International Journal of Computer Trends and Technology
ISSN: 2231 – 2803 / https://doi.org/10.14445/22312803/IJCTT-V69I7P109

Volume 69 Issue 7, 58-69, July 2021
© 2021 Seventh Sense Research Group®

Original Article

Self-Organizing Information for Approaching AI by Relative Entropy

I-HO Lee

Master, Information engineering, I-Shou University,
No.1, Sec. 1, Syuecheng Rd., Dashu District, Kaohsiung City, Taiwan, R.O.C.

Received Date: 08 June 2021
Revised Date: 11 July 2021
Accepted Date: 21 July 2021

Abstract - *In a dynamic environment, creatures can adapt proper actions to interact or deal with other objects that are required to identify the relationship of self-behaviors and outside environmental data. Furthermore, before creatures have properly acted, the creatures have experienced or learned a successful way to handle the same problems or situations. In this article, we want to demonstrate this identifying mechanism of creatures, which leads to the creatures properly interacting and learning response with the environment. Those proper interactions with environmental data can be observed by our sensory organs. Our sensory organs receive data from an environment that is a basic fundamental function to learn the knowledge or model for proper action. However, there is a lot of data, and we need to identify the relation of different data. We use information entropy to measure the happened times of data. This kind of formulation is helping us to find out the relation between different data and data of behaviors. Once we can identify the connection between our behaviors and environmental data, that connection becomes a logic for judgment and guiding our interactions more properly. Furthermore, we had coded a program to demonstrate this dynamic environment and interacting behaviors by combined particle swarm optimization (PSO) and information entropy. Moreover, our program is presenting that the learning way of a creature is constructed by the identifying mechanism to form our logic. The idea of what we propose is basic philosophy, pso, statistics, and behavioral psychology. Those fields help us to figure out the steps of our thinking and design the program.*

Keywords - *Central limit theorem, Information entropy, Max entropy, Knowledge diffusion, PSO.*

I. INTRODUCTION

The creature is continuously collecting data from the environment, then it is according to specific data of events, and it justifies its behaviors for better survival chances. And this mechanism of collecting data and behavioral data is one part of our learning knowledge. We want to express a similar learning knowledge process in that we create the identifying mechanism as a detectable interaction of learning purpose (DILP). DILP uses information entropy to analyze the dynamic environmental data, and we demonstrate a program for explaining the idea. DILP has two major considerations of entropy systems: identifying data of associated interactions and training data by maximum Information.

In identifying data of associated interactions, we are interested in a topic. How do creatures justify their behaviors for better survival chances? It must understand the relation between behavioral results and the environment. Our sensory organs collect data, and our neural system or memory records those cases which are different from the environment or unusual phenomenon, especially during our enfant period. We produce a lot of neural cells to remember the connection of environment data. Those processes are continuous for building our logic, and then our logical thinking is able to adapt the correct response to the environment. Furthermore, identifying the connections between environmental data and behavioral data is forming our logic of reasonable responses. And our behaviors are also creating unique data in any observed environment. So, we can use the relative entropy (Kullback-Leibler divergence) to identify the connections of environmental data and data of behaviors that are able to be the logical identification and learning way for adjusting behaviors.

In training data by maximum Information, we want to ensure our trained data is best for trained models. The most common problems of the best-trained model are under-fitted and overfit that is a trade-off relation. How do we ensure our trained data is best for a model? The maximum entropy is what we use for identifying the best-trained models when the received data are qualified and enough for trained models.

There are a lot of science perspectives for designing our programs, such as pso, philosophy, statistics, data mining, neural network, and behavioral psychology. The initial idea is a problem of AI " reasonable response[1]." We need to settle this issue down and provide a mathematical perspective for our designs. We focus on the behaviors of creatures and the environmental data. Our point is that the creature's behaviors interact with the environment in that it must create data. That data is detectable by our sense organs, and some specific behavior produces unique data. Our sense organs observe those different data types from the environment that we can use information entropy to identify their relations. Furthermore, those relations are similar to our logical thinking that we can accurately identify a reasonable and expected response from many possibilities. The idea of our DILP is based on data quantity or probability to find the relation of different data from physical space. Using probability to describe the reasonable response of different data can construct such logic of handling a dynamic environment that has much more meaning for living creatures.

In the philosophy views, a priori and a posteriori knowledge define how we learn knowledge. And in this article, we can only follow the posteriori of philosophy to design our theory and program. The posteriori knowledge indicates that we can only expect a result from our experiences that those experiences are constructed by "try and error." And a priori knowledge infers an unknown or pre-guess result that requires a lot of posteriori knowledge to be the base of inferring thoughts. The posteriori knowledge has a critical connection with our logical thinking and proper interactions because without the expected result or experiences, and there are many possible actions that could be total chaos of our actions or go to the error side of " try and error." On the other hand, our behaviors are gradually settled down by best survival purposes.

Statistics is the fundamental of many sciences, and our entropy information is also similar to statistics. The current applications, neural networks, and data mining are based on statistical perspectives. However, those still require a lot of human classification and collected data for training models. Normally we consider Gaussian distribution to estimate the trained degree of models by variance and bias of models.

