EDGE DETECTION IN MEDICAL IMAGES

Using Genetic Algorithm

Submitted by
Riya Ghai 2017UEC2083
Abhishek Rawat 2017UEC2116
Avishi Rathore 2017UEC2121
Prateek Yadav 2016UEC2093

Project Guide
Dr. Tarun Kumar Rawat

Table of Contents

1 Abstract

In our project we have considered the problem of edge detection as a problem of cost minimization. The edge configurations are seen as the two-dimensional chromosomes whose cost is inversely proportional to their fitness value. 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. These cost factors record the desirable features of edges like thinness, accuracy in localization and continuity. Edge configurations that decrease the cost function are used to find the edges. We have employed a Genetic Algorithm to optimise the population consisting of random pixels initially to a refined most optimal edge structure matrix for the given image.The genetic algorithm-based technique that we have used is effective in factors like quality of the final edge image, robustness to noise and rate of convergence.

2 Introduction

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. Local search method is used in edge detection algorithm. In this method, different edge configurations are analyzed and out of all configurations best configuration is selected. A decision tree helps in calculating the cost factors. Genetic algorithms using this cost function have been shown to give better results.

The genetic algorithm works on the principle of natural selection, it traverses all possible solutions generated using the fittest parents and propagates desirable features to the future generations and also incorporates a mechanism to introduce new characteristics so as to maintain uniqueness and a natural sense of randomness. Because we have such a huge search space this algorithm is robust from getting stuck in a local optimum.

3 Implementation and Work Done

Genetic Algorithm:

1. Random population is created in which each chromosome pixel is assigned value 0 or 1 and this search space is refined using a threshold defined by the dissimilarity function. After this we iterate through each candidate and evaluate its cost using the cost function.
2. The candidates are passed to the selection function which uses Roulette Wheel selection algorithm to select 2 parents that will act as a genetic base for the next generations.(in our project the candidate with the least cost is the fittest)
3. Parents are passed to the crossover function to generate two offspring and these offspring are passed to the mutation function where they are randomly mutated with the probability of mutation to make the population more rich in characteristics.
4. We pass each child created in the above step through the cost function and make it a part of the new population.
5. New children are created and evaluated until the size of this new population is not equal to that of the old one, and when this happens we replace the pre-existing candidates by the new ones for the next GA iteration.
6. If the number of iterations is not finished and the most optimum cost is above a certain value, repeat step 2-5, if any of these conditions are met we return the best suited chromosome encountered in all of the iterations.

The main operations in the Genetic Algorithm:

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

1. Selection

Selection is the first major step in Genetic Algorithm. Two parents are selected, and then mating is performed to produce children. Parent selection is very important in a Genetic Algorithm as fitter parents produce fitter off springs. We select individuals based on their fitness values.

We have used the selection operator that selects chromosomes using the Roulette Wheel operator, in this operator the probability of selection is directly proportional to their fitness. The individuals that are more fit will have a larger share in the area on the wheel.

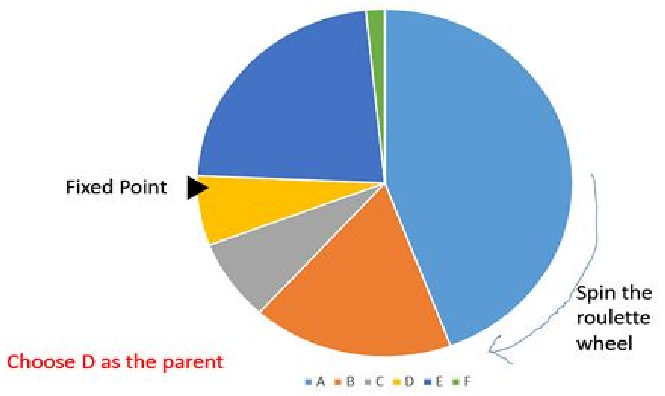
Selection process is implemented as follows:

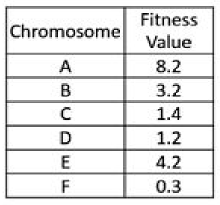
1. For each individual the calculation of the fitness function is performed. This fitness function gives fitness values, which is then normalized. Normalization is the ratio of fitness value of an individual and the sum of all fitness values so that sum of all fitness values is equal to one.

2. Accumulated normalized fitness values are calculated. It is the sum of self and all the ones that come before it. Therefore it is evident that the accumulated value for the last element will be 1.

