Pool Selection Techniques in Genetic Algorithms

Aayushi Dave - 1614009,Divyansh Sahu - 1614010,
Hima George - 1614011,Jash Gopani - 1614012,
Navneet Hingankar - 1614013

Introduction

Genetic Algorithms(GAs) are adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. These are intelligent exploitation of random search provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems. Genetic algorithms simulate the process of natural selection which means those species who can adapt to changes in their environment are able to survive and reproduce and go to the next generation. Brute force solutions are exhaustive in nature and there is less chance of getting desired output from an infinitely large solution space whereas genetic algorithms have variations in generations which makes it more efficient than brute force.

Topic Discussion

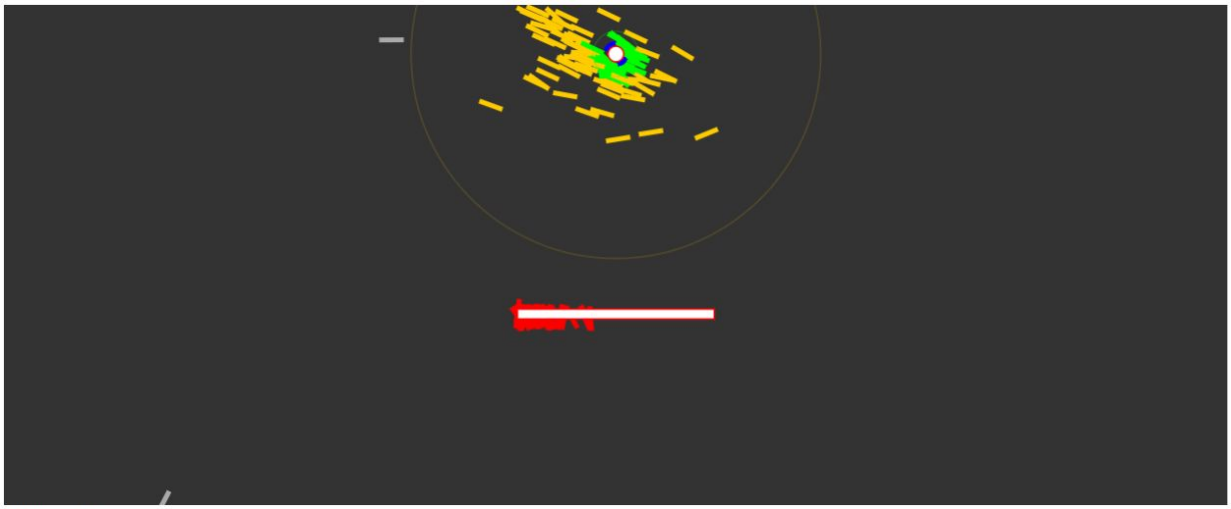
Pool selection is the main focus of this project as the evolution of the next generation i.e the performance of our Genetic algorithm depends on the selection of the fittest parents. Those parents will mate to create the next generation and in this way, the characteristics of the fittest will be passed to the next generation through inheritance. Genes from “fittest” parent propagate throughout the generation, that is sometimes parents create offspring which is better than either parent. Thus each successive generation is more suited for their environment.

The example considered in our project is called “Smart Rockets”.

Problem

The rockets need to reach the target by finding their way from the source and by overcoming the hindrances or obstacles. There are endless permutations and combinations of movements the rocket can make at each spot in the environment, so we want to implement a genetic algorithm for the rockets so that they can find their way on their own.

This report throws light on 3 techniques for selecting the fittest parents from the population. The main objective in pool selection is selecting a parent randomly by its fitness score i.e the more fit a rocket is, the more probability it has of being chosen from the population.



Technology

The simulation is done in HTML5 canvas using a JS library p5.js

Simulation Overview

The complete code can be found here. Referred link mentioned in each code snippet in all the further sections. The algorithm used is the same, code was modified as per the project needs.

