Assignment 4

Pranav Kasela 846965

```
require(curry)
require(GA)
require(ggplot2)
require(reshape2)
require(dplyr)
require(Rcpp)
```

Initially the function that were being using were the written in R, but one of our colleague Federico Moiraghi, was kind enough to provide his code compiled in Cpp for the time function to speed up the process. Some parameters are modified in the function before reusing his code.

```
time <- function(perm,distMatrix){
  n_jobs     <- ncol(distMatrix)
  n_machines <- nrow(distMatrix)
  dist       <- matrix(NA, nrow=n_machines, ncol=n_jobs)
  dist[1,]   <- cumsum(distMatrix[1,perm])
  dist[,1]   <- cumsum(distMatrix[,perm[1]])
  for (i in 2:n_machines){
    for (j in 2:n_jobs){
      dist[i,j] <- distMatrix[i,perm[j]] +
        max(dist[i,j-1],dist[i-1,j])
    }
  }
  makespan   <- dist[n_machines,n_jobs]
  return(makespan)
}

fitness <- function(perm,distMatrix){
  return(1/time(perm,distMatrix))
}
```

From hereon the time function used will be timeCpp, while the fitness code will be fitnessCpp.

```
cppFunction('double fitnessCpp(NumericVector perm,
                           NumericMatrix distMatrix)
    {
        int nrow = distMatrix.nrow();
        int ncol = distMatrix.ncol();
        int norder = perm.size();
        NumericVector order(norder);
        for (int i = 0; i < norder; i++) order[i] = perm[i]-1;
        NumericMatrix time_matrix(nrow, norder);
        time_matrix[0] = distMatrix[nrow * order[0]];
        for (int r = 1; r < nrow; r++)
            time_matrix[r] = time_matrix[r - 1] +
            distMatrix[nrow * order[0] + r];
        for (int c = 1; c < norder; c++)
            time_matrix[nrow * c] = time_matrix[nrow * (c - 1)] +
            distMatrix[nrow * order[c]];
        for (int r = 1; r < nrow; r++)
```

```
            for (int c = 1; c < norder; c++)
                if (time_matrix[nrow * c + (r - 1)] > time_matrix[nrow * (c - 1) + r])
                    time_matrix[nrow * c + r] = time_matrix[nrow * c + (r - 1)] +
            distMatrix[nrow * order[c] + r];
        else
            time_matrix[nrow * c + r] = time_matrix[nrow * (c - 1) + r] +
            distMatrix[nrow * order[c] + r];
        return 1/time_matrix[nrow * norder - 1];
    }')

cppFunction('double timeCpp(NumericVector perm,
                            NumericMatrix distMatrix)
    {
        int nrow = distMatrix.nrow();
        int ncol = distMatrix.ncol();
        int norder = perm.size();
        NumericVector order(norder);
        for (int i = 0; i < norder; i++) order[i] = perm[i]-1;
        NumericMatrix time_matrix(nrow, norder);
        time_matrix[0] = distMatrix[nrow * order[0]];
        for (int r = 1; r < nrow; r++)
            time_matrix[r] = time_matrix[r - 1] +
            distMatrix[nrow * order[0] + r];
        for (int c = 1; c < norder; c++)
            time_matrix[nrow * c] = time_matrix[nrow * (c - 1)] +
            distMatrix[nrow * order[c]];
        for (int r = 1; r < nrow; r++)
            for (int c = 1; c < norder; c++)
                if (time_matrix[nrow * c + (r - 1)] > time_matrix[nrow * (c - 1) + r])
                    time_matrix[nrow * c + r] = time_matrix[nrow * c + (r - 1)] +
            distMatrix[nrow * order[c] + r];
        else
            time_matrix[nrow * c + r] = time_matrix[nrow * (c - 1) + r] +
            distMatrix[nrow * order[c] + r];
        return time_matrix[nrow * norder - 1];
    }')
```