3. A random number R is chosen using the rand() function in MATLAB which is between zero and one.

4. The individual who is selected is the one who has accumulated normalized value >=R.





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

Fig. 1.1

2. Crossover

Crossover is a major step in a genetic algorithm in which new chromosomes of a child are created from the parents at the time of reproduction. In crossover, copies of the two parents (selected using the roulette wheel) are created. Then the sites are chosen randomly or according to some predefined criteria, and the parts of chromosomes are exchanged (Fig. 2.1). After crossover, recombination of genetic material in the parent chromosomes takes place and that results in production of new child chromosomes (hopefully of better fitness values). The crossover operation is a crucial step in genetic algorithms because it helps to look at all possible chromosomes in search space.

The crossover function used in our project is similar to the basic crossover function in genetic algorithms in which we select two points randomly, and the genetic material between these two points is exchanged as shown in Fig 2.1. What we did differently is the use of a vector of length two instead of one to represent the selected points. The major types of crossovers are single point, double points, uniform crossover, shuffle crossover and crossover with reduced surrogate. We have used the reduced surrogate crossover because it evicts the redundant crossover operations. Crossover operations in crossover function are unwanted if parents have identical genes. 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, then parents are supplanted with two new mates and the process is performed again.

We created a 2D crossover function which takes parent 1, parent 2 in matrix form, probability of crossover and dissimilarity matrix as the input and returns two children in matrix form after performing crossover. We also compare the fitness values of parent and child and return the one which has better fitness value.

