```r
H = c(2,    2,  3,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  2,  3,  3)

get.schedule = function(Cohort_1, Cohort_2, H){
  schedule_1 = matrix(list(), nrow=8, ncol=5)
  schedule_2 = matrix(list(), nrow=8, ncol=5)

  # Evaluate objective function
  penalty = 0

  # Generater Schedule for Cohort 1
  j = 1
  Cohort_1_Selected = c(rep(FALSE,23))
  while (j <= 5)
  {
    i = 1
    while (i <= 8)
    {
      feasible = FALSE
      for (g in 1:length(Cohort_1_Selected)) {
        if (!Cohort_1_Selected[g] && Cohort_1[g,1] == 0) {
          schedule_1[[i,j]] = c(0,0)
          i = i + 1
          Cohort_1_Selected[g] = TRUE
          feasible = TRUE
          break
        } else if (!Cohort_1_Selected[g] && (9 - i >= H[Cohort_1[g,1]])) {
          for (h in 1:H[Cohort_1[g,1]]) {
            if (Cohort_1[g,1] %in% c(13,14,19,20,22)) {
              schedule_2[[i,j]] = c(Cohort_1[g,1], Cohort_1[g,2])
            }
            schedule_1[[i,j]] = c(Cohort_1[g,1], Cohort_1[g,2])
            i = i + 1
          }
          Cohort_1_Selected[g] = TRUE
          feasible = TRUE
          break
        }
      }
      if (!feasible) {
        i = i + 1
      }
    }
    j = j + 1
  }

  # Generater Schedule for Cohort 2
  j = 1
  Cohort_2_Selected = c(rep(FALSE,21))
  while (j <= 5)
  {
    i = 1
    while (i <= 8)
    {
      feasible = FALSE
      for (g in 1:length(Cohort_2_Selected)) {
        if (!Cohort_2_Selected[g] && Cohort_2[g,1] == 0) {
          if(length(which(sapply(schedule_2[i,j], is.null) == FALSE)) == 0){
            schedule_2[[i,j]] = c(0,0)
            i = i + 1
            Cohort_2_Selected[g] = TRUE
            feasible = TRUE
            break
          } else {
            break
          }
        } else if (!Cohort_2_Selected[g] && (9 - i >= H[Cohort_2[g,1]])) {
          if(length(which(sapply(schedule_2[i:(i+H[Cohort_2[g,1]]-1),j], is.null) == FALSE)) == 0){
            for (h in 1:H[Cohort_2[g,1]]) {
              schedule_2[[i,j]] = c(Cohort_2[g,1], Cohort_2[g,2])
              i = i + 1
            }
            Cohort_2_Selected[g] = TRUE
            feasible = TRUE
            break
          }else {
            break
          }
        }
      }

      if (!feasible) {
        i = i + 1
      }
    }
    j = j + 1
  }
  list(schedule_1 = schedule_1, schedule_2 = schedule_2)
}

get.fitness = function(Cohort_1, Cohort_2, H){
  schedule_1 = matrix(list(), nrow=8, ncol=5)
  schedule_2 = matrix(list(), nrow=8, ncol=5)
  mondayProfs = c()
  fridayProfs = c()

  # Evaluate objective function
  penalty = 0

  # Get number of courses taught by professors
  profToCourse = as.data.frame(table(c(Cohort_1[,2], Cohort_2[,2])))
  for (i in 2:nrow(profToCourse)) {
    if (profToCourse[6,2] > 2) {
      penalty = penalty + 8
    } else {
      penalty = penalty + 4
    }
  }


  # Penalize number of professors
  penalty = penalty + 10*(length(unique(c(Cohort_1[,2], Cohort_2[,2]))) - 1)

  # Generater Schedule for Cohort 1
  j = 1
  Cohort_1_Selected = c(rep(FALSE,23))
  while (j <= 5)
  {
    i = 1
    while (i <= 8)
    {
      feasible = FALSE
      for (g in 1:length(Cohort_1_Selected)) {
        if (!Cohort_1_Selected[g] && Cohort_1[g,1] == 0) {
