hold both constraints to not get reinitialized.