4M17 Exercise III : Solving the Bird Function

eng-216er

January 13, 2015

`eng-216er.github.io`

Summary: A JavaScript program was written to investigate two different methods of solving the bird function.

The motivation for choosing JavaScript to write optimisation algorithms in is discussed. In brief, this is because it enables problems to be solved A genetic algorithm method and a tabu search method are considered.

Parameters controlling each of these methods are varied, and the resulting changes to the convergence rate are discussed.

The tabu search method is found to perform better than the genetic algorithm method.

Contents

List of Figures

1 Running the Code

The code in this report runs as web app. It can be found in the listings, but is also hosted at `eng-216er.github.io` and can be accessed by launching this URL in a web browser. The code has been tested in the latest versions of the Firefox and Chrome browsers.

It should be noted that when the parameter "Pause Between Iterations" is set to zero, there is still a slight delay introduced since each step of the algorithms is initiated by a callback within the JavaScript event loop. If the code were modified for actual use, it could operate much faster.

2 Rationale behind the use of JavaScript

JavaScript is unique in that programs written in it can be embedded in a html document, and executed in a web browser. No other language can be used for client side web programming without either using a browser extension (Java, Flash) or compiling into JavaScript (CoffeeScript).

This provided the motivation for me to implement the optimisation algorithms in JavaScript. In small part, this was because of the possibility of creating a simple html based UI for controlling the optimization parameters and inspecting the results.

Largely however, I was drawn to using JavaScript because being able to solve optimization problems in a browser could potentially be useful within several web programming contexts. For instance, the development of WebGL allows for hardware accelerated graphics problems to be developed for the web. Optimization can be used to solve useful problems in graphics programming. An example is computing the best possible conformal mapping between texture co-ordinates, and coordinates that make up a mesh of a surface. This cam be used to apply a texture to a 3D surface, while minimising the effect of distortion on the surface.

There are currently very few JavaScript optimization libraries. Although the software provided in this report does very little to rectify that, it does provide a starting point for more complex software.

3 A Note on Random Number Seeds

JavaScript supplies a random number generator via the `Math.random()` function. The random numbers produced by this method are not standardised, and there is no way to provide a seed to it.

Unfortunately this means that any optimization algorithm that utilizes random numbers will not be repeatable. This applies to the genetic algorithm as implemented, however tabu search is unaffected, and is repeatable.

4 Genetic Algorithms

A genetic algorithm based optimization solver was written for the bird function. In the genetic algorithm solver, for each individual each parameter x1 and x2 is represented using 16 bits.

Two selection strategies were implemented: tournament selection, and frequency dependant selection.

Tournament selection was implemented by shuffling the population, and then partitioning the population up into N sets, where N is the number of parents. The best candidate from each set was selected to be a parent. This guarantees that each candidate may only be selected once. It also guarantees that the best candidate will be selected.

Fitness proportionate selection chose N candidates at random from the population, with a probability proportional to $100 - bird(x)$. Selection was carried out with replacement, so it was possible for an individual to be selected several times, especially if that individual was disproportionately good.

The next generation was generated, by selecting from the list of parents in turn, and pairing each parent with another random parent. The parents were bread using two point crossover on each parameter (4 point crossover overall). The crossover points were selected at random, but the first crossover point has to appear in the first 3/4 of the parameter's chromosome.

After crossover, each child may be subjected to a random 1 bit mutation. Mutation took place at a configurable rate. The parameter and the bit to be mutated was selected at random.

The number of parents, and the population size was left configurable.

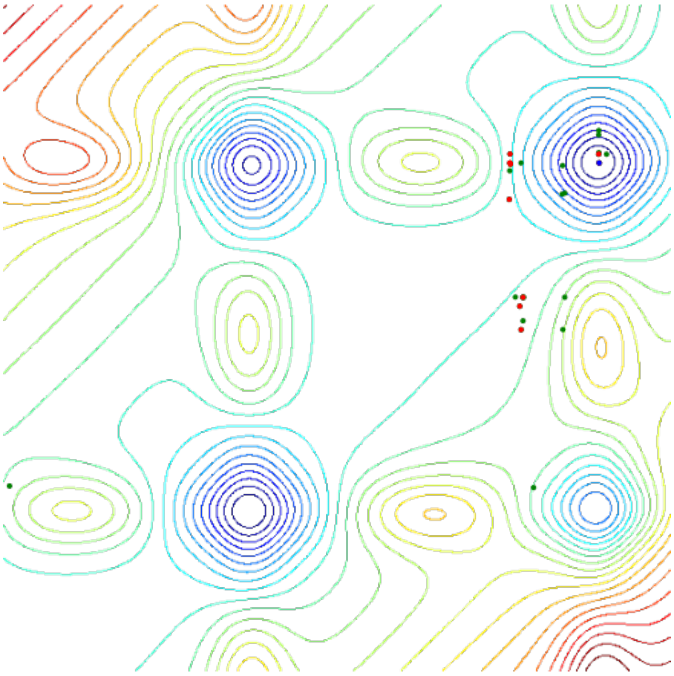


Figure 1: The Genetic Algorithm in Mid Run
Green dots show unselected points
Red dots show parent points
The blue dot shows the current minimum

4.1 Comparing Selection Strategies

Figure 2 shows the results of several runs using the two different selection strategies. All other parameters were kept at their default values. Tournament selection performs better than fitness proportional selection. This involves ignoring one outlier where a tournament selection approach found a local minimum instead.

From observation, fitness proportional selection works well in the earlier stages of the optimization, when there are a wide range of fitness values. However it falls down at precisely finding the optimum in the later stages.

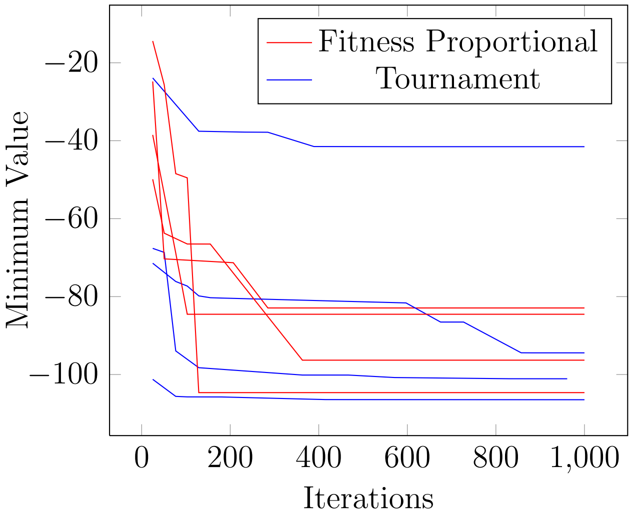


Figure 2: Comparison Between Different GA Selection Strategies

4.2 Varying the Proportion of Parents

Figure 3 shows the effect of varying the number of parents on a genetic algorithm run with the default values, implying the use of tournament selection.