          schedule_1[[i,j]] = c(0,0)
          i = i + 1
          Cohort_1_Selected[g] = TRUE
          feasible = TRUE
          break
        } else if (!Cohort_1_Selected[g] && (9 - i >= H[Cohort_1[g,1]])) {
          for (h in 1:H[Cohort_1[g,1]]) {
            if (Cohort_1[g,1] %in% c(13,14,19,20,22)) {
              schedule_2[[i,j]] = c(Cohort_1[g,1], Cohort_1[g,2])
            }
            schedule_1[[i,j]] = c(Cohort_1[g,1], Cohort_1[g,2])

            # If a course is scheduled on Morning, Lunch of Evening slots penalize it
            if (i == 1) {
              penalty = penalty + 1
            }
            if (i == 4) {
              penalty = penalty + 100
            }
            if (i == 8) {
              penalty = penalty + 1
            }

            i = i + 1
          }
          Cohort_1_Selected[g] = TRUE
          feasible = TRUE

          # If a professor is scheduled on a Monday or Friday penalize it
          if (j  == 1) {
            mondayProfs = c(mondayProfs, Cohort_1[g,2])
          }

          if (j  == 5) {
            fridayProfs = c(fridayProfs, Cohort_1[g,2])
          }

          break
        }
      }
      if (!feasible) {
        i = i + 1
      }
    }
    j = j + 1
  }

  # Generater Schedule for Cohort 2
  j = 1
  Cohort_2_Selected = c(rep(FALSE,21))
  while (j <= 5)
  {
    i = 1
    while (i <= 8)
    {
      feasible = FALSE
      for (g in 1:length(Cohort_2_Selected)) {
        if (!Cohort_2_Selected[g] && Cohort_2[g,1] == 0) {
          if(length(which(sapply(schedule_2[i,j], is.null) == FALSE)) == 0){
            schedule_2[[i,j]] = c(0,0)
            i = i + 1
            Cohort_2_Selected[g] = TRUE
            feasible = TRUE
            break
          } else {
            break
          }
        } else if (!Cohort_2_Selected[g] && (9 - i >= H[Cohort_2[g,1]])) {
          if(length(which(sapply(schedule_2[i:(i+H[Cohort_2[g,1]]-1),j], is.null) == FALSE)) == 0){
            for (h in 1:H[Cohort_2[g,1]]) {
              schedule_2[[i,j]] = c(Cohort_2[g,1], Cohort_2[g,2])

              # If a course is scheduled on Morning, Lunch of Evening slots penalize it
              if (i == 1) {
                penalty = penalty + 1
              }
              if (i == 4) {
                penalty = penalty + 100
              }
              if (i == 8) {
                penalty = penalty + 1
              }

              i = i + 1
            }
            Cohort_2_Selected[g] = TRUE
            feasible = TRUE

            # If a professor is scheduled on a Monday or Friday penalize it
            if (j  == 1) {
              mondayProfs = c(mondayProfs, Cohort_1[g,2])
            }

            if (j  == 5) {
              fridayProfs = c(fridayProfs, Cohort_1[g,2])
            }

            break
          }else {
            break
          }
        }
      }

      if (!feasible) {
        i = i + 1
      }
    }
    j = j + 1
  }

  penalty = penalty + 8*(length(unique(c(mondayProfs))))
  penalty = penalty + 8*(length(unique(c(fridayProfs))))

  if (FALSE %in% Cohort_1_Selected || FALSE %in% Cohort_2_Selected) {
    penalty = penalty + 100000
  }

  1000/penalty
}

#This creates the initial population.