Furthermore, in some sciences, they observed the foraging behaviors of animals, and they proposed statistical models such as levy flight[2]. Whatever the statistical data of levy flight is only based on one target or object, and those statistical data are not fully collecting all relative data or other affecting elements, which those relative data and elements could have associated with target data. In the levy flight, scientists observed albatrosses which we cannot fully understand their behaviors, but behavioral psychology observed human behaviors. Some specific human behaviors are triggered or caused by specific environmental stimulation. Behavioral physiology collected much more about the behaviors of humans, and it has much more statistical data and scientific methods for reference. Therefore, we can infer the trace of interaction between humans and the environment.

When we identify what kind of animals are smarter than we normally will think about the animal which has a social construction or hierarchy. Therefore, their learning efficiency has explicitly increased more than others without society. Society provides learning processes a lot of reference data and experience for the youth generation. For our learning, we need enough samples to identify cause and result relations. And this phenomenon has better mathematically described as what we proposed " training data by maximum
information entropy."

Particle swarm optimization (PSO) has a better explanation of those learning processes. Learning processes of pso construction have two major considerations, global searching, and local searching[3][4][5][6][7]. Global searching is a learning style from older people or parents, and the local searching is similar to self-expedition. Global searching and local searching are trade-off connections.

However, global searching and local searching are not suitable to analyze a dynamic environment for identifying their relations. Therefore, we propose an information entropy system to settle down reasonable response issues for dealing with a dynamic environment. But, global searching has a similar function as society provides a based reference or a guide for those who have not experienced some situation. It is an important learning style for accumulated knowledge from society or elders. In our program, we did not design global and local searching. But, we need to share information to improve our performance.

The procedure of accumulated knowledge is the most important fundamental for our intelligence in nature, and it has become a pattern for guiding our behaviors, such as educating the offspring, socializing for searching food, or protection. The pattern of formed society has an important function that is education or spreading knowledge, and that also influences our thinking[8].

In this article, the identifying data of associated interactions is connected with how we distinguish the relative data of the environment, and the training data by maximum information is settling down how we accumulate or enhance the knowledge of identifying the relative data. We have searched a lot of data for explaining our idea of what is a reasonable response. The pso and the priori and posteriori knowledge of philosophy demonstrate our learning methods, and our idea has only followed their rules to explore and explain the process of learning methods. The process of learning methods has a major function in that it identifies "cause and effect". We use information entropy to design the system, which is based on probability. Moreover, those daily behaviors are accumulating a lot of knowledge of the relative environmental conditions that is the reason for bringing high survival chances and high competing abilities. And we can only use information entropy to similar the processes of creatures learning and adapting behavior.

II. IDENTIFYING DATA OF ASSOCIATED INTERACTIONS

We use information entropy to identify the data of cause and effect or interaction, in which data is reviewed by a happened event of specific conditions. Our sensory organs can detect such determinative data in a dynamic and interacting space that avoids something dreadful affecting us. And the data normally has relative to our behaviors or even logical thinking. Therefore, this information entropy system can collect this quantity of probabilities for analyzing purposes.

To demonstrate this process of our ideas, we have designed a program to display it. In this program, it is a simple hunter and prey game, and we need to record the happened or caught events of data during each time. We need a 2-dimensional array X_{ij} to save happened times of different data types. Properties of environmental data are setting as X_i, and in the same property, "j" is recoding different values of the same X_i condition.

We want to find out the relation between an event and other data. So we have an entropy value of target event α. And we have discrete entropy formulation

$$\mathrm{H}(X_{ij}) = -\sum_{i=1}^{n} P(X_{ij})\, log_2 P(X_{ij})$$

If we can find the H(α), and the α is binary data, then we set log2. We want to compare their relationship that we use relative entropy (Kullback–Leibler divergence) for measuring their data sequence of time.

A. Relative Entropy

When we want to know the relation of two discrete probability distributions P and Q defined on the same probability space, X_{ij}, the Kullback–Leibler divergence form is defined as

DKL(P||Q)=

$$\sum_{x\in X} P(x) log\left(\frac{P(x)}{Q(x)}\right) = H(P,Q) - H(P)$$

If we can find α and β which they satisfy

DKL(α||β) and DKL(β||α)==0

and β ϵ X_{ij},

If we can find the α and β, and their probability distributions are the same. Therefore, we know the α event happened then the β was also existing. The β data is a boundary of decision results for interacting with others during this dynamic space in a probabilistic construction.

III. TRAINING DATA BY MAXIMUM INFORMATION

Creatures can judge or infer an expected result that has a good definition to describe those result data from our experiences by the posteriori knowledge of philosophy, and the posteriori knowledge is also one part of learning methods. How do we ensure that the collected data is enough for building the best model? We have adopted the max information to set the boundary quantity of the data set for training the best model.

We use DKL to identify the relative data of specific events. However, there is a problem when the reference cases are too few. In that situation, the value of each entropy data has a chance to be the same value during DKL calculation. Such as when the target event has only one case that causes DKL values of distinguishing relative data to have a weak identification. On the other hand, there is a lack of information for entropy systems, and this problem is an underfitting problem for trained models. When we try to find the best-trained model, we have to deal with a balance of trade-offs. The underfit and overfit are the common issues of trade-offs. And we can use maximum entropy to solve the balance problem of the trade-off. Moreover, finding the trade-off problem is multiple optimal problems, and the pareto optimal has the same function during catching the reference data.