Rocket (Phenotype)

Each rocket has the following properties associated with it:

- Position, Velocity & Acceleration (Vectors)
- DNA
- Fitness score
- Completed, Crashed Flags

```
// Daniel Shiffman
// http://codingtra.in
// http://patreon.com/codingtrain
```

```
// Code for: https://youtu.be/bGz7mv2vD6g

// Constructor function
function Rocket(dna) {
 // Physics of rocket at current instance
 this.pos = createVector(width / 2, height);
 this.vel = createVector();
 this.acc = createVector();

 this.crashedAt = 0;
 this.completedAt = 0;

 // Checks if rocket had crashed
 this.crashed = false;
 //if rocket crashed and updated the global value
 this.crashUpdated = false;

 // Checkes rocket has reached target
 this.completed = false;

 //if rocket completed and updated the global value
 this.completeUpdated = false;

 // Gives a rocket dna
 if (dna) {
   this.dna = dna;
 } else {
   this.dna = new DNA();
 }
 this.fitness = 0;

 // Object can recieve force and add to acceleration
 this.applyForce = function(force) {
   this.acc.add(force);
 }
```

```
// Calulates fitness of rocket
this.calcFitness = function() {
  // Takes distance to target
  var d = dist(this.pos.x, this.pos.y, target.x, target.y);

  // Maps range of fitness
  this.fitness = map(d, 0, width, width, 0);
  // If rocket gets to target increase fitness of rocket

  if (this.completed) {
    this.fitness *= 100;
  }

  // If rocket does not get to target decrease fitness
  if (this.crashed) {
    this.fitness /= 10;
  }

  if(this.crashed && this.crashedAt<=lifespan*0.5)
    this.fitness /= 10;

  if(this.completed && this.completedAt >=0.4)
    this.fitness *= this.completedAt;

}

// Updates state of rocket
this.update = function() {
  // Checks distance from rocket to target
  var d = dist(this.pos.x, this.pos.y, target.x, target.y);
  // If distance less than 10 pixels, then it has reached target
  if (d < 10) {
    this.completed = true;
    this.completedAt = this.completedAt>0?this.completedAt:count;
    this.pos = target.copy();
  }
  // Rocket hit the barrier
```

```
  if (this.pos.x > rx && this.pos.x < rx + rw && this.pos.y > ry &&
this.pos.y < ry + rh) {
    this.crashed = true;
    this.crashedAt = count;
  }
  // Rocket has hit left or right of window
  if (this.pos.x > width || this.pos.x < 0) {
    this.crashed = true;
    this.crashedAt = count;

  }
  // Rocket has hit top or bottom of window
  if (this.pos.y > height || this.pos.y < 0) {
    this.crashed = true;
    this.crashedAt = count;

  }

  if(this.crashed && !this.crashUpdated){
    crashed+=1;//update the global crashed count defined in sketch.js
    this.crashUpdated=true;
  }

  if(this.completed && !this.completeUpdated){
    completed+=1;//update the global completed count defined in sketch.js
    this.completeUpdated=true;
  }



  //applies the random vectors defined in dna to consecutive frames of
rocket
  this.applyForce(this.dna.genes[count]);
  // if rocket has not got to goal and not crashed then update physics
engine
  if (!this.completed && !this.crashed) {
    this.vel.add(this.acc);
    this.pos.add(this.vel);
```

```
    this.acc.mult(0);
    this.vel.limit(4);
  }
}
// displays rocket to window
this.show = function() {
  // push and pop allow's rotating and translation not to affect other
objects
  push();
  //color customization of rockets
  noStroke();
  // fill(255, 150);
  this.colorize();
  //translate to the postion of rocket
  translate(this.pos.x, this.pos.y);
  //rotatates to the angle the rocket is pointing
  rotate(this.vel.heading());
  //creates a rectangle shape for rocket
  rectMode(CENTER);
  rect(0, 0, 25, 5);
  pop();
}


this.colorize = function(){
  // Checks distance from rocket to target
  var d = dist(this.pos.x, this.pos.y, target.x, target.y);

  if(!this.crashed){
    if(d<=25 && !this.completed)
       fill('rgb(0,255,0)'); //blue
     else if(d<=25 && !this.completed)
        fill(color(0, 0, 255)); //green
     else if(d>25 && d<=200)
       fill(255, 204, 0) //yellow
     else
       fill(255,150); //white
```

```
    }else{
      fill('red');
    }

    if(this.completed)fill(color(0, 0, 255));
 }

}
```

