Genetic Algorithms (ATHENS course): Takehome project report

E. Sidiropoulos
C. Evangelides

Dieter Castel

April 22, 2014

Contents

1 Introduction **2**

2 Objective function **2**

3 Search space specification **4**

4 Genetic Algorithm **4**
4.1 code 4
4.2 results 5
4.3 conclusion 5

5 Particle Swarm Optimization **5**
5.1 code 5
5.2 results 6
5.3 conclusion 6

6 Utility functions **6**

List of Figures

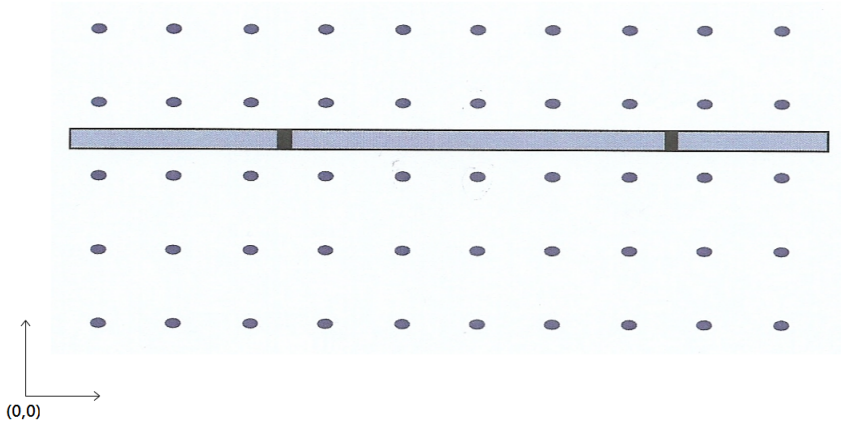

Figure 1: Graphical representation of the town with the 50 possible locations. Each dot represents a possible building site for the fire unit. The rectangle in the center of the image is the river with the two bridges at $(3.5, 3.5)$ and $(8.5, 8.5)$

1 Introduction

The assignment describes the problem a fire department would have if they want to select the best location for placing a fire unit. There are 50 locations suitable and those are spaced out evenly on a grid as shown in figure 1. The river in the center of the town has two bridges. These bridges should be taken into account when calculating the weight each point gets according to the objective function. The objective function models the response time to all these locations according to the distance. In the next section we will look at this function in more detail.

2 Objective function

The objective function for this particular problem is defined as the summed distance to each of the possible building sites. The euclidean distance is being used and the bridges are taken into account. The euclidean distance is proffered over an other distance (e.g. hamming distance) since we have no further information about the layout of the town. The symbolic notation of the objective function is the following:

$$f((a,b)) = \Sigma_{x \in i, y \in j}(x-a)^2 + (y-b)^2$$
$$\text{with } i = 1, \ldots, 10 \text{ and } j = 1, \ldots, 5$$

The matlab implementation of the objective function f can be found in listing 1 on page 6.

In figure 2 you can find a graphical representation of the objective function f. The choice to represent the function in a mesh grid is intentional. Because creating a mesh grid isn't as calculation intensive as a more precise representation, a rough idea can quickly be formed of the function being researched.

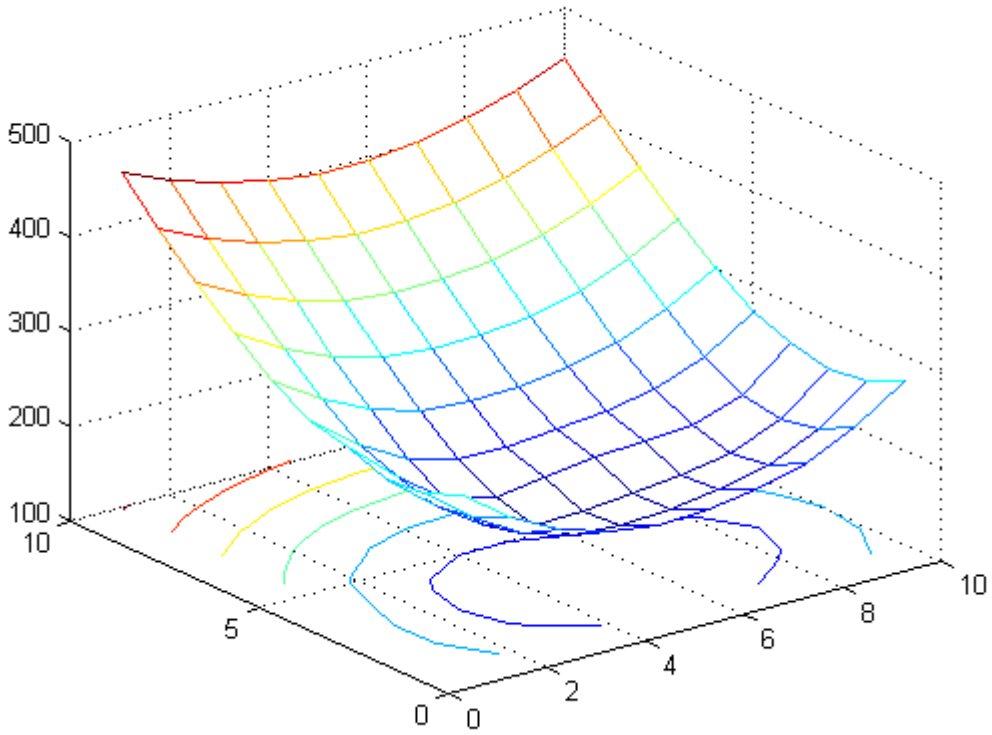

Figure 2: A mesh visualization of the objective function. A grid of 10 on 10 is used starting in point (0,0). Under the mesh the contour plot is visible.