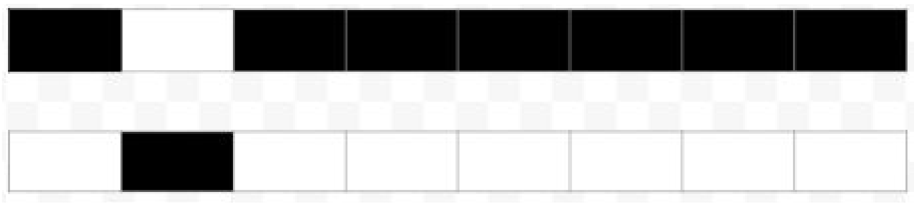


Fig 2.1 : 1D crossover

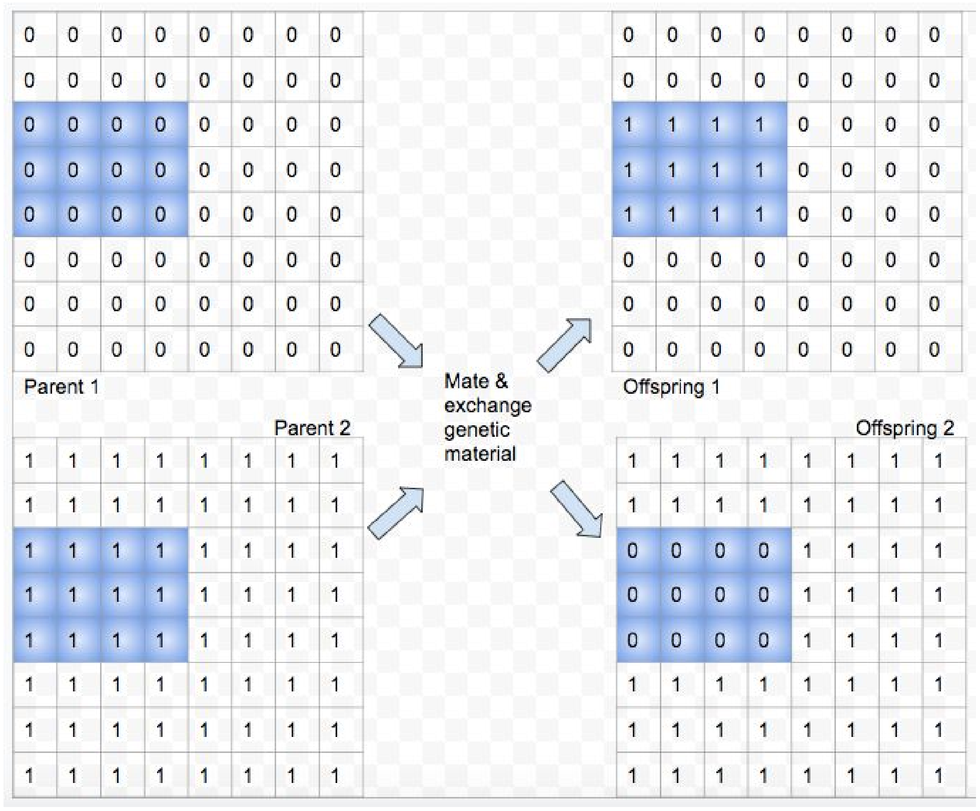


Fig 2.2 : 2D crossover

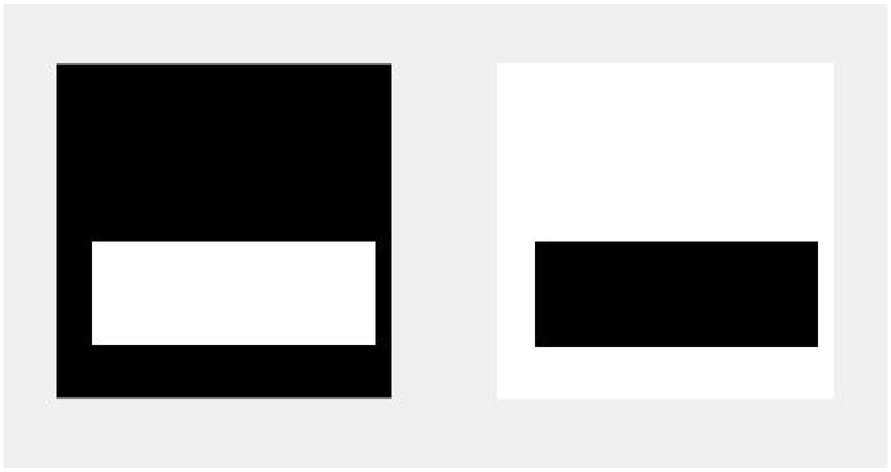


Fig 2.3 : Outputs of crossover function(child1, child2) if parent1 is black image & parent2 is white image

3. 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, there is no provision to bring them back in crossover and selection alone so it is very important to have a mechanism that can incorporate some randomness along with adding features that are more optimized without relying on other functions.

Mutation does not allow our algorithm to get stuck in a population with undesired features with no way out from them and hence makes the algorithm more optimum.

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. In our project 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. The matrices that are worse than the original matrix is given a probability zero and the ones better for cost are given a higher probability. These matrices are selected using Roulette Wheel and then the selected site is altered.

These selected matrices decide the kind of edge structures we will have in our image.

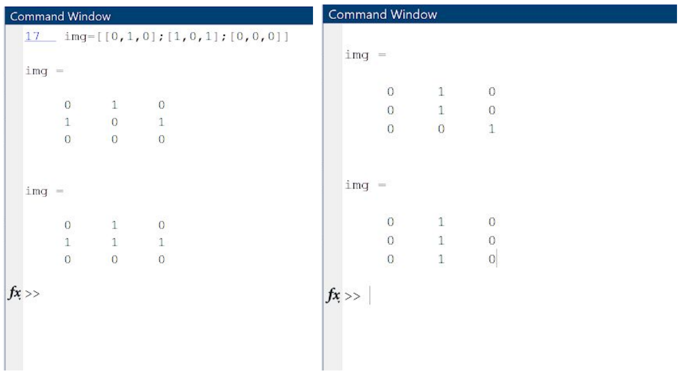
There are many factors that decide the favorability of one edge structure over another:

1. Structures that result is a straight line are provided with a higher probability.
2. Structures that provide curvature of 45 degrees are considered more suitable than those with more than 45 degrees.
3. Resulting structures that are valid and thin are more favored than invalid local edges.
4. 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..
5. A random mutation in a 3 x 3 window is also assigned a certain small probability.

Examples 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

Table 1



```
17   img=[[0,1,0];[1,0,1];[0,0,0]]

img =

     0     1     0
     1     0     1
     0     0     0


img =

     0     1     0
     1     1     1
     0     0     0

fx >>
```

```
img =

     0     1     0
     0     1     0
     0     0     1


img =

     0     1     0
     0     1     0
     0     1     0

fx >>
```

Fig 3.1: outputs of Mutation operator

Here in case 1, option 1 is more favorable as compared to option 2 and hence it is provided a greater probability to be selected as compared to option 2. In case 2, option 1 is more favorable as compared to option 2 and hence it is provided a greater probability to be selected as compared to option 2. There are 21 matrices in the set which are used to match with the current site.

The list of these matrices and possible switches with their respective probabilities is attached as Annexure A.

4. 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.

The enhanced image D(1) which is called a dissimilarity matrix . Procedure used to obtain it is :

(1) An all zero matrix D(1) is created.

(2) We traverse and at pixel site l, we perform the following steps:

(a) We have a basis set that contains all the possible valid edge structures that are shown in Fig 4.1. In the basis set the center pixel is taken to be the site 1. We 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 from the basis set that results in the maximum value of f(Rl, R2) is selected to be the best fitting structure for the given site l. Note that we have only considered thin edges with only 3 pixels per edge. For the best fitting structure, as shown in the figure the crosses are the sites of the 3 edge pixels and they are denoted as l, 11 and 12 (Fig. 4.1).

