Assignment 4

Lorenzo Pastore

7/6/2019

Problem Formulation

The objective of the assignment is to apply different resolution methods to a non linear combina- torial problem. The problem that we will deal with is the Flow shop scheduling problem in which we have n machines and m jobs.

- Each job must be processed in all the machines following the same order, that is the i-th operation of the job must be executed on the i-th machine.
- We assume that machines can not work on more than one process simultaneously and that for each job the operation time in each machine is different and known at the beginning.
- You can assume assume that the order in which jobs are processed is exactly the same for all the machines.

The objective of the problem is to find an ordering of the jobs that minimizes total job execution time (also called makespan) that corresponds to the time on which all jobs get processed.

$$min(Makespan) = min(C_{max}) = min \sum_{i}^{m} \sum_{j}^{n} Pt_{i,j}$$

Loading packages required

```
library(optimx)
library(base)
library(Hmisc)
```

```
## Loading required package: lattice
```

```
## Loading required package: survival
```

```
## Loading required package: Formula
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'Hmisc'
```

```
## The following objects are masked from 'package:base':
##
##     format.pval, units
```

```
library(curry)
library(ggplot2)
```

```
read.Taillard <- function(case){
  url <- paste0("http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/tai",
  text <- readLines(url)
  text.split <- strsplit(text, " ")
  clean <- lapply(text.split, function(x) x[which(nchar(x)!=0)])

  lines <- length(clean)
  k <- 1
  num.instance <- 1
```

```
  instances <- list()

  while(k < lines){

    refs <- as.numeric(clean[[k+1]])
    n <- refs[1]
    m <- refs[2]
    seed <- refs[3]
    upper <- refs[4]
    lower <- refs[5]

    tij <- numeric(0)

    for(i in (k+3):(k+2+m)) tij <- c(tij, as.numeric(clean[[i]]))

    tij <- matrix(tij, m, n, byrow=TRUE)

    instances[[num.instance]] <- list(m=m, n=n, seed=seed, upper=upper, lower=lower, tij=tij)

    num.instance <- num.instance+1
    k <- k + m  + 3
  }
  return(instances)
}
tai_instances_list<- function(Tai){
  n <- length(Tai)
  t1<-Tai[[1]]$tij
  t2<-Tai[[2]]$tij
  t3<-Tai[[3]]$tij
  t4<-Tai[[4]]$tij
  t5<-Tai[[5]]$tij
  t6<-Tai[[6]]$tij
  t7<-Tai[[7]]$tij
  t8<-Tai[[8]]$tij
  t9<-Tai[[9]]$tij
  t10<-Tai[[10]]$tij
  test <- list(t1,t2,t3,t4,t5,t6,t7,t8,t9,t10)
  return(test)
}
```

For time and convenience reason I decided to use only two of the sets of instances, each one containg 10 processing times matrix. In particular I've picked one small set (tai20x5) and one bigger set (tai100x5)

```
tai20.5 <- read.Taillard("20_5")
tai20.5 <- tai_instances_list(tai20.5)
tai100.5 <- read.Taillard("100_5")
tai100.5 <- tai_instances_list(tai100.5)
```

As a start I defined the generic makespan function, we decided to use as objective function, and a swap move function. These will help us to implement different algorithm both in the Heuristic and the Meta-heuristic field.

```
#-makespan function
makespan <- function(matrix, seq){
  m <- dim(matrix)[1]
```

```
  n <- length(seq)
  times <- matrix(as.numeric(), m, n)
  matrix<- matrix[,seq]
  times[1,1] <- matrix[1,1]
  for(i in 2:m) times[i,1] <- times[i-1,1] + matrix[i,1]
  for(j in 2:n) times[1,j] <- times[1,j-1] + matrix[1,j]
  for(i in 2:m){
    for(j in 2:n)
      times[i,j] <- max(times[i-1,j], times[i, j-1]) + matrix[i,j]
  }
  result <- times[m,n]
  return(result)
}

#-swap function
swap <- function(v,i,j){
    aux <- v[i]
    v[i] <- v[j]
    v[j] <- aux
    return(v)
}
```

Sub-optimal Solutions

In the next section a bunch of algorithms will be implemented. Each algorithm proposed will be runned on each of 10 instances of the two choosen sets (tai20.5; tai100.5).

