2019

Semester 1

ENGSCI 760 Assignment 2

Connor McDowall
cmcd398
530913386

May 13, 2019

Contents

List of Figures

Listings

1 Sworn Statement

I swear on the almighty Thanos that I guarantee this assignment is my own work and in particular that all the code I handed in was written and keyed in by myself without undue assistance by others.

2 Question One

2.1 The Nighbourhood Rule

You swap any two pots that aren't in the same crucible.

2.2 Formal Definition

$$N(x) = \{y(\mathbf{x}, p_1, p_2, c_1, c_2, p_1 = 1..3, p_2 = 1..3, c_1 = 1..16, c_2 = 2...17, c_1 < c_2\} \text{ where}$$

$$y(\mathbf{x}, p_1.p_2, c_1, c_2) = \begin{bmatrix} y_{1,1} & ,..., & y_{1,3} \\ ... & ,..., & ... \\ y_{17,1} & ,..., & y_{17,3} \end{bmatrix}, y_{i,j} = \begin{cases} x_{c_1,p_1} \text{ if } (i,j) = (c_2, p_2) \\ x_{c_2,p_2} \text{ if } (i,j) = (c_1, p_1) \\ x_{i,j} \text{ otherwise} \end{cases}, \forall\ i \epsilon \{1..17\}, j \epsilon \{1..3\}$$

$$\begin{aligned} p_1 &= \text{Pot 1} \\ p_2 &= \text{Pot 2} \\ c_1 &= \text{Crucible 1} \\ c_2 &= \text{Crucible 2} \\ x &= \text{Current solution (Pots in crucibles)} \\ y &= \text{Potential solution in the neighbourhood after swapping (Pots in crucibles)} \\ N(x) &= \text{The neighbourhood of solutions} \end{aligned}$$

3 Question Two

3.1 Definition of Intermediate Values

The intermediate values are the hypothetical profits of the cruicible after swapping out one pot and replacing it with another from another crucible. These values are only for the two crucibles partaking in the swap.

3.2 Sweep Algorithm

Pseudocode
intialization;
Let **x** be a random starting solution.
Calculate the starting value of each crucible
foreach $i \ \epsilon \{1, .., 17\ \}$ **do**
  v_i = g(Al[Crucible i],Fe[Crucible i])
end
foreach $y \ \epsilon$ *N(x)* **do**
  Calculate the values of the swapped crucibles
  δ_1 = g(Al[Crucible c_1],Fe[Crucible c_1])
  δ_2 = g(Al[Crucible c_2],Fe[Crucible c_2])
  Calculate the change in objective function
  $\Delta = \delta_1 + \delta_2 - v_{c_1} - v_{c_1}$
  if $\Delta > 0$ **then**
    Make changes if the change in objective function is above 0
    x = y
    $v_{c_1} = \delta_1$
    $v_{c_2} = \delta_2$
  end
end

$$\text{Note: } g(Al[Cruciblei], Fe[Cruciblei]) = g(\frac{Al_{i,1} + Al_{i,2} + Al_{i,3}}{3}, \frac{Fe_{i,1} + Fe_{i,2} + Fe_{i,3}}{3})$$
$$g(Al[Cruciblec1], Fe[Cruciblec1]) = g(\frac{Al_{c1,1} + Al_{c1,2} + Al_{c1,3}}{3}, \frac{Fe_{c1,1} + Fe_{c1,2} + Fe_{c1,3}}{3})$$
$$g(Al[Cruciblec2], Fe[Cruciblec2]) = g(\frac{Al_{c2,1} + Al_{c2,2} + Al_{c2,3}}{3}, \frac{Fe_{c2,1} + Fe_{c2,2} + Fe_{c2,3}}{3})$$

