17

Hunting Search Algorithm

Ferhat Erdal
Department of Civil Engineering
Akdeniz University, Turkey

Osman Tunca
Department of Civil Engineering
Karamanoglu Mehmetbey University, Turkey

CONTENTS

17.1 Introduction

Firstly proposed by Oftadeh et al. in 2009 [1], the Hunting Search (HuS) technique is inspired by the behavior of animals hunting in a group. HuS has the distinguishing features of algorithm simplicity and search efficiency. During recent years, it has been successfully used in areas such as function optimization [1], simulated manufacturing condition improvement [2], hybrid PV-battery storage system optimization [3], a quadratic assignment problem [4], optimization of a traveling salesman problem [2] and steel and composite beam design [5, 6]. In this chapter, we first introduce the underlying inspiration and principles of the basic HuS method. The rest of this chapter is organized as follows: in Section 17.2.1 the pseudo code for the HuS algorithm is presented and discussed, in Section 17.3 the source-code for the HuS algorithm in Matlab is shown with more detail. Then, the pseudo-code of the HuS algorithm in

C++ programing language is demonstrated. In Sections 17.5 and 17.6, the numerical example of the HuS algorithm and some conclusions are provided, respectively.

17.2 Original HuS algorithm

In this section, the mentioned algorithm will be discussed with more detail. Compared to hunting process, in an optimization problem each 'hunter' is replaced with a 'solution' of the design problem. Note that group hunting and our stochastic search algorithm have a primary difference. In group hunting of lions, wolves, and dolphins, hunters can see the prey or when they hunt at night at least they can sense the smell of the prey and determine its position. In contrast, in optimization problems we have no indication of the optimum solution. In group hunting of animals, however, the solution (prey) is dynamic and the hunters (based on the current position of the prey) must correct their position. In optimization problems instead, the optimum solution is static and does not change its position during the search process. In fact, both real and artificial group hunting have their own difficulties. To resemble the dynamics of the hunting process in our algorithm, artificial hunters move towards the leader. The leader is the hunter which has the optimum solution among current solutions at hand. In fact, we assume that the leader has found the optimum point and other members move towards it. If any of them finds a point better than the current leader, it becomes leader in the next stage. Hunters not only gradually move toward the prey but also correct their position. Therefore, in this algorithm, after moving toward the previous leader, the hunters correct their position based on the position of other members. This is accomplished by introducing the 'hunting group consideration rate' (HGCR). In addition in the group hunting of real animals, if the prey escapes out of the ring, the hunters organize themselves to encircle the prey again. In the HuS algorithm, the ability will be given to hunters, so they can search out of the ring of siege. In the algorithm, if the positions of the hunters/solutions are too close to each other, the group is reorganized to find the optimum point in the next effort.

17.2.1 Pseudo-code and description of HuS algorithm

The pseudo-code for the global version of the HuS algorithm is presented in Algorithm 16, the flow chart of HuS algorithm is shown in Figure 17.1.

Algorithm 16 Pseudo code of original HuS.

```
1: Initialize optimization problem and parameters;
   $HGS$: number of hunters- hunting group size
   $MML$: maximum movement toward leader
   $HGCR$: hunting group consideration rate
   $EN$: number of epochs
   $Ra$: distance radius
   $\alpha$ and $\beta$ reorganization parameters
2: Initialize hunting group - generate random population of $HGS$ solutions;
3: Calculate the fitness values of initial members: $fitness(i) = f(x_i)$;
4: Set leader as the best fitness of all hunters
5: while (the termination conditions are not met) do
6:     for each hunter i do
7:         Change the position- move toward leader in view of $MML$
8:         Calculate $fitness(i)$
9:         if $fitness(i)$ is better than leader; leader = $fitness(i)$ then
10:        end if
11:    end for
12:    for each hunter i do;
13:        Correct position on the basis of $HGCR$ and local search
14:        Calculate $fitness(i)$
15:        if $fitness(i)$ is better than leader; leader = fitness(i) then
16:        end if
17:    end for
18:    for each hunter i do
19:        Update the position- reorganize the hunting group
20:        Calculate $fitness(i)$
21:        if $fitness(i)$ is better than leader; leader = $fitness(i)$ then
22:        end if
23:    end for
24: end while
```

Step 1. Initialize the design problem and algorithm parameters.
In this step, HuS algorithm parameters that are required to solve the optimization problem are also specified: number of solution vectors in hunting group (HGS), maximum movement toward the leader (MML), and hunting group consideration rate (HGCR), which varies between 0 and 1. The parameters MML and HGCR are parameters that are used to improvise the hunter solution vector.