The fist solution is given by the Nawaz-Enscore-Ham (NEH) heuristic which is, according to some academic literature, a good starting point for other local search techniques. In particular the optimal NEH sequence is used as starting sequence for the Hill Climbing Algorithm, the Simulated Annealing Algorithm and a second SA algorithm implemented using R package optimx.

NEH algorithm

```
NEH <- function(Instance){
  ptm <- proc.time()
  n <- dim(Instance)[2]
  jobs <- order(apply(Instance, 2, sum), decreasing = TRUE)
  if( makespan(Instance, c(jobs[1], jobs[2])) < makespan(Instance, c(jobs[2], jobs[1])) )
    sol <- c(jobs[1], jobs[2])
  else
    sol <-c(jobs[2], jobs[1])

  for(k in 3:n){
    fit <- Inf
    pos <- -1
    for(i in 1:k){
      if(i==1) test <- c(jobs[k], sol)
      if(i!= 1 & i!=k) test <- c(sol[1:(i-1)], jobs[k], sol[i:(k-1)])
      if(i==k) test <- c(sol, jobs[k])
```

```
      testfit <- makespan(Instance, test)

      if(testfit < fit){
        fit <- testfit
        pos <- i
      }
    }
    if(pos==1) sol <- c(jobs[k], sol)
    if(pos!= 1 & pos!=k) sol <- c(sol[1:(pos-1)], jobs[k], sol[pos:(k-1)])
    if(pos==k) sol <- c(sol, jobs[k])
  }
  fit <- makespan(Instance, sol)
  return(list(sol=sol, fit=fit,time = proc.time() - ptm))
}


NEH_instances <- function(instances, seq){
  n <- length(instances)
  d <- dim(instances[[1]])[2]
  NEH_res <- vector("list",n)
  NEH_result <- list(NEH_res,n)
  fit = matrix(NA,10,1)
  sol = matrix(NA,10,d)
  time = matrix(NA,10,5)
    for (i in 1:n) {
      NEH_result[[i]] <- NEH(instances[[i]])
      fit[i] <- NEH_result[[i]]$fit
      sol[i,] <- NEH_result[[i]]$sol
      time[i,] <- NEH_result[[i]]$time
    }
  x <- c(seq(1:d),"makespan NEH","time NEH")
  time <- as.data.frame(time[,3])
  res <- as.data.frame(cbind(sol,fit,time))
  colnames(res) <-x
  return(res)
}
```

```
set.seed(1029)
seq <- 1:20
seq2 <- 1:100

NEH20.5 <- NEH_instances(tai20.5,seq)
NEH100.5 <- NEH_instances(tai100.5,seq2)
```

Hill Climbing Algorithm

```
HillClimbing <- function(matrix, seq){
  ptm <- proc.time()
  n <- length(seq)
  sol <- seq
  obj <- makespan(matrix, sol)
  k <- TRUE
```

```
  while(k){
    soltest <- numeric(n)
    objtest <- Inf
    for(i in 1:(n-1)){
      for(j in (i+1):n){
        test <- makespan(matrix, swap(sol,i,j))
        if(test < objtest){
          objtest <- test
          soltest <- swap(sol,i,j)
        }
      }
    }
    #comparing the obtained solution with the obtained in previous iteration
    if(objtest < obj){
      obj <- objtest
      sol <- soltest
    }
    else
      k <- FALSE
  }
  return(list(sol=sol, obj=obj, time = proc.time() - ptm))
}

HC_instances <- function(instances, seq){
  n <- length(instances)
  d <- dim(instances[[1]])[2]
  HC_res <- vector("list",n)
  HC_result <- list(HC_res,n)
  obj = matrix(NA,10,1)
  sol = matrix(NA,10,d)
  time = matrix(NA,10,5)
    for (i in 1:n) {
      HC_result[[i]] <- HillClimbing(instances[[i]], seq)
      obj[i] <- HC_result[[i]]$obj
      sol[i,] <- HC_result[[i]]$sol
      time[i,] <- HC_result[[i]]$time
    }
  x <- c(seq(1:d),"makespan HC","time HC")
  time <- as.data.frame(time[,3])
  res <- as.data.frame(cbind(sol,obj,time))
  colnames(res) <-x
  return(res)
}
```

```
set.seed(1029)
seq <- as.numeric(NEH20.5[10,1:20])
seq2 <- as.numeric(NEH100.5[6,1:100])

HC20.5 <- HC_instances(tai20.5,seq)
HC100.5 <- HC_instances(tai100.5,seq2)

#summary(HC100.5[,101:102])
```