Lifespan

All the rockets have a lifespan (an integer variable) which will restart when it reaches 0. Lifespan can be taken as the maximum number of moves a rocket can make in order to reach the target.

```
var generation;
var population;
var crashed,completed;
// Each rocket is alive till 400 frames
var lifespan = 210;
// Made to display count on screen
var lifeP;
// Keeps track of frames
var count = 0;
// Where rockets are trying to go
var target;
// Max force applied to rocket
var maxforce = 0.5;

//some more DOM elements
var generation_sp1,generation_sp2;
var lifespan_sp1,lifespan_sp2;
var crashed_sp1,crashed_sp2;
var completed_sp1,completed_sp2;
var population_sp;


// Dimensions of barrier
```

```
var rx,ry;//top-left coordinates
var rw = 200;
var rh = 10;

function setup() {
 createCanvas(windowWidth*0.95,windowHeight*0.8);
 population = new Population();

 generation=1;
 crashed=0;
 completed=0;
 target = createVector(width / 2, 50);

 //updating dimensions
 rx = width/2 - rw/2;
 ry = height - 200;

 //creating dom elements
 lifespan_sp1 = createSpan("Lifespan : ");
 lifespan_sp2 = createSpan();
 createDiv();
 population_sp = createSpan("Population : "+population.popsize);
 createDiv();
 generation_sp1 = createSpan("Generation : ");
 generation_sp2 = createSpan("");
 createDiv();
 crashed_sp1 = createSpan(" No. of rockets crashed : ");
 crashed_sp2 = createSpan("");
 createDiv();
 completed_sp1 = createSpan(" No. of rockets reached : ");
 completed_sp2 = createSpan("");
}
```

DNA (Genotype)

The DNA of the rocket is made up of genes where each gene represents a vector which will add up to the current position of the rocket in order to

move the rocket. The length of the DNA is equal to the lifespan count since a rocket cannot move once its lifespan is over. This class has a crossover method for simulating the mating of two parents.

```
// Daniel Shiffman
// http://codingtra.in
// http://patreon.com/codingtrain
// Code for: https://youtu.be/bGz7mv2vD6g

function DNA(genes) {
 // Recieves genes and create a dna object
 if (genes) {
   this.genes = genes;
 }
 // If no genes just create random dna
 else {
   this.genes = [];
   for (var i = 0; i < lifespan; i++) {
     // Gives random vectors
     this.genes[i] = p5.Vector.random2D();
     // Sets maximum force of vector to be applied to a rocket
     this.genes[i].setMag(maxforce);
   }
 }
 // Performs a crossover with another member of the species
 this.crossover = function(partner) {
   var newgenes = [];
   // Picks random midpoint
   var mid = floor(random(this.genes.length));
   for (var i = 0; i < this.genes.length; i++) {
     // If i is greater than mid the new gene should come from this
partner
     if (i > mid) {
       newgenes[i] = this.genes[i];
     }
     // If i < mid new gene should come from other partners gene's
     else {
       newgenes[i] = partner.genes[i];
```

```
    }
  }
  // Gives DNA object an array
  return new DNA(newgenes);
}

// Adds random mutation to the genes to add variance.
this.mutation = function() {
  for (var i = 0; i < this.genes.length; i++) {
    // if random number less than 0.01, new gene is then random vector
    if (random(1) < 0.01) {
      this.genes[i] = p5.Vector.random2D();
      this.genes[i].setMag(maxforce);
    }
  }
}

}
```

Population

The population class is responsible for representing a generation. When the lifespan of a generation is over, a new generation is generated by selection and crossover process. The population class has an array of rocket objects and a selection method for the selection phase. The selection techniques are discussed in the next section of the report.

```
// Daniel Shiffman
// http://codingtra.in
// http://patreon.com/codingtrain
// Code for: https://youtu.be/bGz7mv2vD6g

function Population() {
 // Array of rockets
 this.rockets = [];
 // Amount of rockets
 this.popsize = 200;
```

```
// Associates a rocket to an array index
for (var i = 0; i < this.popsize; i++) {
  this.rockets[i] = new Rocket();
}

this.evaluate = function() {

  var maxfit = 0;
  // Iterate through all rockets and calcultes their fitness
  for (var i = 0; i < this.popsize; i++) {
    // Calculates fitness
    this.rockets[i].calcFitness();
    // If current fitness is greater than max, then make max equal to
current
    if (this.rockets[i].fitness > maxfit) {
      maxfit = this.rockets[i].fitness;
    }
  }


  // Normalises fitnesses
  for (var i = 0; i < this.popsize; i++) {
    this.rockets[i].fitness /= maxfit;
  }

}

this.improvedAcceptOrReject = function(){
  var hack = 0;
  //Accept or reject
  var index = 0;
  var R = random(1);

  while(R > 0){
    R -= this.rockets[index].fitness;
    index += 1;
    // console.log("inside while");
```

```
    }
    index-=1;
    return this.rockets[index];


}

// Selects appropriate genes for child
this.selection = function() {
    /*
       Code given in the next section
    */
}

// Calls for update and show functions
this.run = function() {
    for (var i = 0; i < this.popsize; i++) {
      this.rockets[i].update();
      // Displays rockets to screen
      this.rockets[i].show();
    }
}
}
```