(b) Non-maxima suppression is employed to improve the quality of this function and this is done by shifting the matrix of best fitting edge structure in certain directions and evaluating their results of dissimilarity. For horizontal, vertical, and diagonal edge structures, the matched matrix is moved to the direction perpendicular to itself, for eg. a vertical matrix will move a pixel left and a pixel right. For matched edge structures that do not belong to the above category, they are shifted one pixel in each of the 4 directions and then evaluated. The evaluated values are stored in f(R1,R2).

We consider one of the following cases:

(i)If the values of f(R1,R2) that are obtained from the shifts are not larger than the ones of the original site then we simply increment each pixel site that is 1,11,12 at by the value f(R1,R2).

(ii) If any of the values of f(R1,R2) that are obtained from the shifts are larger than the ones of the original site then we simply leave each pixel site that is 1,11,12 unaltered.

(3) We make sure that the values of D(1), D(11) and D(12) do not exceed 1 by truncating them.

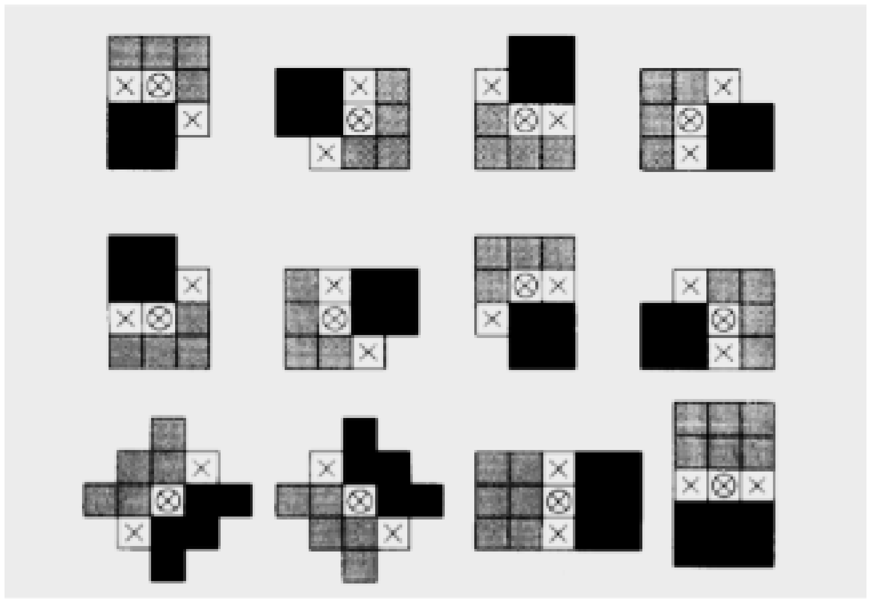


Fig 4.1 : possible edge structures and division of surrounding regions

Here in the Fig 4.1 the center encircled pixel is the site 1 in concern and the crosses besides it are sites 11 and 12. The lighter greyish region that surrounds is region R1 and the black region pixels that surrounds site 1 is region R2.