Simulated Annealing Algorithm

We implement a Simulated Annealing algorithm (SA) based on the example seen at lesson for the Flow-shop problem and integrate the algorithm in a function to apply the (SA) algorithm to all of set's instances.

```
SA <- function(tour, distMatrix, maxIterNoChange=200){
  ptm <- proc.time()
  path <- tour
  n <- length(path)
  tmin <- 0.01 # minimum temperature
  alpha <- 0.999 # update factor
  T <- tini <- 100 # starting temperature
  dist <- makespan(distMatrix, path) # objective function
  bestLength <- dist
  traceBest <- c(dist)
  traceCurrentLength <- c(dist)
  iterNoChange = 0
  while(T >= tmin){ # if the temperature is not at its minimum
    iterNoChange = iterNoChange+1
    #swap
    pair <- sample.int(n,2)
    nmin <- min(pair)
    nmax <- max(pair)
    newpath <- path
    newpath[nmin] <- path[nmax]
    newpath[nmax] <- path[nmin]
    dist_new <- makespan(distMatrix, newpath)

    if(dist_new <= bestLength){
      path <- newpath
      dist <- dist_new
      bestLength <- dist
      iterNoChange <- 0 #print("improve")
  }
  else {
    if (exp((dist-dist_new)/T)>runif(1, 0, 1)){
      dist <- dist_new
      path <- newpath
      iterNoChange <- 0
    }
  }
  traceBest <- append(traceBest, bestLength)
  traceCurrentLength <- append(traceCurrentLength, dist)
  T <- T*alpha
  if(iterNoChange >= maxIterNoChange){ break}
}
res = list(route=path, traceBest = tail(traceBest,1),trace = traceCurrentLength, time = proc.time() - p
class(res) = "SAObj"
return(res)
}

SA_instances <- function(instances, seq){
  n <- length(instances)
  d <- dim(instances[[1]])[2]
```

```
  SA_res <- vector("list",n)
  SA_result <- list(SA_res,n)
  obj = matrix(NA,10,1)
  sol = matrix(NA,10,d)
  time = matrix(NA,10,5)
    for (i in 1:n) {
      SA_result[[i]] <- SA(seq, distMatrix = instances[[i]], maxIterNoChange = 100)
      obj[i] <- SA_result[[i]]$traceBest
      sol[i,] <- SA_result[[i]]$route
      time[i,] <- SA_result[[i]]$time
    }
  x <- c(seq(1:d),"makespan SA","time SA")
  time <- as.data.frame(time[,3])
  res <- as.data.frame(cbind(sol,obj,time))
  colnames(res) <-x
  return(res)
}
```

Looking for a solution

```
set.seed(1029)
seq <- as.numeric(NEH20.5[10,1:20])
seq2 <- as.numeric(NEH100.5[6,1:100])

SA20.5 <- SA_instances(tai20.5,seq)
SA100.5 <- SA_instances(tai100.5,seq2)

#summary(SA100.5[,101:102])
```

Visualizing the SA progression and the SA results.