Pool Selection Techniques

1. Mating Pool: The simplest method of pool selection is by creating a mating pool having all the elements. The trick here is that if an element has a fitness score of M, it occurs M number of times in the mating pool. This simply maps the fitness of the element/ member to its probability of being chosen at random.

Consider this population array (X,f) where X is the rocket identifier and f is its fitness score

[(A,5), (B,7), (C,1), (D,3)]

The mating pool for this population will look like

A	A	A	A	A	B	B	B	B	B	B	B	C	D	D	D

```
this.evaluate = function() {

  var maxfit = 0;
  // Iterate through all rockets and calcultes their fitness
  for (var i = 0; i < this.popsize; i++) {
    // Calculates fitness
    this.rockets[i].calcFitness();
    // If current fitness is greater than max, then make max equal to
current
    if (this.rockets[i].fitness > maxfit) {
      maxfit = this.rockets[i].fitness;
    }
  }
  // Normalises fitnesses
  for (var i = 0; i < this.popsize; i++) {
    this.rockets[i].fitness /= maxfit;
  }

  this.matingpool = [];
  // Take rockets fitness make in to scale of 1 to 100
  // A rocket with high fitness will highly likely will be in the mating
pool
  for (var i = 0; i < this.popsize; i++) {
    var n = this.rockets[i].fitness * 100;
    for (var j = 0; j < n; j++) {
      this.matingpool.push(this.rockets[i]);
    }
  }
}
// Selects appropriate genes for child
this.selection = function() {
  var newRockets = [];
  for (var i = 0; i < this.rockets.length; i++) {
```

```
    // Picks random dna
    var parentA = random(this.matingpool).dna;
    var parentB = random(this.matingpool).dna;
    // Creates child by using crossover function
    var child = parentA.crossover(parentB);
    child.mutation();
    // Creates new rocket with child dna
    newRockets[i] = new Rocket(child);
  }
  // This instance of rockets are the new rockets
  this.rockets = newRockets;
}
```

2. Accept and Reject: This technique does not require a mating pool. Instead, it works by indirectly mapping the random element with the probability scale i.e 0 to 1. This technique needs the probability of all the elements to be calculated using its fitness score using the following formula

   P(X) = (X.fitness)/(∑ fitness of each element of population)

   The steps for selecting a Parent are as follows :
   1. Select two random numbers R1 and R2 from the range [0,population_size) and [0,1) respectively.
   2. R1 corresponds to the index of an element in the population to be picked and R2 is the decision value which accepts or rejects the selected element “population[R1]” based on the following condition :

      If R2 < P(population[R1])
      then Accept
      Else repeat steps 1 and 2

```
this.acceptOrReject = function(){
  var hack = 0;
  //Accept or reject

  while(true){
   hack+=1;

   var index = floor(random(this.popsize));
   var R = random(1);

   if(R < this.rockets[index].fitness){
    // console.log("Parent = "+this.rockets[index].fitness);
    return this.rockets[index];
   }

   if(hack > 10000){
    console.log("Hacked");
    return this.rockets[index];
   }
  }
```

3. Improved Accept and Reject: This is an amelioration of the above method which could go in an infinite loop if the element picked is always rejected. Like its previous version, this one needs the probabilities of the elements too; which can be calculated using the same formula as before. Here a random number R is used which can be visualised a point in the range [0,1) and if any element has a probability greater than R then it will be selected as a parent.