Step 2. Initialize the hunting group.
Based on the solution vectors in the hunting group (HGS), the hunting group matrix is filled with feasible solution vectors. The values of objective function are computed and the leader is defined based on the values of objective functions of the hunters.

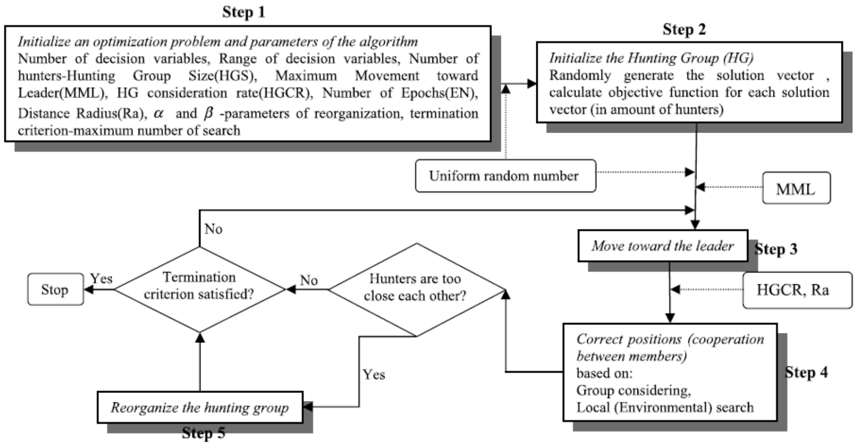
Step 1
Initialize an optimization problem and parameters of the algorithm
Number of decision variables, Range of decision variables, Number of hunters-Hunting Group Size(HGS), Maximum Movement toward Leader(MML), HG consideration rate(HGCR), Number of Epochs(EN), Distance Radius(Ra), α and β -parameters of reorganization, termination criterion-maximum number of search
Step 2
Initialize the Hunting Group (HG)
Randomly generate the solution vector , calculate objective function for each solution vector (in amount of hunters)
Uniform random number
MML
No
Move toward the leader
Step 3
Stop
Yes
Termination criterion satisfied?
No
Hunters are too close each other?
HGCR, Ra
Correct positions (cooperation between members) based on: Group considering, Local (Environmental) search
Step 4
Yes
Reorganize the hunting group
Step 5

FIGURE 17.1
Displays the procedure of the Hunting Search algorithm, which consists of the steps discussed in the text.

Step 3. Moving toward the leader.
The new solution vectors $x' = (x'_1, x'_2, x'_3, ..., x'_N)$ are generated by moving toward the hunter that has the best position in the group as follows:

$$x'_i = x_i + rand \times MML \times (x_i^L - x_i) \tag{17.1}$$

The MML is the maximum movement toward the leader, rand is a uniform random number which varies between 0 and 1, and x_i^L is the position value of the leader for the ith variable.
If the movement toward the leader is successful, the hunter stays in its new position. However, if each hunter's previous position is better than its new position it comes back to the previous position. This will give two main advantages. First, we do not compare the hunter with the worst hunter in the group, so we allow the weak members to search for other solutions; they may find better solutions. Secondly, for prevention of rapid convergence of the group the hunter compares its current position with its previous position; this means that good positions will not be eliminated in the cycle.
The value of parameter MML depends on the number of iterations in each epoch. According to different searches, the range between 0.05 (for epochs with large number of iterations) and 0.4 (for epochs with small number of iterations) gives the best results.
Step 4. Position correction-cooperation between members.
The cooperation among the hunters is required to be modeled in order to conduct the hunt more efficiently. After moving toward the leader, hunters choose another position to find better solutions. Hunters correct their position either following *"real value correction"* or *"digital value correction"*. In real value cor-