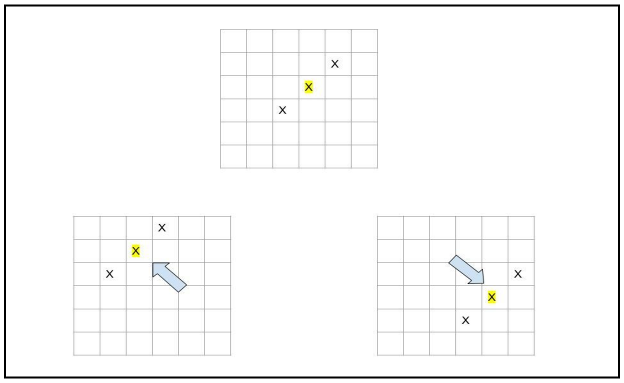


Fig 4.2 : Non Maximum Suppression for diagonal edge

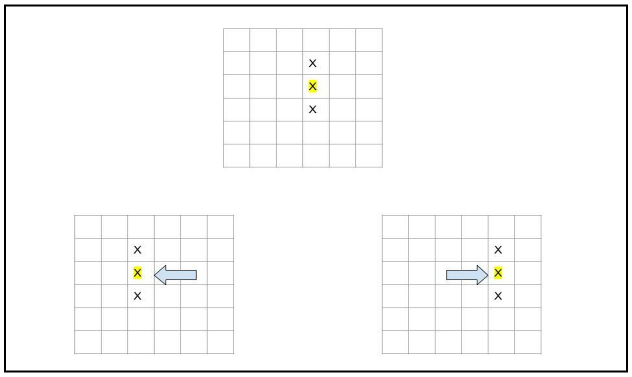


Fig 4.3 : Non Maximum Suppression for vertical edge

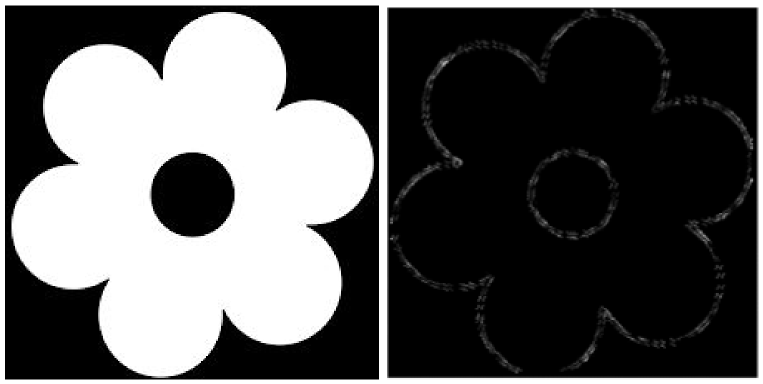


Fig 4.4: Sample Outputs of dissimilarity function

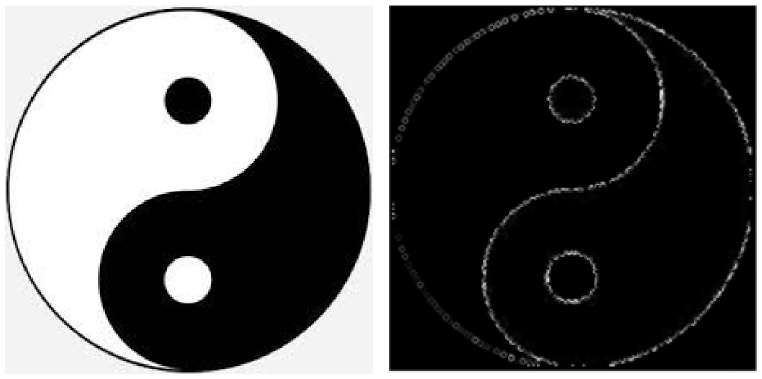


Fig 4.5: Sample Outputs of dissimilarity function

5. Cost Function

The prime idea behind genetic algorithms is to evolve the best possible optimal solution from a population of random elements. A genetic algorithm works with a set of elements, called chromosomes. Fitness of these chromosomes is evaluated using a cost function. The cost function is adapted to represent the problem we have that we want to solve. The function constitutes correlation among the different parameters which we want to optimize.

The edge detection problem is formulated as a cost minimization problem. The minimization of cost function used by Tan et al. [3], [14] is adopted here. This cost function only takes into consideration valid edge structures, which results in only sharp and thin enough edges and connected edge points, with a high enough contrast perpendicular to the contour direction to be allowed.

The quality of edge configurations is evaluated by the cost function. This cost function can be established as a linear sum of weighted cost factors. The cost factor evaluates useful traits of edges such as accuracy in localization, curvature, dissimilarity, continuity and thinness. Edges are then detected by using this cost function, by finding the edge configuration that minimizes the cost function.

We define an edge structure within a 3 X 3 neighborhood Wij(S) around a single central pixel I = s(i, j), Here, the set of all possible edge configurations in an image I is denoted by S. An edge structure is considered as a valid edge structure, if: (1) If the 8-neighborhood Vij (S) of point point s(i, j) and forms a valid edge structure within the window Wij (S) and point s(i, j) is an edge point; (2) at least 3 connected points are present in edge structure, and (3) it only contains thin edges.

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 $i \in \{c, d, e, f, t\}$

Then, the sum of the point costs at every pixel site of the image is the total cost of an edge configuration is : $\mathbf{F(S) = \sum F(S, l)}$

where $l \in L$

Using this cost function, the total cost is obtained for an edge configuration which is then used for evaluating the quality of edge. This is the cost that we want to minimize. Here, Ci represents the five cost factors:, the curvature cost (Cc), the dissimilarity cost (Cd), the cost for thick edges (Ct), the edge pixel cost (Ce), and the fragmentation cost (Cf).

The wi are the respective cost weights wc, wd, wt, we, wf. The shape of the optimised edges is controlled by varying these cost weights. Thin edges are guaranteed in an optimal configuration, if the following equation is satisfied: $wt > 2wf + wc + wd + we$

The corresponding cost weights we use are: wd = 2, wf = 3, wc = 0.5, wt = 6.51, and we = 1.

