Assignment 1

Populacija

Populacijo se generira tako, da se najprej izbere naključno permutacijo tolikih števil, kot je števil v izrazu (n), nato se izbere še (n-1) naključnih števil od n+1 do n+5 s ponavljanjem, ki predstavljajo operatorje, to dvoje pa na koncu združi (izmenično) v en sam vektor.

Fitness

Fitness funkcija evalvira izraz, ki ga prestavlja dani vektor, nato pa vrne absolutno vrednost razlike te vrednosti od pravilne.

Mutation

myMutation1 izbere naključni operator v izrazu in ga zamenja z nekim novim naključnim, števila pa pusti pri miru. myMutation2 izbere dva operatorja ali dve števili v izrazu ter ju med seboj zamenja. myMutation3, dodan kasneje, podobno kot M1 zamenja en operator v izrazu, za tem pa med zamenja še dve števili

Crossover

crossover1 med seboj zamenja naključna istoležna operatorja v dveh izrazih, crossover2 pa med seboj zamenja celotni prvi polovici izrazov, nato pa popravi števila v drugi polovici tako, da se v celotnem izrazu ne ponavljajo.

Koda

```
library(GA)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
## Package 'GA' version 3.2.2
## Type 'citation("GA")' for citing this R package in publications.
```

```
##
## Attaching package: 'GA'
```

