Search Based Software Engineering

Exercise 02 - Genetic Algorithms

Team:

1. Sagar Nagaraj Simha - sagar.nagaraj.simha[at]uni-weimar.de
   Matrikel Nummer - 120797, Degree - Master's Computer Science for Digital Media
2. Mohd Saif Khan - mohd.saif.khan[at]uni-weimar.de
   Matrikel Nummer - 120803, Degree - Master's Human Computer Interaction

Task 1. Theory

a) What are (1 + 1), (1 + λ), (1, λ), (μ, λ) and (μ + λ)?

(1 + 1) : Basic hill climbing
(1 + λ) : Steepest ascent hill climbing
(1 , λ) : Steepest ascent hill climbing with replacement
(μ , λ) : Evolutionary strategy (Children replaces parents in next generation)
(μ + λ) : Evolutionary strategy (Children and parents together makes next generation)

μ : Number of parents
λ : Number of children

b) Explain differences between them.

(μ , λ) Evolutionary strategy algorithm takes μ parents and replaces them with their λ fittest children in the next generation after breeding. Whereas (μ + λ) retains μ parents in the total population of λ individuals. Therefore fitter parents compete with children to be in the next generation in this algorithm. (1 + 1), (1 + λ), (1 , λ) are the degenerate cases of above evolutionary strategy algorithm. (1 + 1) is basic hill climbing in which next children replaces parent if it has better fitness than parent else parent is retained in the next generation. (1 , λ) is steepest ascent hill climbing with replacement in which 1 parent has λ children. We then take the children with best fitness and replace parent with it. (1 , λ) is steepest ascent hill climbing in which next parent is chosen from current parent and the fittest children produced after breeding.

Task 2. Line Recombination

a) How does Line Recombination work? Explain in detail.

Line Recombination is a crossover algorithm for floating point vectors in a multidimensional hyperspace. In this algorithm we draw a line between 2 points in a hyperspace and we could also expand this line beyond the points and take children along this line to prevent premature convergence. It depends on variable p which determine how far out on the line we can choose the children. So if p>0 then children can be picked on the line even outside the hypercube.

We choose 2 random values, (α, β) between -p to 1+p and then breed children (t ,s) from parents (v ,w) based on a bound condition. The pseudo code is as below,

```
α <- -p to 1+p
β <- -p to 1+p
for a pair of parent(v,w):
        child t = αv +(1-α)w
        child s = βw +(1-β)v
  if t and s within bounds:
      replace (v, w) with (t, s)
```

The algorithm is repeated for all the parents in population.

b) How can Line Recombination be extended to get Intermediate Line Recombination?

In Intermediate Recombination algorithm α and β are evaluated for every pair of parent repeatedly till the produced children t and s are within the specified bounds. This way we do not reject recombination if the elements go out of bounds.

```
for a pair of parent(v,w):
        repeat :
            α <- -p to 1+p
            β <- -p to 1+p
            child t = αv +(1-α)w
            child s = βw +(1-β)v
        till t and s within bounds
        replace (v, w) with (t, s)
```

The algorithm is repeated for all the parents in population.

c) Implement Intermediate Line Recombination as a python function.

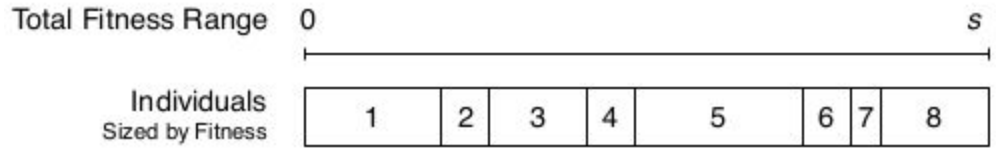
-- IntermediateLineRecombination.py is in the folder.

Task 3. Selection Procedures

a) What is Fitness-Proportionate Selection (FPS) and how does it work?

Fitness Proportionate Selection (FPS), sometimes known as Roulette Selection is one of the 'SelectwithReplacement' algorithms under the 'Selection' step of a Genetic Algorithm. A Genetic Algorithm performs iterative selection, crossover, and mutation while breeding and FPS is used to 'select' parents to breed the next generation of children.