Cost Factors

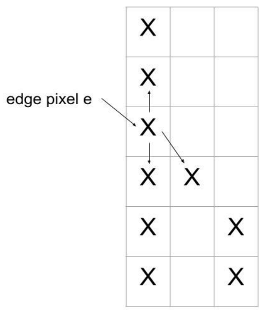
Cost for Curvature:

The cost of curvature designates a cost to non-endpoint edge pixels based on the local curvature measure. This cost factor tends to remove or smooth out curvy edges. For defining the cost of curvature, consider a non-endpoint edge pixel s(l). Then s(l) is the connection point of at least one pair of straight edge segments. Suppose the image lattice is uniformly spaced, then the curvature at any site can take one of four possible values, namely 0, 45, 90 or 135.

Cc(S, l) = 0, if $\theta(l) = 0$

= 0.5, if $\theta(l) = 45$

= 1.0, if $\theta(l) \geq 90$



The curvature at the denoted edge pixel 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. This cost factor tends to place edge pixels at points of region of high dissimilarity. The region dissimilarity point cost factor Cd(S,l) is given by:

Cd(S, l) = 0, is s(l) is an edge pixel

= d(l), 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 for Number of Edge Points:

The cost for the number of edge points allocates a unit cost to each edge pixel. The detection of edge pixels where f(R1, R2) is non-zero is favored by this cost factor Cd(S, I). This cost factor Ce(S, l) is used to reduce the excessive number of edge pixels that are detected. The endpoint point cost factor Ce(S,l) is given by:

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

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

Cost for Fragmentation:

The cost for fragmentation assigns a cost to endpoint edge pixels. This cost factor tends to locally link up or remove fragmented edges. It is devised to reduce the fragmentation of the resulting edge structure. We set Cf(S, l) using the following cases:

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

= 0.5, if s(l) is a non-isolated endpoint edge pixel

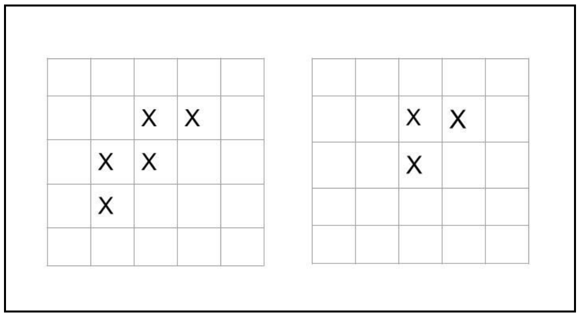
= 0,   else

Cost for Edge Thickness

The cost for edge thickness assigns a unit cost to thick edged pixels. If the presence of a pixel causes multiple links between neighboring (edge) pixels in the edge structure, it is called a thick edge pixel. This means that there exists multiple paths to reach from one pixel to another. The presence of a right-triangle structure of edge pixels indicates a thick edge. The edge thickness point cost factor Ct(S,l) is given by:

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

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

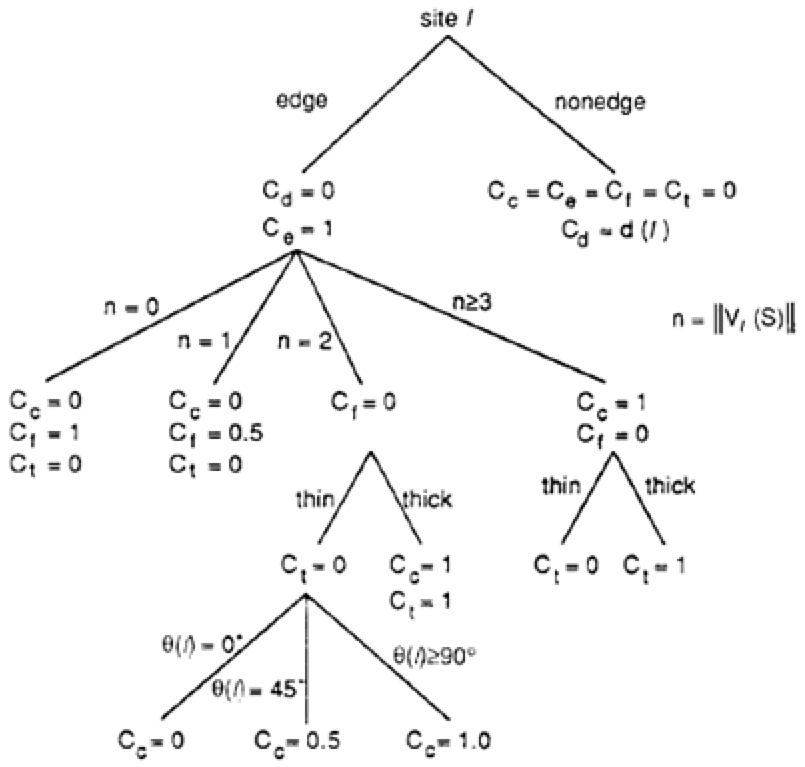


(a) (b)