   The steps for selecting a Parent are as follows :
   1. Select a **random number *R*** in the range **[0,1)** and initialize a variable "***index***" to **0**.
   2. R is the decision value which returns the parent based on the following condition :
      While R > 0

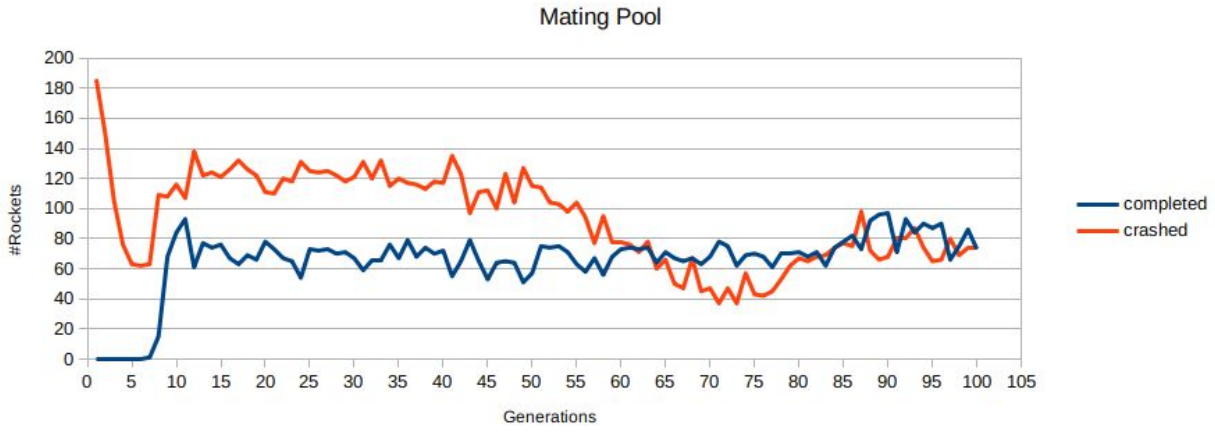
{
R = R - P(*population[index]*)
index = index +1
}

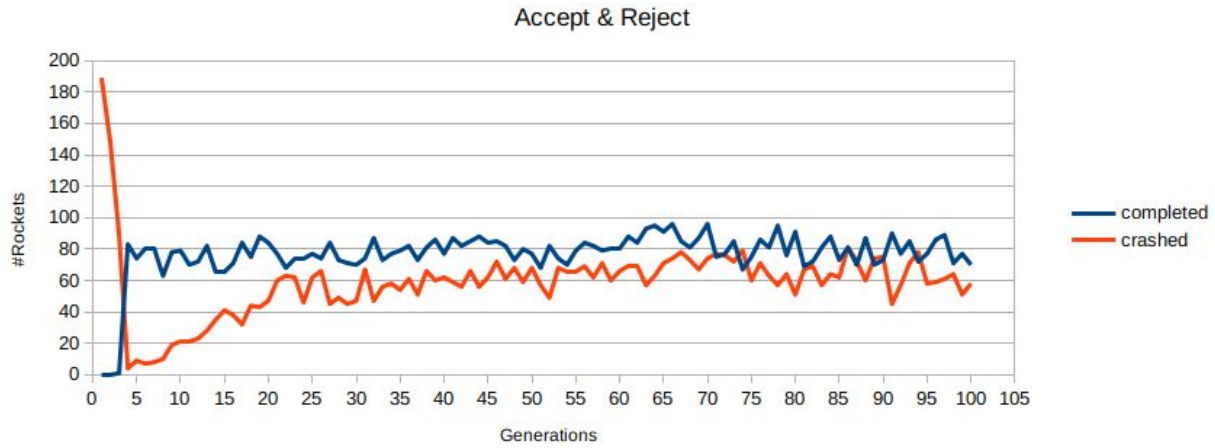
3. Return *population[index-1]* .