rection, the new hunter's position $x' = x'_1, x'_2, ..., x'_n$ is generated from HG, based on hunting group considerations or position corrections. For instance, the value of the first design variable for the j^th hunter $x_i^{j'}$ for the new vector, can be selected as a real number from the specified $HG(x_i^1, x_i^2, ..., x_i^{HGS})$ or corrected using $HGCR$ parameter (chosen between 0 and 1). The variable is updated as follows:

$$x_i^{j'} \leftarrow \begin{cases} x_i^{j'} \in \{x_i^1, x_i^2, i = 1, \ldots, x_i^{HGS}\} & i = 1, \ldots, N \\ x_i^{j'} = x_i^j \pm Ra \ with \ probability \ (1 - HGCR) & j = 1, \ldots, HGS \end{cases} \tag{17.2}$$

The parameter $HGCR$ is the probability of choosing one value from the hunting group stored in the HG. It is reported that selecting values between 0.1 and 0.4 produces better result. Ra is an arbitrary distance radius for the continuous design variable. It can be fixed or reduced during the optimization process. Several functions can be selected for reducing Ra. Eq. 17.3.

$$Ra(it) = Ra_{min}(x_i^{max} - x_i^{min}) \exp\left(\frac{\ln\left(\frac{Ra_{max}}{Ra_{min}}\right) \times it}{itm}\right) \tag{17.3}$$

where, it is the iteration number. x_i^{max} and x_i^{min} are maximum and minimum possible values for x_i. Ra_{max} and Ra_{min} are the maximum and minimum of relative search radius of the hunter, respectively, and itm is the maximum number of iterations in the optimization process.

In digital value correction, instead of using real values of each variable, hunters communicate with each other by the digits of each solution variable. For example, the solution variable with the value of 23.4356 has six meaningful digits. For this solution variable, the hunters choose a value for the first digits *(i.e.2)* based on hunting group consideration or position correction. After the quality of the new hunter position is determined by evaluating the objective function, the hunter moves to this new position; otherwise it keeps its previous position.

Step 5. Reorganizing the hunting group

In order to prevent being trapped in a local optimum, they must reorganize themselves to get another opportunity to find the optimum point. The algorithm does this in two independent conditions. If the difference between the values of objective function for the leader and the worst hunter in the group becomes smaller than a present constant ε_1 and the termination criterion is not satisfied, then the algorithm reorganizes the hunting group for each hunter. Alternatively, after a certain number of searches, the hunters reorganize themselves. The reorganising is carried out as follows: the leader keeps its position and the other hunters randomly choose their position in the design space

$$x'_i = x_i^L \pm rand \times (max(x_i) - min(x_i)) \times \alpha \exp(-\beta \times EN) \tag{17.4}$$

where x'_i is the position value of the leader for the i^{th} variable, *rand* is a uniform random number between [0,1]. x_i^{max} and x_i^{min} are the maximum and minimum possible values of variable x_i, respectively. *EN* counts the number of times that the group has been trapped until this step. As the algorithm goes on, the solution gradually converges to the optimum point. Parameters α and β are positive real values.

Step 6. Repeat Steps 3–5 until the termination criterion is satisfied. In this step, the computations are terminated when the termination criterion is satisfied. If not, Step 3 and Step 5 are then repeated. The termination criterion can be defined as the maximum number of searches. Alternatively, if after reorganizing the function for the leader and the worst hunter in the group remains smaller than a preset constant (ϵ), the search process ends.

