KNAPSACK PROBLEM USING GENETIC ALGORITHM & SIMULATED ANNEALING BY HYDER NABI

Hyder Nabi
Roll No. 20045110005
MCA-II-2020
Sub.: Artificial Intelligence

Report Prepared by
Hyder Nabi
Submitted To
Dr. Sadaf Afzal

Genetic Algorithm

PROBLEM: KNAPSACK PROBLEM

PROBLEM DESCRIPTION: We have *n* objects with weight W and value V of every object and a knapsack of capacity K, we have to put the objects in the knapsack so that the value is maximized and weight of all those objects should be less than or equal to K.

Here,

n = 8

Weights (W) = 25 35 45 5 25 3 2 2
Values (V) = 350 400 450 20 70 8 5 5
Knapsack Capacity (K) = 104
Encoding: Binary Encoding:

e.g.;

1 = item/object is present in knapsack.

0 = item/object is not present in knapsack.

State (C) = [1 0 0 1 0 1 1 0]

Initial Population:

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

Fitness Function:

$$f = \begin{cases} \sum_{i=1}^{n} Ci * Vi, & if \sum_{i=1}^{n} Ci * Wi \leq K \\ 0, & if \sum_{i=1}^{n} Ci * Wi > K \end{cases}$$

C_i = 1 or 0.