Here the testing of the functions are done with the smallest table. The algorithm used are the Genetic Algorithm and the Simulated Annealing to check their performance.

```
time_matrix <- as.matrix(read.csv("j20-m5",sep=" ",header = FALSE))
n_jobs <- ncol(time_matrix)


time_taken <- microbenchmark::microbenchmark(
  GA.fit <- ga(type = "permutation",
    fitness = fitnessCpp,
    distMatrix = time_matrix,
    lower = 1,
    upper = n_jobs,
    popSize = 600,
    maxiter = 10000,
    run = 200,
    pmutation = 0.2,
```

```
    keepBest = TRUE,
    monitor = NULL,
    seed = 1234),
times = 1
)

summary(GA.fit)
```

```
## -- Genetic Algorithm -------------------
##
## GA settings:
## Type                  =  permutation
## Population size       =  600
## Number of generations =  10000
## Elitism               =  30
## Crossover probability =  0.8
## Mutation probability  =  0.2
##
## GA results:
## Iterations             = 259
## Fitness function value = 0.00077101
## Solutions =
##       x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x19 x20
## [1,]  15  6 11 14  9  5  8  7 17   1        10  20
## [2,]  14 19 17  8  5 11 15 13  7  16        10  20
## [3,]  15  6 14  9  1  5  8 17 19  16        12  20
## [4,]  15  6  1 11  9 14 16  8  5  19        12  20
## [5,]  15  6 11 14  5  8  7 17  9   1        10  20
## [6,]  15  1 11  3 19 14  9  6  4  17        10  20
## [7,]  17  8 19 14  5 11 15 13  7  16        10  20
## [8,]  15  6 11  9 14  5  8  7 17   1        10  20
## [9,]  15  6 11  1  9 14 19 17 13  16        10  20
## [10,] 15  6 11 14  5  8  7 17  1  19        10  20
##  ...
## [33,] 15  6 11  9 14 16 13  3  4  17        10  20
## [34,] 15  6  1  9  5 14 17  8  7  11        12  20
```