3 Search space specification

Since only a finite amount of possible locations is given the search space is clearly discrete. The (two-dimensional) points were defined in the previous section as the combination of all the elements from i for the first dimension with every element of j for the second dimension.

4 Genetic Algorithm

4.1 code

```
function [ Generation ] = ga(PopSize , BreedSize, SurvivorSize, MutChance)
%GA Runs the genetic algorithm with the given parameters.

Population = randi(10,PopSize,2);

Generation = zeros(PopSize, 3);

for i=1:PopSize
    Generation(i,:) = [Population(i,:) weight(Population(i,:))];
end

for k=1:1000
SortedGeneration = sortrows(Generation,-3);
%Select Parents (BreedSize lowest).
Parents = SortedGeneration(end-BreedSize:end,:);
Survivors = SortedGeneration(end-(BreedSize+SurvivorSize):end-BreedSize,:);

%Breed to create BreedSize Children
Children = zeros(BreedSize,3);

%Keep first coordinate.
for j=1:BreedSize
    MutMod1 = 0;
    MutMod2 = 0;
    if rand(1)<MutChance
        MutMod1 = round(randi(1));
    end
    if rand(1)<MutChance
        MutMod2 = round(randi(1));
    end
    Gene1 = Parents(end,1)+MutMod1;
    Gene2 = Parents(j,2)+MutMod2;

    Children(j,:) = [Gene1 Gene2 weight([Gene1 Gene2])];
end

Generation = [Parents ; Survivors ; Children];

end

end
```

4.2 results

4.3 conclusion

5 Particle Swarm Optimization

5.1 code

```
function [ g  particles ] = pso( nb_particles, iterations )
%PSO Summary of this function goes here
%   Detailed explanation goes here
particles = [randi(10,nb_particles,1) randi(5,nb_particles,1)]

p = particles;

particlesWeight = zeros(nb_particles, 3);

for i=1:nb_particles
    particlesWeight(i,:) = [particles(i,:) weight(particles(i,:))];
end

particlesSortedWeight = sortrows(particlesWeight,-3);
g = particlesSortedWeight(1,1:2);
omega = 1;
phi_p = 1;
phi_g = 1;

velocities= [(randi(20,nb_particles,1)-10) (randi(10,nb_particles,1)-5)];

for k=1:iterations
    for i=1:nb_particles
        r_p = rand(1);
        r_g = rand(1);
        velocities(i,1) = omega*velocities(i,1)+phi_p*r_p*(p(i,1)-particles(i
            ,1))+phi_g*r_g*(g(1,1)-particles(i,1));
        velocities(i,2) = omega*velocities(i,2)+phi_p*r_p*(p(i,2)-particles(i
            ,2))+phi_g*r_g*(g(1,2)-particles(i,2));
        particles(i,:) = floor(particles(i,:) + velocities(i,:));

        if weight(particles(i,:))< weight(p(i,:))
                p(i,:) = particles(i,:);
                WP = weight(particles(i,:))
                WG = weight(g)
            if weight(particles(i,:))< weight(g)
                g = particles(i,1:2);
            end
        end
    end
end
```

5.2 results

5.3 conclusion

6 Utility functions

Listing 1: Matlab implementation for the objective function. The input parameter V is the vector representing the point for which the weight function f is calculated.

```
function [ W ] = weight( V )
%WEIGHT Summary of this function goes here
%   Detailed explanation goes here
W = 0;
loc = locations;
for i=1:length(loc)
    W = W + minDist(V,loc(i,:));
end

end
```

Listing 2: Function that calculates the matrix holding the weights of the points generated from combining the X and Y matrices for the first and the second coordinate respectively.

```
function [ W ] = meshweight( X,Y )
l1 = length(X);
l2 = length(X(:,1));
W = zeros(l1,l2);
for i=1:l1
    for j=1:l2
        W(i,j) = weight([X(i,j),Y(i,j)]);
    end
end
end
```

Listing 3: Utility function that calculates the minimal euclidean distance between two given vectors taking the bridges into account.

```
function [ Dist ] = minDist( P1, P2 )
% Calculates the shortest distance
Bridge1 = [3.5 3.5];
Bridge2 = [8.5 3.5];

if (P1(2) < 3.5 && P2(2) < 3.5) || (P1(2) > 3.5 && P2(2) > 3.5)
    Dist = norm(P1 - P2);
else
    Dist = min(norm( P1- Bridge1 )+ norm(Bridge1 -P2), norm(P1 - Bridge2)+norm(
        Bridge2 - P2));
end

end
```

Listing 4: Utility function that generates the possible locations for use in verious functions.

```
function [ V ] = locations( )

V = zeros(50,2);

for i=1:10
    for j=1:5
        V(5*(i-1)+j, :) = [i j];
    end
end

end
```