ACOr Examples

Pablo Portillo Garrigues

May 13, 2019

Introduction

The algorithm is a method of heuristic optimization whose objective is to find the global minimum.

The Ant Colony Optimization is based on the behavior of the ants. It is an evolutionary algorithm because it improves iteratively throught generations. Each generation is composed by "ants" and each ant represent a point of the random search space. From generation to generation the results of each ant is compared and each ant is given some weight based on different function. According to the weight that each ant recieves it will serve as a prototype for the next generation.

Algorithm

The algorithm implemented in this paper has the following procedure:

1. Initialize each ant with some random values between some prefixed range.
2. Calculate the errors that each ant have according to the objective function
3. Calculate the weight of each ant throught a variant of the gaussian function
4. Calculate the probability of choosing each ant as a prototype for the next generation
5. Calculate the sigma for each variable of the random search space
6. Calculate the new generation based on the sigma and the probabilty computed before

Initialize the ants

The first thing we need to do is to create randomly the first generation of ants.

This function will recieve the range that each variable can have (paramList) and the number of ants its going to compute (hor).

```
randParam<-function(paramList,hor){
  res<-data.frame(param=numeric())
  for(i in 1:hor){
    for(j in 1:dim(paramList)[2])
      res[i,j]<-runif(1,paramList[1,j],paramList[2,j])
  }
  res
}
```

Calculate the errors

This function will compute the error that each ant (paramList) have based on the objective funcion (costF). The user will indicate through costF his own objective function but it is required that it has as input the parameters datos and paramList. This algorithm has normally been used for meta-parameter tuning of some machine learning algorithms so in **datos** we give the objective function the data we are interested in and in **paramList** the ant (meta-parameters) we want to evaluate. In case the user want to parallelize the computation of the error, we can simply pass the parameter **paralelo** as 1.

```
calcErr<-function(datos,costF,paramList,paralelo){
  res<-data.frame(err=numeric())
  if(paralelo==0){
    coste<-match.fun(costF)
    tParam<-as.data.frame(t(paramList))
    aux<-unlist(lapply(tParam,function(x){coste(datos=datos,paramList=x)}))
    res<-as.data.frame(aux)
    res
  }

  if(paralelo==1){
    library(foreach)
    library(doParallel)
    no_cores <- detectCores() - 1
    cl<-makeCluster(no_cores)
    coste<-match.fun(costF)
    tParam<-as.data.frame(t(paramList))
    cl <- makeCluster(no_cores)
    registerDoParallel(cl)
    aux<-foreach(paramList1=tParam,.combine = c) %dopar% {
      library(caret)
      costF(datos=datos,paramList=paramList1)
    }
    stopCluster(cl)
    res<-as.data.frame(aux)
    res
  }

  res

}
```

Calculate the weights

Once we have computed the errors of eachb ant we need to calculate the weight of each ant. This calculation is based on the gaussian distribution and the formula used is the following:

$$h_i = \frac{1}{q * n * \sqrt{2 * \pi}} * \exp(-\frac{or(h_i)}{2 * (q * n)^2})$$

Where:

- h_i is the ant i whose weith is being computed
- q is a meta-parameter of the algorithm that we will explain later on.
- n is the number of ants
- $or(h_i)$ is the position in increasing order by error that holds the ant being computed

```
pesos<-function(err,q){
  tot<-dim(err)[1]
  res<-data.frame(w=numeric())
  for(i in 1:dim(err)[1]){
    res[i,]<-(1/(q*tot*sqrt(2*pi)))*exp(-(which(err[i,]==err[order(err),])[1]-1)^(2)/(2*(q*tot)^(2)))
```

```
  }
  res
}
```

Calculate the probability

We compute the probability of choosing each ant as a prototype for the next generation

```
probHor<-function(peso){
  res<-peso
  tot<-sum(peso[,1])
  res<-res/tot
  res

}
```

Calculate the sigma for each variable

```
cSigma<-function(paramList,eps){
  hor<-dim(paramList)[1]
  res<-paramList
  for(i in 1:hor){
    for(j in 1:dim(paramList)[2]){
      des<-0
      for(h in 1:hor){
        des<-des+abs(paramList[i,j]-paramList[h,j])
      }
      res[i,j]<-eps*des/(hor-1)
    }
  }

  res
}
```

Compute the new generation

Computing the each new ant of the new generation consist on choosing randomly following the probabilities computed before the ant to be the prototype. The new ant will be computed through the following formula:

$$NH_i = h_p + \sigma_p * r$$

Where: - $NH_{i} is the new new ant. - h_p is the ant chosen randomly following the weights computed before. - $sigma_p$ is the ...