get.random.population = function(n){
  population = vector("list", n)
  for (g in 1:n) {
    courseToProf = list(sample(c(6,9,10)), sample(c(3,4)), sample(c(5,10)), sample(c(3,4)), sample(c(3,4,9)), sample(c(3,4,8)),
                        sample(c(1,2)), sample(c(5,7,8)), sample(c(5,8)), sample(c(1,2)), c(6), sample(c(6,7,9,10)))
    Cohort_1 = Matrix(c(1,2,3,4,7,8,13,14,15,16,17,18,19,20,22,rep(0,31)), nrow=23, ncol=2, byrow=F)
    Cohort_2 = Matrix(c(5,6,9,10,11,12,21,rep(0,35)), nrow=21, ncol=2, byrow=F)
```

```r
    for (i in seq(1, 15, 2)) {
      if (Cohort_1[i,1] == 22) {
        for (j in 1:length(courseToProf[[12]])) {
          if (length(which(Cohort_1[,2]==courseToProf[[12]][j],2)) == 0) {
            Cohort_1[i,2] = courseToProf[[12]][j]
            break
          } else if (j == length(courseToProf[[12]])) {
            Cohort_1[i,2] = 11
          }
        }
      } else {
        index = ceiling(Cohort_1[i,1]/2)
        for (j in 1:length(courseToProf[[index]])) {
          if (length(which(Cohort_1[,2]==courseToProf[[index]][j],2)) == 0) {
            Cohort_1[i,2] = courseToProf[[index]][j]
            Cohort_1[i+1,2] = courseToProf[[index]][j]
            break
          } else if (j == length(courseToProf[[index]])) {
            Cohort_1[i,2] = 11
          }
        }
      }
    }
    Cohort_1 = Cohort_1[sample(nrow(Cohort_1)),]

    for (i in seq(1, 7, 2)) {
      if (Cohort_2[i,1] == 21) {
        for (j in 1:length(courseToProf[[11]])) {
          if (length(which(Cohort_2[,2]==courseToProf[[11]][j],2)) == 0) {
            Cohort_2[i,2] = courseToProf[[11]][j]
            break
          } else if (j == length(courseToProf[[12]])) {
            Cohort_1[i,2] = 11
          }
        }
      } else {
        index = ceiling(Cohort_2[i,1]/2)
        for (j in 1:length(courseToProf[[index]])) {
          if (length(which(Cohort_2[,2]==courseToProf[[index]][j],2)) == 0) {
            Cohort_2[i,2] = courseToProf[[index]][j]
            Cohort_2[i+1,2] = courseToProf[[index]][j]
            break
          } else if (j == length(courseToProf[[index]])) {
            Cohort_1[i,2] = 11
          }
        }
      }
    }
    Cohort_2 = Cohort_2[sample(nrow(Cohort_2)),]

    population[[g]] = list(Cohort_1, Cohort_2)
  }
  population
}

#For each member of the population, this generates the probability of being chosen.
roulette.wheel = function(fitness){
  probs = fitness/sum(fitness)
  sample(length(fitness), length(fitness), replace=T, prob=probs)
}

#This function decides which two members of the population will be mating.
cross.over = function(mating.population, p.c=0.75){
  rlist = runif(length(mating.population),0,1)
  c.indices = which(rlist < p.c)
  if(length(c.indices) <= 1){
    c.indices = order(rlist)[1:2]
  }
  if( (length(c.indices) %% 2) ==1 ){
    c.indices = c.indices[-which(rlist[c.indices] == max(rlist[c.indices]))]
  }
  pairs = matrix(c.indices, ncol=2, byrow=T)
  offsprings = vector("list", nrow(pairs)*2)
  offspring_count = 1
  for(i in 1:nrow(pairs)){
    offsprings[[offspring_count]] = coitus(mating.population[pairs[i,1]][[1]],mating.population[pairs[i,2]][[1]])
    offspring_count = offspring_count + 1
    offsprings[[offspring_count]] = coitus(mating.population[pairs[i,2]][[1]],mating.population[pairs[i,1]][[1]])
    offspring_count = offspring_count + 1
  }
  list(offsprings = offsprings, c.indices = c.indices)
}
```

```
# The chosen parents mate and generate an offspring. One parent will be responsible for the schedule,
# while the other will be responsible for the assignment of professors.
coitus = function(parent_1, parent_2) {
  courseToProf = rep(0,22)
  for (i in 1:23) {
    if (parent_1[1][[1]][i,1] != 0) {
      courseToProf[parent_1[1][[1]][i,1]] = parent_1[1][[1]][i,2]
    }
  }

  for (i in 1:21) {
    if (parent_1[2][[1]][i,1] != 0) {
      courseToProf[parent_1[2][[1]][i,1]] = parent_1[2][[1]][i,2]
    }
  }

  offspring = parent_2
  # Cohort 1
  for (i in 1:23) {
    if (offspring[1][[1]][i,1] != 0) {
      offspring[1][[1]][i,2] = courseToProf[offspring[1][[1]][i,1]]
    }
  }

  # Cohort 2
  for (i in 1:21) {
    if (offspring[2][[1]][i,1] != 0) {
      offspring[2][[1]][i,2] = courseToProf[offspring[2][[1]][i,1]]
    }
  }

  offspring
}

#This allows mutation, where two items of an offspring are swapped in their order.