CODE:

```
%AUTHOR : HYDER NABI
%ROLL NO. : 05
%IMPLEMANTAATON OF GENETIC ALGORITHM USING KNAPSACK PROBLEM
function GeneticII()
%BINARY ENCODED CHROMOSES/STATES
%THE RANDOM INITIAL POPULAITON
population = load('pop.txt');

%THE WEIGHTS OF EVERY OBJECT IN KNAPSACK PROBLEM
weights = [25,35,45,5,25,3,2,2];

%THE VALUES ASSOCIATED WITH EVERY OBJECT
values = [350,400,450,20,70,8,5,5];

%THE CAPACITY OF KNAPSACK
capacity = 104;

%INITIAL GENERATIONS
generation = 0;

%LOOP CONTROLLER
count = 1;

while count

    %#STEP 1
    %CALCULATE THE FITNESS OF EVERY STATE(CHROMOSOME) IN THE POPULATION
    %BY CALLING FITNESS FUNCTION
    population_fit = fitness(population,values,weights,capacity);

    %#STEP 2
    %SELECT THE CHROMOSOMES/STATES/PARENTS FOR REPRODUCTION/CROSSOVER
    %BY CALLING SELECTION FUNCTION
    parents = selection(population_fit,population);

    %#STEP 3
    %PERFORM THE CROSSOVER OF EVERY PAIR OF PARENTS USING PROBABILITY
```

```
    %BY CALLING CROSSOVER FUNCTION
    offsprings = crossover(parents);

    %#STEP 4
    %MUTATE THE CHROMOSOMES USING PROBABILITY
    %BY CALLING MUTATION FUNCTION
    newPopulation = mutation(offsprings);

    %THE NEW POPULATION(SET OF STATES) AT THE END OF FIRST GENERATION
    population = newPopulation;

    %THE NO OF GENERATONS(TERMINATION CRITERAI)
    if generation == 1000
        count = 0;
    end
    generation = generation + 1;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Final RESULT
%CALCULATE THE FITNESS OF EVERY CHROMESOME IN THE FINAL POPULAITON
fit = fitness(population,values,weights,capacity);

%CALCULATE THE INDEX OF CHROMOSOME/STATE WITH THE HIGHEST FITNESS
for i = 1:size(population,1)
    if fit(i) == max(fit);
        index = i;
    end
end

    %The Final Weight of objects in the Knapsack
   FinalWeight = finalResult(index,population,weights);
   %Their Value
   FinalValue = max(fit);
   %Their encoded(binary) representation
   FinalChromosome = population(index,:);

   %DISPLAY VALUES
   disp("WEIGHT");
   disp(FinalWeight);
   disp("VALUE");
   disp(FinalValue);
   disp("BINARY REPRESENTATION OF SOLUTION STATE");
   disp(FinalChromosome);


end


%THIS FUNCTION IS USED TO CALCLATE THE OBJECTIVE VALUE OR FITNESS VALUE
%OF EVERY CHROMOSOME IN THE POPULATION
function population_fit = fitness(population,values,weights,capacity)
 for i=1:size(population,1)
     %CALCULATE THE WEIGHTS OF EVERY CHROMOSOME
     % AND VALUE OF EVERY CHROMOSOME IN THE POPULATION
        temp_ft = 0;
        temp_wt = 0;
        for j = 1:size(population,2)
            temp_ft = temp_ft + (population(i,j)*values(j));
```

```
            temp_wt = temp_wt + (population(i,j)*weights(j));
        end;
        %IF WEIGHT OF CHROMOSOME EXCEEDS THE WEIGHT OF KNAPSACK
        %THEN THE FITNESS IS ZERO
        %ELSE FITNESS IS THE SUM OF VALUES OF CORROSPONDING OBJECTS
        if temp_wt > capacity
            population_fit(i) = 0;
        else
            population_fit(i) = temp_ft;
        end
    end
end


%THIS FUNCTION SELECTS THE CHROMOSOMES FOR CROSSOVER/REPRODUCTION.
%THE SELECTION CRITERIA IS BASED ON THE HIGHEST VALUE,
%WHICH IS PROPORTIONAL TO THE FITNESS OF EVERY CHROMOSOME
%THE MORE FIT IS THE CHROMOSOME , THE MORE CHANCES ARE TO GET SELECTED FOR
%REPRODUCTION/CROSSOVER
%THE PROCEDURE USED FOR SELECTION IS THE ROULETTE WHEEL
function parents = selection(population_fit,population)
%THE SUMO OF FITNESSES
S = sum(population_fit);

parents = [];

for i = 1:size(population,1)

%A RANDOM NUMBER BETWEEN 0 AND S
r = Random(0,S);

partial_sum = 0;

for j = 1:size(population,1)
    partial_sum = partial_sum + population_fit(j);
    %AT WHICH CHROMOSOME THE PARTIAL SUM EXCEEDS THE RANDOM NUMBER
    %SELECT THAT CHROMOSOME AND ROTATE THE ROULETTE WHEEL AGAIN.
    %DUPLICATION MAY BE POSSIBLE
    if partial_sum >= r
        parents(i,:) = population(j,:);
        break;
    end
end
end
end


%THIS FUNCTION PRODUCES THE OFFSPRINGS /CROSSOVER
%IT IS BASED ON PROBABILITY (CROSSOVER RATE)
function offsprings = crossover(parents)
%Cross Over RATE (Probability) = CR
CR = 0.7;
offsprings = [];

%TAKING PAIR OF CHROMOSOMES INDEX 1->(PAIR 1 AND 2), 3->(PAIR 3 AND 4)
for i = [1,3]
    %CHOOSE A RANDOM NUMBER BW 1 AND 0
    r = rand();
```

```
    %IF RANDOM NUMBER IS LESS THAN CR
    %THEN CROSSOVER HAPPENS
    %OTHERWISE SELECT CHROMOSOMES AS THEY ARE
    if r <= CR

        %THE CROSSOVER POINT IS RANDOMLY CHOOSEN
        %ONE POINT CROSSOVER METHOD IS USED

        crossover_point = ceil(Random(1,15));

        %CALL FUNCTION ONEPOINT() TO SWAP THE GENES/BITS IN A CHROMOSOME
        temp_offspring = onepoint(crossover_point,parents(i:i+1,:));

        offsprings(i:i+1,:)  = temp_offspring();

    else
        offsprings(i:i+1,:) = parents(i:i+1,:);

    end

end


    %function which swaps the bits/GENES after crossover point of a pair of
parents
    function temp_offspring = onepoint(crossover_point,parents_pair)
        temp_offspring = [];
        temp = parents_pair(1,crossover_point+1:end);
        parents_pair(1,crossover_point+1:end) =
parents_pair(2,crossover_point+1:end);
        parents_pair(2,crossover_point+1:end) = temp;
        temp_offspring = parents_pair;

    end
end


%THIS FUNCTION IS USED TO MUTATE THE CHROMOSOMES WITH PROBABILITY
function newPopulation = mutation(offsprings)
%Mutation Rate(PROBABILITY)
MR = 0.2;
newPopulation = [];

for i = 1:4
    for j = 1:8
        %CHOOSE A RANDOM NO FOR EVERY BIT/GENE IN EVERY CHROMOSOME.
        r = rand();

        %IF RANDOM NUMBER CHOOSEN IS LESS THAN THE MR
        %MUTATE THE BIT/FLIP THE BIT FROM 1 TO 0 OR 0 TO 1
        %ELSE MUTATION IS NOT PERFORMED FOR THAT BIT/GENE
        if r <= MR
            %flip the bits in a chromosome[mutation]
            %IF BIT IS 0 SET IT TO 1
            if(offsprings(i,j) == 0)
                newPopulation(i,j) = 1;
            else
                %ELSE SET TO TO 0
```

```
                newPopulation(i,j) = 0;
            end
        else
            newPopulation(i,j) = offsprings(i,j);
        end
    end

end
end

%FUNCTION WHICH CALCULATES THE FINAL RESULT/WEIGHT
function FinalWeight = finalResult(index,population,weights)
    FinalWeight = 0;
    for i = 1:size(population,2)
        FinalWeight = FinalWeight + (population(index,i)*weights(i));
    end
end

%FUNCTION WHICH GENERATES THE RANDOM NO WITHIN SPECIFIED INTERVAL
function r = Random(a,b)
    r = (b-a).*rand()+a;
end
```

Output: The output of the 10 runs of the program.