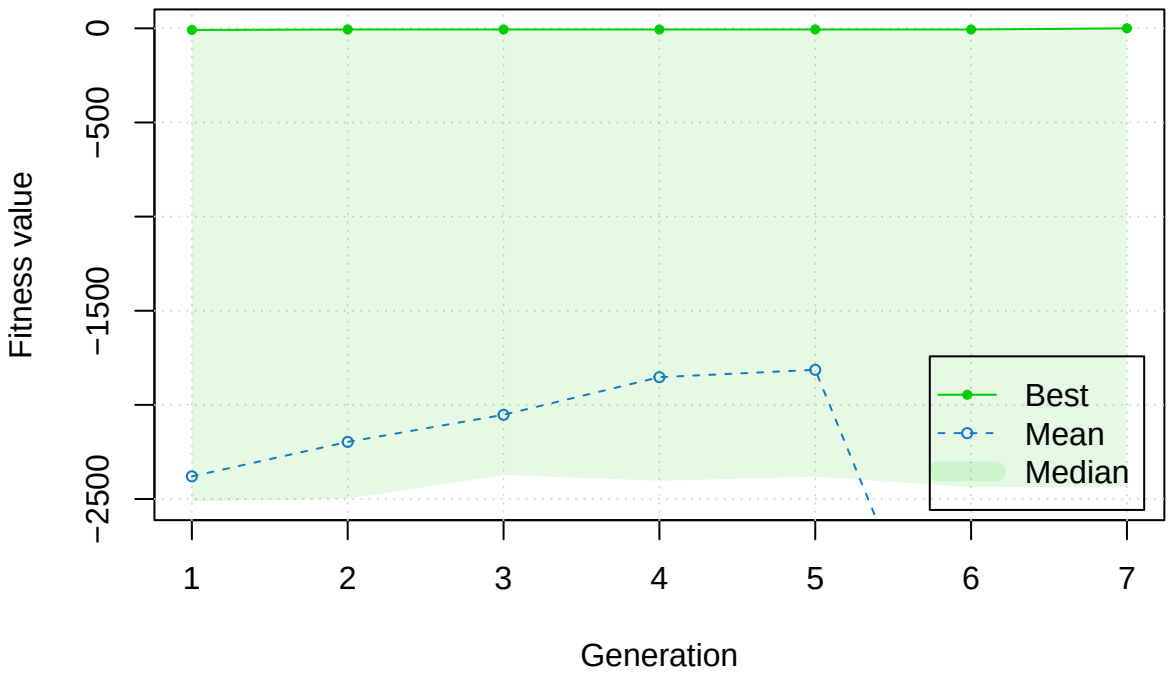
```
## The following object is masked from 'package:utils':
##
##     de
```

```
numbers_and_operators <- c("10", "25", "100", "5", "3", "+", "-", "/", "*")
numbers = c("10", "25", "100", "5", "3")

x = c(3, 9, 2, 6, 1, 6, 4, 7, 5)
target <- 2512

## Population generation (initial)
myInitPopulation <- function(object)
{
  iter <- 1: object@popSize
  m <- matrix(nrow=object@popSize, ncol = 2*length(numbers)-1)
  for (i in iter)
  {

    nums <- 1: length(numbers)
    nums_perm_indexes = sample(nums, replace=FALSE)
    oper_indexes = (length(numbers) +1) : (length(numbers_and_operators))
    oper_perm_repeat = sample(oper_indexes,length(numbers), replace = TRUE) #generirano enega preveč

    izraz <- c(rbind(nums_perm_indexes, oper_perm_repeat))
    izraz <- head(izraz, -1)

    m[i,] <- izraz

  }
  return (m)

}

myMutation1 <-function (object,parent){ # vzame random operator in ga zamenja z naključnim
  mutate <- parent <- as.vector(object@population[parent,])
  n <- length(parent)
  mutat_idx <- sample(1:((n-1)/2), size = 1)*2

  samp_oper <- sample((length(numbers)+1):length(numbers_and_operators),1)
  mutate[mutat_idx] = samp_oper

  return (mutate)
}

myMutation2 <-function (object, parent){ #vzame 2 števili ali dva operatorja in ju zamenja
  mutate <- parent <- as.vector(object@population[parent,])
  n <- length(parent)
  rand_idx = sample(parent, size=2)
  wrong_sample = TRUE

  rnd = sample(0:1, 1)
  if (rnd == 1) { # zamenja stevili
    idx = (sample(1:((n+1)/2), 2)-1)*2 + 1
    rev_idx = rev(idx)
```

```
    mutate[rev_idx] = parent[idx]
  } else { #       zamenja operatorja
    idx = (sample(1:((n-1)/2), 2)-1)*2 + 1
    rev_idx = rev(idx)
    mutate[rev_idx] = parent[idx]
  }

  return (mutate)
}

myMutation3 <-function (object,parent){ # zamenja naključni operator in dve števili
  mutate <- parent <- as.vector(object@population[parent,])
  n <- length(parent)
  mutat_idx <- sample(1:((n-1)/2), size = 1)*2

  samp_oper <- sample((length(numbers)+1):length(numbers_and_operators),1)
  mutate[mutat_idx] = samp_oper

  # zamenja stevili
  idx = (sample(1:((n+1)/2), 2)-1)*2 + 1
  rev_idx = rev(idx)
  mutate[rev_idx] = parent[idx]

  return (mutate)
}

crossover1 <- function(object, parent){  # zamenja istoležna operatorja
  children <- parent <- (object@population[parent,])
  n <- length(numbers)*2 - 1
  i = (sample(1:((n-1)/2), 2)-1)*2
  children[c(1,2), i] = children[c(2,1), i]

  return(list(children=children, fitness=rep(NA, 2)))
}

crossover2 <- function(object, parent){  # zamenja prvi polovici vektorjev in popravi podvajanje števil
  children <- parent <- (object@population[parent,])
  nums <- length(numbers)
  n <- nums*2 - 1
  is = 1:((n+1)/4)*2 - 1
  js = is + 1
  children[c(1,2), is] = children[c(2,1), is]
  children[c(1,2), js] = children[c(2,1), js]

  k1 = 1
  k2 = 1
  for (ii in ((tail(is, n=1)+1)/2+1):((n+1)/2)) {
    i = ii*2 - 1
    if (children[1,i] %in% children[1, is]) {
      while (parent[1,k1] %in% children[1, is]) {
        k1 = k1 + 2
      }
      children[1, i] = parent[1, k1]
```

```
      k1 = k1 + 2
    }
    if (children[2,i] %in% children[2, is]) {
      while (parent[2,k2] %in% children[2, is]) {
        k2 = k2 + 2
      }
      children[2, i] = parent[2, k2]
      k2 = k2 + 2
    }
  }

  return(list(children=children, fitness=rep(NA, 2)))
}

calc <- function(izraz){
  izraz <-numbers_and_operators[izraz]
  str = paste(izraz, collapse='')
  rez <- eval(parse(text=str))
  return (rez)

}

myFitness <- function(izraz)
{
  -abs(target - calc(izraz))
}

GA <- ga(type = "permutation", population = myInitPopulation, fitness = myFitness,
     lower = 1, upper = 2*length(numbers)-1, popSize = 20, maxiter = 2000,
     run = 100, mutation = myMutation3, crossover=crossover1, maxFitness=0)
```

```
plot(GA)
```