This graph seems to suggest that the algorithm works best when the number of parents is under 15.

When the parent population size is 25, the entire population become parents, irrespective of their fitness. Thus this is roughly equivalent to sampling points at random. Similarly, when large sections of the population are allowed to become parents, then points with a low objective function value have only a small advantage, and the algorithm takes a long time to converge.

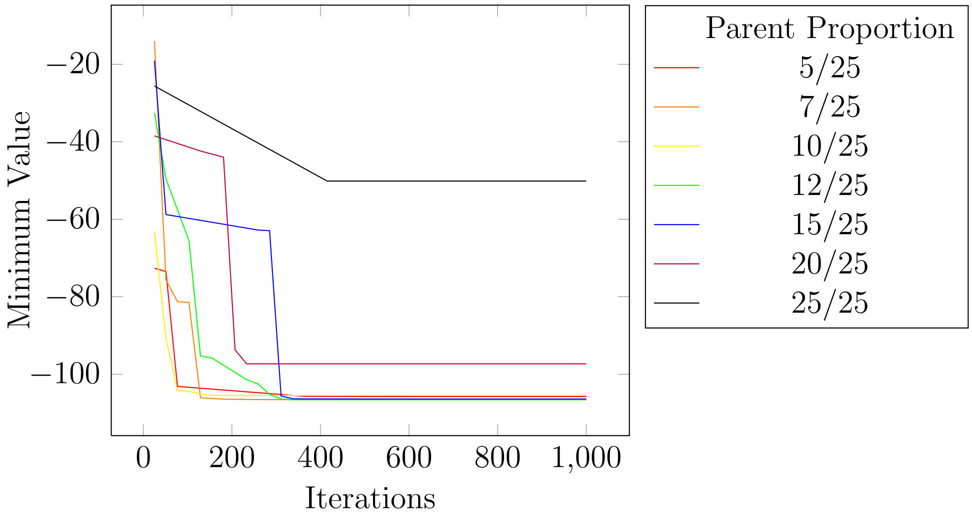


Figure 3: The Effect of Varying the Number of Parents

4.3 The Effect of Varying the Mutation Rate

Figure 4 shows several runs of the Genetic algorithm with varying mutation rates. From inspection, varying the mutation rate within this range seems to have little effect on the convergence rate. Anecdotally, having a relatively high mutation rate decreases the probability of the algorithm getting trapped on a local minima.

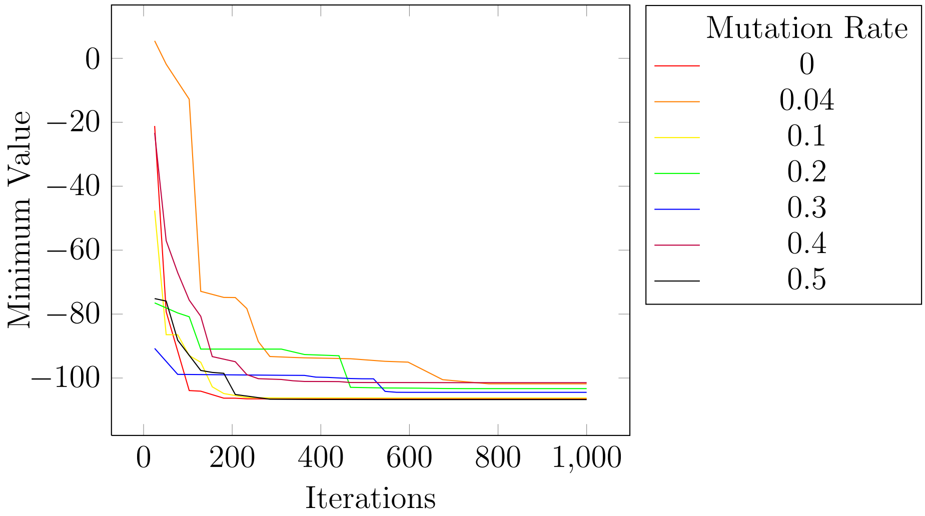


Figure 4: The Effect of Varying the Mutation Rate

4.4 Tendency to get Trapped in a Local Minima

As discussed above, the genetic algorithm optimization will occasionally fail to find either of the global minima, and instead focus on the local minima. Including some mutation seems to decrease the incidence of this. Figure 5 shows a genetic algorithm run that has converged on a local minima. When 100 genetic algorithm runs with the default parameters were sampled, only two of these became stuck on a local minima.

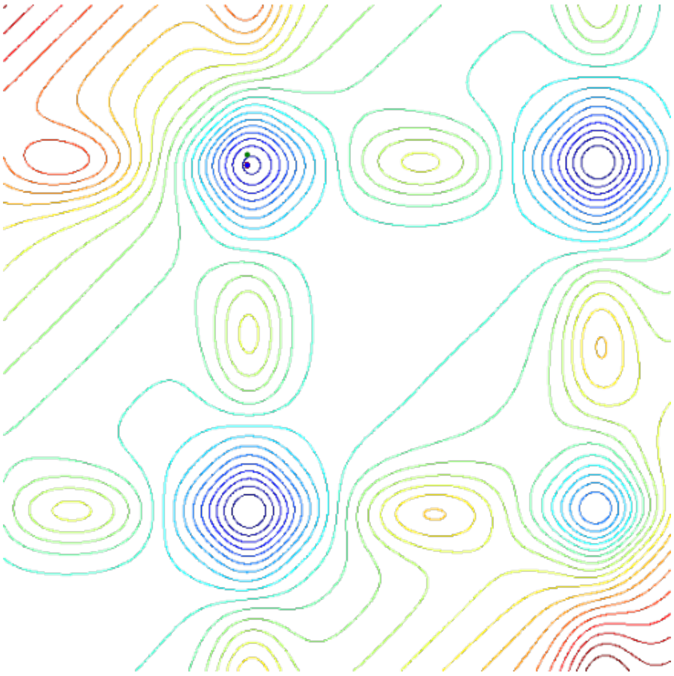


Figure 5: The Genetic Algorithm stuck on a Local Minima

5 Tabu Search

Tabu search was implemented with configurable short and medium term memory sizes. Long term memory was implemented by dividing the search space into cells and increasing a counter when that cell was visited. The number of cells was left configurable. Diversification involved moving to the center of the first cell that hadn't been visited yet, and resetting the interval length. Step size reduction was implemented by moving to the current lowest point visited, and setting the interval length to half it's value when this point was visited.