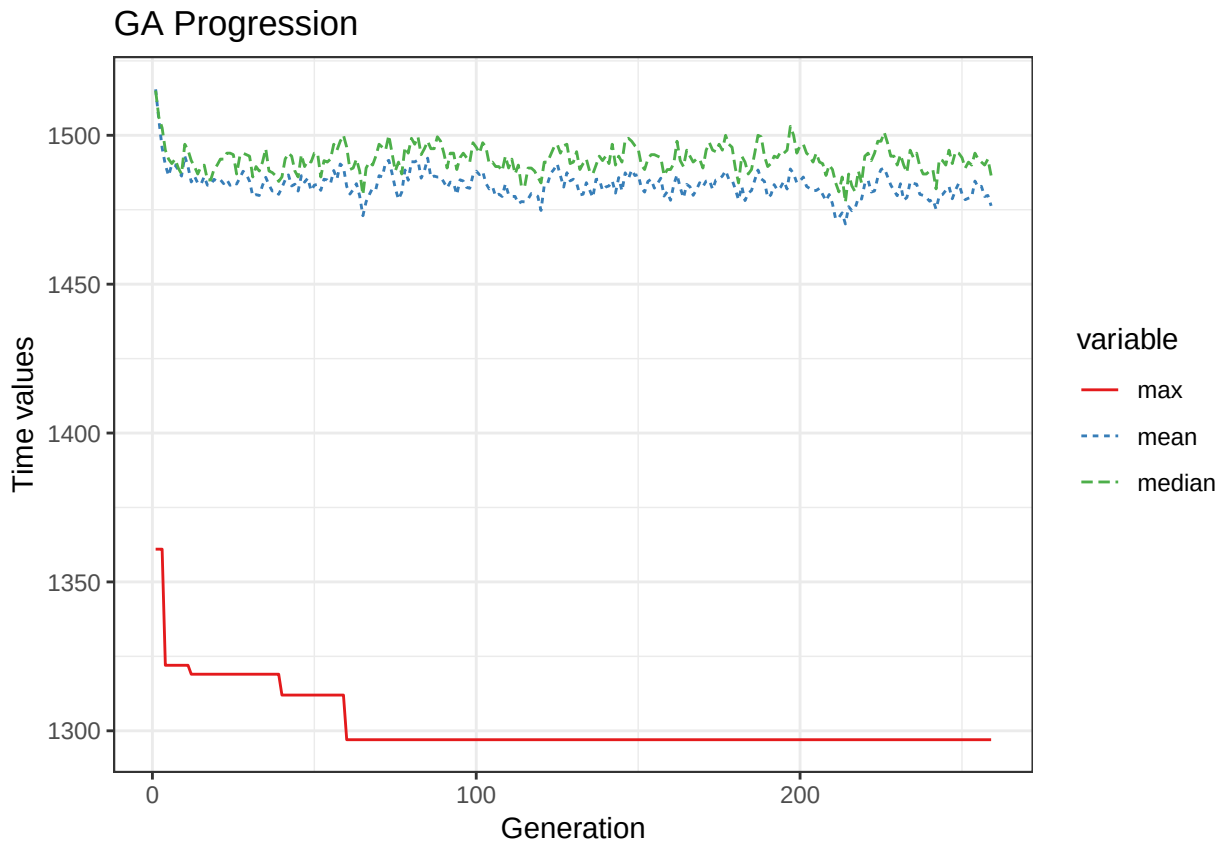
```
timeCpp(GA.fit@solution[1,],time_matrix) #best time
```

```
## [1] 1297
```

```
out <- plot(GA.fit, main = "GA progression")
```

```
melt(out[,c(1:3,5)],id.var="iter") %>%
  mutate(time=1/value) -> df1

ggplot(df1, aes(x = iter, y = time,
                group = variable, colour = variable)) +
  xlab("Generation") + ylab("Time values") +
  #geom_point(aes(shape = variable)) +
  geom_line(aes(lty = variable)) +
  scale_colour_brewer(palette = "Set1") +
  theme_bw() +
  labs(title = "GA Progression")
```




```
swapJobs <- function(perm){
    perm <- as.numeric(perm)
    n <- length(perm)
    n_change <- min(sample(1:floor(n/2),1),5) #no more than 5 changes at a time
    change <- sort(sample.int(n,n_change))
    newperm <- replace(perm,change,perm[sort(change,decreasing = TRUE)])
    return(as.numeric(newperm))
}

SA <- function(tour, distMatrix, maxIterNoChange=2000, T_ini = 50, T_min = 1){
  path <- tour
  n <- length(path)
  tmin <- T_min    # minimum temperature
  alpha <- 0.999  # update factor
  T <- T_ini
  tini <- T_ini   # starting temperature
  dist <- timeCpp(path, distMatrix)
  bestLength <- dist
  traceBest <- c(dist)
  traceCurrentLength <- c(dist)
  iterNoChange = 0
  while(T >= tmin){        # if the temperature is not at its minimum
    iterNoChange = iterNoChange+1
    newpath <- swapJobs(path) #swap
    dist_new <- timeCpp(newpath, distMatrix)
    if(dist_new <= bestLength){
      path <- newpath
```

```
      dist <- dist_new
      bestLength <- dist
      iterNoChange <- 0
    }
    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 # the temperature is updated
    if(iterNoChange >= maxIterNoChange){ break}
  }
  res = list(route=path, traceBest = traceBest, trace = traceCurrentLength)
  class(res) = "SAObj"
  print(paste("best=", toString(bestLength), sep=" "))
  return(res)
}

start <- as.numeric(sample(1:n_jobs,n_jobs)) #start randomly
res <- SA(start, time_matrix, maxIterNoChange = 10000)
```

```
## [1] "best= 1297"
```

Even when trying with the 100x20 table, the ga algorithm finished in approximately 15 minutes, while SA algorithm finished in a few seconds giving more or less the same result so thr best choice is to procceed using only the SA algorithm. The result for the 100x20 table is not provided here in order to avoid waiting 15 minutes more every compilation of the results.

Final Results using 500 jobs with 20 machines.

```
time_matrix <- as.matrix(read.csv("j500-m20",sep=" ",header = FALSE))
n_jobs <- ncol(time_matrix)

funObj <- tail_curry(timeCpp,time_matrix)


start <- as.numeric(sample(1:n_jobs,n_jobs))

res <- SA(start,time_matrix,maxIterNoChange = 10000)
```

```
## [1] "best= 27619"
```