In this algorithm, the individuals are selected in proportion to their respective fitnesses. If an individual has higher fitness, then it's selected more often. First, the individuals are scaled/sized according to their fitnesses by means of creating a cumulative distribution function of individual fitnesses, cdf ={0 to S}. Then, a random number is generated between 0 and S. The range of the individual under which this random number falls, is then selected. The method can by chance select certain individuals over and over again, and it can also by chance occasionally select low-fitness individuals as well. The generation of cdf is done once per generation (n individuals) and the random number generation and the corresponding individual selection is done n times. The method and the pseudo algorithm are as shown below:



Pseudo Code Fitness-Proportionate Selection:

Perform once per generation:

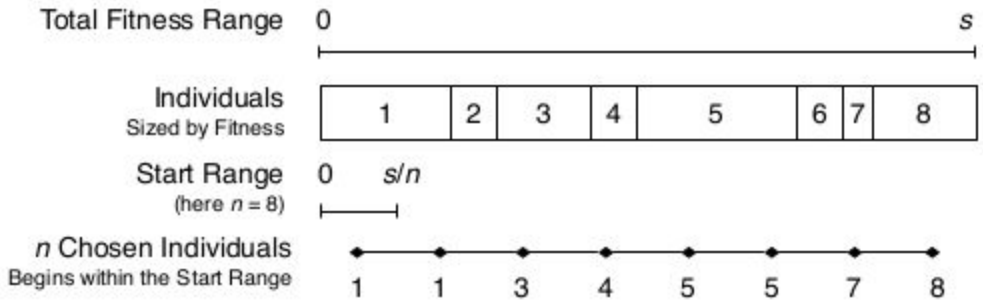
```
    p ← (p1, p2, p3….pn) - current population of n individuals
    f ← (f1, f2, f3…..fn) - fitnesses of corresponding n individuals in p
    For i in 2 to n:
        cdf[i] ← f[i-1] + f[i]
    CDF←(CDF1, CDF2, CDF3….CDFn) -cumulative fitness array built using f
For n times
    r ← random number between 0 to CDFn inclusive
    for i from 2 to n:
        If f[i-1 <= n <= f[i]:
            return pi
    return p1
```

- ***Images courtesy [2]***

b) What is Stochastic Universal Sampling (SUS) and how does it work?

Stochastic Universal Sampling is again one of the 'SelectwithReplacement' algorithms under the 'Selection' step of a Genetic Algorithm to 'select' parents to breed the next generation of children. It is a variant of Fitness-Proportionate Selection. Here again the individuals are selected in proportion to the fitness (using cdf) but biased so that fit individuals always get picked at least once. The SUS also selects n individuals at a time, i.e the size of the next generation.

First, the individuals are scaled/sized according to their fitnesses by means of creating a cumulative distribution function of individual fitnesses, cdf ={0 to S}. Then, a random number (position) is selected between 0 and S/n. The individual which occupies that position is then selected. We then increment the position by s/n and pick the next individual. The process is done in total 'n' times to pick 'n' individuals of the next generation. The method and the pseudo algorithm are as shown below:



Pseudo Code Stochastic Universal Sampling:

Perform once per generation:

```
    p ← (p1, p2, p3….pn) - current population of n individuals after a random shuffle
    f ← (f1, f2, f3…..fn) - fitnesses of corresponding n individuals in p
    index ← 0
    for i in 2 to n:
        cdf[i] ← f[i-1] + f[i]
    CDF←(CDF1, CDF2, CDF3….CDFn) - cumulative fitness array built using f
    value ← random number between 0 to (CDFn / n) inclusive
For n times:
    while f[index] < value do:
        index ← index +1
    value ← value + CDFn/n
    return p[index]
```

- ***Images courtesy [2]***

c) Implement SUS as a python function.

-- StochasticUniversalSampling.py is in the folder

References:

[1] Lecture Slides

[2] Essentials of Metaheuristics - Sean Luke