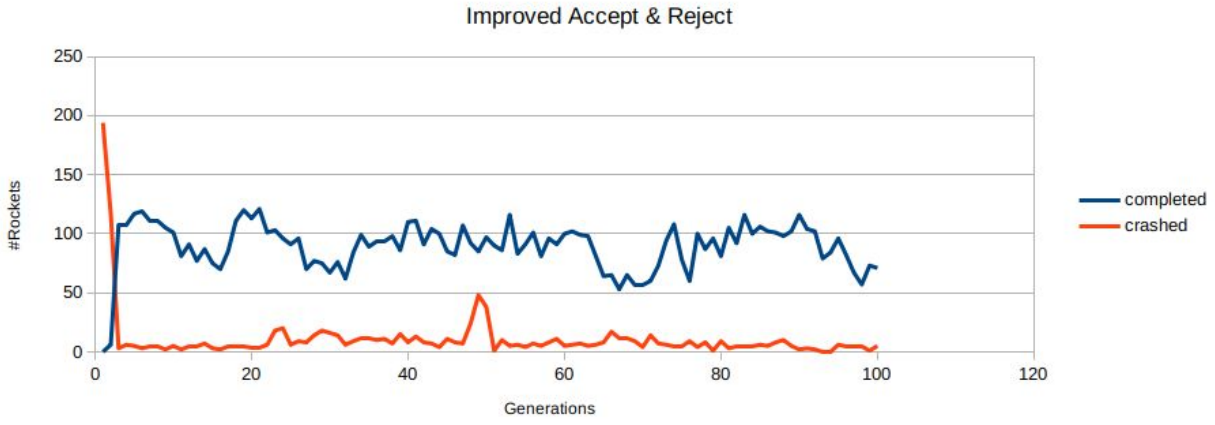
```
this.improvedAcceptOrReject = function(){
   var hack = 0;
   //Accept or reject
   var index = 0;
   var R = random(1);

   while(R > 0){
     R -= this.rockets[index].fitness;
     index += 1;
     // console.log("inside while");
   }
   index-=1;
   return this.rockets[index];


 }
```

This technique works on the logic that if the value R can only become negative or 0 only when the probability of the *population[index]* is greater than R; which implies that the element has more chance of being chosen. If R is positive after subtraction means that the element's probability isn't large enough to overcome R.

Results







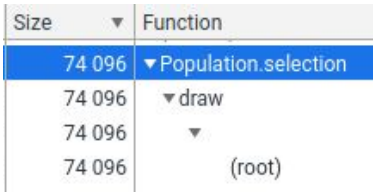
All the pool selection techniques were implemented keeping other parameters like population size, the lifespan of rockets, fitness function, source, target and obstacle. Along with this, the HTML5 canvas size was also kept the same for uniform conditions.

The mating pool technique was the simplest of all to implement and the number of crashes were highest in the initial generations and the number of rockets reaching the target or the rockets who have “completed” the mission were 0. After a few succesful rockets being able to reach closest to the target or at the target , the DNA of the generations changed drastically which lead to the sudden bump in the number of “completed” rockets. Throughout the simulation, the number of crashes reduced significantly but on an average were half the population. On the other hand, the number completed rockets almost remained constant till the end.

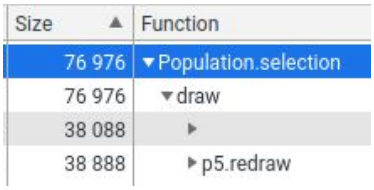
Next, was the Accept and Reject selection. The major change observed compared to the previous selection is that the parents selected were fitter than the previous model and that is why the rockets reaching the target is almost the same but the number of crashes is even further decreased! This shows that the generations inherited better genes throughout the simulation which led to this output.

Improved Accept and Reject minimised the number of crashes throughout the simulation. Coincidently, the first generation of rockets reached extremely close to the target due to which the number of rockets reaching the target went up in the very initial phase itself.

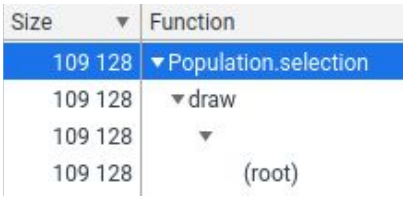
Since the target here is very small, many of the rockets neither reach the target nor crash; they are mostly deviated by the random vectors in their DNA.



(3)



(2)



(1)

The above 3 screenshots are from the chrome dev tools which was used to analyse the memory usage of the program. The number below the images correspond to the technique number respectively. So from the screenshots, it is clear that the array size of the mating pool depends on the fitness of the elements in the population; more number of fit elements will lead to a larger array size whereas in the other 2 algorithms, the memory allocated is approximately the same.

Conclusion

Genetic algorithms are a great replacement for brute force solutions for problems which have infinite solution possibilities. The genetic algorithm reduces the cost significantly by passing down some genes of the fittest elements of the population to the next generation. But it is still important to note that it all starts with a set of random values and hence the simulation will not be the same each time. It depends mainly on the genotype of the object i.e the encoding of the data as a DNA of the element and the Phenotype i.e the object itself (its properties and its methods). The combination of max speed(in this case), lifespan, population size, fitness function, crossover technique, uniformity and independence of random number generator also affect the results significantly. The mating pool algorithms is memory inefficient compared to the other two.