B.E. PROJECT ON
EDGE DETECTION IN MEDICAL IMAGES USING GENETIC ALGORITHM

Under the guidance of
Dr. Tarun Kumar Rawat

Submitted By:
Riya Ghai (2017UEC2083)
Abhishek Rawat (2017UEC2116)
Avishi Rathore (2017UEC2121)
Prateek (2016UEC2093)

Introduction

- In our project we have considered the problem of edge detection as a problem of cost minimization
- The edge configurations are seen as two-dimensional chromosomes whose fitness values are inversely proportional to their costs
- We have used the knowledge-augmented mutation operator which uses the principle of local edge structure that results in rapid convergence
- To minimize the cost function we evaluated the quality edge configuration.The function is a linear sum of weighted cost factors.

Introduction (cont..)

- Edges are detected by finding the edge configurations that minimize the cost function
- There are many edge detection techniques which exist like the gradient operator and the laplacian operator techniques. Most techniques out there are not able to identify the true edges that are related to physical boundaries of the object in that image. In these techniques it is considered that the edge detection problem is dependent on the output of the edge detector at a particular pixel location i.e the characteristic of the edge structure at a particular pixel location is mostly ignored.
- To overcome these problems, a new technique has emerged i.e edge detection by minimizing the cost function. In this technique, a cost measure is allotted to the edge image. This cost measure is a function of the local edge structure which is dependent on the factors like edge thickness, dissimilarity, fragmentation and continuity.

Genetic Algorithm

1. An initial population of chromosomes is created randomly and this is refined using the threshold in dissimilarity matrix to narrow down our region of concern, and each individual is evaluated by the objective function
2. Two mates are selected for reproduction with probabilities in proportion to their fitness values (in our project the candidate with the least cost is the fittest) using roulette wheel selection
3. Crossover and mutation operators are applied to the selected mates passing on the probability of mutation and probability of crossover, and offsprings are generated.
4. Each individual offspring generated using crossover and mutation is evaluated by the objective function

Genetic Algorithm

5. Steps 2 to 4 are repeated until an entirely new candidate population of chromosomes is generated

6. The pre-existing population is replaced by the new population

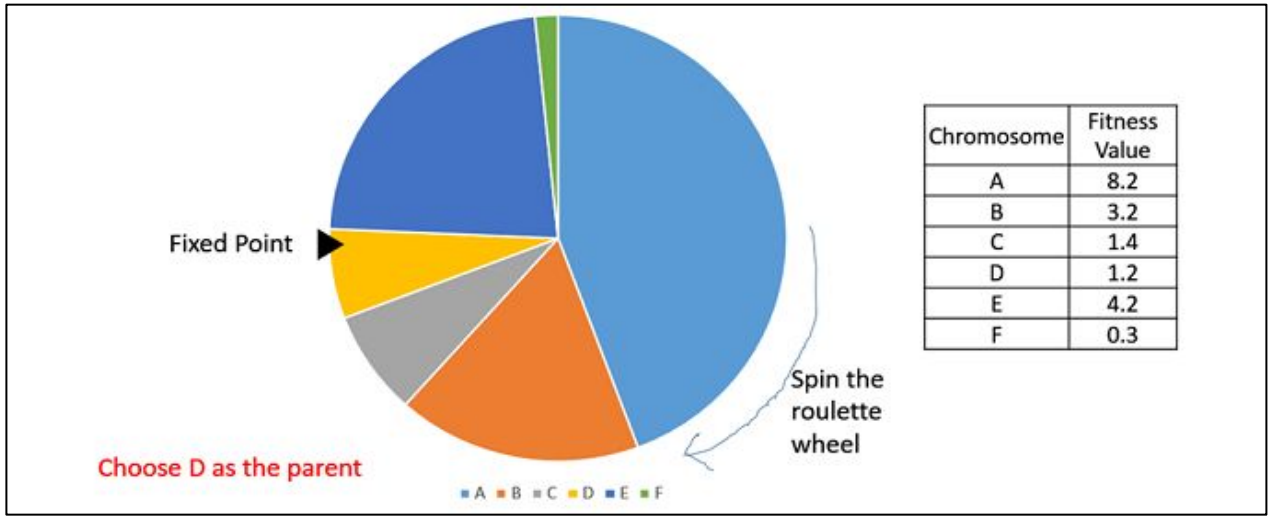
7. If the number of iterations is not finished and the most optimum cost is above a certain value, go to step 2, otherwise return the best chromosome that is the candidate edge image in the new population as the solution to the problem

Main Components of Genetic Algorithm

- **Selection**
- **Crossover**
- **Mutation**
- **Dissimilarity Function**
- **Cost Function**

Selection

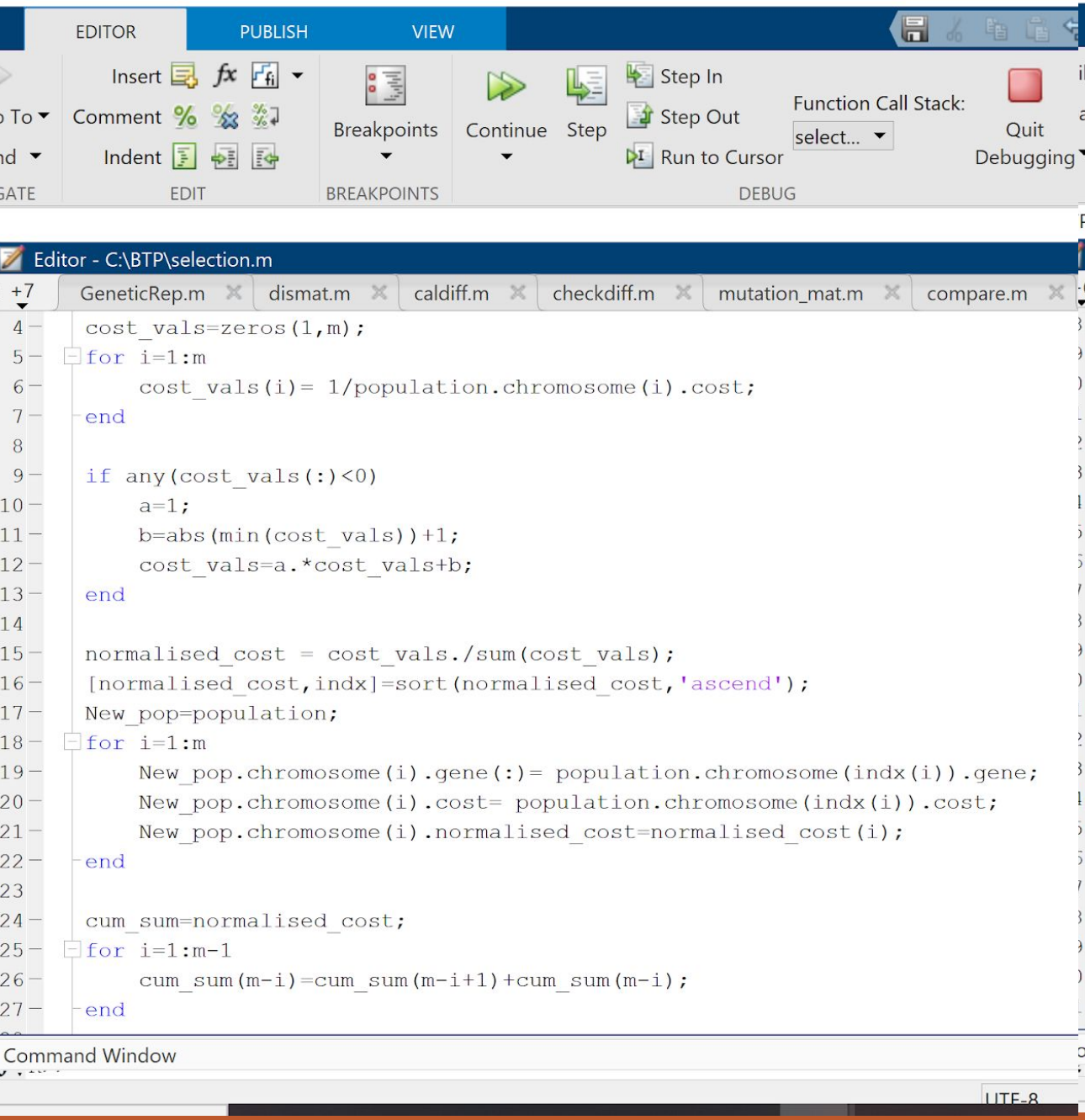
- Two parents are selected, and then mating is performed to produce children
- Individuals are selected based on their fitness values
- Roulette wheel is used in which every chromosome has a roulette wheel slot proportional to its fitness value

