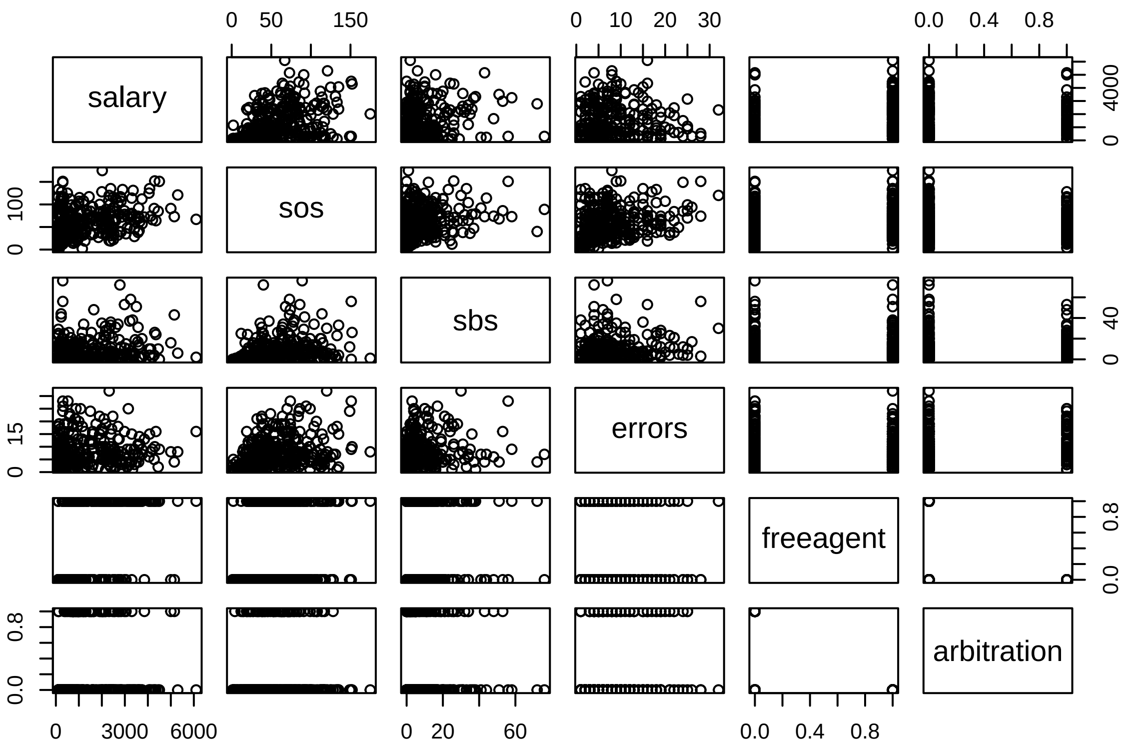
Computational Statistics. Chapter 2: Combinatorial optimization. Solution of exercises

Thierry Denoeux

2024-02-15

Question 1

```
baseball <- read.table("/Users/Thierry/Documents/R/Data/Compstat/baseball.dat",
                       header=TRUE)
attach(baseball)
plot(baseball[,c(1,11:15)])
```



We notice that we cannot plot more than 5-6 variables in a matrix plot.

Question 2

Question 2a

We use function `sample`. The following function simulates p random draws from the set $\{0, 1\}$ with replacement:

```
initialize<-function(p) theta0<-sample(c(0,1),p,replace=TRUE)
```

Question 2b

In the following function, we treat separately the case where all predictors have been removed (i.e., all elements of vector `theta` are equal to zero), in which case we run the linear regression with the intercept only:

```
aic <-function(theta,data){
  if(all(theta==0))  # if all predictors have been removed, we have ony the intercept
    crit<-AIC(lm(salary~0,data=data)) else
    crit<-AIC(lm(salary~.,data=data[,c(1,which(theta==1)+1)]))
  return(crit)
}
```

Question 2c

The following function `neighborhood` generates a matrix of size $p \times p$ containing the p neighbors of the current solution encoded in vector `theta0`:

```
neighborhood <- function(theta0){
    p<-length(theta0)
    Theta<-matrix(theta0,p,p,byrow=TRUE)
    diag(Theta)<-1-diag(Theta)
    return(Theta)
}
```