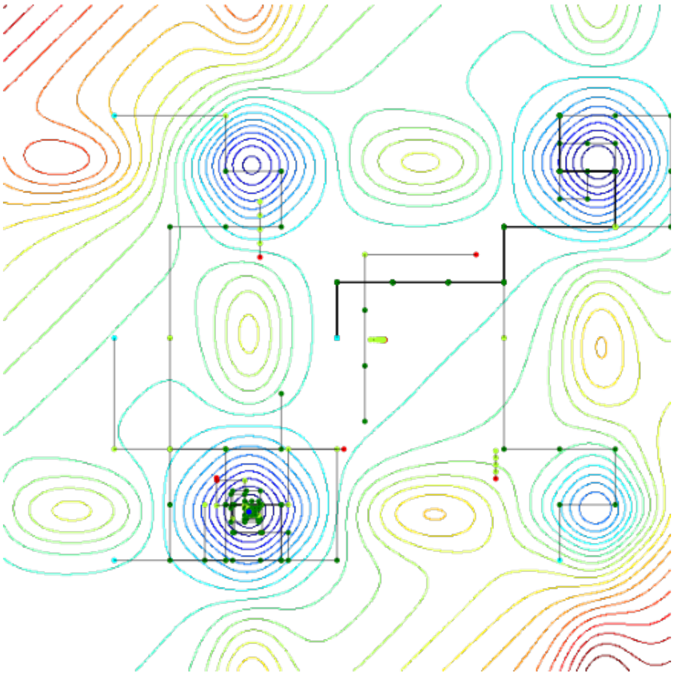


Figure 6: Result of Performing a Tabu Search with the Default Parameters
Green dots show points reached via a normal move
Light green dots show points reached via a pattern move
Red dots show points reached by intensification
Cyan dots shows points reached by diversification
Yellow dots shows points reached by step size reduction
The blue dot shows the final minimum value

5.1 Varying the Short Term Memory Size

Figure 7 shows the effect of varying the short term memory size on the convergence. Varying the short term memory size has little effect on the convergence.

When a small short term memory is used, it becomes apparent by inspection that certain points are being computed visited several times. Figure 8 demonstrates this, as there is a loop in the graph when a small short term memory is used, and no loop with a larger memory. This is obviously inefficient, however it does not stop the algorithm from converging efficiently to the minimum.

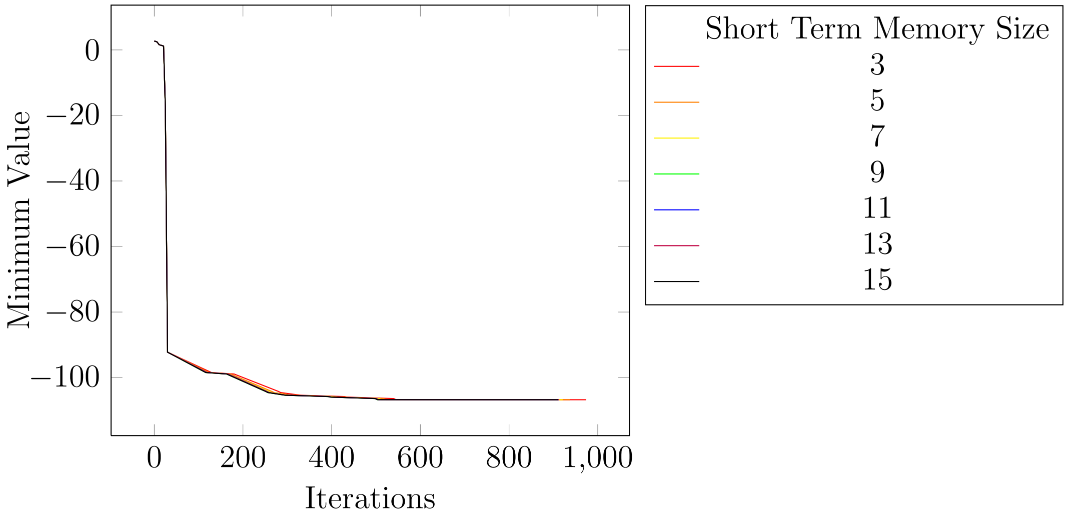
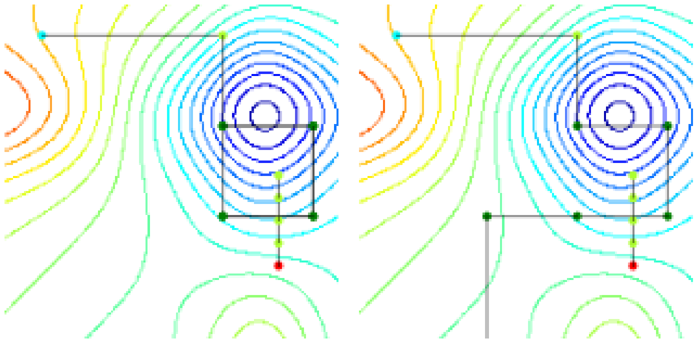


Figure 7: Effect of Varying the Short Term Memory Size



(a) 3 Elements

(b) 20 Elements

Figure 8: Section of the Tabu Search Graph for different Short Term Memory Sizes

5.2 Effect of Varying the Medium Term Memory Size

Figure 9 shows the effect of varying the size of the medium term memory on the convergence rate. Like the short term memory, varying the medium term memory has little discernible effect.

