Article

A Matlab method to fit an image with polygons using genetic algorithm

WenJun Zhang
School of Life Sciences, Sun Yat-sen University, Guangzhou 510275, China; International Academy of Ecology and Environmental Sciences, Hong Kong
E-mail: zhwj@mail.sysu.edu.cn

Received 25 December 2020; *Accepted 28 January 2021*; *Published 1 June 2021*

Abstract
Genetic algorithms are a kind of algorithms to simulate genetic evolution, which belong to the subject of selforganizology. They have been used in function optimization, automatic control, robotics, and image processing, etc. In this article, a Matlab algorithm of the genetic algorithm to fit the given image with polygons is presented. Full codes are given for further use.

Keywords Matlab; genetic algorithm; image fitting; polygons.

Selforganizology
ISSN 2410-0080
URL: http://www.iaees.org/publications/journals/selforganizology/online-version.asp
RSS: http://www.iaees.org/publications/journals/selforganizology/rss.xml
E-mail: selforganizology@iaees.org
Editor-in-Chief: WenJun Zhang
Publisher: International Academy of Ecology and Environmental Sciences

1 Introduction

Genetic algorithms (GAs) are a kind of algorithms to simulate genetic evolution, which belong to the subject of selforganizology (Yadav et al., 2018, 2018, 2019; Zhang, 2013, 2014, 2016).

GAs are stochastic global search methods that simulate biological evolution (Chipperfield et al., 2014). In essence, they are evolutionary algorithms. Also, GAs are adaptive plans (Holland, 1975). In an adaptive plan, structures in a search space are selected and continuously modified by some rules according to the quality of their performance in the past (Zhang, 2016). In a GA, a set of structures is a population and each structure is a genotype/individual. A genotype/individual is the expressions of features in the structures of the problem domain. Adaptive operators (i.e., genetic operations) include selection, crossover and mutation. GAs operate on a population using the principle of survival of the fittest, in order to produce better and better approximations to the solution. Genetic operations are used in each generation to mimic natural adaptation, which leads to the evolution of individuals that are better suited to their environment than their parent individuals (Chipperfield et al., 2014). GAs differ from conventional search and optimization methods. The most significant differences include (Chipperfield et al., 2014): (1) GAs search a population in parallel; (2) GAs do not require derivative information or other auxiliary knowledge - only the objective function and corresponding fitness values will determine the direction of search; (3) GAs use probabilistic transition rules rather than deterministic ones, and (4) GAs work on an encoding of the parameter set, rather than the parameter set itself (Zhang, 2016).

So far, GAs have been used in various areas, for example, Feng and Zhang (2004) used a GA to improve

BP algorithm. Function optimization is the classic applications of GAs. For some optimization problems of non-linear, multi-model, and multi-objective functions, GAs perform better than conventional methods (Zhang, 2016). GAs have been successfully used to solve the traveling salesman problem, packing, layout optimization, graphic delineation, and other optimization problems with NP difficulty. In addition, they are used in automatic control, robotics, and image processing (Zhang, 2016, and internet resources).

In this article, a Matlab algorithm of the GA to fit the given image is presented. Full codes are given also for further use.

2 Genetic Algorithm for Image Fitting

Given a target image, use genetic algorithm to fit the image with *p* polygons of certain transparency and certain vertices. The algorithm is as follows:

(1) Transform the target image into numerical data. The size of the target image is supposed to be *n***m* pixels. The color of each pixel is recorded with three primary colors (RGB) and stored in the array.

(2) Initialization: For each individual image, randomly generate the locations and colors of *p* polygons.

(3) Iteration.

(I) Calculate the fitness of all individuals in the new generation of population and sort them from low to high;

(II) Retain the most fitted individuals directly to the next generation.

(III) For all individuals in the new generation of population, normalization is used to make their probability being selected proportional to their fitness.

(4) Crossover. Randomly select the parents (individual images) and locations (polygons) for crossover. After making crossover, the offspring individuals are obtained, and the offspring individuals are classified into the next generation.

(5) Mutation. Randomly select the locations and colors to make change.

(6) Calculate the fitness of the new generation of population. If the fitness is lower than the expected value, go to (3), and if it is higher than the expected value, then calculation stops.