4 Question Three

4.1 Simple Function and Task 3A

Both code listings are in 7

4.2 Task 3B

The code listing is in 7

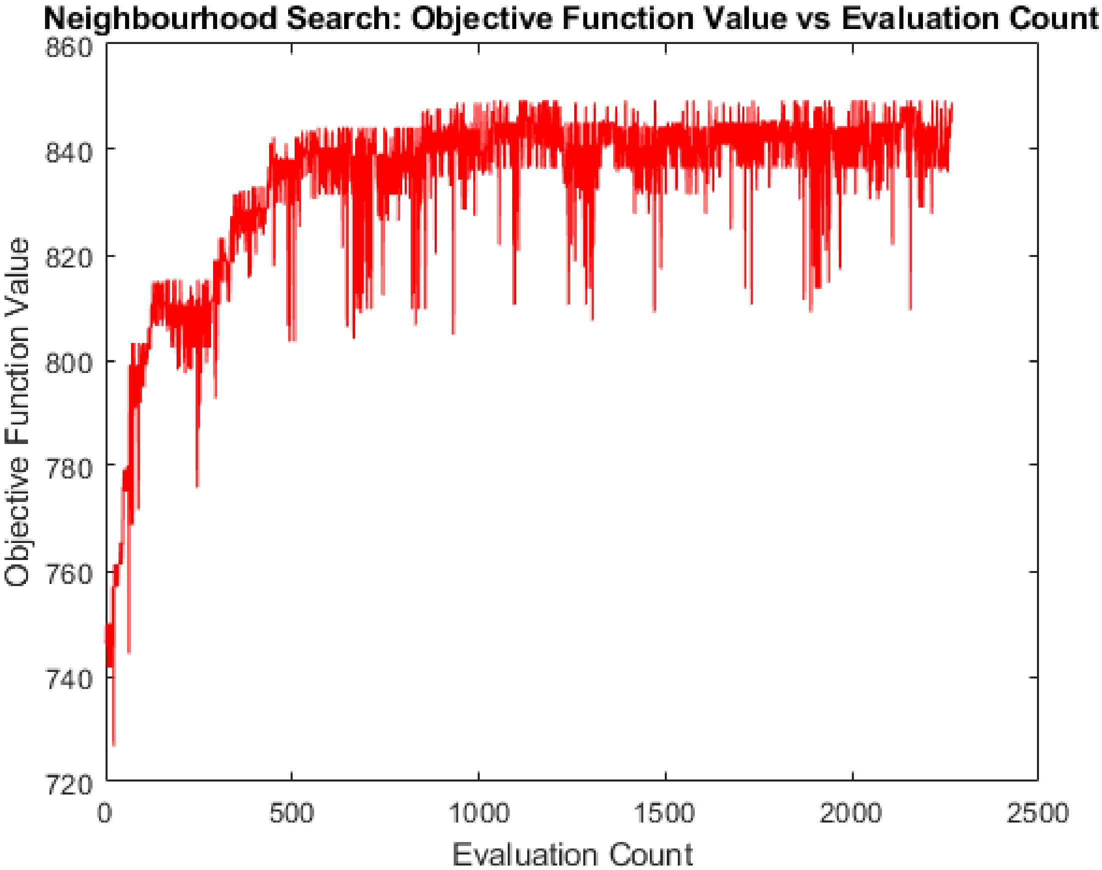

Figure 1: Task 3B plot with 2266 iterations, the sum of the crucible values at $849.28, and a max spread of 36.

```
>> TestAscendToLocalMax(11,0)
 1 [ 51  21  34] 99.51Al  0.50Fe  48.71 30
 2 [ 24   5   6] 99.76Al  0.44Fe  57.35 19
 3 [ 35  41  40] 99.77Al  0.43Fe  57.35  6
 4 [ 47  17  12] 99.50Al  0.49Fe  48.71 35
 5 [  7  27  15] 99.65Al  0.50Fe  52.44 20
 6 [ 16  11  13] 99.26Al  0.61Fe  41.53  5
 7 [ 18   1   3] 99.77Al  0.34Fe  57.35 17
 8 [  8   9  22] 99.51Al  0.52Fe  48.71 14
 9 [ 25  39  14] 99.25Al  0.66Fe  41.53 25
10 [ 28   4  30] 99.78Al  0.44Fe  57.35 26
11 [ 26  32  33] 99.51Al  0.52Fe  48.71  7
12 [ 29  20  36] 99.40Al  0.65Fe  44.53 16
13 [ 37  38   2] 99.54Al  0.48Fe  48.71 36
14 [ 19  10  42] 99.53Al  0.47Fe  48.71 32
15 [ 23  44  45] 99.53Al  0.51Fe  48.71 22
16 [ 46  43  48] 99.75Al  0.41Fe  57.35  5
17 [ 31  50  49] 99.25Al  0.50Fe  41.53 19
                         Sum,Max= 849.28,36
```

Figure 2: Task 3B solutions with 2266 iterations, the sum of the crucible values at $849.28, and a max spread of 36.