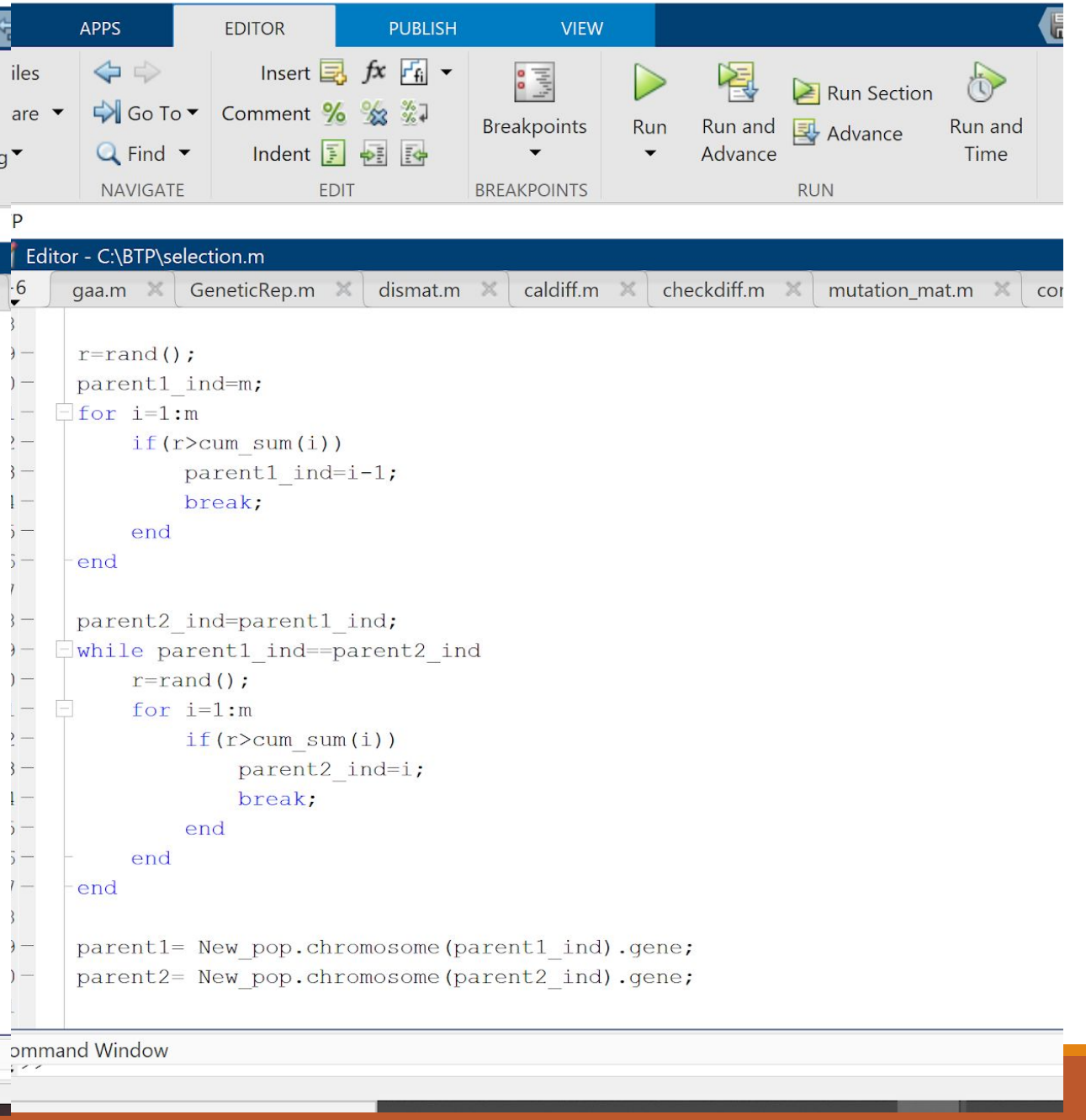


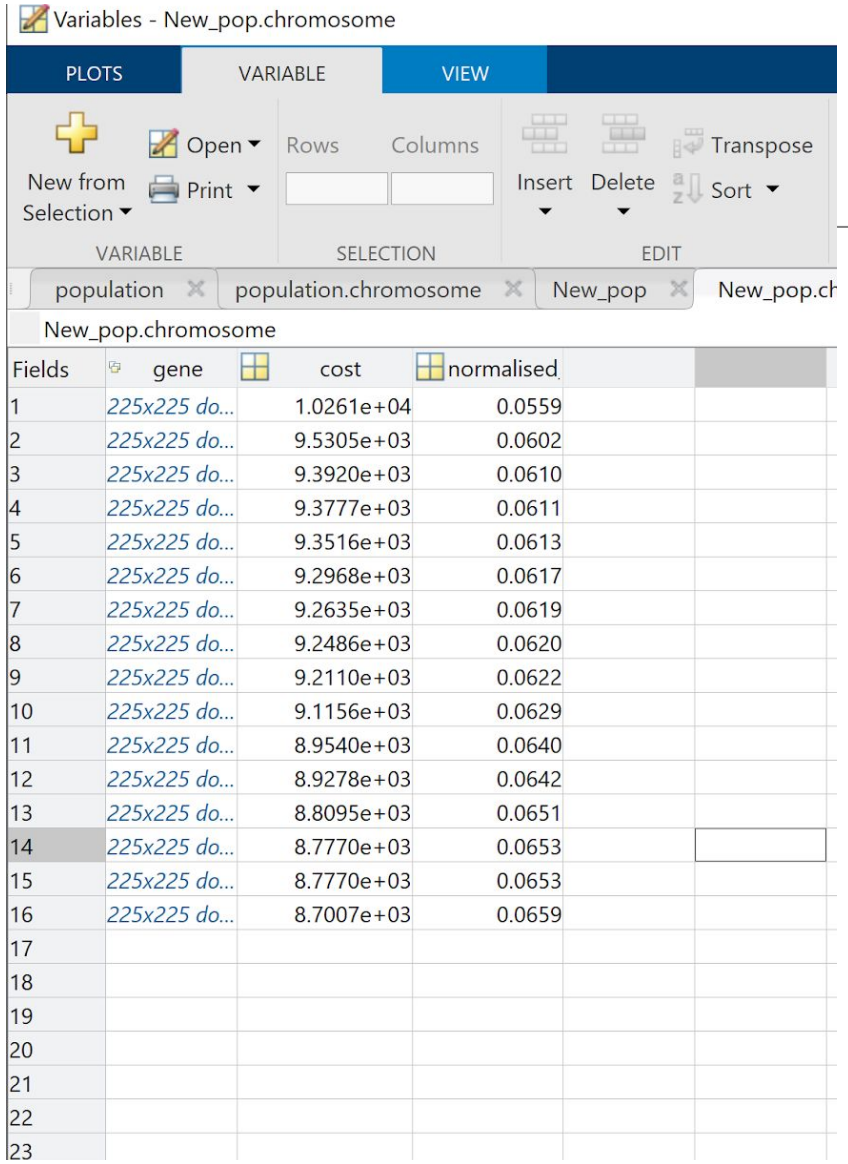
Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