Random search

Še implementacija naključnega iskanja:

```
n_random_samples_fitn <- function(n){
  optimum_izraz <- c()
  optim_fitness = -1e10
  iter <- 1: n
  m <- c()
  for (i in iter)
  {
    # generiranje naključnega
    nums <- 1: length(numbers)
    nums_perm_indexes = sample(nums, replace=FALSE)
    oper_indexes = (length(numbers) +1) : (length(numbers_and_operators))
    oper_perm_repeat = sample(oper_indexes,length(numbers), replace = TRUE) #generirano enega preveč

    izraz <- c(rbind(nums_perm_indexes, oper_perm_repeat))
    izraz <- head(izraz, -1)

    val <- myFitness(izraz)
    if(val >= optim_fitness){
      optimum_izraz <- izraz
      optim_fitness <- val
    }
```

```
  }
  return (optimum_izraz)

}


random_search <- function(n, max_iters){
  hit_target = FALSE
  fitnesses <- c()
  global_best_izraz <- c()
  global_best_fitness <- -1e10

  i = 1
  while(!hit_target){
    naj_izraz <- n_random_samples_fitn(n)
    fitness_naj_izraz <- myFitness(naj_izraz)
    fitnesses <- c(fitnesses, fitness_naj_izraz)

    if(fitness_naj_izraz >= global_best_fitness){
      global_best_fitness <- fitness_naj_izraz
      global_best_izraz <- naj_izraz
    }
    if(global_best_fitness == 0){
      hit_target = TRUE
    }
    if (i >= max_iters) {
      hit_target = TRUE
    }

    i = i + 1
  }
  rez <- list(fitnesses, global_best_izraz)
  return (rez)
}

l = random_search(20, 100)
print(l[[1]])
```

```
##  [1] -1529.000   -23.000 -1676.667  -977.000 -2160.000   -18.000   -13.000
##  [8] -1442.000    -1.400   -13.000 -1912.000   -14.500   -13.000 -2077.000
## [15]    -9.500 -1365.000     0.000
```

Evaluation

```
evaluate <- function(nn, mut, cross) {
  opt = 0
  its = 0
  fit = numeric(1)
  c=0
  for (i in 1:nn) {
```

```
    GA <- ga(type = "permutation", population = myInitPopulation, fitness = myFitness,
       lower = 1, upper = 2*length(numbers)-1, popSize = 20, maxiter = 2000,
       run = 100, mutation = mut, crossover=cross, maxFitness=0, monitor=FALSE)
    f = GA@summary[,"max"]
    len = max(length(fit), length(f))
    length(fit) <- length(f) <- len
    f[is.na(f)] = 0
    fit[is.na(fit)] = 0
    fit = fit + f
    if (GA@fitnessValue == 0) {
      opt = opt + 1
      its = its + GA@iter
    }
    c = c + 1
  }
  opt1 = opt/nn
  its1 = its/nn
  fit1 = fit/nn
  pr = (paste("% of found optimal:", opt1*100, "       # of iters until opt. found:", its1))
  return (list(opt1, its1, fit1, pr))
}
nn = 20
```

```
e11 = evaluate(nn, myMutation1, crossover1)
print(paste("m1, c1:   ", e11[[4]]))
```

```
## [1] "m1, c1:    % of found optimal: 50        # of iters until opt. found: 19.5"
```

```
e21 = evaluate(nn, myMutation2, crossover1)
print(paste("m2, c1:   ", e21[[4]]))
```

```
## [1] "m2, c1:    % of found optimal: 35        # of iters until opt. found: 4.2"
```

```
e12 = evaluate(nn, myMutation1, crossover2)
print(paste("m1, c2:   ", e12[[4]]))
```

```
## [1] "m1, c2:    % of found optimal: 65        # of iters until opt. found: 10.6"
```

```
e22 = evaluate(nn, myMutation2, crossover2)
print(paste("m2, c2:   ", e22[[4]]))
```

```
## [1] "m2, c2:    % of found optimal: 40        # of iters until opt. found: 2.65"
```

```
e31 = evaluate(nn, myMutation3, crossover1)
print(paste("m3, c1:   ", e31[[4]]))
```