R Algorithm Function

implementing Simulated Annealing with package Optimx

```
swap1 <- function(path){
  n <- length(path)
  pair <- sample.int(n,2)
  nmin <- min(pair)
  nmax <- max(pair)
  newpath <- path
  newpath[nmin] <- path[nmax]
  newpath[nmax] <- path[nmin]
  return(as.numeric(newpath))
}

# this function will help us to apply the R (SA) optim function to all instances
SAR_instances <- function(instances,seq){
  f <- length(instances)
  d <-dim(instances[[1]])[2]
  SAR_res <- vector("list",f)
  SAR_result <- list(SAR_res,f)
  obj = matrix(NA,10,1)
  sol = matrix(NA,10,d)
```

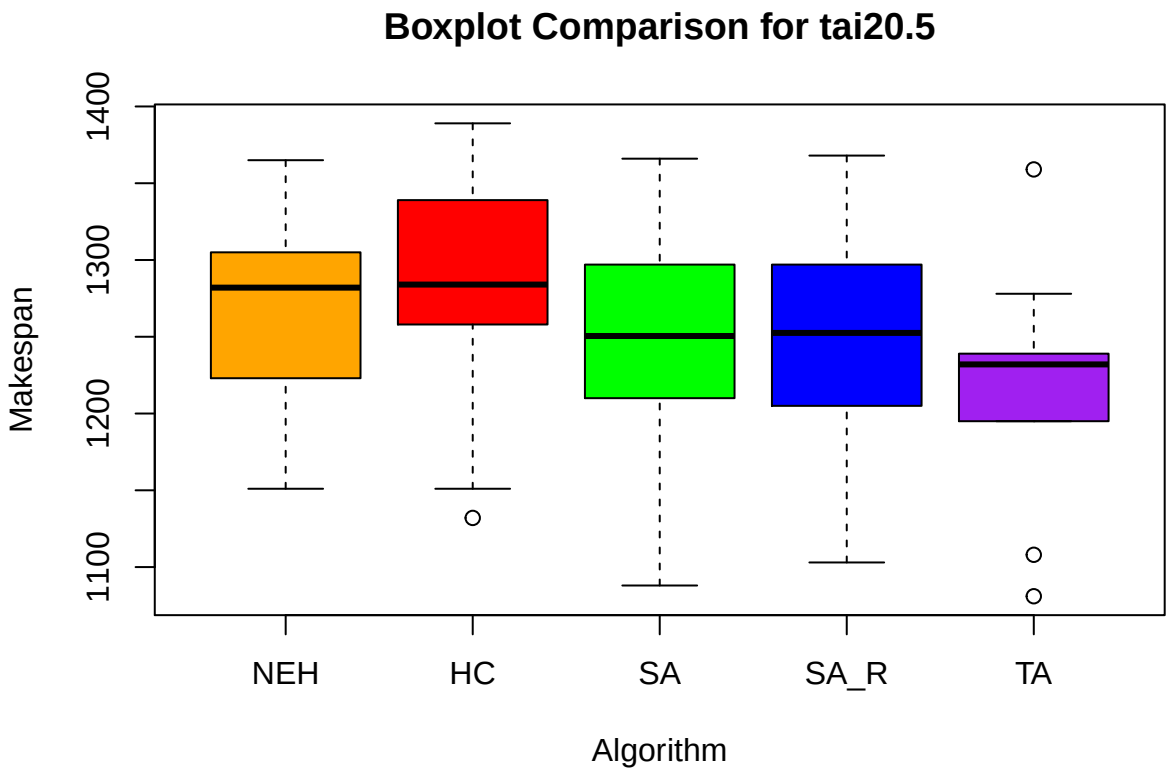
```
  time = matrix(NA,10,1)
  for (k in 1:f){
    start_time <- Sys.time()
    fnObj <- function(seq){
    m <- dim(instances[[1]])[1]
    n <- dim(instances[[1]])[2]
    times <- matrix(as.numeric(), m, n)
    matrix <- instances[[k]][,seq]
    times[1,1] <- matrix[1,1]
    for(i in 2:m) times[i,1] <- times[i-1,1] + matrix[i,1]
    for(j in 2:n) times[1,j] <- times[1,j-1] + matrix[1,j]
    for(i in 2:m){
      for(j in 2:n)
        times[i,j] <- max(times[i-1,j], times[i, j-1]) + matrix[i,j]
    }
   result <- times[m,n]
   return(result)
    }
  SAR_result[[k]] <- optim(seq, fnObj, gr= swap1, method = "SANN", control = list(maxit=10000, temp=100
  end_time <- Sys.time()
  time[k] = end_time -start_time
  obj[k] <- SAR_result[[k]]$value
  sol[k,] <- SAR_result[[k]]$par
  }
  x <- c(seq(1:d),"makespan","time SA-R")
  time <- as.data.frame(time)
  res <- as.data.frame(cbind(sol,obj,time))
  colnames(res) <- x
  return(res)
}
```

```
set.seed(1029)

SAR20.5 <- SAR_instances(tai20.5,seq)
SAR100.5 <- SAR_instances(tai100.5,seq2)
```