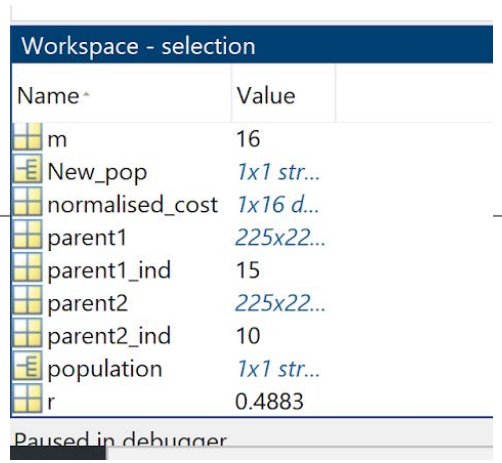
Implementation of Selection Process

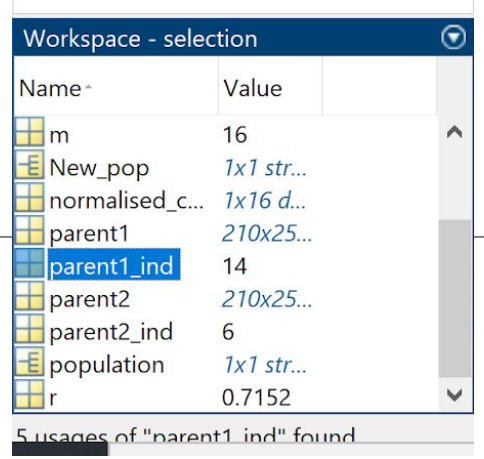
- For each individual the fitness function is calculated
- This fitness function gives fitness values, which is then normalized
- Calculate accumulated normalized fitness values
- A random number R is chosen using the rand() function in MATLAB
- The selected individual is the first one whose accumulated normalized value is greater than or equal to R


EDITOR
PUBLISH
VIEW
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Continue
Step
Step In
Step Out
Run to Cursor
Function Call Stack:
select...
Quit Debugging
DEBUG
Editor - C:\BTP\selection.m
+7
GeneticRep.m
dismat.m
caldiff.m
checkdiff.m
mutation_mat.m
compare.m
cost_vals=zeros(1,m);
for i=1:m
cost_vals(i)= 1/population.chromosome(i).cost;
end
if any(cost_vals(:)<0)
a=1;
b=abs(min(cost_vals))+1;
cost_vals=a.*cost_vals+b;
end
normalised_cost = cost_vals./sum(cost_vals);
[normalised_cost,indx]=sort(normalised_cost,'ascend');
New_pop=population;
for i=1:m
New_pop.chromosome(i).gene(:)= population.chromosome(indx(i)).gene;
New_pop.chromosome(i).cost= population.chromosome(indx(i)).cost;
New_pop.chromosome(i).normalised_cost=normalised_cost(i);
end
cum_sum=normalised_cost;
for i=1:m-1
cum_sum(m-i)=cum_sum(m-i+1)+cum_sum(m-i);
end
Command Window
UTF-8


APPS
EDITOR
PUBLISH
VIEW
Insert
Comment
Indent
Go To
Find
NAVIGATE
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
Editor - C:\BTP\selection.m
-6
gaa.m
GeneticRep.m
dismat.m
caldiff.m
checkdiff.m
mutation_mat.m
r=rand();
parent1_ind=m;
for i=1:m
if(r>cum_sum(i))
parent1_ind=i-1;
break;
end
end
parent2_ind=parent1_ind;
while parent1_ind==parent2_ind
r=rand();
for i=1:m
if(r>cum_sum(i))
parent2_ind=i;
break;
end
end
end
parent1= New_pop.chromosome(parent1_ind).gene;
parent2= New_pop.chromosome(parent2_ind).gene;
Command Window







Here are the outputs of 2 runs of the selection function and we can observe that the genes which provide better fitness are favoured.

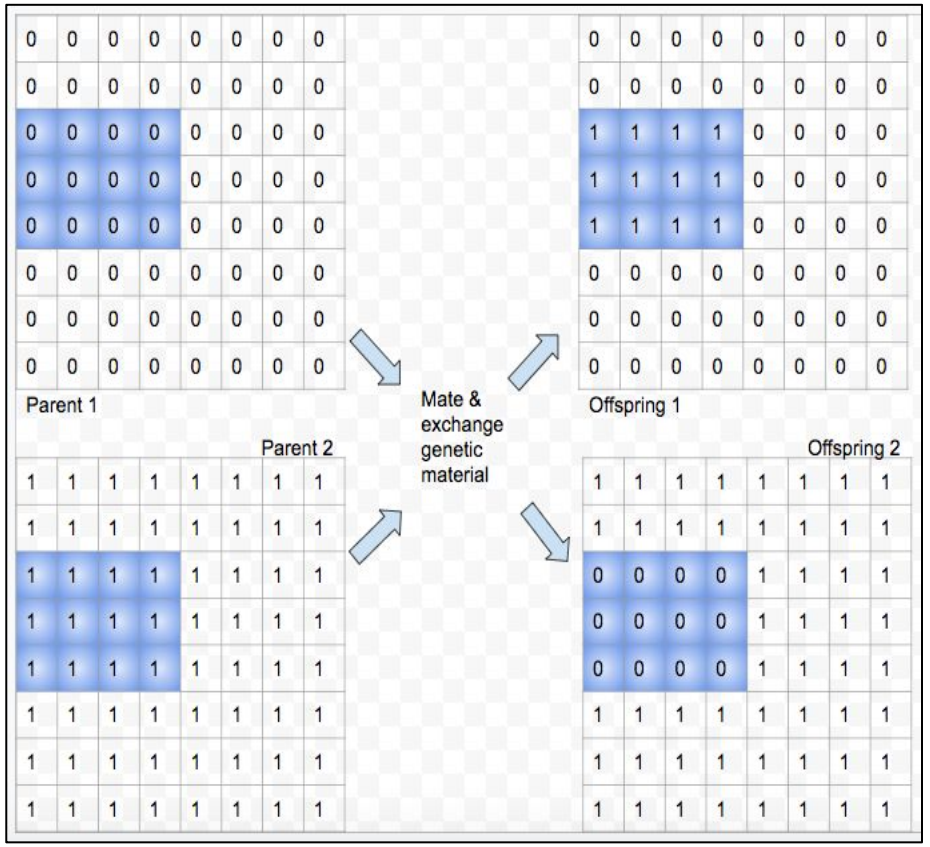
Crossover

- In crossover new child chromosomes are created from parent chromosomes
- The crossover operator makes copies of the two mates selected by the roulette wheel selection operator, and exchanges parts of chromosomes
- After crossover, genetic material in the parent chromosomes is recombined and new child chromosomes (hopefully of better fitness values) are created

1-D Crossover

Crossover

- We took a vector of length two to represent the selected points
- Used the reduced surrogate crossover
- In case a part of the gene is common between the two parents, we select crossover points again.
- We try this activity at most 5 times, the parents are supplanted with two new mates and the process is repeated



2 D Crossover

Editor - C:\Users\AIR\Desktop\crossOverBTP.m

crossOverBTP.m | crossover22.m

```
%crossOverBTP
function [child1,child2] = crossOverBTP(parent1,parent2,Pc,dis_mat)

child1=parent1;
child2=parent2;
[m,n]= size(parent1);% size of parent1 matrix
lb=2;
ub1=m-1;

x1=lb+round((ub1-lb)*rand());
y1=lb+round((u1b-lb)*rand());

ub2=n-1;
x2=lb+round((ub2-lb)*rand());
y2=lb+round((ub2-lb)*rand());

if(x1>x2)
    temp= x1;
    x1= x2;
    x2 =temp;
end
if(y1>y2)
    temp= y1;
    y1 = y2;
    y2= temp;
end

iter=5; % after 5 unsuccessful trails, the parents are replaced with two new mates and the process is repeated
while iter>=0
    for i=1:m
        for j=1:n
            if(i>=x1 && i<=x2 && j>=y1 && j<=y2)
                child1(i,j)=parent2(i,j);
                child2(i,j)=parent1(i,j);
```