3 Matlab Algorithm

The following are Matlab codes for the algorithm above.

```
clear;
close all;
[filename,pathname]=uigetfile({'*.jpg;*.tif;*.png;*.gif','Choose an image file'});
originpic=imread(fullfile(pathname,filename)); %Raw data; RGB colors of polygons
popusize=input('Input the number of individuals in the population (i.e., population size, e.g., 1000): ');
mostfittedind=input('Input the number of the most fitted individuals that need not to make mutation operations (e.g., 10): ');
accuracy=input('Input the target accuracy (e.g., 0.95): ');
generationmax=input('Input the permitted number of maximum generations (e.g., 10000): ');
probcross=input('Input the probability for crossover (e.g., 0.7): ');
locnumcross=input('Input the number of locations for crossover (e.g., 0.3): ');
probmuta=input('Input the probability for mutation (e.g., 0.001): ');
polygonnum=input('Input the number of polygons (e.g., 100): ');
polygonvert=input('Input the number of vertices of the polygon (e.g., 3): ');
polygontransp=input('Input the transparency of of polygons (e.g., 0.2): ');
figure;
imshow(originpic);
```

```
%Run genetic algorithm
%Pre-operation to calculate image size
[w,l,unuse]=size(originpic);
%Maximal fitness
maxdiff=255^3*3;
%Present numerical expression of original image
picresize=double(imresize(originpic,[256,256]));
totalamount=popusize+mostfittedind;
%Create initialized population
%Population is represented by a binary matrix of 8*(2*polygonvert+3)*polygonnum¡Átotalamount, where 8 bits of binary value represent a value of 0-255, and each polygon has polygonvert vertices (x,y) and a RGB color
%In ASCII table, 48 and 49 correspond to 0 and 1 respectivelty. For the genes (polygonnum of polygons) of each individual, the
%locations and colors of polygonnum of polygons (genes) are stochastic
si=8*(2*polygonvert+3)*polygonnum;
population=char(randi([48,49],si,totalamount));
%Initial optimal fitness is 0
optimalfitness=0;
%Initial optimal individual genes are empty
optimalgene=char(zeros(si,1));
%The matrix for fitness of every individual
fitnessarray=zeros(1,totalamount);
%Gentic algorithm
figure('Name','Optimal Gene');
for generationcounter=1:generationmax %Run the bank of individuals
%Calculate fitness
for i=1:totalamount
gene=population(:,i);
fitnessarray(i)=calFitness(picresize,gene,maxdiff,polygonnum,polygonvert,polygontransp);
if fitnessarray(i)>optimalfitness %Find the optimal solution in population
optimalfitness=fitnessarray(i);
optimalgene=gene;
end
end
if optimalfitness>=accuracy break;
%Find the expected solution and calculation ends
end
%(1) Choose several optimal offspring individuals to the next generation. Sort them from low to high
[unuse,mostfittedindex]=sort(fitnessarray);
%Choose the most fitted ones
mostfittedindex=mostfittedindex(end-mostfittedind+1:end);
%Find them with their indexes
mostfitted=population(:,mostfittedindex);
%(2) Randomly select individuals, and their probability being selected proportional to their fitness.
%Calculate the probability being selected and further obtain the cumulative probability, and compare to a random value in [0¬1]
%If the random value <= the cumulative probability of individuals and > 1, select the individual to the next generation
```

```
fitnessarray=1-fitnessarray/maxdiff; %Fitness
fitnessarray=fitnessarray/sum(fitnessarray(:)); %Normalization
accumulation=cumsum(fitnessarray); %Cumulative probability
%Obtain the column down to up (row right to left) summation matrix corresponding to every element in input matrix
roulettenumber=rand(1,popusize);
surviveindex=discretize(roulettenumber,[0,accumulation]);
newpopulation=population(:,surviveindex);
newpopulation=[newpopulation,mostfitted];
newpopulation=newpopulation(:,randperm(totalamount)); %Random disperse the orders of individuals in the new generation
%(3) Crossover. Part of two parent individuals crossover and generate new individuals
%1-single location crossover: there is one random location for crossover
%2-mutiple locations crossover: there are mutiple random locations for crossover
%3-even crossover: randomly generate a shield word to determine how the offspring individuals obtain genes from parent
%individuals
%Generate shield model
crossoverindex=rand(1,totalamount/2)<probcross;
%Shield word string (0-1 string)
crossovergeneindex=rand(si/8,totalamount/2)<locnumcross;
crossovergeneindex=repelem(crossovergeneindex,8,1);
crossovergeneindex(:,~crossoverindex)=0;
geneindexdiff=zeros(si,totalamount);
geneindexdiff(:,1:2:totalamount)=crossovergeneindex;
geneindexdiff(:,2:2:totalamount)=-crossovergeneindex;
[X,Y]=meshgrid(1:totalamount,1:si);
newgeneindex=sub2ind([si,totalamount],Y,X+geneindexdiff);
newpopulation=newpopulation(newgeneindex);
newpopulation=[newpopulation(1:si,mostfittedind+1:end),mostfitted];
%(4) Mutation
mutationindex=rand(si,totalamount)<probmuta;
population=char(uint8(xor(newpopulation>48,mutationindex))+48); %or operation
population=[population(1:si,mostfittedind+1:end),mostfitted];
disp(['Generation_',num2str(generationcounter),': ',num2str(optimalfitness)]);
pic=drawPicture(optimalgene,polygonnum,polygonvert,polygontransp);
pic=imresize(pic,[w,l]);
imshow(uint8(pic));
end
pic=drawPicture(optimalgene,polygonnum,polygonvert,polygontransp);
pic=imresize(pic,[W,L]);
figure;
imshow(pic); %Present the final fitted image

function fitness=calFitness(pic,gene,maxdiff,polygonnum,polygonvert,polygontransp)
%For every individual, calculate its fitness
[w,l,unuse]=size(pic);
%Gene expression (transform a series of data to an image)
```

```
si=8*(2*polygonvert+3)*polygonnum;
nloc=2*polygonvert*polygonnum*8;
polygonloc=gene(1:nloc);
polygoncolor=gene(nloc+1:si);
polygonloc=reshape(polygonloc,2*polygonvert*polygonnum,8);
polygoncolor=reshape(polygoncolor,3*polygonnum,8);
polygonloc=bin2dec(polygonloc)+1; %binary to decimal
polygoncolor=bin2dec(polygoncolor);
polygonloc=reshape(polygonloc,polygonnum,2*polygonvert); %For polygonnum of polygonangles, the subordinates of all
%vertices of every polygon, (x1,y1),(x2,y2),(x3,y3), ..., (x-polygonnum,y-polygonnum)
polygoncolor=reshape(polygoncolor,polygonnum,3); %For polygonnum of polygons, color (RGB) of every polygon
%Generate image
picgene=insertShape(ones(w,l,3)*255,'Filledpolygon',polygonloc,'Color',polygoncolor,'Opacity',polygontransp);
fitness=abs(picgene);
fitness=1-sum(fitness(:))/maxdiff;

function pic=drawPicture(gene,polygonnum,polygonvert,polygontransp)
si=8*(2*polygonvert+3)*polygonnum;
nloc=2*polygonvert*polygonnum*8;
triloc=gene(1:nloc);
tricolor=gene(nloc+1:si);
triloc=reshape(triloc,2*polygonvert*polygonnum,8);
tricolor=reshape(tricolor,3*polygonnum,8);
triloc=bin2dec(triloc)+1;
tricolor=bin2dec(tricolor);
triloc=reshape(triloc,polygonnum,2*polygonvert);
tricolor=reshape(tricolor,polygonnum,3);
pic=insertShape(ones(256,256,3)*255,'FilledPolygon',polygonloc,'Color',polygoncolor,'Opacity',polygontransp);
pic=uint8(pic);
```