17.3 Source code of HuS algorithm in Matlab

The Matlab source code for objective function which will optimize by HuS algorithm is shown below with detail.

```
% initialize the optimization problem and algorithm parameters
    [such as hunting group size (HGS), maximum movement toward the
     leader (MML), and hunting group consideration rate (HGCR)...
HGS=input(‘‘), MML=input(‘‘), HGCR=input(‘‘),    Hmin=input(‘‘),
    Hmax=input(‘‘);
% initialize the hunting group (HG) based on the number of
    hunters (HGS).
HG=zeros(N,HGS)+Hmin(1,:)+rand*(Hmax(1,:)-Hmin(1,:))
% starting main algorithm loop
while (iter<maxiteration);
% number of iteration is increasing by one
iter=iter+1;
% evaluating the hunters by using objective function and
    specifying the leader (HGL).
Eval=OF(HG);
A=[Eval,HG];
B=sort(A,1,'descend');
HGL=B(2,:);
% moving toward the leader.
HG=HG(i,j)+rand*MML*(HGL(1,j)-HG(i.j));
% position correction-cooperation between members.
for i=1:HGS;
   if rand(0,1)<HGCR;
   d=int(HGS*rand);
   for j=0:N;
   HG(i,j)=HG(d,j);
   end
else
   for j=0:N;
   HG(i,j)=HG(i,j)+2*(rand-0.5))*Ramin*(max(HG(i))-min(HG(i)))*exp
     (log(Ramax/Ramin)*iter/maxiter);
```

```
  end
end
end
% reorganizing the hunting group.
tt=iter/HGS*epochiter
for i=0:N;
   HG(i,j)=best(j)+ 2*(rand-0.5))*(max(HG(i))-min(HG(i))*alfa*exp
     (-betha*tt);
end
% calculate the new fitness
Eval=OF(HG);
A=[Eval,HG];
B=sort(A,1,'descend');
NewHGL=B(2,:);
% if NewHGL is better than leader, NewHGL is leader
if NewHGL>HGL;
HGL=NewHGL
end %if
end %for
end %while
% the result of HuS algorithm
disp('HGL')
```

Listing 17.1
Source code of global version of the HuS in Matlab.

17.4 Source code of HuS algorithm in C++

The C++ source code for the objective function which will optimize by HS algorithm is shown in below with detail.

```
//initialize the optimization problem and algorithm parameters [
     such as hunting group size (HGS), maximum movement toward the
     leader (MML), and hunting group consideration rate (HGCR)...
int main()
{
  int HGS, MML, HGCR, Hmin, Hmax;
  cout << "Enter HGS, MML, HGCR, Hmin, Hmax:";
  cin >>HGS >>MML >>HGCR >>Hmin >>Hmax;
}
//initialize the hunting group (HG) based on the number of
     hunters (HGS).
{\bf float} HG[N][HGS];    {\bf float} Eval[N];
{\bf float} HGL[N];        {\bf float} EvalHGLbest[N];
HG[i][j]= Hmin(j)+r()*(Hmax(j)-Hmin(j));
HGL[i][j]=HG[i][j];
//evaluating the hunters by using objective function and
     specifying the leader (HGL).
Eval[i]=OF(HG[i],HGS);
EvalHGLbest=Eval(i);
If (Eval[i]<Eval[Best]) Best=i;
}
//starting main loop
while (iter<maxiteration)
```

```
{
//number of iterations is increasing by one
iter=iter+1;
// moving toward the leader.
HG [i][j]= HG [i][j]+r()*MML*(HGL[j]-HG[j]);
//position correction-cooperation between members.
for (i=1,i<HGS, i++)
{
if (r()<HGCR)
{
d=int(HGS*rand);
for (j=0,j<N, j++);
HG [i][j]=HG[d][j];
end
}
else
{
for (j=0,j<N, j++);
HG[i][j]= HG[i][j]+2*(r() -0.5))*Ramin*(max(HG[i])-min(HG[i]))*exp
      (log(Ramax/Ramin)*iter/maxiter);
}
}
//reorganizing the hunting group.
tt=iter/HGS*epochiter;
for (i=0, i<N, i++)
{
  HG[i][j]=best[j]+ 2*(r() -0.5))*(HGmax[i]-HGmin[i])*alfa*exp(-
      betha*tt);
//calculate the fitness
Eval[i]=OF(HG[i],HGS);
NewEvalHGLbest=Eval(i);
If (Eval[i]<Eval[Best]) Best=i;
{
//if NewEvalHGL is better than leader, NewEvalHGL is leader
if NewEvalHGL>EvalHGL
{
EvalHGL=NewEvalHGL;
}
}
}
//the result of HuS algorithm
count<<EvalHGL;
```

Listing 17.2
Source code of global version of the HuS in C++.