Editor - C:\Users\AIR\Desktop\crossOverBTP.m

crossOverBTP.m | crossover22.m

```
                child2(i,j)=parent1(i,j);
            else
                child1(i,j)=parent1(i,j);
                child2(i,j)=parent2(i,j);
            end
        end
    end
    for i= x1:x2
        for j=y1:y2
            if parent1(i,j)~=parent2(i,j)
                break;
            else
                continue;
            end;
        end
    end
    iter= iter-1;
    x1=lb+round((ub1-lb)*rand());
    y1=lb+round((u1b-lb)*rand());

    ub2=n-1;
    x2=lb+round((ub2-lb)*rand());
    y2=lb+round((ub2-lb)*rand());

    if(x1>x2)
        temp= x1;
        x1= x2;
        x2 =temp;
    end
    if(y1>y2)
        temp= y1;
        y1 = y2;
        y2= temp;
    end

end;
    c1 = costfunction(image,parent1,dis_mat);
    c2 = costfunction(image,parent2,dis_mat);
    c3 = costfunction(image,child1,dis_mat);
    c4 = costfunction(image,child2,dis_mat);
    r=rand();
    if (c1<c3 && c2<c4 && r<Pc)
        child1=parent1;
        child2=parent2;
    end
end
```

Child 1

Child 2

Outputs of crossover function(child1, child2) if parent1 is black image & parent2 is white image

Mutation

- Mutation is used for genetic diversity in a genetic algorithm
- During crossover it is possible that some traits that may be beneficial for evolution be lost or sometimes during selection some special features of the population are not carried forward to the next generations
- In the context of optimization, mutation prevents a genetic algorithm from being stuck in an undesired local optimum

Mutation

- There is a parameter called probability of mutation.
- It is passed to the genetic algorithm and is chosen to be small during the initial iterations and is increased during the latter part of the algorithm.
- We choose a site of mutation based on the probability of mutation and then consider a 3X3 matrix around it to be mutated.
- This matrix is matched to a predefined set of matrices, once a match is found we consider all the matrices it can be converted into using only one alteration and all these possibilities are assessed and assigned a probability.
- These matrices are selected using Roulette Wheel and then the selected site is altered.

Favorability Factors of an Edge Structure

- Structures that result is a straight line are provided with a higher probability
- Structures that provide curvature of 45 degrees are considered more suitable than those with more than 45 degrees
- Resulting structures that are valid and thin are more favored than invalid local edges.A certain non-zero probability is given to a mutation that would result in the local edge to be a 3 x 3 window of all zeros
- A random mutation in a 3 x 3 window is also assigned a certain small probability.

Example of Mutation

Site Matrix		Option 1	Option 2
0 1 1 0 1 0 1 0 0	Mutated to:	0 0 1 0 1 0 1 0 0	0 1 0 0 1 0 1 0 0
0 0 1 1 1 0 0 0 0	Mutated to:	0 0 0 1 1 1 0 0 0	0 1 1 1 0 0 0 0 0

Editor - C:\BTP\mutation_mat.m*

+7 GeneticRep.m | dismat.m | caldiff.m | chec

```
76
77 -  mat13=[[1,1,0];[1,1,0];[0,0,0]];
78 -  mat131=[[1,0,0];[0,1,0];[0,0,0]] ;
79 -  mat132=[[1,1,0];[0,0,0];[0,0,0]] ;
80 -  mat133=[[0,1,0];[0,1,0];[0,0,0]] ;
81 -  mat134=[[0,1,0];[1,0,0];[0,0,0]];
82 -  p13=[0.4,0.1,0.25,0.25];
83
84 -  mat14=  [[1,0,0];[0,1,0];[0,0,0]];
85 -  mat141=[[1,0,0];[0,1,0];[0,0,1]] ;
86 -  mat142=[[1,0,0];[0,1,1];[0,0,0]] ;
87 -  mat143=[[1,0,0];[0,1,0];[0,1,0]] ;
88 -  mat144=[[1,0,0];[0,0,0];[0,0,0]];
89 -  p14=[0.4,0.1,0.4,0.1];
90
91 -  mat15=  [[0,0,0];[0,1,0];[0,1,0]];
92 -  mat151=[[0,1,0];[0,1,0];[0,1,0]] ;
93 -  mat152=[[0,1,0];[0,1,0];[1,0,0]] ;
94 -  mat153=[[0,1,0];[0,1,0];[0,0,1]] ;
95 -  mat154=[[0,1,0];[0,0,0];[0,0,0]];
96 -  p15=[0.4,0.25,0.25,0.1];
97
98 -  mat16=  [[0,1,0];[0,1,0];[0,1,0]];
99 -  mat161=[[0,1,0];[1,0,0];[0,1,0]] ;
```

Command Window

itor - C:\BTP\checkdiff.m

GeneticRep.m | dismat.m | caldiff.m | checkdiff.m | mutatio

```
        end
    elseif(compare(img,x,y,mat13))
          r=rand();
        matn=1;
        sum=0;
        for i=1:size(p13,2)
            sum=sum+p13(i);
            if(sum>=r)
                matn=i; break;
            end
        end
        if(matn==1)
            img=convert_gene(img,x,y,mat13_1);
        end
        if(matn==2)
            img=convert_gene(img,x,y,mat13_2);
        end
        if(matn==3)
            img=convert_gene(img,x,y,mat13_3);
        end
        if(matn==4)
            img=convert_gene(img,x,y,mat13_4);
        end
    elseif(compare(img,x,y,mat14))
```

nand Window

Command Window

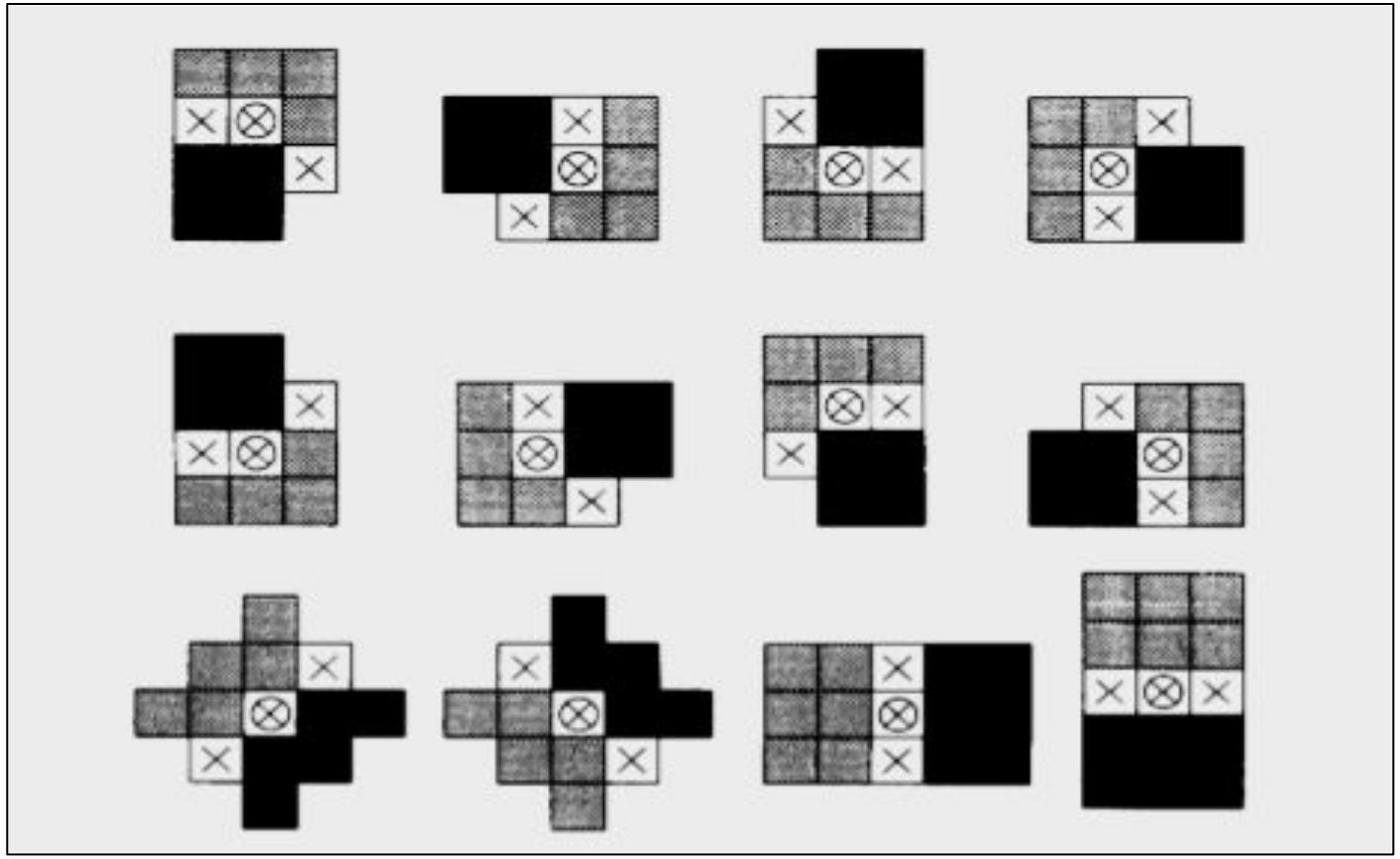
```
img =

     0     0     0
     0     1     0
     0     1     0


img =

     0     1     0
     0     1     0
     0     1     0
```