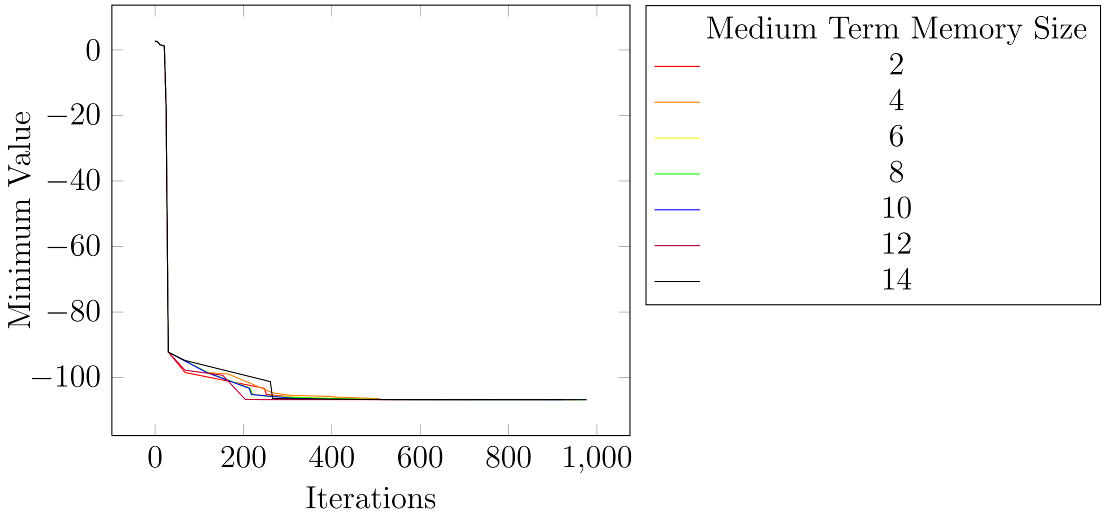


Figure 9: Effect of Varying the Tabu Medium Term Memory Size

5.3 Effects of Varying the Long Term Memory Size

Figure 10 shows the effect of varying the size of the long term memory on the convergence rate.

Like the short and medium term memory sizes, the long term memory has little effect on the convergence rate. For large long term memory sizes, it is interesting to see that only a small proportion of the cells are diversified to. Note how in Figure 11 the only cells that are visited due to diversification are at the extreme left of the graph. This indicates that diversification doesn't take place often enough for all parts of the graph to be visited.

This could have lead to us missing the global minima if it was in one of the cells that we didn't visit. In this case if there had been a larger minima in the bottom right, the Tabu search would have missed it. This implies that it is important to set the number of cells in the long term memory to be small enough that all regions of the search space are investigated.

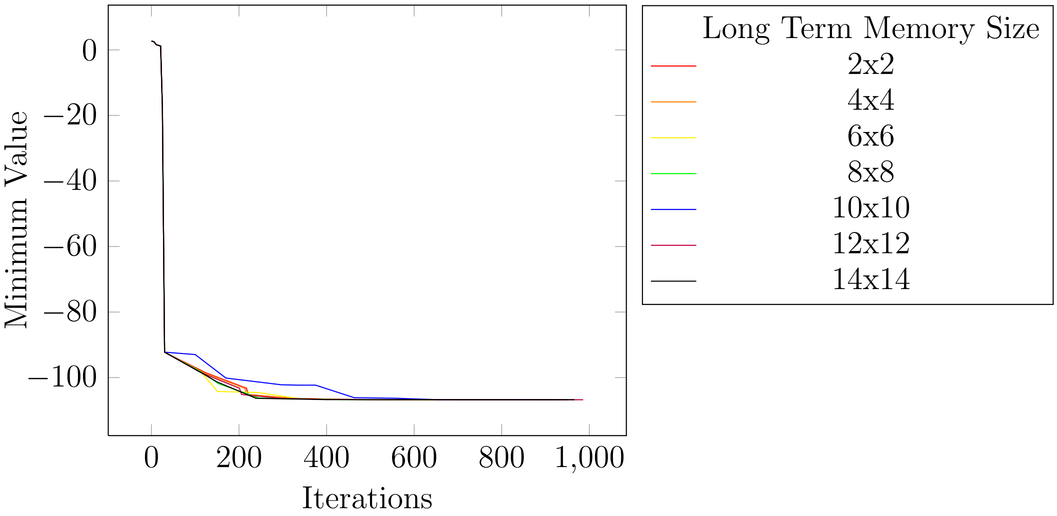


Figure 10: Effect of Varying the Tabu Long Term Memory Size

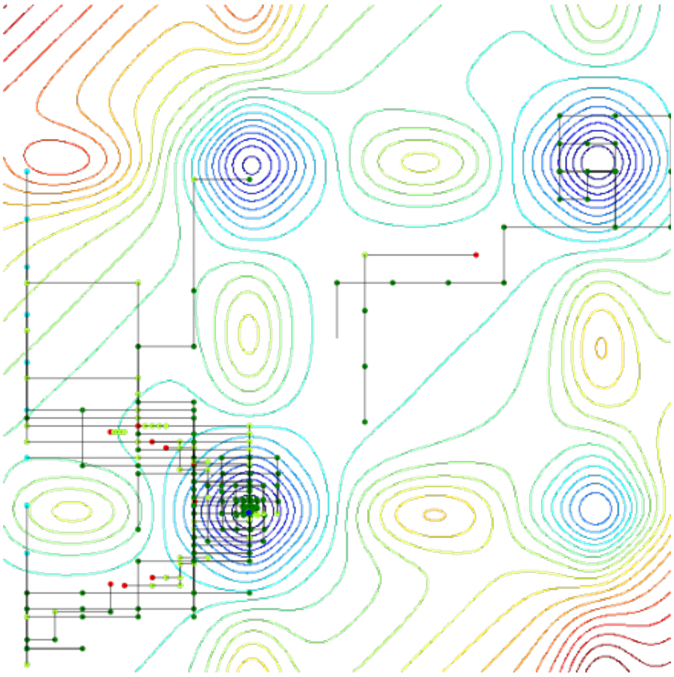


Figure 11: Tabu Search with a 14x14 Long Term Memory Size
Note the cyan dots at the right hand side indicating the cells that are visited due to diversification.

6 Comparison of Genetic Algorithm and Tabu Search Methods

Both genetic algorithm and tabu Search Methods are effective at finding the global minimum to the bird function.

Of the two search methods tabu search converges faster. The genetic algorithm method also exhibits a problem with converging to local minima that tabu search does not. Despite this it is clear that if used incorrectly tabu search could easily converge on a local minima.

Additionally, inspection of the tabu search graph (Figure 6) suggests that there are lots of starting points in the search space where the naive search at the heart of the tabu algorithm would lead to the correct solution. This fact would seem to work in favour of the tabu search.