When CR (Crossover Rate) = 0.7

MR (Mutation Rate) = 0.2

Optimal Solution

Weight = 104

Value = 900

State = 1 0 1 1 1 0 1 1

```
>> GeneticII
WEIGHT
   94

VALUE
  850

BINARY REPRESENTATION OF SOLUTION STATE
     1     1     0     1     1     0     1     1

>> GeneticII
WEIGHT
   92

VALUE
  888

BINARY REPRESENTATION OF SOLUTION STATE
     0     1     1     1     0     1     1     1
```

>> GeneticII

WEIGHT

104

VALUE

900

BINARY REPRESENTATION OF SOLUTION STATE

1 0 1 1 1 0 1 1

>> GeneticII

WEIGHT

92

VALUE

845

BINARY REPRESENTATION OF SOLUTION STATE

1 1 0 1 1 0 0 1

>> GeneticII

WEIGHT

72

VALUE

788

BINARY REPRESENTATION OF SOLUTION STATE

1 1 0 1 0 1 1 1

>> GeneticII

WEIGHT

92

VALUE

888

BINARY REPRESENTATION OF SOLUTION STATE

0 1 1 1 0 1 1 1

>> GeneticII

WEIGHT

100

VALUE

890

BINARY REPRESENTATION OF SOLUTION STATE

1 0 1 1 1 0 0 0

>> GeneticII

WEIGHT

98

VALUE

878

BINARY REPRESENTATION OF SOLUTION STATE

1 0 1 0 1 1 0 0

>> GeneticII

WEIGHT

92

VALUE

888

BINARY REPRESENTATION OF SOLUTION STATE

0 1 1 1 0 1 1 1

>> GeneticII

WEIGHT

88

VALUE

828

BINARY REPRESENTATION OF SOLUTION STATE

1 1 0 0 1 1 0 0

Simulated Annealing

Problem: Knapsack

Weights (W) =
70,73,77,80,82,87,90,94,98,106,110,113,115,118,120
Values (V) =
135,139,149,150,156,163,173,184,192,201,210,214,221,229,240

Capacity of Knapsack (K) = 750

n = 15

Initial State = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Initial Temperature = 1000

Alpha (Temperature Reduction Factor) = 0.1

Objective function:

$$MAX \sum_{i=1}^{n} Vi * Ci;$$

$$where\ Ci = \begin{cases} 1; if\ the\ item\ is\ in\ the\ knapsack \\ 0; \qquad\qquad\qquad\qquad otherwise \end{cases}$$