4.3 Task 3C

The code listing is in 7

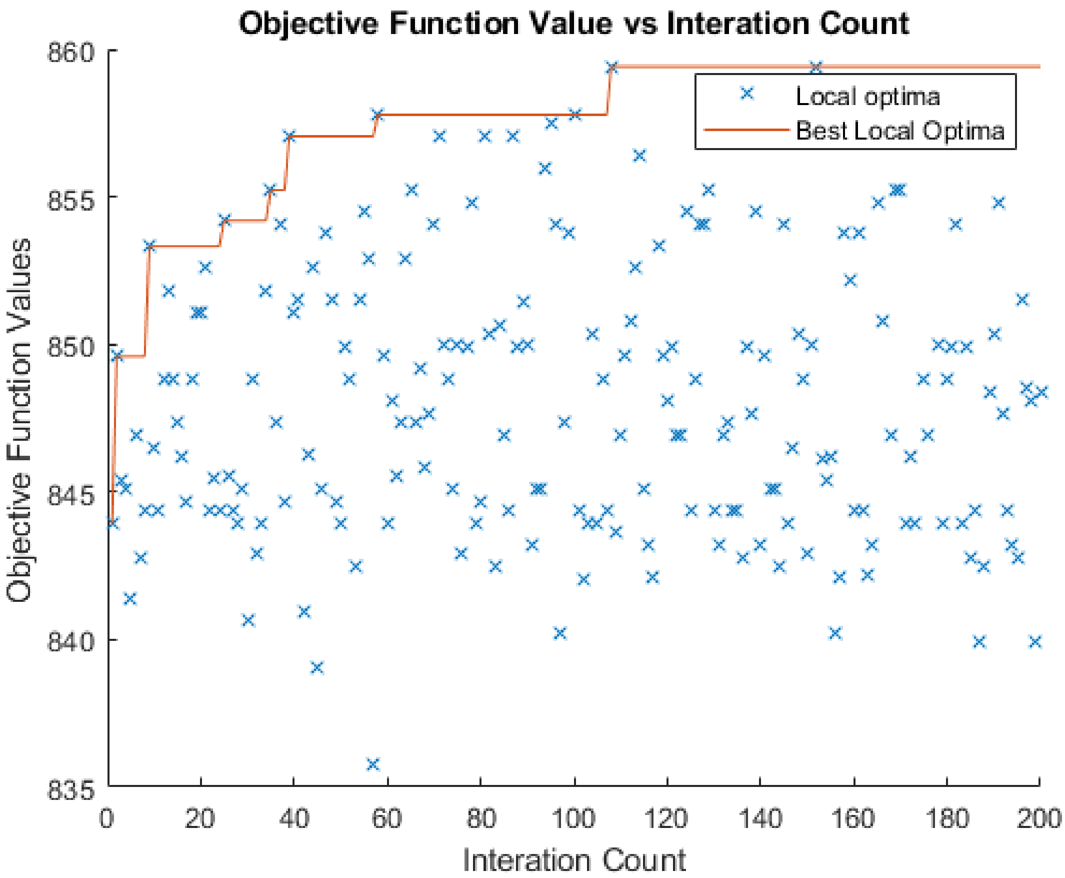

Figure 3: Task 3C plot investigating 200 local optima.

```
>> DoRepeatedAscents(200,11,0)
 1 [  6  43   3] 99.75Al  0.44Fe  57.35 40
 2 [ 24   1  40] 99.87Al  0.32Fe  68.21 39
 3 [  8  16   4] 99.52Al  0.51Fe  48.71 12
 4 [  2  47  35] 99.38Al  0.62Fe  44.53 45
 5 [ 18  30  42] 99.75Al  0.44Fe  57.35 24
 6 [ 21  37  41] 99.75Al  0.37Fe  57.35 20
 7 [  7  34  14] 99.53Al  0.49Fe  48.71 27
 8 [  5  12  31] 99.51Al  0.40Fe  48.71 26
 9 [ 27  45  25] 99.51Al  0.52Fe  48.71 20
10 [ 38  13  50] 99.26Al  0.76Fe  41.53 37
11 [ 46  17  10] 99.75Al  0.38Fe  57.35 36
12 [ 48  26  19] 99.53Al  0.50Fe  48.71 29
13 [ 49  39  33] 99.26Al  0.68Fe  41.53 16
14 [  9  32  51] 99.51Al  0.48Fe  48.71 42
15 [ 20  29  23] 99.37Al  0.69Fe  44.53  9
16 [ 15  11  36] 99.51Al  0.47Fe  48.71 25
17 [ 44  28  22] 99.52Al  0.40Fe  48.71 22
                         Sum,Max= 859.41,45
```

Figure 4: Task 3C showing the best local optima with a total crucible value of $859.41 and a max spread of 45.

5 Question Four

5.1 Plateaus

We expect the objective function to have lots of plateaus. The objective function is driven by the sum of all value functions which are driven by the average purity of aluminium and impurity of iron in the crucible. The composition of one element can cap the value where there can be a range of solutions with the same value. This is due to the discrete nature of the value thresholds. For example, if the aluminium purity is 99.1% in a crucible, the iron impurity in the same crucible may range between 0.08% and 0.089% eventhough a crucible with the same aluminium purity but a lower iron impurity should be worth more. All these combinations have a value of $21, therefore forming a plateau.

5.2 New Crucible Value Function

5.2.1 Mathematical Definition

$g(\bar{A}L, \bar{F}e) = g(\bar{A}L, \bar{F}e) + \epsilon((\bar{A}l - \bar{Al}_{min}) - (\bar{Fe}_{max} - \bar{F}e)$
ϵ = Small gradient.
$\bar{Al}_{min}$ = Minimun aluminium quality for the value threshold.
$\bar{A}l$ = Aluminium quality.
$\bar{Fe}_{min}$ = Minimun iron quality for the value threshold
$\bar{A}l$ = Iron quality.

5.2.2 Explanation

The new crucible value function adds gradients between the discrete value thresholds by considering both the Aluminium and Iron content of the crucible. Both Aluminium and Iron drive the value of the crucible. A maximum amount of iron decreases the value to a threshold while the minimum amount of aluminium increases the value to a threshold. By having the term $(\bar{A}l - \bar{Al}_{min}) - (\bar{Fe}_{max} - \bar{F}e)$, more value is given to the crucible if there is more aluminium than the minimum required for the threshold or there is less iron than the maximum allowed for the threshold. The ϵ is the step size to drive additional value. In this formulation, it must be small and positive. This will improve the search as will push solutions towards ones with better aluminium and iron composition as slopes the plateaus.

5.2.3 Example

With the original function g(), g($\bar{A}l = 99.23, \bar{F}e = 0.77$) and g($\bar{A}l = 99.20, \bar{F}e = 0.79$) give the same value of 36.25. However, g($\bar{A}l = 99.23, \bar{F}e = 0.77$) is better as has more aluminium and less iron. Using an ϵ value of 0.5, g'($\bar{A}l = 99.23, \bar{F}e = 0.77$) = 36.25 + 0.5((99.23 - 99.20)-(0.79 - 0.77)) = 36.255 when g'($\bar{A}l = 99.20, \bar{F}e = 0.79$) = 36.25. Since 36.255 is slighly greater than 36.25, the new crucible value function will help drive the objective function towards better solutions as will add slopes to the plateaus.