In our program, the best training model is based on max entropy to sort the relative data from a binary event result. This binary event result is dependent on a particle's active behavior, and each particle's active behavior is an independent random variable. Therefore, we can use the central limit theorem to judge the best training data using max entropy. S={$O1$,$O2$, Om} S is a simple space of the random experiment
I(X) is self-information from outcome Xϵ S
And we have n trails of random experiment X
occurs C_n(X) times in n trails
each occurrence produces I (X) bits of information
The average information of the random variable X is associated with S as n approaches infinity.

$$H(X) = \lim_{n\to\infty}\left(\frac{1}{n} * \sum_{x=1}^{m} C_n(X) I(X)\right)$$
$$= \sum_{x=1}^{m} P(X) I(X)$$
$$= -\sum_{x=1}^{m} P(X) log_2(P(X))$$

Where I(X)= log_2(P(X)) is the self-information due to an occurrence of outcome X.
Our outcome is a binary result that we can expect the best training data on a certain form of probability. In Binomial distribution, we have

$$\text{P(data | t)} = \binom{n}{k} t^k * (t-1)^{n-k}$$

In this formulation, we cannot control the probability value t, but we can control the n and k values, and when k=1/2n is equal to max entropy, and when the n is large enough that we can have P-value near 1/2 during the first iteration. But after the first iteration, the behaviors are not as independent of the normal distribution, and the P-value is not expected. However, the case is still needed for analysis, and the max entropy qualifies the condition to set the best training data.

IV. CONCLUSION

In this article, we designed a program to express that using information entropy is able to analyze a reasonable relation between self-behaviors and environmental data. Our major point is using information entropy to design the analysis system. The information entropy is based on a probabilistic definition, and the probabilistic inspires us to consider different perspectives. Moreover, that reasonable relation data is forming one part of our logic and thinking. And this one part of our logic and thinking relies on probabilities, and it forms our reasonable thinking and decisions by mathematical support.

In our article, we created this learning process of creatures identifying logic data. It has mathematical support in the result. But the most powerful proof of one part of our logic is based on probability, and we have to mention knowledge diffusion. In knowledge diffusion, it is a better way to prove that probability forms some part of our logic layers. In a social network, there are some people or media that influence us to accept a new idea or product. Therefore, knowledge diffusion wants to know in what conditions can influence people to accept a new product or idea. And knowledge diffusion normally uses cascade and threshold models to analyze the complicated social network interaction[8].

As we can expect, the cascade and threshold models are based on probability, and those systems are fully present that the probability forms our logic layer. However, we analyze the environmental data and find the unique relation of each interaction to demonstrate what data forms reasonable actions or behaviors for a specific question, and those unique relative data are accumulated to become a large knowledge for society.

If we simply observe the distribution of behavioral data that the data is present as a normal distribution, then the creature has recognized the problem and adapted different behaviors that the behavioral data is no longer a normal distribution form as our program demonstrated.

In the end, the data in the dynamic environment are normally in a chaotic situation, if a creature can properly interact with the problem from the dynamic environment that the creature had recognized what data or condition caused the problem, the cause and effect relationship between behavior and environmental data is required by our learning mechanism to continuously improve and fix, and our program is present as this function. The creature's learning method is following case by case. Although it may have a lack of simples, this is true of the learning steps, and this problem is only possibly solved by the construction of society.

REFERENCES

[1] Stuart Russell and Peter Norvig., Artificial Intelligence: A Modern Approach, Edition 3, Prentice Hall., (2002)
[2] X.-S. Yang; S. Deb., Cuckoo search via Lévy flights, World Congress on Nature & Biologically Inspired Computing. IEEE Publications. Paper core summary, 210–214.http://papercore.org/Yang2009
[3] Yang, X. S., Firefly Algorithm, Stochastic Test Functions, and Design Optimisation., Paper core summary., http://papercore.org/Yang2009., (2008) 1-11.
[4] Kennedy, J.; Eberhart, R.., Particle Swarm Optimization., Proceedings of IEEE International Conference on Neural Networks., 4(1942–1948).
[5] Ujjwal Maulik, Sanghamitra Bandyopadhyay (29 April 1999) " Genetic algorithm-based clustering technique, Pattern Recognition., 33(1455) (1465) 1455-1465.
[6] Ahmad Rabanimotlagh Bursa., An Efficient Ant Colony Optimization Algorithm for Multiobjective Flow Shop Scheduling Problem" World Academy of Science, Engineering and Technology., 51(2011) 127-133.
[7] Kirkpatrick, S.; Gelatt, C. D.; Vecchi, M. P., Optimization by Simulated Annealing., Science 220(4598) (1983) 671-680.
[8] Nishith Pathak; Arindam Banerjee; Jaideep Srivastava., A Generalized Linear Threshold Model for Multiple Cascades., 2010 IEEE International Conference on Data Mining, DOI: 10.1109/ICDM.2010.153, ISSN: 2374-8486