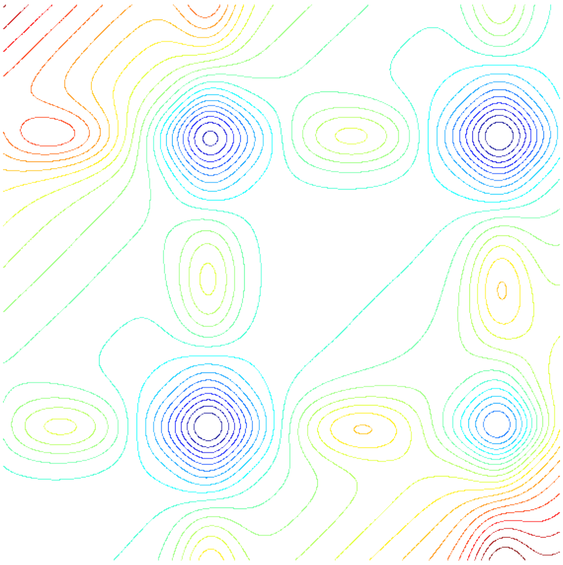
A Listings

A.1 PlotImage.m

```
function [  ] = PlotImage(  )
% Plot a contour map of the Bird Function
% This is to be used as the background of the canvas
n = 1000;
range = linspace(-6,6,n);
y = Bird( ones( n,1 ) * range , range' * ones( 1,n )  );

contour( range, range, y, 23);
axis equal
axis off
end
```

A.2 Bird.png



A.3 index.html

```
<!DOCTYPE html>

<html>
<head>
  <title>Optimization Ex3</title>
  <meta charset="utf-8">
  <script type="text/javascript" src="minmax.js">
</script>
  <script type="text/javascript" src="ex3.js">
</script>
  <script type="text/javascript" src="genetic_algorithm.js">
</script>
  <script type="text/javascript" src="tabu.js">
```

```
</script>
  <style type="text/css">
 body{
     font-family: monospace;
   }
   h1 {
     text-align: center;
     font-size: 16pt;
   }
   .parent {
     max-width: 500px;
     max-height: 500px;
     margin: 1em auto;
     border: 1px solid black;
   }
   .buttons {
     margin: 1em auto;
     text-align: center;
   }
   .animParent{
     margin: 0.5em auto;
     padding: 0;
     height: 4em;
     overflow: hidden;
     text-align: center;
   }
   #loadingAnim{
     margin: 0 auto;
     display: none;
     background-color: black;
     border: 1px solid black;
     -webkit-animation: loader 2s ease infinite;
     animation: loader 2s ease infinite;
   }
   @-webkit-keyframes loader {
     from {
       opacity: 1.0;
       margin-top: 1em;
       width: 0em;
       height: 0em;
       border-bottom-right-radius: 0em;
       border-bottom-left-radius: 0em;
       border-top-right-radius: 0em;
       border-top-left-radius: 0em;
     }
     to {
       opacity: 0;
       margin-top: 0em;
       width: 4em;
       height: 4em;
       border-bottom-right-radius: 2em;
       border-bottom-left-radius: 2em;
       border-top-right-radius: 2em;
       border-top-left-radius: 2em;
     }
   }
   @keyframes loader {
     from {
       opacity: 1.0;
       margin-top: 2em;
       width: 0em;
       height: 0em;
       border-bottom-right-radius: 0em;
       border-bottom-left-radius: 0em;
       border-top-right-radius: 0em;
       border-top-left-radius: 0em;
     }
     to {
       opacity: 0;
       margin-top: 0em;
       width: 4em;
       height: 4em;
       border-bottom-right-radius: 2em;
       border-bottom-left-radius: 2em;
       border-top-right-radius: 2em;
       border-top-left-radius: 2em;
     }
   }
   #controls {
     text-align: left;
     margin: 0 auto;
     width: 30em;
   }
  </style>
</head>

<body>
  <h1>Bird Function Optimizer</h1>

  <div class="parent">
    <canvas id="c" width="500" height="500"></canvas>
  </div>

  <div class="buttons">
    <button class="algorithm" onclick=
    "genetic_algorithm(␣bird_function␣)">Genetic Algorithm</button>
    <button class="algorithm" onclick=
    "tabu_search(␣bird_function␣)">Tabu Search</button>

    <div class="animParent">
      <div id="loadingAnim">
        &nbsp;
      </div>
    </div>

    <div id="urlsection"></div>

  </div>

  <div id="controls">
    <h3>Settings:</h3>

    <span>Pause Between Iterations</span><br>
    <input type="range" class="range" min="0" max="1000" step="1"
    value="500" id="pause" onchange="updateSlider('pause')">
    <span id="pausespan"></span> ms<br>

    <h3>GA Specific Settings:</h3>

    <span>Population Size</span><br>
    <input type="range" class="range" min="10" max="50" step="5"
    value="25" id="gapopulation" onchange=
    "updateSlider('gapopulation')"> <span id=
    "gapopulationspan"></span><br>

    <span>Parent Count</span><br>
    <input type="range" class="range" min="5" max="25" step="1"
    value="10" id="gaparents" onchange="updateSlider('gaparents')">
    <span id="gaparentsspan"></span><br>

    <span>Selection Strategy</span><br>
    <select id="gastratergy">
      <option value="tournament">
        Tournament Select
      </option>
      <option value="fitness">
        Fitness Proportionate Select
      </option>
    </select><br>

    <span>Mutation Rate</span><br>
    <input type="range" class="range" min="0" max="0.5" step="0.02"
    value="0.02" id="gamutate" onchange="updateSlider('gamutate')">
    <span id="gamutatespan"></span><br>

    <h3>Tabu Specific Settings</h3>

    <span>Short Term Memory</span><br>
    <input type="range" class="range" min="1" max="20" step="1"
    value="7" id="tabushort" onchange="updateSlider('tabushort')">
    <span id="tabushortspan"></span><br>

    <span>Medium Term Memory</span><br>
    <input type="range" class="range" min="1" max="20" step="1"
    value="4" id="tabumedium" onchange=
    "updateSlider('tabumedium')"> <span id=
    "tabumediumspan"></span><br>

    <span>Long Term Memory Grid Length</span><br>
    <input type="range" class="range" min="1" max="20" step="1"
    value="3" id="tabulong" onchange="updateSlider('tabulong')">
    <span id="tabulongspan"></span> per side<br>

    <span>Intensification Step</span><br>
    <input type="range" class="range" min="1" max="40" step="1"
    value="10" id="tabuintensify" onchange=
    "updateSlider('tabuintensify')"> <span id=
    "tabuintensifyspan"></span><br>

    <span>Diversification Step</span><br>
    <input type="range" class="range" min="1" max="40" step="1"
    value="15" id="tabudiversify" onchange=
    "updateSlider('tabudiversify')"> <span id=
    "tabudiversifyspan"></span><br>

    <span>Step Size Reduction Step</span><br>
    <input type="range" class="range" min="1" max="40" step="1"
    value="25" id="tabustepreduce" onchange=
    "updateSlider('tabustepreduce')"> <span id=
    "tabustepreducespan"></span><br>

  </div><img id="birdFunctionContour" src="Bird.png" alt=
  "hidden␣image" style="display:␣none">
</body>
</html>
```