mutation = function(offsprings, p.m1=0.30, p.m2=0.15, p.m3=0.05){
  for(i in 1:length(offsprings)){
    v = runif(1)
    if (v < p.m3) {
      for (j in 1:4) {
        cohort_choice = runif(1)
        if (cohort_choice < 0.5) {
          swap = sample(1:23, 2, replace=F)
          tmp = offsprings[i][[1]][1][[1]][swap[1],]
          offsprings[i][[1]][1][[1]][swap[1],] = offsprings[i][[1]][1][[1]][swap[2],]
          offsprings[i][[1]][1][[1]][swap[2],] = tmp
        } else {
          swap = sample(1:21, 2, replace=F)
          tmp = offsprings[i][[1]][2][[1]][swap[1],]
          offsprings[i][[1]][2][[1]][swap[1],] = offsprings[i][[1]][2][[1]][swap[2],]
          offsprings[i][[1]][2][[1]][swap[2],] = tmp
        }
      }
    }
    else if (v < p.m2) {
      for (j in 1:3) {
        cohort_choice = runif(1)
        if (cohort_choice < 0.5) {
          swap = sample(1:23, 2, replace=F)
          tmp = offsprings[i][[1]][1][[1]][swap[1],]
          offsprings[i][[1]][1][[1]][swap[1],] = offsprings[i][[1]][1][[1]][swap[2],]
          offsprings[i][[1]][1][[1]][swap[2],] = tmp
        } else {
          swap = sample(1:21, 2, replace=F)
          tmp = offsprings[i][[1]][2][[1]][swap[1],]
          offsprings[i][[1]][2][[1]][swap[1],] = offsprings[i][[1]][2][[1]][swap[2],]
          offsprings[i][[1]][2][[1]][swap[2],] = tmp
        }
      }
    }
    else if (v < p.m1) {
      for (j in 1:2) {
        cohort_choice = runif(1)
        if (cohort_choice < 0.5) {
          swap = sample(1:23, 2, replace=F)
          tmp = offsprings[i][[1]][1][[1]][swap[1],]
          offsprings[i][[1]][1][[1]][swap[1],] = offsprings[i][[1]][1][[1]][swap[2],]
          offsprings[i][[1]][1][[1]][swap[2],] = tmp
        } else {
          swap = sample(1:21, 2, replace=F)
          tmp = offsprings[i][[1]][2][[1]][swap[1],]
          offsprings[i][[1]][2][[1]][swap[1],] = offsprings[i][[1]][2][[1]][swap[2],]
          offsprings[i][[1]][2][[1]][swap[2],] = tmp
        }
      }
    }
  }
  offsprings
}

N = 100
population = get.random.population(N)

x.best = NULL
z.best = 0
z.list = NULL
mean.z.list = NULL

max.iter = 101
iter = 0

while (iter < max.iter) {
  fitness = c()
  for (i in 1:N) {
    fitness = c(fitness ,get.fitness(population[i][[1]][1][[1]], population[i][[1]][2][[1]], H))
  }

  z.list = c(z.list, max(fitness))
  mean.z.list  = c(mean.z.list, mean(fitness))
  if(max(fitness) > z.best){
    z.best = max(fitness)
    x.best = population[which(fitness==max(fitness))[1]][[1]]
  }

  m.indices = roulette.wheel(fitness)
  mating.population = population[m.indices]

  cover = cross.over(mating.population)
  c.indices = cover$c.indices
  offsprings = cover$offsprings
  offsprings = mutation(offsprings)

  population = mating.population
  population[c.indices] = offsprings

  iter = iter + 1
}

x.best
z.best

schedules = get.schedule(x.best[1][[1]], x.best[2][[1]], H)
schedule_1 = schedules$schedule_1
schedule_2 = schedules$schedule_2
```