Appendix
Matlab code

```
%This program is demonstrating a learning system that is similar to animals
%This program has two major parts, identifying the key data when events are
%caused. And enhance the training data.  There are two parts of our article to demonstrate the idea,
%"Identifying data of associated interactions, and Training data by maximum
%Information."
function [ output_args ] = EvolutionSI( input_args )
%EVOLUTIONSI Summary of this function goes here %   Detailed
explanation goes here
%Grid environments sizeT=100;
T=zeros(sizeT,sizeT);
%create 10 swarms,2-D, and state(size T) of initial individuals, for training data by maximum
%Information we need to set the swarmN=10, which is the particle number, swarmN=10;
%Swarm
S=zeros(10,swarm N);%XXX?6 format
%building reference properties for estimated NN
%1, the distance between current predator and S.2, S and S. %3, mark a shortest neighbor 4, S(state=0) and P latest
position. 5,S(state=0) and P latest second position.
Properties N N=zeros(swarmN,5);
%create simple train Data for behaviors train Data A=[100 100 100];%just for
initial number
```

```
%position
            for ii=1:swarmN,
%x,y,states,moving directions,end X,endY,
S(:,ii)=[51,51,100,0,0,0,0,0,0,0];
    end
%temp
TableEP=zeros(5,3*swarmN,5); SelectionEN=zeros(swarmN,4);
CountSelEN=zeros(swarmN,4);%debug1
ComparingEE=zeros(15,2,swarmN);
%unidentified targets
%position predatorA=[1,1,100,0,51];
%Loop for generations %set 100
iterations for i=1:100,
    %reset All but keep trainData for ii=1:swarmN,
    %x,y,states,moving directions,end X,endY,
    S(:,ii)=[51,51,100,0,0,0,0,0,0,0]; end
    PropertiesNN=zeros(swarmN,5);
    TableEP=zeros(5,3*swarmN,5); SelectionEN=zeros(swarmN,4);
    CountSelEN=zeros(swarmN,4);%debug1
    ComparingEE=zeros(15,2,swarmN);
%reset predatorA predatorA=[1,1,100,0,51];
%point predatorOnly=0;
    %action times %if (j==1)
    for j=1:2000,
%Swarm group action for x=1:swarmN,
            %S call the moving function to decide which position is better for approache if S(3,x)>0
        S(7,x)=S(1,x);%recalling last position
        S(8,x)=S(2,x);%recalling last position end
    S(:,x)=move(S(:,x),trainDataA,predatorA);%no Neural Network
    end

    %predator acts time
    [predatorA,S]=shortestdistancePS(predatorA,S);
        %calculating entropy values-array if
        (predatorA(4)>predatorOnly)
            predatorOnly=predatorA(4); PropertiesNN=analysis(S,predatorA);
            TableEP=EntropyProc(PropertiesNN,S);
                                    [ComparingEE SelectionEN]=EntropyFormulation(S,TableEP,PropertiesNN);
            %accumulating SelectionEN results for ASR=1:swarmN,
                for ASC=1:4,
                    if (SelectionEN(ASR,ASC)~=0)
                    CountSelEN(ASR,ASC)=CountSelEN(ASR,ASC)+1; else
                    CountSelEN(ASR,ASC)=0; end end
            end
        end
    %show the actions
    figure(1); set(gcf,'position',[0,0,1000,1000]);
    plot(S(1,:),S(2,:),'O'); hold on;
    plot(predatorA(1),predatorA(2),'X'); xlim([0 100]);
    ylim([0 100]);
    grid on; grid minor;
    pause(.1);
    clf('reset')
    %hold off;
    %set break point
    %if (predatorA(4)==5)
   %    j=0;
   %    break
  %  end
  %If we set the swarmN=10,and according our "Training data by maximum %Information", the
>4,which it is stop when we have 5 cases, which is %fitting the maximum information rule. if
predatorA(4)>4 j=200; break end
    end
%training Data set
%position and best direction
%
MAXSEN=max(CountSelEN(:));
%must be better than this for training data
%find the Max points point TopMAX=[0,0];
%find the most populating value
Point Culmulating = [PropertiesNN(1,4),0];
%collect data to generate points Cumulative
Can Train=0; for ASC2=4:5,
         for ASR2=1:swarmN,
            if Properties NN(ASR2,1)==300%new1 CanTrain=CanTrain+1; if max(is member(Properties
                NN(ASR2,ASC2),point Culmulating(:,1)))
                    [tempX tempY]=is member (PropertiesNN(ASR2,ASC2),point Culmulating(:,1)); point Culmulating
                    (tempY,2)=point Culmulating (tempY,2)+1;
                else point Culmulating(end+1,:)=[Properties NN(ASR2,ASC2) 1];
                end end
         end
     end
%create suitable logic layers
%find the most populating number anchor P=max(point
Culmulating(:,2)); anchor S=zeros(anchorP,2);
nouse=0;