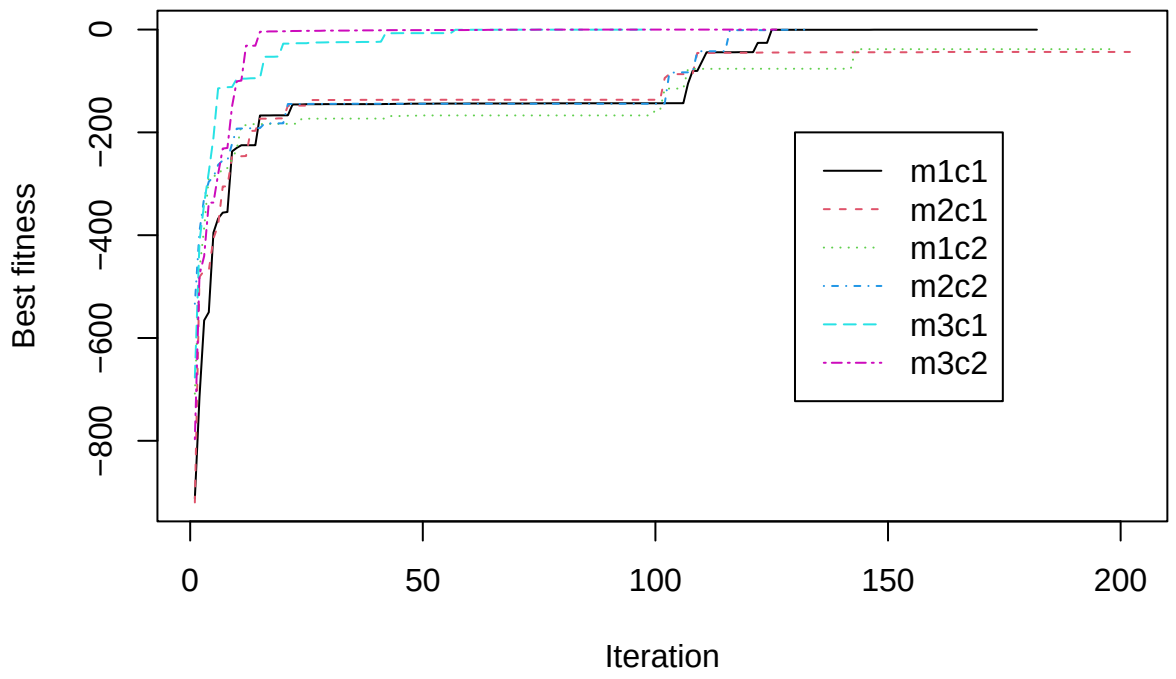
```
## [1] "m3, c1:    % of found optimal: 100        # of iters until opt. found: 26.55"
```

```
e32 = evaluate(nn, myMutation3, crossover2)
print(paste("m3, c2:   ", e32[[4]]))
```

```
## [1] "m3, c2:    % of found optimal: 95        # of iters until opt. found: 42.05"
```

Iz zgornjih podatkov lahko vidimo, da le kombinacije z Mutation3 najdejo rešitev v 100%. Najbolje se izkaže kombinacija Mutation3 in crossover1.

```
f11 = e11[[3]]
f12 = e12[[3]]
f21 = e21[[3]]
f22 = e22[[3]]
f31 = e31[[3]]
f32 = e32[[3]]
l = max(length(f11), length(f12), length(f21), length(f22), length(f31), length(f32))
length(f11) <- length(f12) <- length(f21) <- length(f22) <- length(f31) <- length(f32) <- l

matplot(cbind(f11,f21,f12,f22,f31,f32), type="l", pch=1, col=1:6, lty=1:6, xlab="Iteration", ylab="Best
legend(130, -200, c("m1c1", "m2c1", "m1c2", "m2c2", "m3c1",  "m3c2"), col=1:6, lty=1:6)
```