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

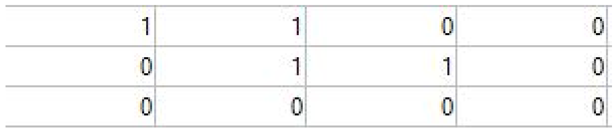
Decision Tree for Cost Function

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. Our code implements this decision tree to obtain cost at each pixel location for the given edge configuration.



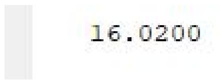
Example:- For the following edge configuration, code outputs cost = 16.02.

<u>Input edge configuration</u>:



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

<u>Code Output</u>:



```
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: wd = 2, wf = 3, wc = 0.5, wt = 6.51, and we = 1

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

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

4 Future Work

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

5 References

- Suchendra M. Bhandarkar, " An Edge Detection Technique Using Genetic Algorithm-Based Optimization", Pattern Reco(lnition, 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

6 Annexure A

The following list contains all the possible matrices the site is matched with and when matched the alternatives they get converted to.

The format is as follows:

matx is the site matched, where x is between 1 to 21.

matx_1, matx_2, mat_3 … matx_n are the n combination matrices the site can switch into.

px is an array of size n containing respective probabilities of the matrices matx_1, matx_2, mat_3 … matx_n.

mat1=[[1,0,0];[0,1,0];[0,1,0]] ;

mat1_1=[[0,1,0];[0,1,0];[0,1,0]];

mat1_2=[[1,0,0];[0,1,0];[0,0,1]];

p1=[0.6,0.4];

mat2=[[1,0,1];[0,1,0];[0,0,0]];

mat2_1=[[0,0,1];[1,1,0];[0,0,0]] ;

mat2_2=[[1,0,0];[0,1,1];[0,0,0]];

p2=[0.5,0.5];

mat3=[[0,1,0];[0,1,0];[0,0,1]];

mat3_1 = [[0,1,0];[0,1,0];[0,1,0]] ;

mat3_2=[[1,0,0];[0,1,0];[0,0,1]];

p3=[0.6,0.4];

mat4=[[1,0,0];[0,1,0];[1,0,0]] ;

mat4_1=[[1,0,0];[0,1,0];[0,1,0]] ;

```
mat4_2=[[0,1,0];[0,1,0];[1,0,0]];

p4=[0.5,0.5];

mat5=[[1,1,0];[0,1,0];[0,0,0]];

mat5_1=[[1,0,0];[0,1,0];[0,0,0]] ;

mat5_2=[[1,1,0];[0,0,0];[0,0,0]];

mat5_3=[[0,1,0];[0,1,0];[0,0,0]];

p5=[0.4,0.2,0.4];

mat6=    [[0,1,0];[1,1,0];[0,0,0]];

mat6_1=[[0,1,0];[0,1,0];[0,0,0]] ;

mat6_2=[[0,1,0];[1,0,0];[0,0,0]] ;

mat6_3=[[0,0,0];[1,1,0];[0,0,0]];

p6=[0.4,0.4,0.2];

mat7=[[0,1,1];[0,1,0];[1,0,0]];

mat7_1=[[0,0,1];[0,1,0];[1,0,0]] ;

mat7_2=[[0,1,0];[0,1,0];[1,0,0]];

p7=[0.6,0.4];

mat8=    [[1,1,0];[0,1,0];[1,0,0]];

mat8_1=[[0,1,0];[0,1,0];[1,0,0]] ;

mat8_2=[[1,0,0];[0,1,0];[1,0,0]];

p8=[0.5,0.5];

mat9=[[0,1,0];[0,1,0];[1,1,0]];

mat9_1=[[0,1,0];[0,1,0];[1,0,0]] ;
```

```
mat9_2=[[0,1,0];[0,1,0];[0,1,0]];

p9=[0.3,0.7];

mat10=    [[1,0,0];[1,1,0];[1,0,0]];

mat10_1=    [[1,0,0];[1,0,0];[1,0,0]] ;

mat10_2=    [[1,0,0];[0,1,0];[1,0,0]];

p10=[0.6,0.4];