[nouse anchorP]=is member (anchorP, point Culmulating(:,2)); anchorP=point
Culminating (anchorP,1); nouse=1;
%fetch logic layer and position,4=last,5=last second for AnchC=4:5,
    for           AnchS=1:swarmN,            if          Properties
    NN(AnchS,AnchC)==anchorP   anchor   S   (nouse,:)=[AnchS
    AnchC]; nouse=nouse+1; end
    end
end
%setting rules,S,
[sizeAry nouse]=size(anchorS); if
CanTrain>=2%new2
    for anchS=1:sizeAry,
        %negative behaviors create logic actions
        %negative or positive results,and PropertiesNN transport S.
        %distance S-P=(X,Y)
       [comp nouse]=size(trainDataA); compwell=1;
        for ij=1:comp,
        TempTrain=[S(1,anchorS(anchS,1))-S(5,anchorS(anchS,1))
S(2,anchorS(anchS,1))-S(6,anchorS(anchS,1)) S(4,anchorS(anchS,1))];
            if isequal(TempTrain,trainDataA(ij,:)) compwell=0;
            end
        end
        if compwell trainDataA(end+1,:)=TempTrain; end
       % end
        %distance 2
       % if S(anchorS(anchS,1),3)==0 && anchorS(anchS,2)==5
       % end
        %moving distance between predator and prey
    end end end
trainDataA
%sub Functions
```

```
%move function,input parament:nn,randomly move function
[positionOut]=move(positionIn,trainDa,Pred) %Logic layers to decide which
machines handle job
[trDR trDC]=size(trainDa); if
positionIn(3)>0
    if trDR<=1 positionIn=randomwalk(positionIn); else
    %find the same dirction of predator and S,and abandon actions
    %find the longest distance between predator and S, distancePS=0; for
    ist=2:trDR,
        tempDistan=abs(trainDa(ist,1))+abs(trainDa(ist,2)); if
        tempDistan>distancePS distancePS=tempDistan; end
    end
    tempDistPS=abs(positionIn(1,1)-Pred(1,1))+abs(positionIn(2,1)-Pred(1,2));
    %If the longest distance of S and Predator is low than present distance if
    distancePS>=tempDistPS
    %find the same position and dirction in the predator and S,
        %except actions
        Paction=10; for ist2=2:trDR,
                        if isequal([trainDa(ist2,1) trainDa(ist2,2) ],[positionIn(1,1)-Pred(1,1)
positionIn(2,1)-Pred(1,2)]) Paction(end+1)=trainDa(ist2,3);
        end
    end
    %except actions %find illegal
    action rightAct=100; for ixt=0:4,
        if ismember(ixt,Paction) else
        rightAct(end+1)=ixt;
        end
    end
    %deal with edge of size [idf
    idf2]=size(rightAct); tempAidf=[100];
    if idf2>1
        for ixt3=2:idf2, %edge detection
            if rightAct(ixt3)==1
                if positionIn(2,1)+1>100
                tempAidf(end+1)=ixt3; end
            end
            if rightAct(ixt3)==2
                if positionIn(2,1)-1<0
                tempAidf(end+1)=ixt3; end
            end
            if rightAct(ixt3)==3
                if positionIn(1,1)+1>100
                tempAidf(end+1)=ixt3; end
            end
            if rightAct(ixt3)==4
                if positionIn(1,1)-1<0
        tempAidf(end+1)=ixt3; end end end
    end
    %remove illegal action
    [iddf iddf2]=size(tempAidf); for ixt4=0:iddf2
        tempps=rightAct(rightAct~=tempAidf(iddf2));
        rightAct=tempps; end
    %estimate possible action
    [ittt ittt2]=size(rightAct); direction2=ceil(rand*ittt2);
    direction2 if direction2~=0 direction2=rightAct(direction2);
                    switch direction2 case 0    %dont move
                        positionIn(4)=0; %do nothing case 1    %move
                        up positionIn(2)=positionIn(2)+1;
                        positionIn(4)=1; case 2    %move down
                        positionIn(2)=positionIn(2)-1;
```

```
                              positionIn(4)=2; case 3    %move right
                              positionIn(1)=positionIn(1)+1;
                              positionIn(4)=3; case 4  %move left
                              positionIn(1)=positionIn(1)-1;
                              positionIn(4)=4; otherwise %not decide end%
                              end switch direction
                      else
                      positionIn=randomwalk(positionIn); end
   else position In=randomwalk(positionIn); end end end
positionOut=positionIn; end
function[comparingE signal]=EntropyFormulation(S2,TableEF,relation2)
%size of TableEF
[ni,nj]=size(S2);
[rni,rnj]=size(relation2);
%boolean data sT=0; sF=0;
%Entropy survival for ij=1:rni if
    relation2(ij,1)~=300 sT=sT+1; else
    sF=sF+1;
        end