A.4 minmax.js

```
Array.prototype.min = function(comparer) {

    if (this.length === 0) return null;
    if (this.length === 1) return this[0];

    comparer = (comparer || Math.min);

    var v = this[0];
    for (var i = 1; i < this.length; i++) {
        v = comparer(this[i], v);
    }

    return v;
}

Array.prototype.max = function(comparer) {

    if (this.length === 0) return null;
    if (this.length === 1) return this[0];

    comparer = (comparer || Math.max);

    var v = this[0];
    for (var i = 1; i < this.length; i++) {
        v = comparer(this[i], v);
    }

    return v;
}
```

A.5 ex3.js

```
"use␣strict";

function draw_background( canvas, ctx ){
  var backgroundElem = document.getElementById( "birdFunctionContour" );
  ctx.clearRect ( 0 , 0 , canvas.width, canvas.height );
  ctx.drawImage( backgroundElem, 0, 0, canvas.width, canvas.height );
}

function clear_screen(){
  var canvas = getCanvasAndContext()[0];
  var context = getCanvasAndContext()[1];
  draw_background( canvas, context );
}

var evalCount = 0;
var resetEvalCount = function(){
  evalCount = 0;
};
var getEvalCount = function(){
  return evalCount;
}

function bird_function( x1, x2 ){
  evalCount = evalCount + 1;
  var y = Math.sin(x1) * Math.exp( Math.pow(1 - Math.cos(x2),2 ) ) +
      Math.cos(x2) * Math.exp ( Math.pow(1 - Math.sin(x1), 2) )+
      Math.pow(x1 - x2, 2);
  return y;
}

var getCanvasAndContext;

var drawPoint = function( x1, x2, colour ){

  var canvas = getCanvasAndContext()[0];
  var context = getCanvasAndContext()[1];

  var centerX = canvas.width * ( x1 + 6 )/12;
  var centerY = canvas.height * ( -x2 + 6 ) /12;
  var radius = 2;
  //window.console.log( "X1: " + x1 + " X2: " + x2 );
  context.beginPath();
  context.arc(centerX, centerY, radius, 0, 2 * Math.PI, false);
  context.fillStyle =  colour || 'green';
  context.fill();
  context.lineWidth = 0.5;
  context.strokeStyle = 'black';
  //context.stroke();
}
var connectPoints = function( a1, a2, b1, b2 ){

  var canvas = getCanvasAndContext()[0];
  var context = getCanvasAndContext()[1];

  var aX = canvas.width * ( a1 + 6 )/12;
  var aY = canvas.height * ( -a2 + 6 ) /12;
  var bX = canvas.width * ( b1 + 6 )/12;
  var bY = canvas.height * ( -b2 + 6 ) /12;

  context.beginPath();
  context.moveTo(aX, aY);
  context.lineTo(bX, bY);
  context.stroke();
}

function clearChildren( node ){
  while (node.firstChild) {
    node.removeChild(node.firstChild);
  }
}

function displayCsvStringAsURL( string ){
  var d = document.getElementById("urlsection");
  var a = document.createElement("a");
  a.href = "data:text/csv," + encodeURIComponent( string );
  a.textContent = "data";
  clearChildren( d );
  d.appendChild(a);
}

function logMinimumHistory( minimumHistory ){
  var csvString = "Evaluations,␣x1,␣x2,␣y\n"
  minimumHistory.forEach( function( h ){
    csvString += h.evaluations + ",␣" + h.x1 + ",␣" + h.x2 + ",␣" + h.y + "\n";
  } );
  displayCsvStringAsURL( csvString );
}

```

```
function getItterationPause(){
  return document.getElementById( "pause" ).value;
}


function setRunning(){
  var d = document.getElementById("urlsection");
  clearChildren(d);
  d.textContent = "Running";
  document.getElementById("loadingAnim").style.display = "block";

  var buttons = document.getElementsByClassName( "algorithm" );
  [].forEach.call( buttons, function(b){
    b.disabled = true;
  } );
}

function finishRunning(){
  document.getElementById("loadingAnim").style.display = "";
  var buttons = document.getElementsByClassName( "algorithm" );
  [].forEach.call( buttons, function(b){
    b.disabled = false;
  } );
}

function updateSlider(id){
  var slider = document.getElementById( id );
  var label = document.getElementById( id + "span" );
  label.textContent = slider.value;
}

window.onload = function(){
  var canvas = document.getElementById( "c" );
  var ctx=canvas.getContext("2d");
  draw_background( canvas, ctx );

  var ranges = document.getElementsByClassName( "range" );
  [].forEach.call( ranges, function(r){
    updateSlider( r.id );
  } );

  getCanvasAndContext = function(){
    return [canvas, ctx];
  }
  finishRunning();
}
```