Question 2d

This is the main function. The arguments are: `fun` (the function to be optimized), `neighbor` (the function that computes the neighborhood of the current solution), the initial solution `theta0`, the maximum number `N` of iterations, the dataset `data` passed to function `fun`, and an argument `disp` with default value `TRUE`; if the argument has the value `FALSE`, intermediate results are not printed. The output is a list with two elements: the minimum of the objective function (`objective`) and the corresponding optimum value of the parameter (`optimum`).

```
local_search <- function(fun,neighbor,theta0,N=1000,data,disp=TRUE){
  p<-length(theta0)
  obj0<-fun(theta0,data)
  go_on<-TRUE
  t <- 0
  while ((t<N) & go_on){
    t<-t+1
    Theta<-neighbor(theta0)
    Obj <- apply(Theta,1,fun,data)
    i_best<-which.min(Obj)
    obj<-Obj[i_best]
    if (obj>=obj0){   # solution has not improved
      go_on <- FALSE # stop
    } else{   # solution has improved
      theta0<-Theta[i_best,]
      obj0<-obj
    }
```

```
    if(disp) print(c(t,obj0))
  } # end while
  return(list(objective=obj0,optimum=theta0))
}
```

Let us now use this function with a random search strategy, running it 50 times from 50 random starting points, and keeping the best solution:

```
p<-ncol(baseball)-1
M<-50
AICbest<-Inf
for(i in 1:M){
  theta0<-initialize(p)
  opt<-local_search(aic,neighborhood,theta0,100,baseball,disp=FALSE)
  if(opt$objective<AICbest){
    opt_best<-opt
    AICbest<-opt$objective
  }
}
```

We print the AIC value and names of the predictors for the best model:

```
print(opt_best$objective)
```

```
## [1] 5375.362
```

```
Names<-names(baseball)
var_best<-which(opt_best$optimum==1)
print(Names[var_best+1])
```

```
## [1] "homeruns"    "rbis"        "walks"       "sos"         "freeagent"
## [6] "arbitration" "walksperso"  "sbsobp"
```

Question 2e

To generate the 2-neighborhood, we generate the 1-neighborhood of each neighbor in the 1 neighborhood. We must be careful to remove the initial solution at each stage, and to remove duplicate solutions:

```
neighborhood2 <- function(theta0){
  p <- length(theta0)
  Theta <- neighborhood(theta0)
  for(i in 1:p){
    Theta1 <- neighborhood(Theta[i,])
    Theta <- rbind(Theta,Theta1[-i,]) # remove initial solution
  }
  Theta <- unique(Theta,MARGIN=1) # remove duplicates
  return(Theta)
}
```

Let us test the local search algorithm with one- and two-neighborhood, starting from the same random initial solution:

```
set.seed(20240214)
ptm <- proc.time()
opt<-local_search(aic,neighborhood,theta0,100,baseball)
```

```
## [1]    1.000 5396.884
```

```
## [1]    2.000 5391.291
## [1]    3.000 5389.072
## [1]    4.000 5387.124
## [1]    5.000 5385.437
## [1]    6.000 5383.974
## [1]    7.000 5382.758
## [1]    8.000 5381.661
## [1]    9.000 5380.881
## [1]   10.000 5379.374
## [1]   11.000 5378.396
## [1]   12.000 5376.702
## [1]   13.00 5376.51
## [1]   14.000 5375.956
## [1]   15.000 5375.956
```

```
time1 <- proc.time() - ptm
print(which(opt$optimum==1))
```

```
##  [1]  3  4  8 10 11 13 14 15 16 17 18 19 24 25
```

```
ptm <- proc.time()
opt<-local_search(aic,neighborhood2,theta0,100,baseball)
```

```
## [1]    1.000 5391.291
## [1]    2.000 5387.124
## [1]    3.000 5383.974
## [1]    4.000 5381.661
## [1]    5.000 5379.374
## [1]    6.000 5376.702
## [1]    7.000 5375.956
## [1]    8.00 5375.85
## [1]    9.00 5375.85
```

```
time2 <- proc.time() - ptm
print(which(opt$optimum==1))
```

```
##  [1]  3  4  8 10 11 12 13 14 15 16 17 18 19 25
```

```
print(rbind(time1,time2))
```

```
##       user.self sys.self elapsed user.child sys.child
## time1     0.843    0.007   0.852          0         0
## time2     7.579    0.101   7.973          0         0
```

In this case, using the 2-neighborhood allowed us to reach a slightly better solution, at the expense of a much higher computing time. We also observe that local search with the 2-neighborhood converges in a smaller number of steps. The results depend on the initialization, but this trend is generally observed.