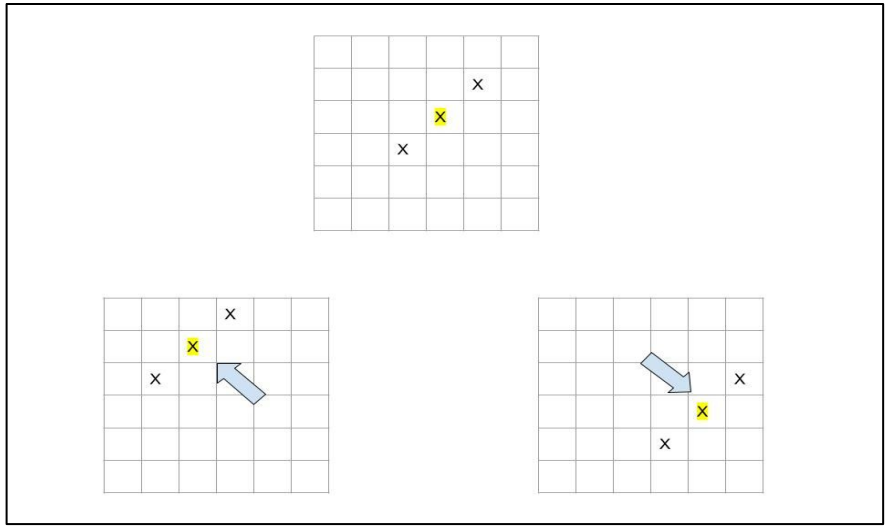
Command Window

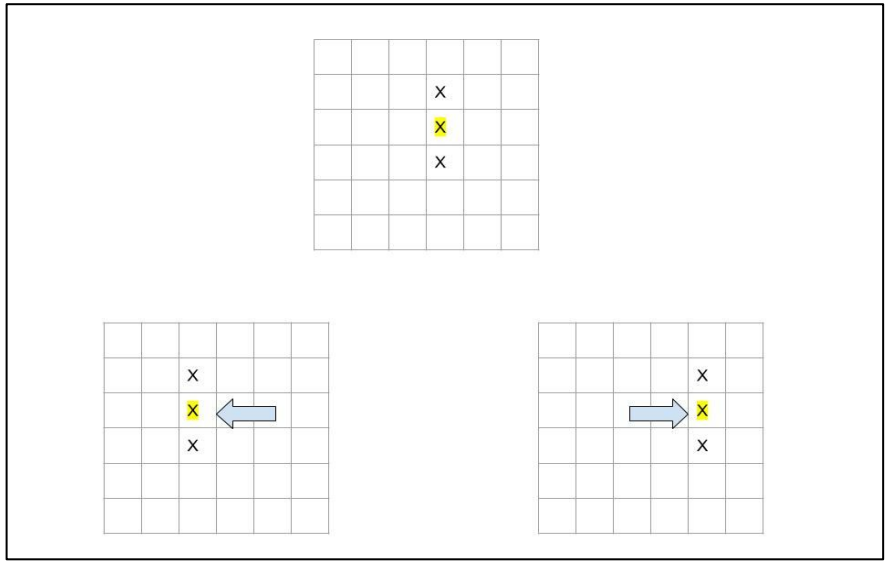
```
img =

     1     1     0
     1     1     0
     0     0     0


img =

     1     0     0
     0     1     0
     0     0     0

fx >>
```

Dissimilarity Function

- This function is used to separate out the regions we need to work with by helping us eliminate regions with dissimilarity below a certain threshold so that we can make our algorithm faster and more optimized.
- We have a basis set that contains all the possible valid edge structures that are shown in Fig 4.1. The pixel site l is chosen as the center pixel for each edge structure from the basis set. We then calculate the difference between the average gray level of region R1 and region R2 that is the value of f(R 1, R2) for each fitted edge structure. The edge structure that results in the maximum value of f(Rl, R2) is deemed to be the best fitting structure for the given site l.







Set of all possible edge structures with region division and Non Maxima Suppression

Different Cases in Dissimilarity Function

- Next, we perform non-maxima suppression by shifting the location of the best fitting edge structure in a direction determined by the matched edge structures.
- If no larger value of f(R1, R2) results from shifting the best fitting edge structure, we set d =f(R1,R2)/3 where f(R1, R2) is computed from the best fitting structure for edge. We increment the value of each of the pixels D(1), D(11) and D(12) by d.
- If the shifting results in a larger value of f(R1, R2) than the original site l then the pixel values D(1), D(11) and D(12) are left unaltered.