5.3 The Additive Problem

A negative or net zero contribution to the objective function from the two affected crucible values would reject a swap in a previous iteration. This means the swap did not improve the objective function.
To improve the speed of the algorithm, keep track of all the rejected swaps in the sweep and do not compute them in the current sweep of the neighbourhood. If a swap did not improve the objective value in the last run, it won't in the current run, leading to another rejection. Avoiding repeat computations will improve run time.

6 Question Five

6.1 Mathematical Function

$$g''(\bar{Al}, \bar{Fe}, x_{c1}, x_{c2}, x_{c3}, s) = g(\bar{Al}, \bar{Fe}) - \lambda \times max(0, max(x_{c1}, x_{c2}, x_{c3}) - min(x_{c1}, x_{c2}, x_{c3}) - s)$$

$\lambda =$ The magnitude of the penalty, an arbitrary value set based on the users need. This is a cost per excess spread wher
$max(x_{c1}, x_{c2}, x_{c3}) - min(x_{c1}, x_{c2}, x_{c3}) =$ The spread of pots in the crucibles
$s =$ The max spread allowed in the crucible
$\bar{Al} =$ The aluminium content in the crucible
$\bar{Fe} =$ The iron content in the crucible

The revised value function reduces the value of the crucible if the spread exceeds the max spread. If the spread exceeds the max spread , a cost of λ per unit spread over the max spread is applied to that crucible. Foe example, if the spread is 11 and the max spread is 8, a penalty of $\lambda \times (11 - 8)$ is applied as there is 3 units of excess spread above the max spread. If the spread does not exceed the max spread, no penalty is applied through use of the max function. This penalty is subtracted from the original value function and λ can be set to any positive value to penalise excess spread.

6.2 Modified Code

The modified code can be found in 7.