A.6 tabu.js

```
"use␣strict";

// Tabu Point class
function TabuPoint( f, x1, x2 ){
  this.x1 = x1 || 0;
  this.x2 = x2 || 0;

  this.getValue = function(){
    return [this.x1, this.x2];
  }

  // Get the value of the tabu point, the value is cached
  var lastCalled;
  this.getFValue = function(){
    if( lastCalled === undefined || !(this.isEqual( lastCalled )) ){
      this.fValue = f( this.x1, this.x2 );
      lastCalled = this.clone();
    }
    return( this.fValue );
  }
  // Is this tabu point equal to another
  this.isEqual = function( p ){
    if( p.x1 === this.x1 && p.x2 === this.x2 ){
      return true;
    }
    return false;
  }
  // Duplicate an instance of this class
  this.clone = function(){
    var o = new TabuPoint( f, this.x1, this.x2 );
    o.lastCalled = this.lastCalled;
    o.fValue = this.fValue;
    return o;
  }
  //Check if the point is within the function range
  this.valid = function(){
    if( Math.abs( this.x1 ) <= 6.0 && Math.abs( this.x2 ) < 6.0 ){
      return true;
    }
    return false;
  }
}

// function used for searching arrays

// Return the lowest value of two points
function tabuMin( a, b ){
  return a.getFValue() < b.getFValue() ? a : b;
}

// Return the highest value of two points
function tabuMax( a, b ){
  return a.getFValue() < b.getFValue() ? a : b;
}

function considerForMediumTermMemory( memory, point, size ){
  if( memory.length < size ){
    memory.push( point );
  } else {
    var max = memory.max( tabuMax )
    if( max.getFValue() > point.getFValue() ){
      // add the point
      var rIndex = memory.indexOf( max );
      memory.splice( rIndex, 1, point );
    }
  }
}


function addToMemory( memory, value, memSize ){
  memory.push( value );
  if( memory.length > memSize ){
    memory.splice( 0, memory.length - memSize );
  }
```

```
}

function getAveragePoint( memory, f ){
  var x1 = 0, x2 = 0;
  memory.forEach( function(m){
    x1 += m.x1/memory.length;
    x2 += m.x2/memory.length;
  } );
  return new TabuPoint( f, x1, x2 );
}

function setupLongTermMemory(size){
  var m = Array(size);
  for( var i = 0; i< size; i++ ){
    m[i] = Array( size );
    for( var j = 0; j< size; j++ ){
      m[i][j] = 0;
    }
  }
  return m;
}

function addToLongTermMemory(memory, point){
  var i =  Math.floor( memory.length*( point.x1 + 6 )/12.01 );
  var j =  Math.floor( memory.length*( point.x2 + 6 )/12.01 );
  memory[i][j] +=1;
}

function getDiversePointFromLongTermMemory(memory, f){
  for( var i = 0; i< memory.length; i++ ){
    for( var j = 0; j< memory[i].length; j++ ){
      if( memory[i][j] === 0 ){
        memory[i][j] = 1;
        window.console.log( "i:␣" + i + ",␣j:␣" + j + ",␣m:" + memory[i][j]);
        return new TabuPoint( f,
          12 * ((i+0.5)/memory.length) - 6,
          12 * ((j+0.5)/memory.length) - 6
        );
      }
    }
  }
  return new TabuPoint( f, 0, 0 );
}

function updateTabuMinimumHistory( minimumHist, minimum ){
  var o = {
    evaluations: getEvalCount(),
    x1: minimum.x1,
    x2: minimum.x2,
    y: minimum.getFValue()
  };
  minimumHist.push( o );
}

function getShortSize(){
  return document.getElementById( "tabushort" ).value;
}
function getMediumSize(){
  return document.getElementById( "tabumedium" ).value;
}
function getLongSize(){
  return document.getElementById( "tabulong" ).value;
```

```
}
function getIntensifyStep(){
  return document.getElementById( "tabuintensify" ).value;
}
function getDiversifyStep(){
  return document.getElementById( "tabudiversify" ).value;
}
function getReduceStep(){
  return document.getElementById( "tabustepreduce" ).value;
}


function tabu_search( f ){

  var shortTerm = [];
  var mediumTerm = [];
  var longTerm = setupLongTermMemory(getLongSize());

  var minimumHist = [];

  var point = new TabuPoint( f, 0, 0 );
  var minimum = point;

  var initialInterval = 1;
  var interval = initialInterval;
  minimum.interval = initialInterval;

  resetEvalCount();
  updateTabuMinimumHistory( minimumHist, minimum );

  setRunning();

  var isInShortTermMem = function( p ){
    var found = false;
    shortTerm.forEach( function( s ){
      if( s.isEqual( p ) ){
        found = true;
      }
    });
    return found;
  };

  var improvementCounter = 0;

  clear_screen();

  var step = function(){
    var nextSteps = [];
    ["x1", "x2"].forEach( function( param ){
      var inc = point.clone();
      var dec = point.clone();
      inc[param] += interval;
      dec[param] -= interval;
      if( !isInShortTermMem( inc ) && inc.valid() ){
        nextSteps.push( inc );
      }
      if( !isInShortTermMem( dec ) && dec.valid() ){
        nextSteps.push( dec );
      }
    } );
    var best = nextSteps.min( tabuMin );
    if( nextSteps.length === 0 ){
      window.console.log( "All␣points␣are␣Tabu" );
      best = point;
    }

    // Colour to draw the next point
    var colour = "green";

    if( best.getFValue() < point.getFValue() ){
      //pattern move
      var change = { x1: best.x1 - point.x1, x2: best.x2 - point.x2 };
      var pattern = point.clone();
      pattern.x1 += 2.0 * change.x1;
      pattern.x2 += 2.0 * change.x2;
      if( pattern.getFValue() < best.getFValue() && pattern.valid() ){
        best = pattern;
        colour = "GreenYellow";
      }
    }

    // store old point for Line Drawing purposes
    var oldPoint = point;

    point = best;

    addToMemory( shortTerm, best, getShortSize() );
    considerForMediumTermMemory( mediumTerm, best, getMediumSize() );
    addToLongTermMemory( longTerm, best );

    improvementCounter += 1;
    if( point.getFValue() < minimum.getFValue() ){
      improvementCounter = 0;
      minimum = point;
      minimum.interval = interval;
      updateTabuMinimumHistory( minimumHist, minimum );
    }

    window.console.log
    if( improvementCounter == getIntensifyStep() ){
      //Intensify
      window.console.log( "intensifying" );
      point = getAveragePoint( mediumTerm, f );
      colour = "red";
    } else if( improvementCounter == getDiversifyStep() ) {
      //Diversify
      window.console.log( "diversifying" );
      point = getDiversePointFromLongTermMemory(longTerm, f);
      window.console.log( "␣point:␣" + point.x1 + ",␣" + point.x2 );
      // clear the minimum term memory
      mediumTerm = [];
      //reset the step Size
      interval = initialInterval;
      colour = "cyan";
    } else if( improvementCounter == getReduceStep() ){
      //Step Size Reduction
      window.console.log( "Step␣Size␣Reduce" );
      point = minimum;
      minimum.interval = 0.5 * minimum.interval;
      interval = minimum.interval;
      colour = "yellow"
      improvementCounter = 0;
    } else {
      connectPoints( oldPoint.x1, oldPoint.x2, point.x1, point.x2 );
    }
    drawPoint( point.x1, point.x2, colour );

    if( point.getFValue() < minimum.getFValue() ){
      improvementCounter = 0;
      minimum = point;
      minimum.interval = interval;
      updateTabuMinimumHistory( minimumHist, minimum );
    }

    if( getEvalCount() < 1000 - 10 ){
      window.setTimeout( step, getItterationPause() );
    } else {
      drawPoint( minimum.x1, minimum.x2, "blue" );
      logMinimumHistory( minimumHist );
      finishRunning();
    }
  }
  step();
}
```