17.5 Elaboration on HuS algorithm with constrained minimization problem

The hunting search algorithm explained in the previous subsections is used to determine the optimum solutions of the Kuhn-Tucker problem. This bench-

mark problem is a minimization with two design variables (x_1 and x_2) and one inequality constraint.
The objective function of the minimization problem is

$$f(x) = 5x_1^2 - 9x_1x_2 + 5x_2^2$$

which is subjected to

$$g(x) = 25 - 16x_1x_2 \leq 0.$$

The variables can be selected from following set of values

$$\{0.5, 1.0, 1.5, 2.0, 2.5, 3.0, \ldots, 10.0\}.$$

The constrained minimum solution is located in a narrow crescent-shaped region. The parameterization of the technique is conducted in Table 17.1 (Step 1) with its recommended settings in preceding studies, as well as extensive numerical experimentations conducted in the present study [1].

TABLE 17.1
HuS parameter values for constrained minimization problem.

Parameters	**Values**
Number of epochs (NE)	2
Iteration per epoch (IE)	30
Hunting Group Size (HGS)	10
Maximum movement toward to leader (MML)	0.3
Hunting group consideration rate (HGCR)	0.3
Ra_max, Ra_min	1e-2, 1e-7
Reorganization parameters-(α, β)	0.1, -1

The HG was initially structured with randomly generated solution vectors within the bounds prescribed for this example (i.e., 0 to 6.0) and the leader is defined (Step 2). After 10 searches the initial hunting group matrix is obtained as given in Table 17.2. The solution vectors are sorted according to the values of the objective function. Next, based on Eq. (17.1), the hunters move toward the leader, and if their new positions are better than the previous positions, they stay there. Otherwise, they come back to their previous positions (Step 3). The 11^{th}, 12^{th} and 13^{th} iteration cannot find a better solution than the ones shown in Table 17.2. The 14^{th} search gives a better hunting group matrix as shown in Table 17.3. A new hunting solution vector was improvised based on followed rules: group considerations with a 30.0% probability and position corrections with a 70.0% probability. As the objective function value of the new

TABLE 17.2
After 10 searches the initial hunting group matrix.

Row Number	$\mathbf{x_1}$	$\mathbf{x_2}$	**f(x)**
1	3.0	4.5	24.75
2	2.0	4.0	28.00
3	7.5	6.0	56.25
4	4.5	7.0	62.75
5	5.5	8.0	75.25
6	8.5	5.5	91.75
7	10.0	9.0	95.00
8	10.0	10.0	100.00
9	5.5	1.0	106.75
10	**1.0**	**9.0**	**329.00**

TABLE 17.3
After 14 searches the hunting group matrix.

Row Number	$\mathbf{x_1}$	$\mathbf{x_2}$	**f(x)**
1	3.0	4.5	24.75
2	2.0	4.0	28.00
3	7.5	6.0	56.25
4	4.5	7.0	62.75
5	5.5	8.0	75.25
6	8.5	5.5	91.75
7	10.0	9.0	95.00
8	**6.0**	**9.0**	**99.00**
9	10.0	10.0	100.00
10	5.5	1.0	106.75

harmony is 99.00, the new harmony is included in the hunting group matrix and the worst candidate is excluded from the matrix, as shown in Table 17.3 (Subsequent HM). After 20 iterations based on Eq. (17.4), the hunters reorganize themselves and improvise an optimal solution (Step 5). After 50 iterations (1000 function evaluations), the HuS algorithm still has the same optimal solution vector $x = (1.5; 1.5)$, which has a function value of 2.25, as shown in Table 17.4 and Figure 17.2. It is noticed that these design vectors remained the same even though the design cycles were continued, reaching 1000, which was the pre-selected maximum number of iterations. Consequently, the technique can be recommended for its application to optimize engineering design problems. It is interesting to see the function we have optimized and we give both a contour plot in Figure 17.2 and three-dimensional plot in Figure 17.3.