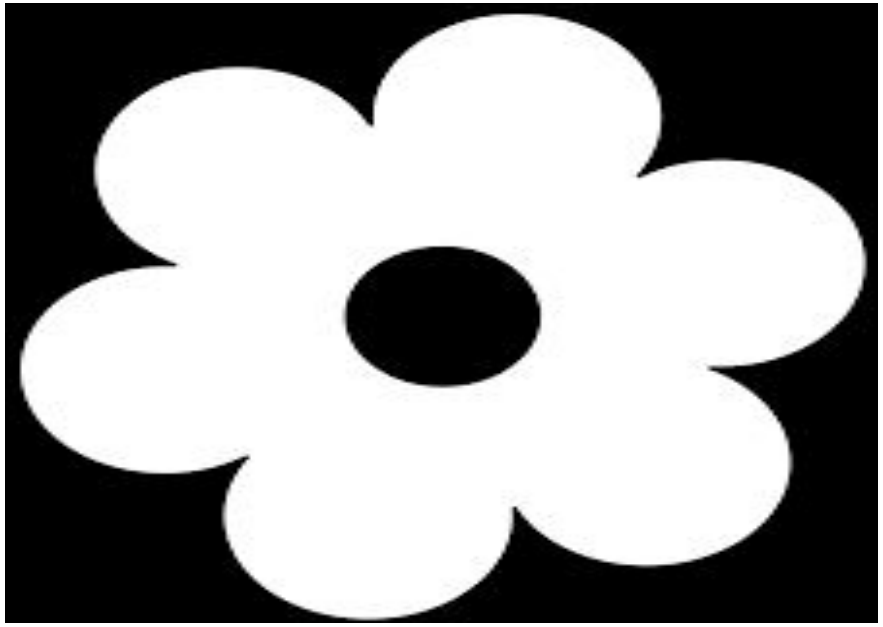
Editor - C:\BTP\caldiff.m

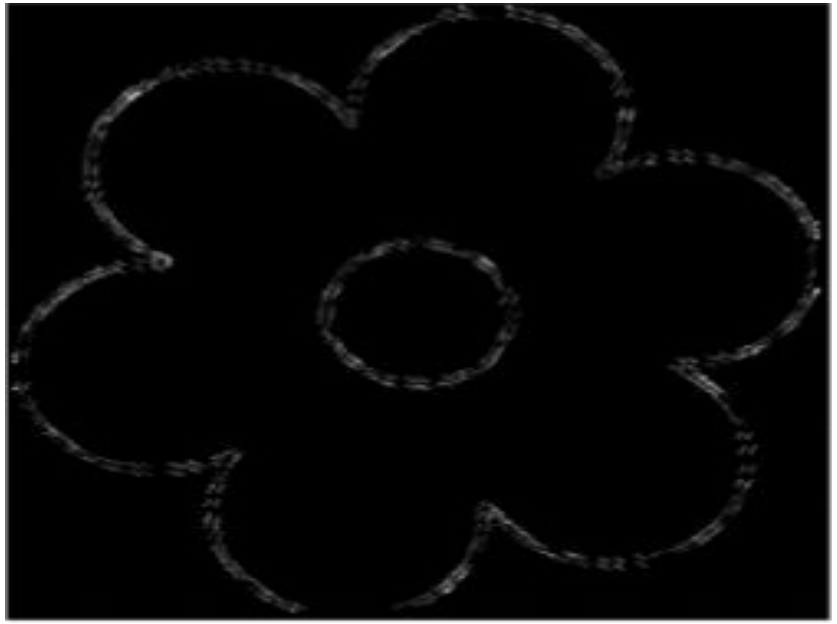
+7 | GeneticRep.m | dismat.m | caldiff.m | checkdiff.m | mutation_mat.m | compare.m | checkcomp.m | convert_gene.m

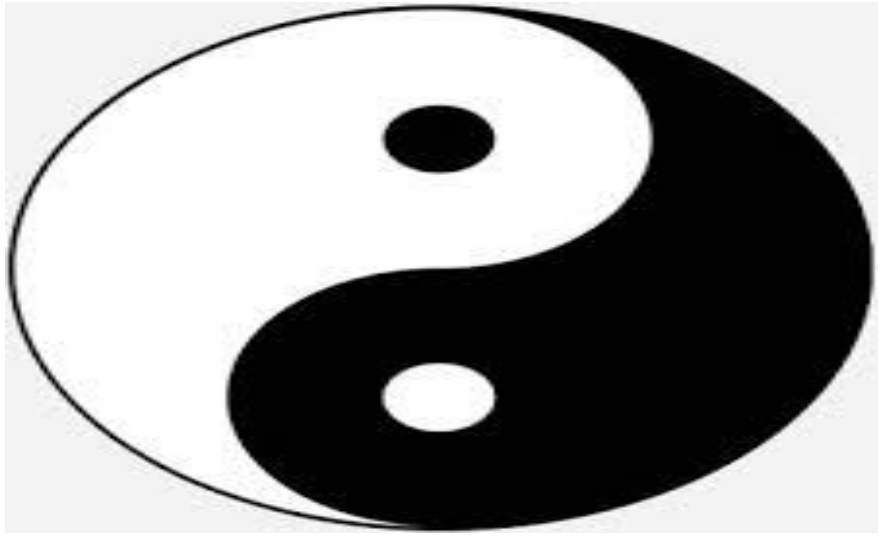
```
%cal
function[answer]= caldiff(img,x,y)

R1x= [[-1,-1,-1,0];[1,1,0,-1];[0,1,1,1];[-1,-1,0,1];[0,1,1,1];[-1,0,1,1];[0,-1,-1,-1];[-1,-1,0,1]];
R1y= [[-1,0,1,1];[0,1,1,1];[-1,-1,0,1];[0,-1,-1,-1];[1,1,0,-1];[-1,-1,-1,0];[-1,-1,0,1];[0,1,1,1]];
R11x= [[0,0,-1,-1,-2,0];[0,0,1,1,2,0];[-1,-1,0,0,1,1];[-2,-2,-2,-1,-1,-1]];
R11y=[[-2,-1,-1,0,0,0];[-2,-1,-1,0,0,0];[-2,-1,-2,-1,-2,-1];[-1,0,1,-1,0,1]];

R2x=[[1,1,2,2];[-1,-1,0,0];[-2,-2,-1,-1];[0,0,1,1];[-2,-2,-1,-1];[-1,-1,0,0];[1,1,2,2];[0,0,1,1]];
R2y=[[-1,0,-1,0];[-2,-1,-2,-1];[0,1,0,1];[1,2,1,2];[-1,0,-1,0];[1,2,1,2];[0,1,0,1];[-2,-1,-2,-1]];
R22x=[[0,0,1,1,2,0];[-2,-1,-1,0,0,0];[-1,-1,0,0,1,1];[1,1,1,2,2,2]];
R22y=[[1,2,0,1,0,0];[0,0,1,1,2,0];[1,2,1,2,1,2];[-1,0,1,-1,0,1]];

        diffmax=0.0;
        matn=1;
        for k=1:8
            a=0.0;
            b=0.0;
            for m=1:4
                a=a+img(x+R1x(k,m),y+R1y(k,m));
                b=b+img(x+R2x(k,m),y+R2y(k,m));

            end
             c=a-b;
```

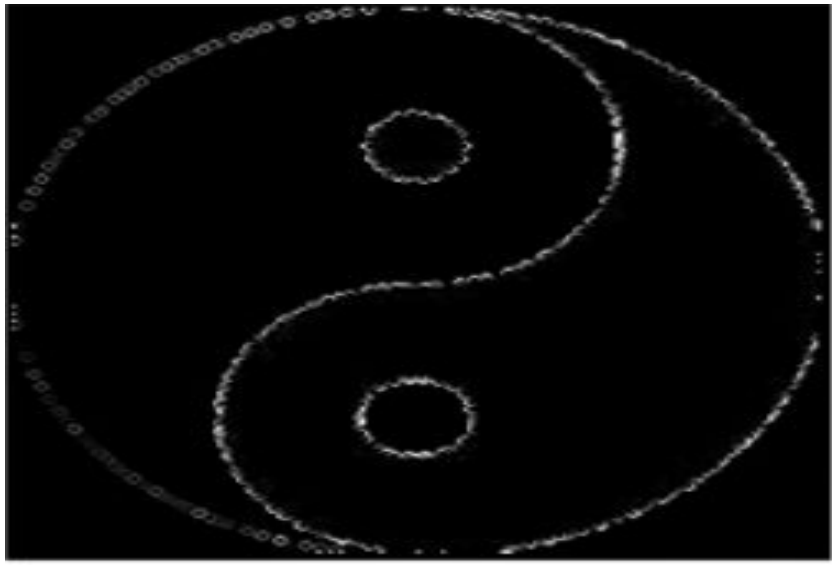
```
        diffmax=0.0;
        matn=1;
        for k=1:8
            a=0.0;
            b=0.0;
            for m=1:4
                a=a+img(x+R1x(k,m),y+R1y(k,m));
                b=b+img(x+R2x(k,m),y+R2y(k,m));

            end
             c=a-b;
                if(c<0)
                    c=-1.0*c;
                end
                c=(c*1.0)/4.0;
                if(c>diffmax)
                    diffmax=c;
                    matn=k;
                end
        end

        for k=1:4
```

and Window









Output of Dissimilarity Function

Cost Function

- A genetic algorithm works with a set of elements, called chromosomes. Fitness of these chromosomes is evaluated using a cost function.
- The problem of edge detection is formulated as a problem of cost minimization. We adopt the minimization of cost function used by Tan et al. [3], [14] here.
- This cost function is a linear sum of weighted cost factors. The cost factor evaluates useful traits of edges such as accuracy in localization, curvature, dissimilarity, thinness and continuity.