     end
%Entropy survival
EntropyS=-(sT/(sT+sF))*log2((sT/(sT+sF)))-(sF/(sT+sF))*log2((sF/(sT+sF)));
%problem Nan
if is nan(Entropy S)
    Entropy S=0; end
% Entropy of after selection comparingE=zeros(15,2,nj);%estimating values with R and L2 logic layers(18),2 properties
and 10 entities
BfT=0;
BfF=0;
%2logic layers,2 properties and 10 entities jj=0; for j=4:rnj,
jj=jj+1; for i=1:nj,
        %estimating values comparing E(1,jj,i)=relation2(i,j);
         %>,<=
         %>,Entropy R, properties ,S T,S F
         %LT,LF
         %first logic layer comparing E(2,jj,i)=Table EF(1,nj+(i*2-1),j);%<,L comparing
         E(3,jj,i)=Table EF(1,nj+(i*2),j); %<,D comparing E(4,jj,i)=TableEF(5,nj+(i*2-
         1),j);%>=,L comparing E(5,jj,i)=Table EF(5,nj+(i*2),j); %>=,D
         %calculating Entropy values
         L2jji=comparingE(2,jj,i)/((comparingE(3,jj,i)+comparingE(2,jj,i)));
         D2jji=comparingE(3,jj,i)/((comparingE(3,jj,i)+comparingE(2,jj,i)));
         %bug fixing
         if isnan(L2jji) L2jji=0;
         end
         if isnan(D2jji) D2jji=0;
         end
         %Entropy
         Comparing E(6,jj,i)=-L2jji*log2(L2jji)-D2jji*log2(D2jji); %<,Entropy
         if(L2jji==0) comparing E(6,jj,i)=-D2jji*log2(D2jji); %<,Entropy end
         if(D2jji==0) comparing E(6,jj,i)=-L2jji*log2(L2jji); %<,Entropy
         end
         if (L2jji==0 && D2jji==0)
          comparing E(6,jj,i)=0; end
         L4jji=comparing E(4,jj,i)/((comparing E(4,jj,i)+comparing E(5,jj,i)));
         D4jji=comparingE(5,jj,i)/((comparingE(4,jj,i)+comparingE(5,jj,i)));
         %bug fixing
         if is nan(L4jji) L4jji=0;
         end
         if is nan(D4jji) D4jji=0;
         end %entropy
         comparingE(7,jj,i)=-L4jji*log2(L4jji)-D4jji*log2(D4jji);%>=,Entropy
         if(L4jji==0)
         comparing E(7,jj,i)=-D4jji*log2(D4jji); %>=,Entropy end
         if(D4jji==0) comparing E(7,jj,i)=-L4jji*log2(L4jji); %>=,Entropy
         end
         if (L4jji==0 && D4jji==0)
          comparing E(7,jj,i)=0; end
         Enallf=comparing E(2,jj,i)+comparing E(5,jj,i)+comparing E(3,jj,i)+comparingE(4,jj,i);
         EnR=comparing E(2,jj,i)+comparing E(3,jj,i);
         EnL=comparing E(4,jj,i)+comparing E(5,jj,i);
comparingE(8,jj,i)=EntropyS-((EnR/Enallf*comparingE(6,jj,i))+(EnL/Enallf*comparingE(7,jj,i))) ; %>=,final total
Entropy
         %second logic layer comparing E(9,jj,i)=TableEF(4,nj+(i*2-1),j);%>,L
         comparingE(10,jj,i)=TableEF(4,nj+(i*2),j); %>,D comparingE(11,jj,i)=TableEF(2,nj+(i*2-
         1),j);%<=,L
          comparing E(12,jj,i)=Table EF(2,nj+(i*2),j); %<=,D
       %calculating Entropy values
         L9jji=comparingE(9,jj,i)/((comparingE(9,jj,i)+comparingE(10,jj,i)));
         D9jji=comparingE(10,jj,i)/((comparingE(9,jj,i)+comparingE(10,jj,i)));
         %bug fixing
         if is nan(L9jji)
         L9jji=0;
         end
         if is nan(D9jji) D9jji=0;
         end
         %Entropy
         Comparing E(13,jj,i)=-L9jji*log2(L9jji)-D9jji*log2(D9jji); %>,Entropy
         if(L9jji==0)
         comparing E(13,jj,i)=-D9jji*log2(D9jji); %>,Entropy end
         if(D9jji==0) comparing E(13,jj,i)=-L9jji*log2(L9jji); %>,Entropy
         end
         if (L9jji==0 && D9jji==0)
          comparing E(13,jj,i)=0;
         end
         L11jji=comparingE(11,jj,i)/((comparingE(11,jj,i)+comparingE(12,jj,i)));
         D11jji=comparingE(12,jj,i)/((comparingE(11,jj,i)+comparingE(12,jj,i)));
         %bug fixing
         if is nan(L11jji) L11jji=0;
         end
         if is nan (D11jji) D11jji=0;
         end %entropy
         comparingE(14,jj,i)=-L11jji*log2(L11jji)-D11jji*log2(D11jji);%<=,L Entropy
         if(L11jji==0) comparing E(14,jj,i)=-D11jji*log2(D11jji); %<=,Entropy
         end
         if(D11jji==0) comparing E(14,jj,i)=-L11jji*log2(L11jji); %<=,Entropy
         end
         if (L11jji==0 && D11jji==0)
          comparing E(14,jj,i)=0;
         end
         Enalls=comparing E(9,jj,i)+comparing E(10,jj,i)+comparingE(11,jj,i)+comparingE(12,jj,i);
         EnRs=comparing E(9,jj,i)+comparing E(10,jj,i);
         EnLs=comparing E(11,jj,i)+comparing E(12,jj,i);
comparingE(15,jj,i)=EntropyS-((EnRs/Enalls*comparingE(13,jj,i))+(EnLs/Enalls*comparingE(
14,jj,i))); %>=,final total Entropy
         end
     end
    jj=0;
    signal=zeros(nj,4);
    %comparing E 8and15 have the estimating value to decide relations for ix=1:nj, %<,>=,First