mat11=    [[1,0,1];[0,1,0];[1,0,0]];

mat11_1=[[0,0,1];[0,1,0];[1,0,0]] ;

mat11_2=[[1,0,0];[0,1,0];[1,0,0]] ;

mat11_3=[[1,0,1];[0,1,0];[0,0,0]] ;

mat11_4=[[1,0,1];[0,0,0];[1,0,0]];

p11=[0.4,0.25,0.25,0.1];

mat12=    [[0,1,0];[0,1,0];[1,0,1]];

mat12_1=[[0,0,0];[0,1,0];[1,0,1]] ;

mat12_2=[[0,1,0];[0,1,0];[1,0,0]] ;

mat12_3=[[0,1,0];[0,1,0];[0,0,1]] ;

mat12_4=[[0,1,0];[0,0,0];[1,0,1]];

p12=[0.4,0.25,0.25,0.1];


mat13=[[1,1,0];[1,1,0];[0,0,0]];

mat13_1=[[1,0,0];[0,1,0];[0,0,0]] ;

mat13_2=[[1,1,0];[0,0,0];[0,0,0]] ;
```

```
mat13_3=[[0,1,0];[0,1,0];[0,0,0]] ;

mat13_4=[[0,1,0];[1,0,0];[0,0,0]];

p13=[0.4,0.1,0.25,0.25];


mat14=    [[1,0,0];[0,1,0];[0,0,0]];

mat14_1=[[1,0,0];[0,1,0];[0,0,1]] ;

mat14_2=[[1,0,0];[0,1,1];[0,0,0]] ;

mat14_3=[[1,0,0];[0,1,0];[0,1,0]] ;

mat14_4=[[1,0,0];[0,0,0];[0,0,0]];

p14=[0.4,0.1,0.4,0.1];


mat15=    [[0,0,0];[0,1,0];[0,1,0]];

mat15_1=[[0,1,0];[0,1,0];[0,1,0]] ;

mat15_2=[[0,1,0];[0,1,0];[1,0,0]] ;

mat15_3=[[0,1,0];[0,1,0];[0,0,1]] ;

mat15_4=[[0,1,0];[0,0,0];[0,0,0]];

p15=[0.4,0.25,0.25,0.1];


mat16=    [[0,1,0];[0,1,0];[0,1,0]];

mat16_1=[[0,1,0];[1,0,0];[0,1,0]] ;

mat16_2=[[0,1,0];[0,0,1];[0,1,0]] ;

p16=[0.5,0.5];
```

```
mat17=    [[0,1,0];[0,1,0];[0,0,1]];

mat17_1=[[0,1,0];[0,0,1];[0,0,1]];

p17=[1];

mat18=    [[0,0,0];[1,1,1];[0,0,0]];

mat18_1=    [[0,1,0];[1,0,1];[0,0,0]] ;

mat18_2=    [[0,0,0];[1,0,1];[0,1,0]];

p18=[0.5,0.5];

mat19=    [[0,1,0];[0,1,0];[1,0,0]];

mat19_1=[[0,1,0];[1,0,0];[1,0,0]];

p19=[1];

mat20= [[1,0,0];[0,1,0];[0,0,1]];

mat20_1=[[1,0,0];[0,1,0];[0,1,0]] ;

mat20_2=[[1,0,0];[0,1,1];[0,0,0]] ;

mat20_3=[[0,1,0];[0,1,0];[0,0,1]] ;

mat20_4=[[0,0,0];[1,1,0];[0,0,1]] ;

mat20_5=[[0,0,0];[1,1,1];[0,0,0]] ;

mat20_6=[[0,1,0];[0,1,0];[0,1,0]] ;

mat20_7=[[0,0,1];[0,1,0];[1,0,0]];

p20=[1.0/7,1.0/7,1.0/7,1.0/7,1.0/7,1.0/7,1.0/7];

mat21=[[0,0,0];[1,1,1];[0,0,0]];

mat21_1=[[1,0,0];[0,1,1];[0,0,0]] ;

mat21_2=[[0,0,0];[1,1,0];[0,0,1]] ;
```

```
mat21_3=[[0,0,1];[1,1,0];[0,0,0]] ;

mat21_4=[[0,0,0];[0,1,1];[1,0,0]] ;

mat21_5=[[0,0,1];[0,1,0];[1,0,0]] ;

mat21_6=[[1,0,0];[0,1,0];[0,0,1]] ;

mat21_7=[[0,1,0];[0,1,0];[0,1,0]];

p21=[1.0/7,1.0/7,1.0/7,1.0/7,1.0/7,1.0/7,1.0/7]
```