Results Comparison

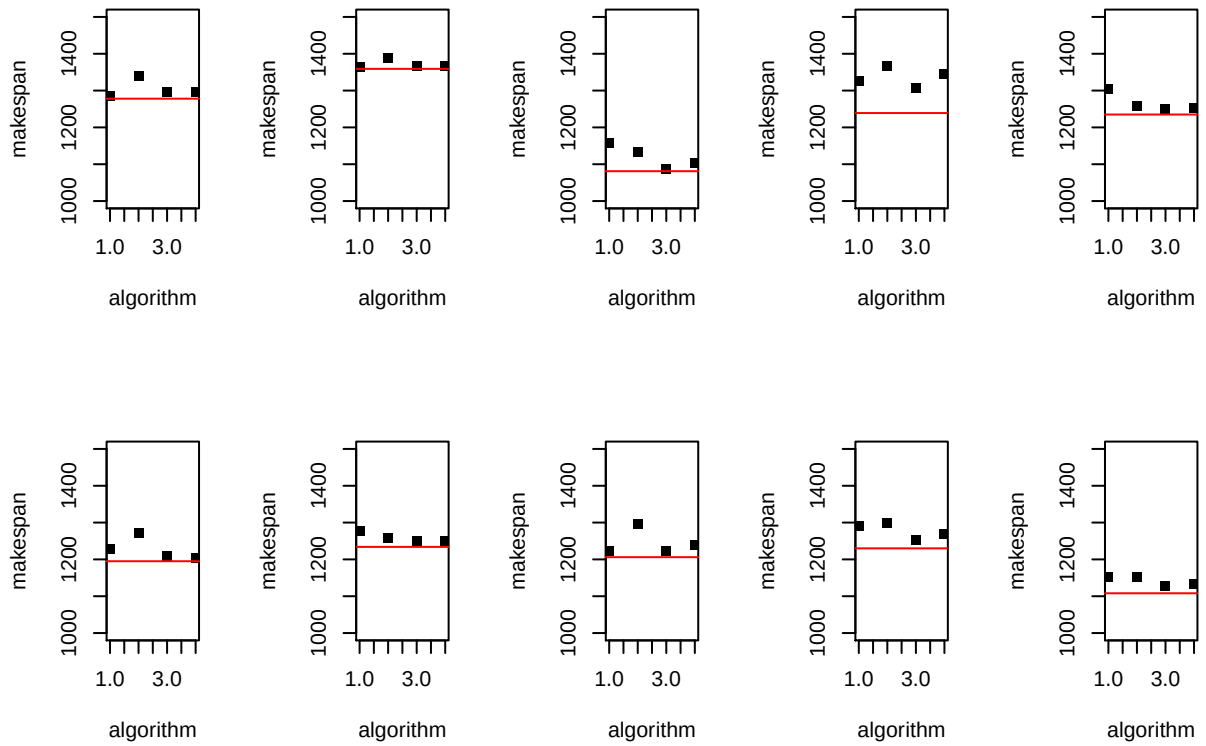
As requested by the assignment out task is also to compare the obtained results against the best known solution and the solution obtained with an R package

```
res <- c(1278,1359,1081,1239,1235,1195,1234,1206,1230,1108)

Comp <- cbind(NEH20.5[,21],HC20.5[,21],SA20.5[,21],SAR20.5[,21],res)
colnames(Comp) <- c("NEH","HC","SA","SA_R","TA")
boxplot(Comp,
        main = "Boxplot Comparison for tai20.5",
        xlab = "Algorithm",
        col = c("orange","red","green","blue","purple"),
        ylab = "Makespan",
        horizontal = F)
```