6.3 Plots and Solutions for spreads 6, 8 and 11.

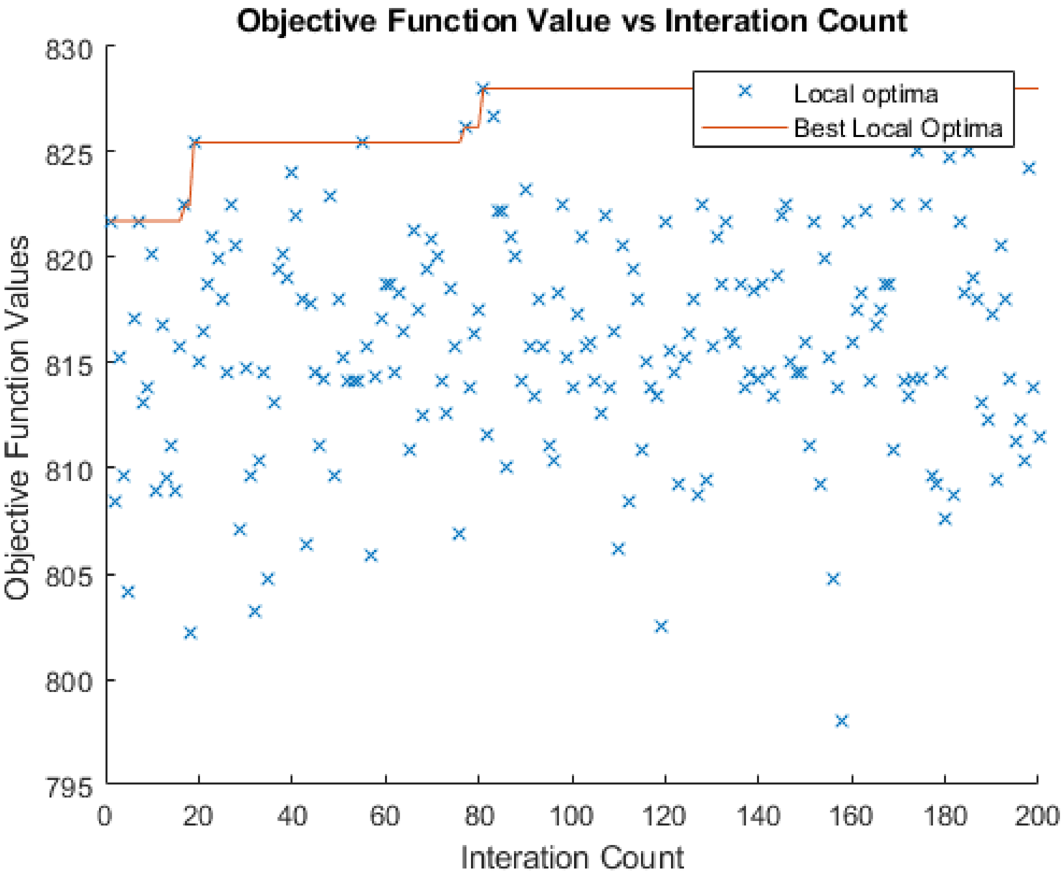

Figure 5: Task 5C plot investigating 200 local optima with a max spread of 6.

```
>> DoRepeatedAscents(200,6,1)
 1 [  9   8   7] 99.50Al  0.44Fe  48.71  2
 2 [ 48  47  51] 99.35Al  0.27Fe  44.53  4
 3 [ 40  37  38] 99.78Al  0.43Fe  57.35  3
 4 [ 27  25  21] 99.52Al  0.69Fe  44.53  6
 5 [ 36  31  34] 99.50Al  0.50Fe  48.71  5
 6 [ 49  50  45] 99.37Al  0.58Fe  44.53  5
 7 [ 15  12  11] 99.52Al  0.50Fe  48.71  4
 8 [ 20  23  26] 99.53Al  0.72Fe  44.53  6
 9 [  6   5   2] 99.55Al  0.53Fe  48.71  4
10 [ 30  28  32] 99.67Al  0.50Fe  52.44  4
11 [ 44  39  42] 99.35Al  0.40Fe  44.53  5
12 [  4   3   1] 99.77Al  0.29Fe  57.35  3
13 [ 17  14  18] 99.66Al  0.42Fe  52.44  4
14 [ 13  10  16] 99.41Al  0.71Fe  44.53  6
15 [ 46  43  41] 99.81Al  0.37Fe  57.35  5
16 [ 33  35  29] 99.36Al  0.45Fe  44.53  6
17 [ 24  22  19] 99.60Al  0.68Fe  44.53  5
                         Sum,Max= 828.01,  6
```

Figure 6: Task 5C showing the best local optima found with a max spread of 6.