     if (comparing E(8,1,ix)==0)
```

```
            signal(ix,1)=1; end
        %>,<=,F
        if (comparing E(15,1,ix)==0)
           signal(ix,2)=1; end
        %<,>=,Second
         if (comparingE(8,2,ix)==0)
           signal(ix,3)=1; end
        %>,<=,S
        if (comparingE(15,2,ix)==0)
            signal(ix,4)=1; end
    end
end
function[]=food()
%search
%flight
end
function[matries]=analysis(object1s,target1p)
[An,Ann]=size(object1s);
matries=zeros(Ann,5);
%1,distance between current predator and S.2, S and S. %3, mark a shortest neighbor 4,S(state=0) and P latest position.
5,S(state=0) and P latest second position.
    for ix=1:Ann,
        %1,distance between current predator and S if(object1s(3,ix)==100)
                          matries(ix,1)=abs(object1s(1,ix)-target1p(1))+abs(object1s(2,ix)-target1p(2));
        else matries(ix,1)=300; end
        %2,  S and S.,shortest distance between S.
        %3, mark a shortest neighbor shortestXY=200;
        for ixx=1:Ann, if(ix~=ixx) distanceX=abs(object1s(1,ix)-
            object1s(1,ixx)); distanceY=abs(object1s(2,ix)-object1s(2,ixx));
            distanceXY=distanceX+distanceY;
                 if(distanceXY<shortestXY)
                    shortestXY=distanceXY; matries(ix,3)=ixx;
                    matries(ix,2)=shortestXY;
                end end
        end
        %4,S(state=0) and P lastest position. 5,S(state=0) and P lastest second position. if(object1s(3,ix)==0)
        matries(ix,4)=abs(object1s(1,ix)-object1s(5,ix))+abs(object1s(2,ix)-object1s(6,ix));
        matries(ix,5)=abs(object1s(7,ix)-object1s(5,ix))+abs(object1s(8,ix)-object1s(6,ix)); else matries(ix,4)=300;
        matries(ix,5)=300; end
    end
%adjust arrary1 to arrarty 4 and 5 for i=4:5,
   for j=1:Ann, if matries(j,i)==300
       matries(j,i)=matries(j,1); end end end
       end
%building a logical,relation1=s,2=predator function[EntropyTable]=EntropyProc(relation1,S1)
%setting logic layers
[Cn,Cnn]=size(relation1);%10,5
%There are 5 logic layers and 10 for entropy boolean,which were set by authour,EntropyTable(5 logic
%layers,5Elements,10=S numbers),
EntropyTable=zeros(5,3*Cn,Cnn);%5X30X5,3*Cn=30 for ij=1:Cnn, for
    ijj=1:Cn,
            %comparing each others
            for ijjj=1:Cn
                %excluding self
                if relation1(ijj,ij)~=300
                if (ijj~=ijjj)
                                % compared results accumulate 5 logic layers,run (Cn-1) times
                    %first logic layer,">" ,smaller
                    if relation1(ijj,ij)>relation1(ijjj,ij)&(relation1(ijjj,ij)~=300)
                    EntropyTable(1,ijj,ij)=EntropyTable(1,ijj,ij)+1; %adjusting the
                        state for entropy if S1(3,ijjj)==100
                        EntropyTable(1,(Cn+(ijj*2)-1),ij)=EntropyTable(1,(Cn+(ijj*2)-1),ij)+1; else
                        EntropyTable(1,(Cn+(ijj*2)),ij)=EntropyTable(1,(Cn+(ijj*2)),ij)+1; end
                    end
                    %second logic layers,">="
                    if relation1(ijj,ij)>=relation1(ijjj,ij)&(relation1(ijjj,ij)~=300)
                    EntropyTable(2,ijj,ij)=EntropyTable(2,ijj,ij)+1; if S1(3,ijjj)==100
                        EntropyTable(2,(Cn+(ijj*2)-1),ij)=EntropyTable(2,(Cn+(ijj*2)-1),ij)+1; else
                        EntropyTable(2,(Cn+(ijj*2)),ij)=EntropyTable(2,(Cn+(ijj*2)),ij)+1; end
                    end
                    %Third logic layers."="
                    if (relation1(ijj,ij)==relation1(ijjj,ij)&(relation1(ijjj,ij)~=300))
                    EntropyTable(3,ijj,ij)=EntropyTable(3,ijj,ij)+1; if S1(3,ijjj)==100
                        EntropyTable(3,(Cn+(ijj*2)-1),ij)=EntropyTable(3,(Cn+(ijj*2)-1),ij)+1; else
                        EntropyTable(3,(Cn+(ijj*2)),ij)=EntropyTable(3,(Cn+(ijj*2)),ij)+1; end
                    end
                    %Fourth logic layers."<"
                    if (relation1(ijj,ij)<relation1(ijjj,ij)&(relation1(ijjj,ij)~=300))
                    EntropyTable(4,ijj,ij)=EntropyTable(4,ijj,ij)+1; if S1(3,ijjj)==100
                        EntropyTable(4,(Cn+(ijj*2)-1),ij)=EntropyTable(4,(Cn+(ijj*2)-1),ij)+1; else
                        EntropyTable(4,(Cn+(ijj*2)),ij)=EntropyTable(4,(Cn+(ijj*2)),ij)+1; end
                    end
                    %Fifth logic layers."<="
                    if (relation1(ijj,ij)<=relation1(ijjj,ij)&(relation1(ijjj,ij)~=300))