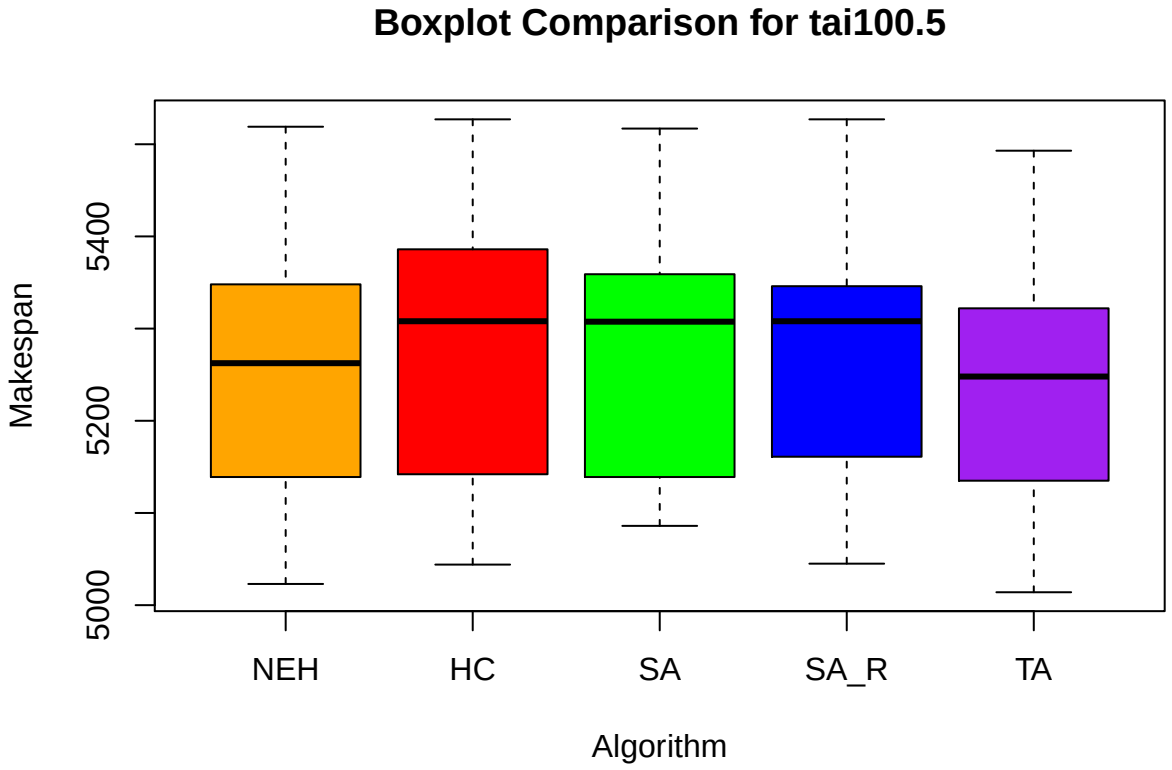
As we expected from this graph is possible to state that all of our algorithms have on average worst result respect to the set of best known solutions values. Although this graph does not tell us much more about the single algorithm performances on each instance, it is interesting to notice how the SA performances are on average slightly better than SA_R, which is the Optim R package Simulated Annealing function.

```
par(mfrow=c(2,5))
for (i in 1:10){
  plot(Comp[i,1:4],
  xlab = "algorithm",
  ylab = "makespan",
  pch = 15,
  xlim = c(1,4),
  ylim = c(1000,1500))
  abline(h=Comp[i,5], col= "red")
}
```