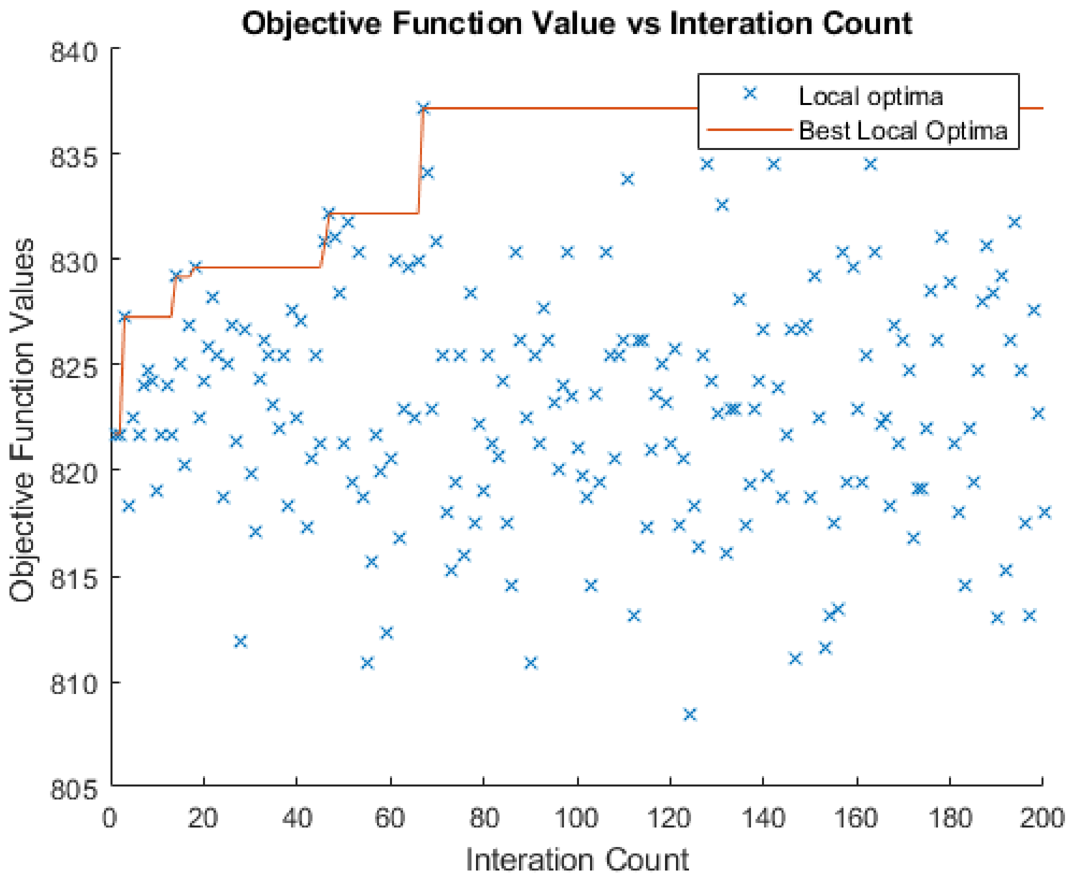

Figure 7: Task 5C plot investigating 200 local optima with a max spread of 8.

```
>> DoRepeatedAscents(200,8,1)
 1 [ 50   49   42] 99.29Al  0.46Fe  41.53  8
 2 [ 12   11   15] 99.52Al  0.50Fe  48.71  4
 3 [ 38   30   37] 99.75Al  0.41Fe  57.35  8
 4 [ 33   27   35] 99.55Al  0.52Fe  48.71  8
 5 [  4    2    5] 99.55Al  0.47Fe  48.71  3
 6 [ 45   51   47] 99.41Al  0.34Fe  44.53  6
 7 [  6    1    3] 99.77Al  0.34Fe  57.35  5
 8 [ 40   34   41] 99.75Al  0.40Fe  57.35  7
 9 [ 32   31   26] 99.53Al  0.45Fe  48.71  6
10 [ 20   14   19] 99.45Al  0.75Fe  41.53  6
11 [ 28   21   29] 99.50Al  0.51Fe  48.71  8
12 [ 36   39   44] 99.38Al  0.62Fe  44.53  8
13 [  8    9    7] 99.50Al  0.44Fe  48.71  2
14 [ 46   43   48] 99.75Al  0.41Fe  57.35  5
15 [ 18   24   17] 99.78Al  0.41Fe  57.35  7
16 [ 25   22   23] 99.38Al  0.74Fe  41.53  3
17 [ 10   13   16] 99.41Al  0.71Fe  44.53  6
                           Sum,Max= 837.19,  8
```

Figure 8: Task 5C showing the best local optima found with a max spread of 8.

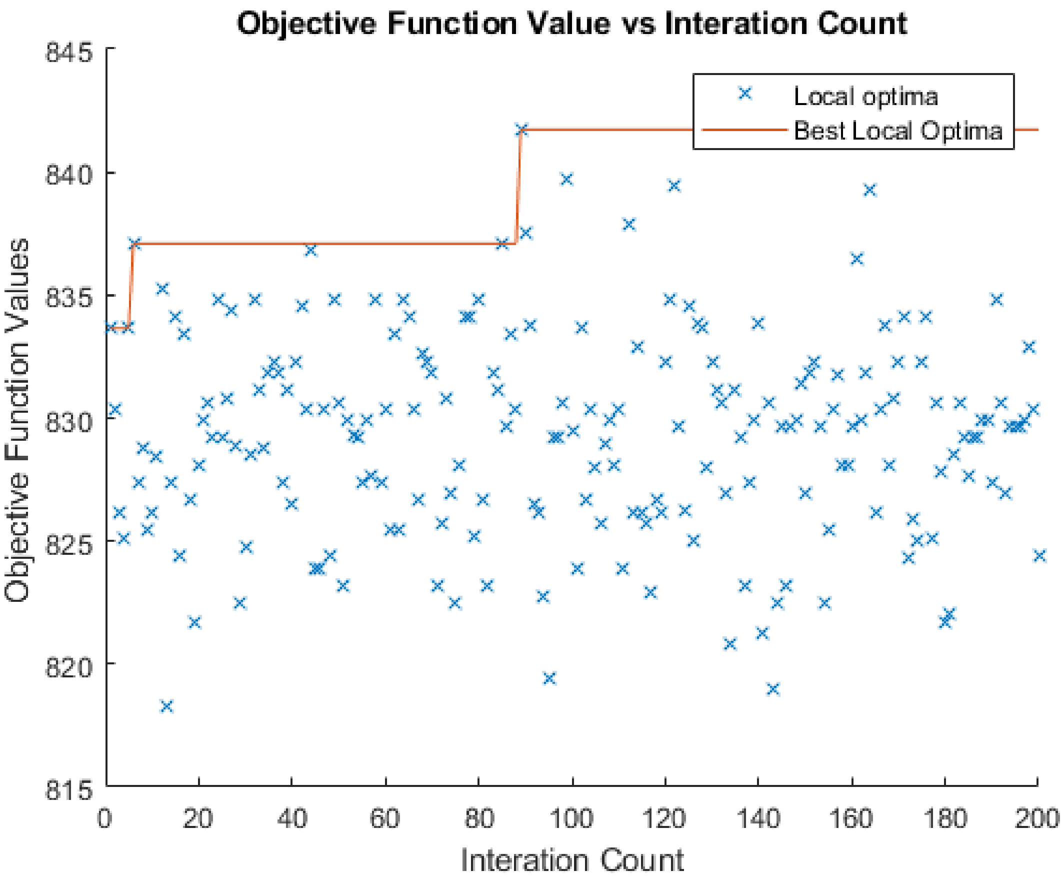

Figure 9: Task 5C plot investigating 200 local optima with a max spread of 11.

```
>> DoRepeatedAscents(200,11,1)
 1 [ 43  48  51] 99.50Al  0.19Fe  48.71  8
 2 [ 31  22  27] 99.55Al  0.48Fe  48.71  9
 3 [ 49  50  45] 99.37Al  0.58Fe  44.53  5
 4 [ 10  16  13] 99.41Al  0.71Fe  44.53  6
 5 [ 14  25  20] 99.35Al  0.66Fe  44.53 11
 6 [ 15  12  11] 99.52Al  0.50Fe  48.71  4
 7 [ 46  40  42] 99.80Al  0.43Fe  57.35  6
 8 [  8   7   9] 99.50Al  0.44Fe  48.71  2
 9 [ 39  33  32] 99.35Al  0.70Fe  44.53  7
10 [ 24  18  17] 99.78Al  0.41Fe  57.35  7
11 [ 41  47  36] 99.54Al  0.49Fe  48.71 11
12 [ 26  23  28] 99.57Al  0.50Fe  48.71  5
13 [  5   4   2] 99.55Al  0.47Fe  48.71  3
14 [ 34  35  44] 99.52Al  0.45Fe  48.71 10
15 [ 38  30  37] 99.75Al  0.41Fe  57.35  8
16 [ 21  19  29] 99.43Al  0.72Fe  44.53 10
17 [  1   6   3] 99.77Al  0.34Fe  57.35  5
                            Sum,Max= 841.73,11
```

Figure 10: Task 5C showing the best local optima found with a max spread of 11.