Subject to Constraints

$$\sum_{i=1}^{n} Ci * Wi \ \leq K$$

CODE:

```
function SimulatedAnnealing()
%KNAPSACK PROBLEM USING SIMULATED ANNEALING

%WEIGHTS OF 15 OBJECTS
Weights = [70,73,77,80,82,87,90,94,98,106,110,113,115,118,120];

%VALUES/COST/PROFIT OF THE RESPECTIVE OBJECTS
Values = [135,139,149,150,156,163,173,184,192,201,210,214,221,229,240];

%NO OF OBJECTS(LENGTH OF WEIGHTS/VALUES)
len = size(Weights,2);

%TOTAL CAPACITY OF THE KNAPSACK
Capacity = 750;

%INITIAL TEMPERATURE = 100
Temperature = 1000;

%TEMPERATURE REDUCTION FACTOR/SCHEDULE
alpha = 0.1;

%INITIAL CURRENT NODE/STATE
CurrentNode = zeros(1,len);

    %REPEAT UNTIL TEMPEATURE IS NEARLY 0
    while floor(Temperature) ~= 0


        %RANDOMLY GENERATE THE SUCCESSOR OF THE CURRENT NODE.
        %BY CALLING GenerateSuccessor() FUNCTION
        NextNode = GenerateSuccessor(CurrentNode,Weights,Capacity);

        %CALCULATE THE OBJECTIVE COST OF THE CURRET NODE
        %THE OBJECTIVE COST HERE IS THE VALUE/PROFIT OF INDIVIDUAL OBJECTS
        %THE OBJECTIVE VALUE IS SUPPOSED TO MAXIMISE
        Value_Current = CalculateValue(CurrentNode,Values,len);

        %CALCULATE THE OBJECTIVE COST OF THE NEXT/SUCCESSOR NODE
        Value_Next = CalculateValue(NextNode,Values,len);

        %DELTA E
        %THE ENERGY: THE CHANGE IN THE VALUES OF CURRENT AND NEXT
NODE/STATE
        %
        Change_in_Energy = Value_Next - Value_Current;

        %IF CHANGE IS POSITIVE
        %POSITIVE INDICATES THE GOODNESS OF THE NEXT NODE
        if Change_in_Energy > 0

            %MAKE NEXT NODE AS CURRENT NODE
            CurrentNode = NextNode;
```

```
        else
            %CALCULATE A RANDOM PROBABILITY
            randomProbability = rand(0,1);

            %CALCULATE THE THRESHOLD (PROBABILITY).
            %IT WILL INDICATE WHEATHER THE BAD NODE(NEXT NODE) WILL BE
            %SELECT AS CURRENT OR NOT
            %THE DOWN HILL STEP(IN OTHER TERMINOLIGY)
            Thrshold = exp(Change_in_Energy/Temperature);

            %IF RANDOM CALCULATED PROBABILITY IS LESS THAN THE THROSHOLD
            %THEN MAKE NEXT NODE AS CURRENT
            if randomProbability <= Thrshold
                CurrentNode = NextNode;
            end
        end

        %ADJUST THE TEMPERATURE
        %REDUCE BY ALPHA
        Temperature = Temperature - alpha;


    end

    %PRINT THE FINAL VALUE
    Value_Current = CalculateValue(CurrentNode,Values,len)

end


%THIS FUNCTION IS USED TO GENERATE THE SUCCESSOR OF THE CURRENT NODE
%WITH RANDOM PROBABILITY
function NextNode = GenerateSuccessor(CurrentNode,Weights,Capacity)
%SET VALUES IN THE CURRENT NODE TO VALUES IN THE NEXT NODE;
NextNode = CurrentNode;

    while true
        %GENERATE A RANDOM INDEX B/W 1 TO LAST INDEX OF STATE(NODE)
        randomIndex = floor(Random(1,size(CurrentNode,2)));

        %IF THE VALUE OF STATE AT THIS INDEX IS 0
        %BREAK AND CHANGE IT TO 1 OUTSIDE THE LOOP
        if NextNode(randomIndex) == 0
            break;
        end
    end


    %ADD THE ITEM IN THE KNAPSACK WITH RANDOM PROBABILITY
    %BY SETTING THE RANDOMLY CHOOSEN POSITION TO 1
    %(WE CAN SAY WE ARE ADDING AN ITEM TO THE KNAPSACK).
    NextNode(randomIndex) = 1;

    %IF THE WEIGHT OF THE KNAPSACK EXCEEDS THE LIMIT/CAPACITY
    %THEN DROP AN RANDOMLY CHOOSEN ITEM FROM THE KNAPSACK
    %BY SETTING THAT POSITIN TO 0
    while sum(NextNode .* Weights)>Capacity
```

```
        while true
            %CHOOSE A RANDOM INDEX TO DROP AN ITEM
            randomIndex = floor(Random(1,size(CurrentNode,2)));

            %IF THE VALUE OF THE STATE AT THIS LOCATION IS 1
            %BRAEK AND SET IT TO 0 OUTSIDE THE LOOP
            if NextNode(randomIndex) == 1
                break;
            end
        end

        %DROP THE ITEM IF CONSTRAINTS ARE NOT SATISFIED
        %MEANS IF KNAPSACK CAPACITY IS EXCEEDED
         NextNode(randomIndex) = 0;
        %LOOP AGAIN TO CHECK THE CONSTRAINTS
    end

end





%THIS SIMPLE FUNCTION IS USED TO CALCULATE THE OBJECTIVE VALUE OF THE STATE
function stateValue = CalculateValue(state,Values,len)
    stateValue = 0;
    for i = 1:len
        %THE VALUE OF THE STATE IS THE ADDITION OF VALUES OF ALL THOSE
        %OBJECTS WHICH ARE PRESENT IN THE KNAPSACK
        stateValue = stateValue + state(i)*Values(i);
    end

end

%FUNCTION USED TO CALCULATE THE RANDOM NUMBER BETWEEN A SPEIFIED RANGE
function r = Random(a,b)
    r = (b-a).*rand()+a;
end
```

OUTPUT:

The output of the 10 runs of the Program.

```
>> SimulatedAnnealing

Value_Current =

    1433

>> SimulatedAnnealing

Value_Current =

    1437

>> SimulatedAnnealing

Value_Current =

    1440

>> SimulatedAnnealing

Value_Current =

    1434
```

```
>> SimulatedAnnealing

Value_Current =

        1440

>> SimulatedAnnealing

Value_Current =

        1439

>> SimulatedAnnealing

Value_Current =

        1444

>> SimulatedAnnealing

Value_Current =

        1440

>> SimulatedAnnealing
```

```
Value_Current =

    1432

>> SimulatedAnnealing

Value_Current =

    1440
```