Question 3

Question 3a

This function returns one randomly selected element in the neighborhood of `theta0`:

```
new <- function(theta0){
  p<- length(theta0)
```

```
  i<-sample(p,1)
  theta <- theta0
  theta[i]<-1-theta[i]
  return(theta)
}
```

Question 3b

The arguments of function `simulated_annealing` below are: `fun` (the function to be optimized), `new` (the function that returns a random candidate solution), the initial solution `theta0`, the initial temperature `tau0`, the initial stage length `m0`, parameters `a` and `b` of the cooling schedule, the minimum temperature `taumin` used as a stopping criterion, the dataset `data` passed to function `fun`, and the trace flag `disp`. The output is a list with four elements: the minimum of the objective function (`objective`), the corresponding optimum value of the parameter (`optimum`), and two vectors: the trace of the error function (`Obj`) and the temperature (`Tau`).

```
simulated_annealing <-function(fun,new,theta0,tau0=10,m0=10,a=0.9,
                               b=10,taumin=0.01,data,disp=TRUE){
  p<-length(theta0)
  obj0<-fun(theta0,data)
  tau<-tau0
  m<-m0
  Tau<-tau0
  Obj<-obj0
  while (tau>taumin){
    for(t in 1:m){
      theta<-new(theta0)
      obj<-fun(theta,data)
      proba<-min(1,exp((obj0-obj)/tau))
      if(runif(1)<proba){
        theta0<-theta
        obj0<-obj
      }
      Tau<-c(Tau,tau)
      Obj<-c(Obj,obj0)
    } # end for
    tau<-tau*a
    m<-m+b
    if(disp) print(c(tau,m,obj0))
  } # end while
  return(list(objective=obj0,optimum=theta0,Obj=Obj,Tau=Tau))
}
```

Question 3c

We now run this function after giving values to the parameters:

```
tau0<-10
m0<-10
a<-0.9
b<-10
taumin<-0.1
```

```r
opt <- simulated_annealing(fun=aic,new=new,theta0,tau0,m0,a,b,taumin,
                           data=baseball,disp=FALSE)
```

We print the final value of the error function as well as the selected predictors:

```r
print(opt$objective)
```

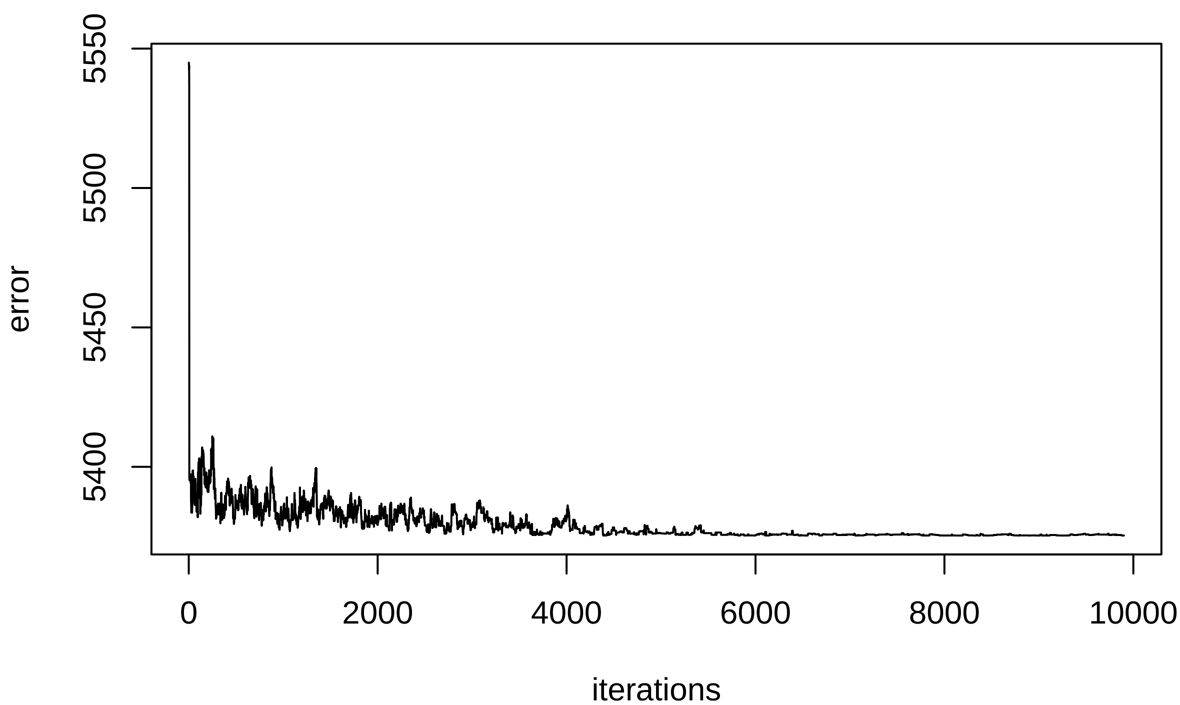
```
## [1] 5375.362
```

```r
print(Names[which(opt$optimum==1)+1])
```

```
## [1] "homeruns"    "rbis"         "walks"        "sos"          "freeagent"
## [6] "arbitration" "walksperso"  "sbsobp"
```

We draw the trace of the error and the temperature vs. the number of iterations:

```r
plot(opt$Obj,type="l",xlab="iterations",ylab="error")
```




```r
plot(opt$Tau,type="l",xlab="iterations",ylab="temperature")
```