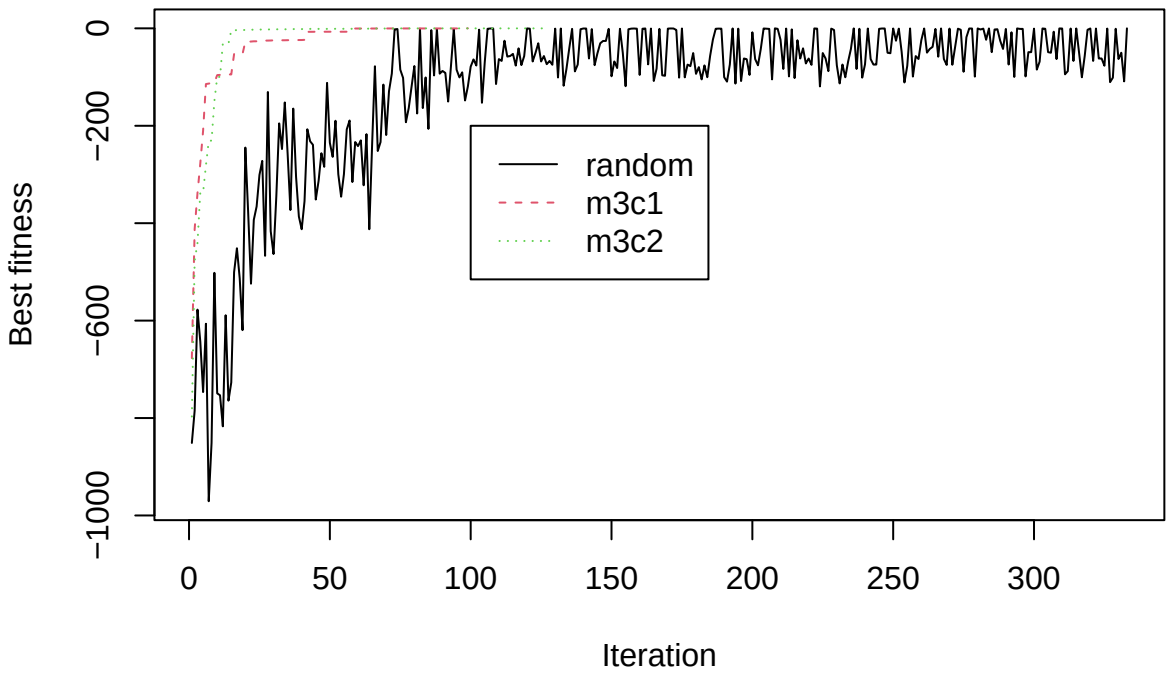
Še primerjava med random_search in kombinacijama z Mutation3:

```
evaluate_random <- function(nn) {
  opt = 0
  its = 0
```

```
  fit = numeric(1)
  c=0
  for (i in 1:nn) {
    res = random_search(20, 500)
    f = res[[1]]
    len = max(length(fit), length(f))
    length(fit) <- length(f) <- len
    f[is.na(f)] = 0
    fit[is.na(fit)] = 0
    fit = fit + f
    if (tail(res[[1]], n=1) == 0) {
      opt = opt + 1
      its = its +length(res[[1]])
    }
    c = c + 1
  }
  opt1 = opt/nn
  its1 = its/nn
  fit1 = fit/nn
  pr = (paste("% of found optimal:", opt1*100, "       # of iters until opt. found:", its1))
  return (list(opt1, its1, fit1, pr))
}

er = evaluate_random(nn)
print(paste("random:   ", er[[4]]))
```