4 Discussion

The present Matlab algorithm is the first version. Further possible versions may be updated in the supplementary files package with present article.

Acknowledgment

We are thankful to the support of Discovery and Crucial Node Analysis of Important Biological and Social Networks (2015.6-2020.6), from Yangling Institute of Modern Agricultural Standardization. Special thank is for Mrs WenZhu Tan, Sun Yat-sen University.

References

Chipperfield A, Fleming P, Pohlheim H, et al. 2014. Genetic Algorithm Toolbox User’s Guide. MathWorks, USA

Feng YJ, Zhang WJ. 2004. Network software of Genetic BP algorithm and its application in biodiversity research. Application Research of Computers, Suppl: 387-389

Goldberg DE. 1989a. Genetic algorithms and Walsh functions: Part I A gentle introduction. Complex Systems, 3: 129-152

Goldberg DE. 1989b. Genetic algorithms in Search, Optimization, and Machine Learning. Addison-Wisely, Reading, MA, USA

Holland JH. 1975. Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. University of Michigan Press, Ann Arbor, MI, USA

https://blog.csdn.net/qq_41742361/article/details/108091365

http://www.matrix67.com/blog/archives/1113

https://www.cnblogs.com/yang3wei/archive/2012/06/30/2739392.html

Wang XP, Cao LM. 2002. Generic Algorithms: Theory, Applications and Software Implementation. Xian Jiaotong University Press, Xian, China

Yadav AS, Swami A, Gupta CB, Garg A. 2017. Analysis of electronic component inventory optimization in six stages supply chain management for warehouse with ABC using genetic algorithm and PSO. Selforganizology, 4(4): 52-64

Yadav AS, Sharma P, Swami A, Pandey G. 2018. A supply chain management of chemical industry for deteriorating items with warehouse using genetic algorithm. Selforganizology, 5(1-2): 1-9

Yadav AS, Swami A, Kher G. 2019. Blood bank supply chain inventory model for blood collection sites and hospital using genetic algorithm. Selforganizology, 6(3-4): 13-23

Zhang WJ. 2013. Selforganizology: A science that deals with self-organization. Network Biology, 3(1): 1-14

Zhang WJ. 2014. Selforganizology: A more detailed description. Selforganizology, 1(1): 31-46

Zhang WJ. 2016. Selforganizology: The Science of Self-Organization. World Scientific, Singapore