Cost Function

- The point cost at a pixel site of an edge configuration is defined as the following linear sum of weighted point cost factors Ci(S,l):

$$\mathbf{F(S, l) = \sum w_i C_i(S, l)}$$

where $w_i \geq 0$, $0 \leq C_i \leq 1$, and $i \in \{c, d, e, f, t\}$

- The total cost of an edge configuration is the sum of the point costs at every pixel site of the image:

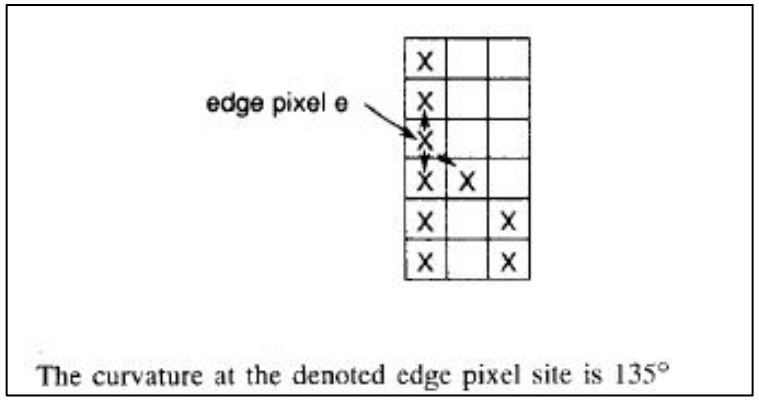
$$\mathbf{F(S) = \sum F(S, l)}$$

where $l \in L$

Cost Factors

- **Cost for Curvature :** The cost of curvature designates a cost to non-endpoint edge pixels based on a local measure of curvature

$$\mathbf{Cc(S, l)} = 0, \quad \text{if } \theta(l) = 0$$
$$= 0.5, \quad \text{if } \theta(l) = 45$$
$$= 1.0, \quad \text{if } \theta(l) \geq 90$$



The curvature at the denoted edge pixel site is 135°

Cost for Region Dissimilarity : This cost for region dissimilarity assigns a cost to non-edged pixels that is proportionate with the degree of dissimilarity

$$\mathbf{Cd(S, l)} = 0, \quad \text{is s(l) is an edge pixel}$$
$$= d(l), \quad \text{is s(l) is not an edge pixel}$$

Where d(l) is the dissimilarity value of pixel location l obtained from the dissimilarity function.

Cost Factors

- **Cost for Number of Edge Points:** The cost for the number of edge points allocates a unit cost to each edge pixel. this cost factor Ce(S, l) is used to suppress this tendency of an excessive number of edge pixels to be detected.

  Ce(S, l) = 1, if s(l) is an edge pixel

  = 0, if s(l) is not an edge pixel

- **Cost for Fragmentation:** This cost factor tends to locally link up or remove fragmented edges.

  Cf(S, l) = 1.0, if s(l) is an isolated endpoint edge pixel

  = 0.5, if s(l) is a nonisolated endpoint edge pixel

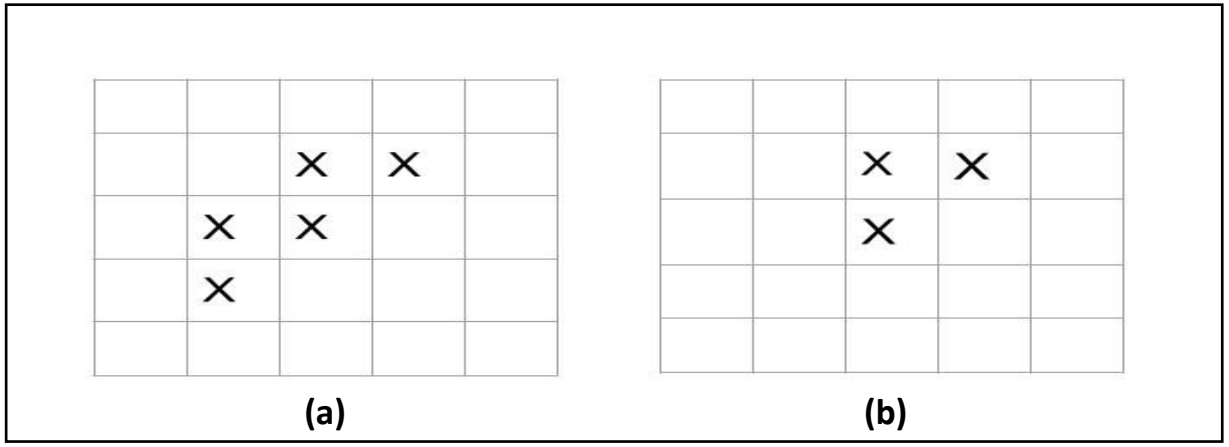
  = 0, else

Cost Factors

- **Cost for Edge Thickness:** The cost for edge thickness assigns a unit cost to thick edged pixels.

Ct(S, l) = 1, if s(l) is a thick edge pixel

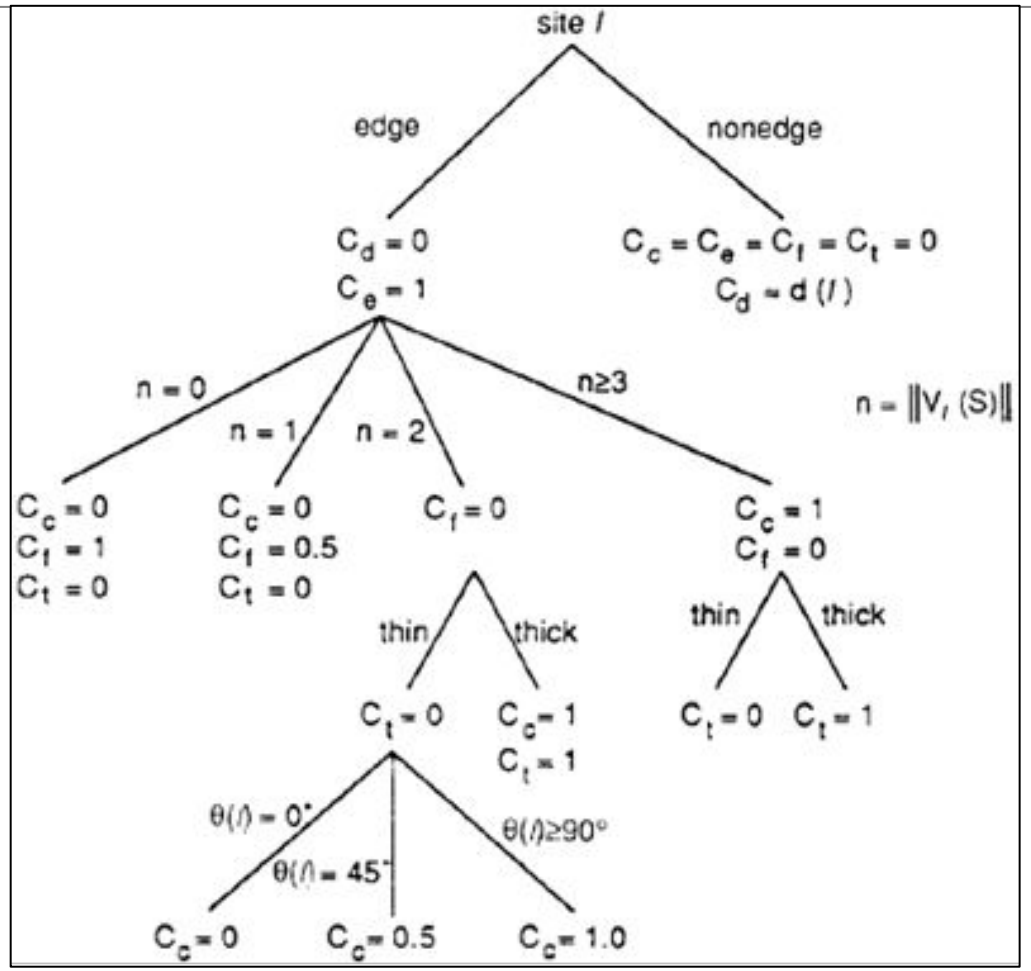
= 0 , if s(l) is not a thick edge pixel



Examples of thick edges (a) with 5 pixels (b) with 3 pixels

Decision Tree

Significant reduction in computation time can be obtained by integrating information influencing each of the different cost factors and establishing it in a form that will enable efficient computation. This is achieved by the decision tree structure as shown.