A.7 genetic_algorithm.js

```
"use␣strict";

function GAPoint(){
  var precision = 16;
  this.x1 = Array(precision);
  this.x2 = Array(precision);

  var getSinglePoint = function( xval ){
    var val = -6;
    for( var i = 0; i< precision; i++ ){
      if( xval[i] ){
        val += 6.0 * Math.pow(0.5, i);
      }
    }
    return val;
  }

  this.getValue = function(){
    return [ getSinglePoint( this.x1 ), getSinglePoint( this.x2 ) ];
  }
  this.getFunctionValue = function( f ){
    if( this.fvalue === undefined ){
      var x = this.getValue();
      this.fvalue = f( x[0], x[1] );
    }
    return this.fvalue;

  }
}

function getRandomGAPoint(){
  var point = new GAPoint();
   var randomizeBool = function(){
    var a;
    if(Math.random()<.5){
      a = true;
    } else {
      a = false;
    }
    return a;
  };
  for( var i = 0; i< point.x1.length; i++ ){
    point.x1[i] = randomizeBool();
    point.x2[i] = randomizeBool();
  }
  return point;
}

function getSeveralRandomGAPoints( N ){
  var points = Array(N);
  for( var i = 0; i< N; i++ ){
    points[i] = getRandomGAPoint();
  }
  return points;
}

function drawGAPoints( points, colour ){

  points.forEach( function(p) {
    var x = p.getValue();
    drawPoint( x[0], x[1], colour );
  } );

}


function swapPoints( points, i, j ){
  var tmp = points[i];
  points[i] = points[j];
  points[j] = tmp;
}

function getRandomInt(min, max) {
    return Math.floor(Math.random() * (max - min + 1)) + min;
}

function fisherYatesShuffle( list ){
  for( var i = 0; i< list.length; i++ ){
    var j = getRandomInt( i, list.length -1 );
    swapPoints( list, i,  j);
  }
}


function split(a, n) {
  var len = a.length;
  var out = [];
  var i = 0;
  while (i < len) {
    var size = Math.ceil((len - i) / n--);
    out.push(a.slice(i, i += size));
  }
  return out;
}

function tournamentSelect( points, f, groups ){

  var size = points.length / groups;
  fisherYatesShuffle( points );
  var splitGroups = split( points, groups );
  var comparer = function( a, b ){
    return a.getFunctionValue(f) < b.getFunctionValue(f) ? a : b;
  }

  var selected = [];

  splitGroups.forEach( function(g){
    selected.push( g.min(comparer) );
  } );
  return selected;
}

function fitnessProportionateSelect( points, f, N ){
  var sum = 0;
  //the function has range that's roughly -100 to 100
  // it goes a little lower than this, but we can still use 100 - function as a
  // fitness score
  points.forEach( function(p){
    sum += 100 - p.getFunctionValue(f);
  } );
  var selected = [];
  for( var i = 0; i < N; i++ ){
    var r = Math.random();
    var j = 0;
    while( r > 0 && j < points.length ){
      r -= (100-points[j].getFunctionValue(f))/sum;
      j++;
    }
    if( points[j-1] === undefined ) alert("scary");
    selected.push( points[j-1] );
  }
  return selected;
}

function doSingleValCrossover( p1, p2, c1, c2, val ){
  var crossoverPoint1 = getRandomInt( 1,
    Math.floor( ( p1[val].length -1) * 0.75 ) );
  var crossoverPoint2 = getRandomInt( crossoverPoint1,  p1[val].length );
  for( var i = 0; i < p1.x1.length; i++ ){
    if( i < crossoverPoint1 || i > crossoverPoint2 ){
      c1[val][i] = p1[val][i];
      c2[val][i] = p2[val][i];
    } else {
      c2[val][i] = p1[val][i];
      c1[val][i] = p2[val][i];
    }
  }
}

function crossTwoPoints( p1, p2 ){
  var c1 = new GAPoint();
  var c2 = new GAPoint();
  doSingleValCrossover( p1, p2, c1, c2, "x1" );
  doSingleValCrossover( p1, p2, c1, c2, "x2" );
  return [c1, c2];
}


function breedPoints( selectedPoints, nextGenSize ){
  var newPoints = [];
  for( var i = 0; i < nextGenSize/2; i += 1){
    // pair each point in turn with a random point, and breed them
    newPoints = newPoints.concat(
      crossTwoPoints(
        selectedPoints[i%selectedPoints.length],
        selectedPoints[getRandomInt(0, selectedPoints.length-1)]
      )
    );
  }
  return newPoints;
}


function mutatePoint( point, value ){
  var bit = getRandomInt( 0, point[value].length -1 );
  if( point[value][bit]  ){
    point[value][bit] = false;
  } else {
    point[value][bit] = true;
  }
}

function getMutationRate(){
  return document.getElementById( "gamutate" ).value;
}

function mutatePoints(points){

  var mutationProbability = getMutationRate();

  points.forEach( function(p){
    if( Math.random() < mutationProbability ){
      var value = "x1";
      if(Math.random() < 0.5 ){
        value = "x2";
      }
      mutatePoint( p, value );
    }
  } );
}

function updateGAMinimumHistory( points, f, minimumHistory ){
  var comparer = function( a, b ){
    return a.getFunctionValue(f) < b.getFunctionValue(f) ? a : b;
  }

  var min = points.min(comparer)
  var x = min.getValue();
  var y = min.getFunctionValue( f );

  if( minimumHistory.length === 0 ||
    minimumHistory[minimumHistory.length-1].y > y ){
    var o = {};
    o.x1 = x[0];
    o.x2 = x[1];
    o.y = y;
    o.evaluations = getEvalCount();
    minimumHistory.push( o );
  }
}


function getGAPopulation(){
  return document.getElementById( "gapopulation" ).value;
```

```javascript
}

function getGAParentsCount(){
  return document.getElementById( "gaparents" ).value;
}

function getGAParents(points, f){
  var strategy = document.getElementById( "gastratergy" ).value;
  if( strategy === "tournament" ){
    return  tournamentSelect( points, f, getGAParentsCount() );
  } else if( strategy === "fitness" ){
    return fitnessProportionateSelect( points, f, getGAParentsCount() );
  }
}

function GAItteration( points, f, minimumHistory ){
  clear_screen();

  drawGAPoints( points, "green" );

  var reproducing = getGAParents(points,  f );

  drawGAPoints( reproducing, "red" );

  var nextGen = breedPoints(reproducing, points.length);

  updateGAMinimumHistory( points, f, minimumHistory);

  var last = minimumHistory[minimumHistory.length-1];

  drawPoint( last.x1, last.x2, "blue" );

  mutatePoints( nextGen );

  if( getEvalCount() < 1000 - nextGen.length){
    window.setTimeout( function(){
      GAItteration( nextGen, f, minimumHistory );
    }, getItterationPause() );
  } else {
    logMinimumHistory( minimumHistory );
    finishRunning();
  }
}

function genetic_algorithm( f ){
  resetEvalCount();
  var nPoints = getGAPopulation();
  var points = getSeveralRandomGAPoints(nPoints);
  setRunning();
  GAItteration(points, f, []);
}
```