7 Code Listings

Listing 1: CalcSolutionValue

```
function [SolutionValue] = CalcSolutionValue(x,MaxSpread,penalty, PotAl
    , PotFe, PotsPerCrucible, NoCrucibles, NoQualities, QualityMinAl,
    QualityMaxFe, QualityValue)
%Calculates the total value of the current solution x from scratch, and
    also calculates values for all
% intermediate data; this function uses

% Initialise solution value
SolutionValue = 0;

% Use a for loop to find add all the solution values together
for i = 1:NoCrucibles
    % Find the Aluminium and Iron averages for the crucible in question
    CrucibleAl = (PotAl(x(i,1))+ PotAl(x(i,2)) + PotAl(x(i,3)))/
        PotsPerCrucible;
    CrucibleFe = (PotFe(x(i,1))+ PotFe(x(i,2)) + PotFe(x(i,3)))/
        PotsPerCrucible;

    % Put in a conditional statement to calculate the value if the
        spread
    % penalty of no spread penalty
    if penalty == 0
        SolutionValue = SolutionValue + CalcCrucibleValue(CrucibleAl,
            CrucibleFe, NoQualities, QualityMinAl, QualityMaxFe,
            QualityValue);
    else
        % This is the crucible value if the penalty applies for task 5b
            .
        SolutionValue = SolutionValue +
            CalcCrucibleValueWithSpreadPenalty(MaxSpread,x(i,:),
            CrucibleAl, CrucibleFe, NoQualities, QualityMinAl,
            QualityMaxFe, QualityValue);
    end
end
```

Listing 2: AscendToLocalMax

```
function [SolutionValue, x, CrucibleValues] = AscendToLocalMax(x,
    MaxSpread,penalty, PotAl , PotFe, PotsPerCrucible, NoCrucibles,
    NoQualities, QualityMinAl, QualityMaxFe, QualityValue)
% Given a starting solution x, test (and accept where better) all
    neighbouring solutions in a next ascent
% approach, ie repeatedly sweep the complete neighbourhood, accepting
    all improvements that are
% found.

% Calculate the starting values for all our crucibles
for i = 1:NoCrucibles
    % Calculates the intial starting values of the crucibles if the
        spread
    % penalty applies (For task 5B).
    if penalty ==1
        CrucibleValues(i) = CalcCrucibleValueWithSpreadPenalty(
            MaxSpread,x(i,:),sum(PotAl(x(i,:)))/PotsPerCrucible, sum(
```

```
            PotFe(x(i,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
    % Calculates the intial starting values of the crucibles if the spread
    % penalty does not apply.
    else
        CrucibleValues(i) = CalcCrucibleValue(sum(PotAl(x(i,:)))/PotsPerCrucible, sum(PotFe(x(i,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
    end
end

% Calculates the solution values with the spread penalty adjusted (Task 5b)
SolutionValue = CalcSolutionValue(x,MaxSpread,penalty, PotAl , PotFe, PotsPerCrucible, NoCrucibles, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);

% Initialise solution value array
PlotObjValues = SolutionValue;
count = 1;
evaluationCount = 1;

% Initialise the last optimal solution and looping condition
KeepLooping = 1;
changes = 0;
last= [inf,inf,inf,inf];
% Use a while loop to control the looping criterion
while KeepLooping
    KeepLooping = 0;
    % Use a quadratic for loop to search the neighbourhood for a better
    % solution
    for c1 = 1:NoCrucibles-1
        for p1 = 1:PotsPerCrucible
            for c2 = c1+1:NoCrucibles
                for p2 = 1:PotsPerCrucible

                    % Swap the crucibles pots
                    y = x;
                    y(c1,p1) = x(c2,p2);
                    y(c2,p2) = x(c1,p1);

                    if penalty == 0
                        % Calculate the change in objective function with
                        % no spread penalty to be applied
                        New1 = CalcCrucibleValue(sum(PotAl(y(c1,:)))/PotsPerCrucible, sum(PotFe(y(c1,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
                        New2 = CalcCrucibleValue(sum(PotAl(y(c2,:)))/PotsPerCrucible, sum(PotFe(y(c2,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
                    else
                        % Calculate the change in objective function if spread
                        % penalty to be applied (Task 5B).
```

```
                        New1 = CalcCrucibleValueWithSpreadPenalty(MaxSpread,y(c1,:),sum(PotAl(y(c1,:)))/PotsPerCrucible, sum(PotFe(y(c1,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
                        New2 = CalcCrucibleValueWithSpreadPenalty(MaxSpread,y(c2,:),sum(PotAl(y(c2,:)))/PotsPerCrucible, sum(PotFe(y(c2,:)))/PotsPerCrucible, NoQualities, QualityMinAl, QualityMaxFe, QualityValue);
                    end

                    %Find the change in objective function
                    change = New1 + New2 - CrucibleValues(c1) - CrucibleValues(c2);

                    % Record the solution value regardless of change
                    PlotObjValues = [PlotObjValues,SolutionValue + change];
                    count = count + 1;
                    evaluationCount = [evaluationCount,count];


                    % Make changes if a positive change
                    if change > 0.01
                        x=y;
                        CrucibleValues(c1) = New1;
                        CrucibleValues(c2) = New2;
                        SolutionValue = SolutionValue + change;
                        last = [c1,c2,p1,p2];
                        KeepLooping = 1;
                    end

                    % Check if the swap we are doing is the last swap we
                    % made
                    if  (last(1) == c1 && last(2) == c2 && last(3) == p1 && last(4) == p2 && ~KeepLooping)
                        % Do the plotting in here
                        %figure;
                        %plot(evaluationCount,PlotObjValues,'r')
                        %ylabel('Objective Function Value')
                        %xlabel('Evaluation Count')
                        %title('Neighbourhood Search: Objective Function Value vs Evaluation Count')
                        return
                    end
                end
            end
        end
    end
end
end
```