                    EntropyTable(5,ijj,ij)=EntropyTable(5,ijj,ij)+1; if S1(3,ijjj)==100
                        EntropyTable(5,(Cn+(ijj*2)-1),ij)=EntropyTable(5,(Cn+(ijj*2)-1),ij)+1; else
                        EntropyTable(5,(Cn+(ijj*2)),ij)=EntropyTable(5,(Cn+(ijj*2)),ij)+1; end
                    end
                else
                   %fix(ijj~=ijjj) cause problems of entropy %firt logic layer,">" ,smaller if
                        relation1(ijj,ij)>relation1(ijjj,ij)&(relation1(ijjj,ij)~=300)
                        %adjusting the state for entropy if S1(3,ijjj)==100
                            EntropyTable(1,(Cn+(ijj*2)-1),ij)=EntropyTable(1,(Cn+(ijj*2)-1),ij)+1; else
                            EntropyTable(1,(Cn+(ijj*2)),ij)=EntropyTable(1,(Cn+(ijj*2)),ij)+1; end
                        end
                        %second logic layers,">="
                        if relation1(ijj,ij)>=relation1(ijjj,ij)&(relation1(ijjj,ij)~=300)
                            if S1(3,ijjj)==100
                            EntropyTable(2,(Cn+(ijj*2)-1),ij)=EntropyTable(2,(Cn+(ijj*2)-1),ij)+1; else
                            EntropyTable(2,(Cn+(ijj*2)),ij)=EntropyTable(2,(Cn+(ijj*2)),ij)+1; end end
                        %Third logic layers."=" if
                        (relation1(ijj,ij)==relation1(ijjj,ij)&(relation1(ijjj,ij)~=300)) if S1(3,ijjj)==100
                            EntropyTable(3,(Cn+(ijj*2)-1),ij)=EntropyTable(3,(Cn+(ijj*2)-1),ij)+1; else
                            EntropyTable(3,(Cn+(ijj*2)),ij)=EntropyTable(3,(Cn+(ijj*2)),ij)+1; end
                        end
                        %Fourth logic layers."<"
                        if (relation1(ijj,ij)<relation1(ijjj,ij)&(relation1(ijjj,ij)~=300)) if S1(3,ijjj)==100
                            EntropyTable(4,(Cn+(ijj*2)-1),ij)=EntropyTable(4,(Cn+(ijj*2)-1),ij)+1; else
                            EntropyTable(4,(Cn+(ijj*2)),ij)=EntropyTable(4,(Cn+(ijj*2)),ij)+1; end
                        end
                        %Fifth logic layers."<=" if
                        (relation1(ijj,ij)<=relation1(ijjj,ij)&(relation1(ijjj,ij)~=300)) if S1(3,ijjj)==100
                            EntropyTable(5,(Cn+(ijj*2)-1),ij)=EntropyTable(5,(Cn+(ijj*2)-1),ij)+1; else
                            EntropyTable(5,(Cn+(ijj*2)),ij)=EntropyTable(5,(Cn+(ijj*2)),ij)+1; end
                        end end
                % if(relation1(ijj,ij~=300)) else
                EntropyTable(1,ijj,ij)=300; EntropyTable(2,ijj,ij)=300;
                 EntropyTable(3,ijj,ij)=300; EntropyTable(4,ijj,ij)=300;
                    EntropyTable(5,ijj,ij)=300;
```

```
                    %adjusting the state for entropy
                end
            end
        end
    end
end
function[object,others]=shortestdistancePS(object,others)
[N,NN]=size(others);
%selected a target for predator Tempj=1;
if others(3,1)>0 shortX=others(1,1); shortY=others(2,1); totalD=abs(object(1)-
others(1,1))+abs(object(2)-others(2,1)); else shortX=100; shortY=100; totalD=200;
end
%searching shortest targets for j=2:N,
    if others(3,j)>0 totalT=abs(object(1)-others(1,j))+abs(object(2)-others(2,j));
        if totalD>totalT
    shortX=others(1,j);
    shortY=others(2,j); totalD=totalT;
    Tempj=j; end end
end
%moving direction updown=object(2)-
shortY; rightleft=object(1)-shortX;
tempC=1; tempOldX=object(1);
tempOldY=object(2); if updown<0
object(2)=object(2)+1; tempC=0;
end if updown>0
    object(2)=object(2)-1;
    tempC=0;
end
if rightleft<0&tempC
    object(1)=object(1)+1;
end
if rightleft>0&tempC
    object(1)=object(1)-1; end %catch one if
object(2)==others(2,Tempj)&object(1)==others(1,Tempj) others(3,Tempj)=0;
others(5,Tempj)=tempOldX; others(6,Tempj)=tempOldY; object(4)=object(4)+1;
end
end function[objec]=randomwalk(objec)
direction=floor(rand*5); switch direction case 0
%dont move objec(4)=0; %do nothing case 1
%move up objec(2)=objec(2)+1; objec(4)=1; case 2
%move down objec(2)=objec(2)-1; objec(4)=2; case 3
%move right objec(1)=objec(1)+1; objec(4)=3; case 4
%move left objec(1)=objec(1)-1; objec(4)=4;
otherwise %not decide end% end switch direction
            if objec(1)>100 objec(1)=100;
            end
            if objec(2)>100 objec(2)=100;
            end
            if objec(1)<0
            objec(1)=0; end
            if objec(2)<0 objec(2)=0;
end end end
```