We will multiply the typical deviation of each variable by a randon number. The new variable will be somethi

```
newGen<-function(paramList,desv,prob,paramListR){
  res<-paramList
  hor<-dim(paramList)[1]
  pars<-dim(paramList)[2]
```

```
  rNum<-matrix(abs(rnorm(hor*pars,1,0.1)),ncol=pars)
  auxR<-matrix(sample(c(-1,1),size=hor*pars,replace=TRUE),ncol=pars)
  rNum<-rNum*auxR
  for(i in 1:dim(paramList)[2]){
    idx<-sample(1:hor,size=hor,replace = TRUE,prob=prob[,1])
    res[,i]<-paramList[idx,i]+desv[idx,i]*rNum[,i]
    res[,i]<-ifelse(res[,i]<paramListR[1,i],paramListR[1,i],res[,i])
    res[,i]<-ifelse(res[,i]>paramListR[2,i],paramListR[2,i],res[,i])
  }
  res
}
```

Main function

This function is the one that ensembles all the previous methods. It will also display or save the results of each generation

The inputs are:

- datos is the data we are interested in
- costF is the cost Function defined by the user
- ParamListR are the ranges for each variable
- hor is the number of ants in each generation
- gen is the number of generations to be computed
- tip is the type of variable (number or integer)
- paralelo allow parallel computation
- q is a meta-parameter, with a range of (0,1). If q is close to 0 it will give more relevance to better ants, in the other hand, if its closer to 1 it will distribute the weight between all the ants more equitatively
- eps

```
ACO<-function(datos,costF,paramListR,hor=50,q=0.2,eps=0.5,gen=20,tip,paralelo=0,printIt=1){
  genP<-randParam(paramListR,hor)
  meanErrP<-0.1
  bestHorP<-0.1
  while(gen>0){
    genP[,!( tip %in% "num")]<-round(genP[,!( tip %in% "num")])
    errGen<-calcErr(datos,costF,genP,paralelo=paralelo)
    if(((gen %% printIt) ==0) | (gen == 1)){
        cat(as.character(Sys.time()),append = TRUE,sep = "\n")
        cat(paste("Generation",gen),append = TRUE,sep = "\n")
        cat(paste("Mean error of: ",sum(errGen)/hor),append = TRUE,sep = "\n")
        cat(paste("Mean value for the varaibles are: ",colMeans(genP)),append = TRUE,sep = "\n")
        cat(paste("Best ant's values: ",genP[order(errGen)[1],]),append = TRUE,sep = "\n")
        cat(paste("Best ant's error: ",min(errGen)),append = TRUE,sep = "\n")
    }
    pesosGen<-pesos(errGen,q)
    probGen<-probHor(pesosGen)
    desv<-cSigma(genP,eps)
    genP<-newGen(genP,desv,probGen,paramListR)
    gen<-gen-1
  }
}
```

Example 1

We will try the algorithm on the function of Mishra Bird:

$$f(x_1, x_2) = sin(x_2)exp(1 - cos(x_1))^2 + cos(x_1)exp(1 - sin(x_2))^2 + (x_1 - x_2)^2$$

For the region:

$$-10 < x_1 < 0$$
$$-6.5 < x_2 < 0$$

Its minimum is around

$$f(-3.1302468, -1.5821422) = -106.7645367$$

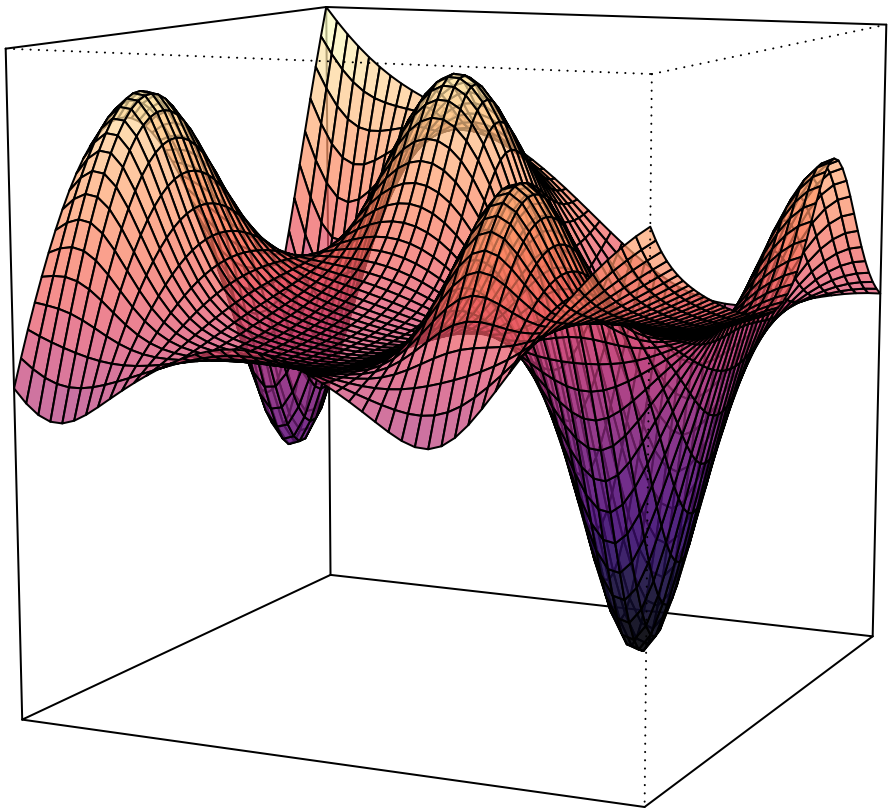
Objective Function

```
costF1 <- function(datos,paramList){
            x1<-paramList[1]
            x2<-paramList[2]
             sin(x2)*exp(1-cos(x1))^2 + cos(x1)*exp(1-sin(x2))^2 + (x1-x2)^2
}
```

Its 3d plot is:

```
## Warning: package 'viridis' was built under R version 3.5.3
```

```
## Loading required package: viridisLite
```



ACO

We will apply the algorithm with 100 ants and 100 generations.

```
set.seed(120)
vars<-data.frame(x1=c(-10,0),x2=c(-6.5,0))
ACO(datos="NA",costF=costF1,paramListR=vars,hor=100,q=0.7,eps=0.5,gen=100,tip=c("num","num"),paralelo=0
```

```
## 2019-05-13 17:45:09
## Generation 100
## Mean error of:  18.0856714431287
## Mean value for the varaibles are:  -4.91058305189945
## Mean value for the varaibles are:  -3.2676522897894
## Best ant's values:  -2.98671490279958
## Best ant's values:  -1.33150260685943
## Best ant's error:  -100.00431855816
## 2019-05-13 17:45:19
## Generation 75
## Mean error of:  -16.3711551512116
## Mean value for the varaibles are:  -3.78695698432542
## Mean value for the varaibles are:  -2.36411105807421
## Best ant's values:  -3.19798613785628
## Best ant's values:  -1.60536429430988
## Best ant's error:  -106.30214205268
## 2019-05-13 17:45:28
## Generation 50
## Mean error of:  -102.040355394839
## Mean value for the varaibles are:  -3.10191921257342
## Mean value for the varaibles are:  -1.56437223149559
## Best ant's values:  -3.12543459556172
## Best ant's values:  -1.5917448440395
## Best ant's error:  -106.78678919365
## 2019-05-13 17:45:37
## Generation 25
## Mean error of:  -106.060248814241
## Mean value for the varaibles are:  -3.11246546180676
## Mean value for the varaibles are:  -1.58804522030038
## Best ant's values:  -3.11917392729778
## Best ant's values:  -1.59003769663862
## Best ant's error:  -106.786583583354
## 2019-05-13 17:45:46
## Generation 1
## Mean error of:  -106.751987770513
## Mean value for the varaibles are:  -3.12432238789448
## Mean value for the varaibles are:  -1.59208222501869
## Best ant's values:  -3.12346919747308
## Best ant's values:  -1.58942244065079
## Best ant's error:  -106.787701909501
```

As we can see, only in 100 generations it reach the minimum of -106.787701909501.

Example 2

We will try the algorithm on the function of Ackley:

$$f(x_1, x_2) = -20exp[0.2\sqrt{0.5(x_1^2 + x_2^2)}] - exp[0.5(cos(2\pi x_1) + cos(2\pi x_1))] + e + 20$$

For the region:

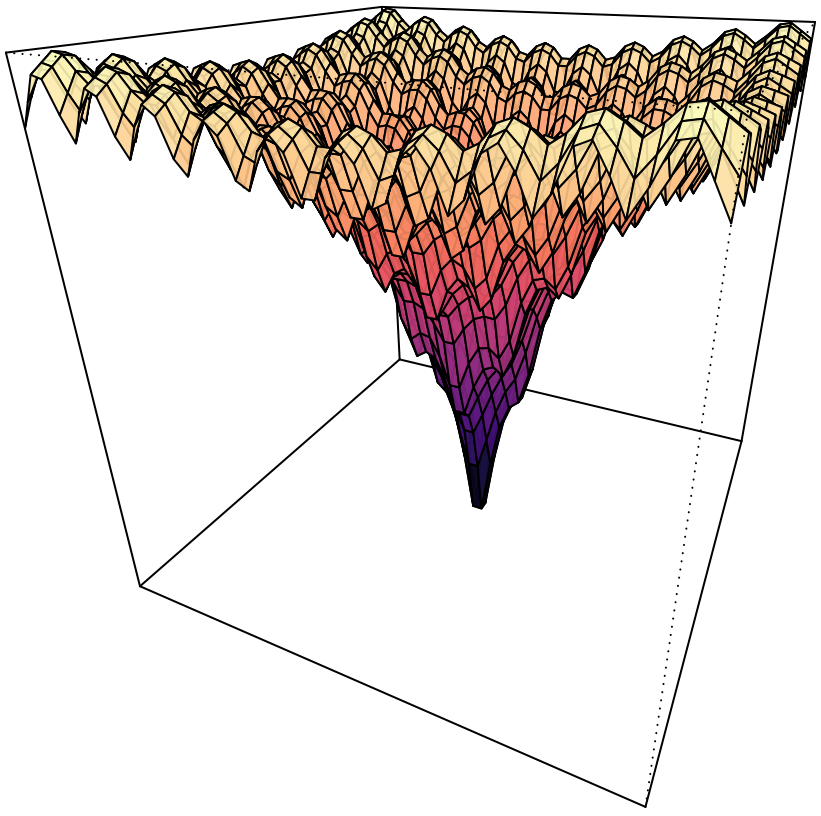
$$-5 < x_1 < 5$$
$$-5 < x_2 < 5$$

Its minimum is around

$$f(0, 0) = 0$$

Objective Function

```
costF2 <- function(datos,paramList){
            x1<-paramList[1]
            x2<-paramList[2]
             -20*exp(-0.7*sqrt(0.5*(x1^2 + x2^2))) - exp(0.5*(cos(2*pi*x1) + cos(2*pi*x2))) + exp(1) + 
}
```



Its 3d plot is:

ACO

We will apply the algorithm with 100 ants and 400 generations.

```
set.seed(120)
vars<-data.frame(x1=c(-5,5),x2=c(-5,5))
ACO(datos="NA",costF=costF2,paramListR=vars,hor=100,q=0.7,eps=0.5,gen=400,tip=c("num","num"),paralelo=0
```

```
## 2019-05-13 17:46:02
## Generation 360
## Mean error of:  1.18892524872626
## Mean value for the varaibles are:  -0.00432734619629633
## Mean value for the varaibles are:  0.000379257894418612
## Best ant's values:  -0.00738781319694598
## Best ant's values:  -0.000657504057863148
## Best ant's error:  0.0747652981222444
## 2019-05-13 17:46:35
## Generation 270
## Mean error of:  0.0032490548467376
## Mean value for the varaibles are:  -4.7037096530352e-06
## Mean value for the varaibles are:  3.27340767518546e-05
## Best ant's values:  4.47651643031571e-05
## Best ant's values:  6.77551013106809e-06
## Best ant's error:  0.000448249806730416
## 2019-05-13 17:47:09
## Generation 180
## Mean error of:  1.5748769368642e-05
## Mean value for the varaibles are:  -3.81423467752652e-07
## Mean value for the varaibles are:  1.41062330447729e-07
## Best ant's values:  -2.86398970016825e-07
## Best ant's values:  -5.36952955455135e-08
## Best ant's error:  2.88460609354502e-06
## 2019-05-13 17:47:42
## Generation 90
## Mean error of:  7.90448352461226e-08
## Mean value for the varaibles are:  4.60342973922218e-10
## Mean value for the varaibles are:  9.7858510224597e-10
## Best ant's values:  -1.22822916661368e-09
## Best ant's values:  -4.03555523472415e-10
## Best ant's error:  1.27983419417887e-08
## 2019-05-13 17:48:15
## Generation 1
## Mean error of:  2.65786468389706e-10
## Mean value for the varaibles are:  -3.56326149502433e-14
## Mean value for the varaibles are:  3.48877164019641e-12
## Best ant's values:  -1.07405670504867e-12
## Best ant's values:  -1.75630042419134e-12
## Best ant's error:  2.03783656615997e-11
```

As we can see in 400 generations the minimum reached is 2.03783656615997e-11