Editor - C:\BTP\costFunction1.m*

checkdiff.m | mutation_mat.m | compare.m | checkcomp

```
function [cost] = costFunction1 (img,dis_mat)

wd = 2;
wc = 0.5;
wf = 3;
wt = 6.51;
% load disMap
cost=0.0; %total cost

for i=2:size(img,1)-1
    for j=2:size(img,2)-1
        we = img(i,j)*1.0;
        if img(i,j)==0
            cd=dis_mat(i,j);
            cc=0;
            ce=0;
            cf=0;
            ct=0;
        else
            cd=0;
            ce=1.0; %Ce
            n=-1; %no of neighbour edges
            for ki=i-1:i+1
                for kj=j-1:j+1
```

Editor - C:\BTP\costFunction1.m*

checkdiff.m | mutation_mat.m | compare.m | checkcomp.m | convert_gene.m | se

```
n=-1; %no of neighbour edges
for ki=i-1:i+1
    for kj=j-1:j+1
        if img(ki,kj)==1
            n=n+1;
        end
    end
end
if n==0
    cc=0;
    ct=0;
    cf=1.0; %Cf
elseif n==1
    cc=0.0;
    ct=0.0;
    cf=0.5; %Cf
elseif n==2
    cf=0.0;
    %thick egde
    if ((img(i-1,j)==1)&&((img(i-1,j-1)==1)||(img(i-1,j+1)==1)))
        ||((img(i,j-1)==1)&&((img(i-1,j-1)==1)||(img(i+1,j-1)==1)))
        ||((img(i,j+1)==1)&&((img(i-1,j+1)==1)||(img(i+1,j+1)==1)))
        ||((img(i+1,j)==1)&&((img(i+1,j-1)==1)||(img(i+1,j+1)==1)))
```

Command Window

Editor - C:\BTP\costFunction1.m*

checkdiff.m | mutation_mat.m | compare.m | checkcomp.m

```
        cc=1.0; %Cc=1 Ct=1
    else %thin edge
        ct=0.0;
        nbs=zeros(1,8);%store neighbors clockwise
        nbs(1,1)=img(i-1,j-1);
        nbs(1,2)=img(i-1,j);
        nbs(1,3)=img(i-1,j+1);
        nbs(1,4)=img(i,j+1);
        nbs(1,5)=img(i+1,j+1);
        nbs(1,6)=img(i+1,j);
        nbs(1,7)=img(i+1,j-1);
        nbs(1,8)=img(i,j-1);
        jmp=0;
        if nbs(1,1)==0
            jmp=4;
        end
        minjmp=7;
        %finding min jump
        for k=2:8
            if nbs(1,k)==1
                if jmp>4
                    jmp=8-jmp;
                end
```

Command Window

Editor - C:\BTP\costFunction1.m

checkdiff.m | mutation_mat.m | compare.m | checkcomp.m | convert_

```
                    jmp=8-jmp;
                end
                minjmp=min(minjmp,jmp);
                jmp=0;
            end
            jmp=jmp+1;
        end
        %curvature cost
        if (minjmp==1)||(minjmp==2)
            cc=1.0; %>=90  Cc=1
        elseif minjmp==3
            cc=0.5; %=45   Cc=0.5
        else
            cc=0.0;
        end
    end
% n>=3
else
    cc=1.0; %Cc=1
    cf=0.0;
    if ((img(i-1,j)==1)&&((img(i-1,j-1)==1)||(img(i-1,j+1)
        ct=1.0; %Ct=1
    else
```

Command Window

Editor - C:\BTP\costFunction1.m

checkdiff.m | mutation_mat.m | compare.m | checkcomp.m | convert_gene.m | selection.m | costI

```
                    cc=0.5; %=45   Cc=0.5
                else
                    cc=0.0;
                end
            end
        % n>=3
        else
            cc=1.0; %Cc=1
            cf=0.0;
            if ((img(i-1,j)==1)&&((img(i-1,j-1)==1)||(img(i-1,j+1)==1)))||((img
                ct=1.0; %Ct=1
            else
                ct=0.0;
            end

        end
    end
    wd = 2.0; wc = 0.5; wf = 3.0;  wt = 6.51;
    c =double((ce*we)*1.0+(cd*wd)+(cc*wc)+(cf*wf)+(ct*wt));
    cost=cost+c;

    end
end
end
```

Input and Output of MATLAB

Input Edge Configuration :

1	1	0	0
0	1	1	0
0	0	0	0

Code Output:

```
16.0200
```

(If we calculate the cost manually, following the decision tree structure established above, we will obtain the same result)

- o An edge structure is defined only in a 3x3 neighborhood, so only pixel locations (2,2) and (2,3) are evaluated.

Cost weights are: we = 1, wd = 2, wc = 0.5, wf = 3, and wt = 6.51

Cost at each pixel location is given by: C = Ce*we + Cd*wd + Cc*wc + Cf*wf + Ct*wt

- <u>For pixel location (2,2):</u>
  - Since it is an edge (=1), Cd=0 and Ce=1.
  - Number of neighbors for this pixel is 3, i.e. (1,1), (1,2) and (2,3). So, Cc=1 and Cf=0.
  - Now, there exists multiple links between neighboring pixels, hence it is a thick edge, and Ct=1.
  - Total cost for pixel (2,2) = Ce*we + Cd*wd + Cc*wc + Cf*wf + Ct*w
  - = (1*1)+(0*2)+(1*0.5)+(0*3)+(1*6.51) = 8.01
- <u>For pixel location (2,3):</u>
  - Since it is an edge (=1), Cd=0 and Ce=1.
  - Number of neighbors for this pixel is 2, i.e. (1,2) and (2,2). So Cf=0.
  - Now, there exists multiple links between neighboring pixels, hence it is a thick edge, and Cc=1 and Ct=1.
  - Total cost for pixel (2,3) = Ce*we + Cd*wd + Cc*wc + Cf*wf + Ct*wt
  - =(1*1)+(0*2)+(1*0.5)+(0*3)+(1*6.51)= 8.01

<u>Total cost for given edge configuration</u> = Cost (2,2) + Cost (2,3) = 8.01+8.01 = 16.02 (which is equal to the output obtained)

Future Work & Conclusion

- We aim to improve our algorithm to fetch better results and a decent convergence curve
- We will explore more parameters that we can include in the cost function for it to work more efficiently and use strategies like elitism.
- We aim to write a function that calculates the efficiency and compares it to that of different algorithms

References

- Suchendra M. Bhandarkar, “ An Edge Detection Technique Using Genetic Algorithm-Based Optimization”, Pattern Recognition, Vol. 27, No. 9, pp. 1159 1180, 1994 Elsevier Science Ltd
- Saul.B Gelfand, Edward J. Delp, “ A Cost Optimization Approach to Edge Detection Using Simulated Annealing, IEEE Transaction on Pattern Analysis and Machine Intelligence, Vol 14, No. 1, January 1991
- Markus Gudmundsson, and Mansur R. Kabuka, ”Edge Detection in Medical Images Using a Genetic Algorithm”, IEEE Transactions on Medical Imaging, VOL. 17, NO. 3, JUNE 1998

THANK YOU