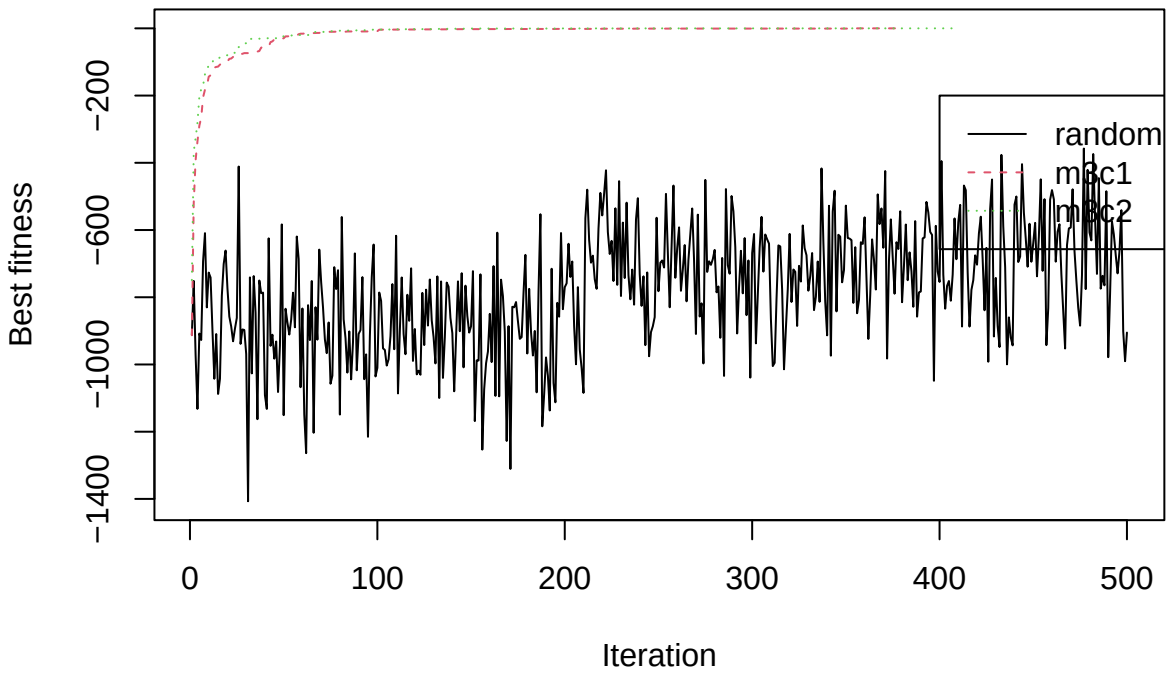
```
## [1] "random:    % of found optimal: 100        # of iters until opt. found: 45.95"
```

```
f31 = e31[[3]]
f32 = e32[[3]]
fr = er[[3]]
l = max(length(fr), length(f31), length(f32))
length(fr) <- length(f31) <- length(f32) <- l

matplot(cbind(fr,f31,f32), type="l", pch=1, col=1:3, lty=1:3, xlab="Iteration", ylab="Best fitness")
legend(100, -200, c("random", "m3c1",  "m3c2"), col=1:3, lty=1:3)
```




```
numbers_and_operators <- c("10", "20", "30", "40", "50", "60", "70", "80", "90", "1" , "500", "+", "-",
numbers = c("10", "20", "30", "40", "50", "60", "70", "80", "90", "1" , "500")
target <- 3700

e31 = evaluate(nn, myMutation3, crossover1)
e32 = evaluate(nn, myMutation3, crossover2)
er = evaluate_random(nn)
```

```
f31 = e31[[3]]
f32 = e32[[3]]
fr = er[[3]]
l = max(length(fr), length(f31), length(f32))
length(fr) <- length(f31) <- length(f32) <- l

matplot(cbind(fr,f31,f32), type="l", pch=1, col=1:3, lty=1:3, xlab="Iteration", ylab="Best fitness")
legend(400, -200, c("random", "m3c1",  "m3c2"), col=1:3, lty=1:3)
```



Tukaj na večjem primeru iz 11 števil je večja hitrost konvergence GA proti random še bolj izrazita.

Zaključimo torej lahko, da se med funkcijami za mutacijo najbolje izkaže Mutation3 (menjava operatorja in zamenjava dveh števil), verjetno zato, ker omogoča največ variacije v populaciji (Mutation2 le med seboj zamenjuje, kar je že v izrazu, zato verjetno nastane problem, če v nekem izrazu še ni npr. množenja, v rešitvi pa je; Mutation1 pa deluje le na operatorjih), med crossoverji pa crossover1 (menjava le operatorjev) - morda zato, ker crossover2 s popravljanjem desnega dela vektorja le-tega preveč spremeni. GA pri uporabi M3 konvergira bistveno hitreje kot random search, kar je bolj izrazito pri večjih izrazih.