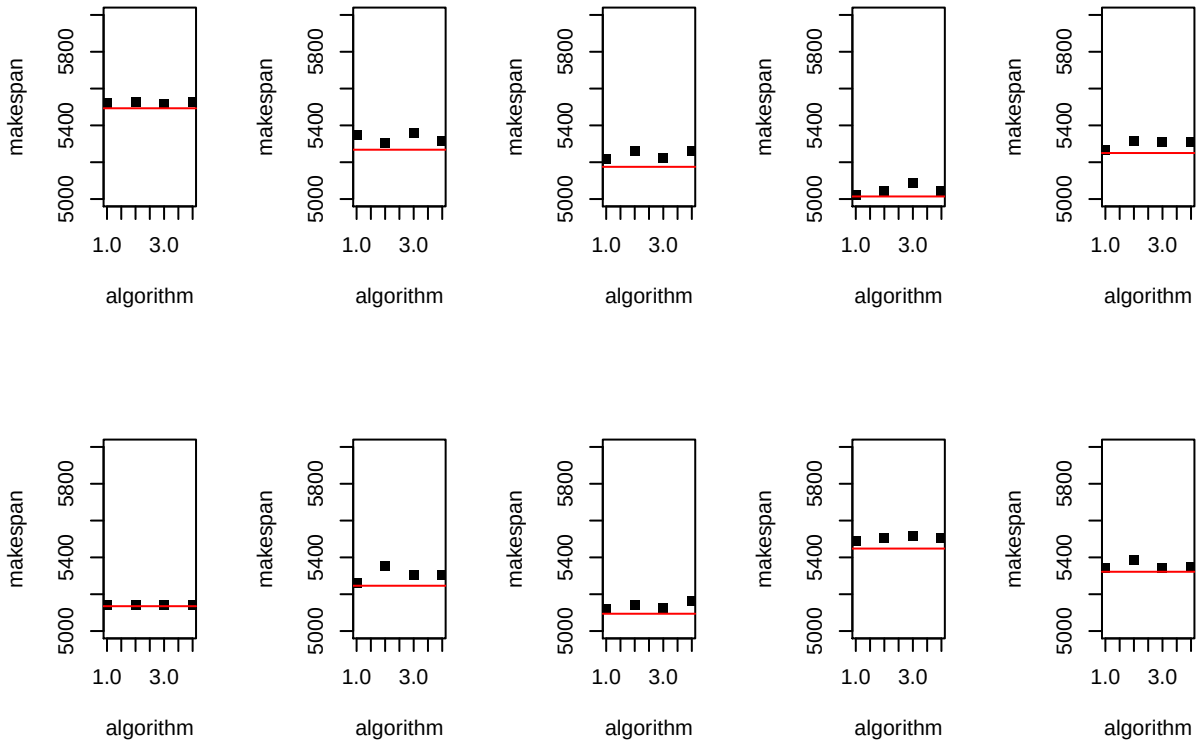
The graph above gives us a much better overview of the performances of all algorithm on each instance of tai20.5. In detail are represented the solution values for NEH, HillClimbing, SA, SA-R (from left to right) while the best known solution value is represented by a red line.

```
res1 <- c(5493,5268,5175,5014,5250,5135,5246,5094,5448,5322)

Comp100.5 <- cbind(NEH100.5[,101],HC100.5[,101],SA100.5[,101],SAR100.5[,101],res1)
colnames(Comp100.5) <- c("NEH","HC","SA","SA_R","TA")
boxplot(Comp100.5,
        main = "Boxplot Comparison for tai100.5",
        xlab = "Algorithm",
        col = c("orange","red","green","blue","purple"),
        ylab = "Makespan",
        horizontal = F)
```



Also in this case our expectation to find on average worst results is meet. By the way it highligths some differences respect to the boxplot graph of 20x5 flow shop problem, in fact we notice that SA performances are still slightly better than SA-R, but this time HC is performing on the same average values and even more surprisingly the NEH algorithm is giving on average better performances than all these three.

```
par(mfrow=c(2,5))
for (i in 1:10){
  plot(Comp100.5[i,1:4],
  xlab = "algorithm",
  ylab = "makespan",
  pch = 15,
  xlim = c(1,4),
  ylim = c(5000,6000))
  abline(h=Comp100.5[i,5], col= "red")
}
```


makespan
5000
5400
5800
1.0
3.0
algorithm