TABLE 17.4
After 20 iterations the hunting group matrix.

Row Number	$\mathbf{x_1}$	$\mathbf{x_2}$	**f(x)**
1	**1.5**	**1.5**	**2.25**
2	2.0	1.5	4.25
3	2.5	1.5	6.25
4	3.0	2.5	8.75
5	3.0	3.0	9.00
6	2.0	3.0	11.00
7	3.0	3.5	11.75
8	3.5	3.5	12.25
9	2.5	3.5	13.75
10	3.5	4.0	15.25

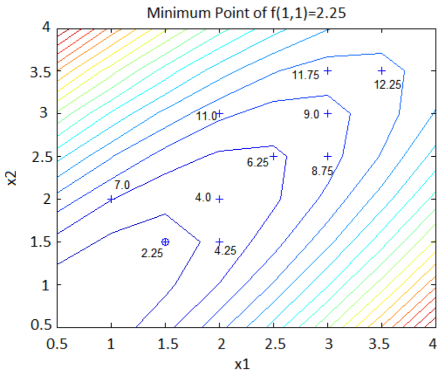

FIGURE 17.2
The contour plot of the Kuhn-Tucker function.

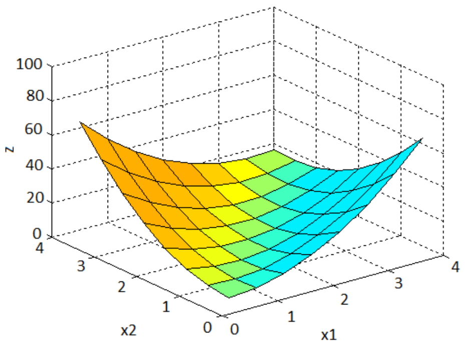

FIGURE 17.3
The three dimensional plot of the Kuhn-Tucker function.

17.6 Conclusion

In this study, besides the basic principles of the hunting search algorithm, operation principles of the algorithm have been explained. Then, source codes of the mentioned algorithm have been provided in Matlab and C++ mathematical programming languages. In the last part of the chapter, numerical example of the hunting search algorithm has been presented with more detail. We strongly claim that this chapter will make the implementation of one's own HuS technique in any programming easier for researchers to achieve.

References

1. R. Oftadeh, M.J. Mahjoob. "A new meta-heuristic optimization algorithm: Hunting Search". *Fifth International Conference on Soft Computing, Computing with Words and Perceptions in System Analysis, Decision and Control*, pp. 1-5, 2009.
2. R. Oftadeh, M.J. Mahjoob, M. Shariatpanahi. "A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search". *Computers and Mathematics with Applications*, vol. 60(7), pp. 2087-2098, 2010.
3. A. Agharghor, M.E. Riffi. "First Adaptation of Hunting Search Algorithm for the Quadratic Assignment Problem". *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pp. 263-267, Springer, 2017.
4. B. Naderi, M. Khalili, A.A. Khamseh. "Mathematical models and a hunting search algorithm for the no-wait flowshop scheduling with parallel machines". *International Journal of Production Research*, vol. 52(9), pp. 2667-2681, 2014.
5. O. Tunca, Erdal F., E. Dogan. "Optimum design of composite corrugated web beams using hunting search algorithm". *International Journal Of Engineering & Applied Sciences*, vol. 9(2), pp. 156-168, 2017.
6. F. Erdal, E. Doğan, O. Tunca, S. Taş. "Optimum Design of Corrugated Web Beams Using Stochastic Search Techniques". *Third International Conference on Advances in Civil, Structural and Environmental Engineering- ACSEE 2015*, pp. 121-125, 2015.