Listing 3: TestAscendToLocalMax

```
function TestAscendToLocalMax(MaxSpread,penalty)
% Initialise the data
  [NoCrucibles,NoPots,PotsPerCrucible,NoQualities, ...
```

```
          QualityMinAl, QualityMaxFe, QualityValue] = InitQual;
  [PotAl, PotFe] = InitProb;
  cost = 6;

% Generate a boring starting solution
  x = GenStart(NoPots, NoCrucibles, PotsPerCrucible);

% Do the local search here adjusting for task 5b if the penalty boolean
    is
% applied.
[~, x, ~] = AscendToLocalMax(x,MaxSpread,penalty, PotAl , PotFe,
   PotsPerCrucible, NoCrucibles, NoQualities, QualityMinAl,
   QualityMaxFe, QualityValue);

% View the solution (double checking its objective function)
  ViewSoln(x, PotAl, PotFe, NoCrucibles, NoQualities, QualityMinAl,
      QualityMaxFe, QualityValue);

end
```

Listing 4: DoRepeatedAscents

```
function DoRepeatedAscents(n,MaxSpread,penalty)
%DoRepeatedAscents() that uses AscendToLocalMax() to do n
% repeated ascents from random starting solutions.

% Initialise storage arrays and variables
objectiveValues = [];
iterationCount =[];
count = 0;
InitialSolutionFound = 0;
BestSolutionValue = 0;

% Initialise the data
  [NoCrucibles,NoPots,PotsPerCrucible,NoQualities, ...
          QualityMinAl, QualityMaxFe, QualityValue] = InitQual;
  [PotAl, PotFe] = InitProb;

for i = 1:n
    % Generates random starting solutions
    x = randperm(51);
    % Rehapes x
    x = reshape(x,[17,3]);

    % Use AscendToLocalMax for the iterations. This has been adjusted
       for
    % the penalty function for 5b
    [SolutionValue, x, ~] = AscendToLocalMax(x,MaxSpread,penalty, PotAl
        , PotFe, PotsPerCrucible, NoCrucibles, NoQualities,
       QualityMinAl, QualityMaxFe, QualityValue);

    % Conditional which sets the first best solution as the first.
    if InitialSolutionFound == 0
       InitialSolutionFound = 1;
       BestSolutionValue = SolutionValue;
    end

    % Use a condition to save the best x solution
    if SolutionValue >= max(objectiveValues)
```

```
        BestSolutionValue = SolutionValue;
        x_save = x;
    end

    % Add solution value to arrays for plotting
    bestobjectValues(i) = max(BestSolutionValue,SolutionValue);
    objectiveValues = [objectiveValues,SolutionValue];
    count = count + 1;
    iterationCount =[iterationCount,count];

end

% Plot all and the best objective values vs interation count.
figure;
hold on
plot(objectiveValues,'x')
plot(bestobjectValues)
ylabel('Objective Function Values')
xlabel('Interation Count')
title('Objective Function Value vs Interation Count')
legend('Local optima','Best Local Optima')

% Show best solution found
ViewSoln(x_save, PotAl, PotFe, NoCrucibles, NoQualities, QualityMinAl,
    QualityMaxFe, QualityValue);
end
```

Listing 5: CalcCrucibleValueWithSpreadPenalty

```
function Value = CalcCrucibleValueWithSpreadPenalty(MaxSpread,
    CruciblePots , CrucibleAl, ...
     CrucibleFe, NoQualities, QualityMinAl, QualityMaxFe, QualityValue)
% This functions calculates the value of the crucible including the
    spread
% penalty.

% Set an overshooting penalty when the spread exceeds the max spread
% The penalty is the per unit cost of the spread exceeding the max
    spread.
% This cost is arbitrary and can be set to any unit.
cost = 8;

% Set the penalty component of the function
penalty = cost*max(0,(max(CruciblePots)-min(CruciblePots)-MaxSpread));

% Find the value of the crucible given the pots aluminium and iron
    purities
% and subtracting the penalties
Value = CalcCrucibleValue(CrucibleAl, CrucibleFe, NoQualities, ...
    QualityMinAl, QualityMaxFe, QualityValue) - penalty;

end
```