genalg para otimização de funções em R

Caio Alberth de Morais, Felipe Muradas Azevedo, Isadora Meireles do Amaral, Luís Fernando Israel Assunção e Pedro Henrique Ribeiro dos Reis

25 de maio de 2018

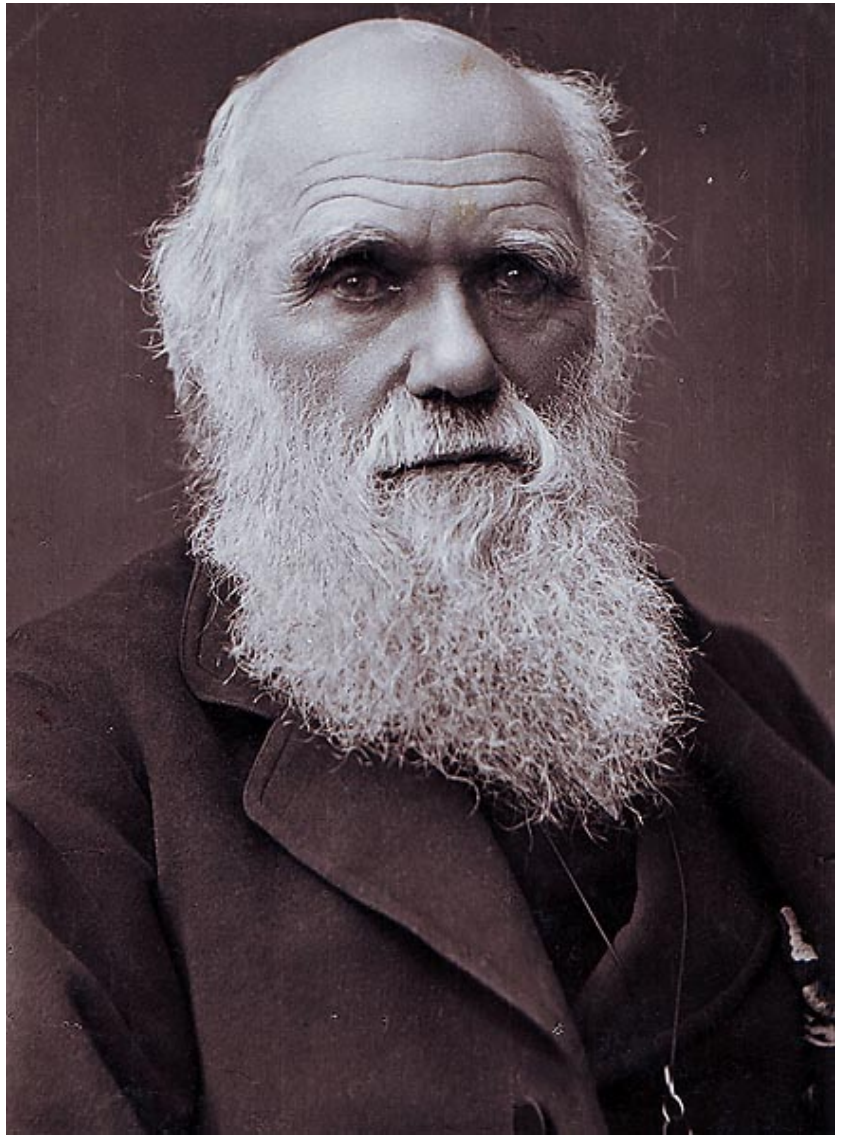
Algoritmos genéticos

Algoritmos genéticos são heurísticas de busca inspiradas pela teoria da evolução de Charles Darwin.

Heurísticas são técnicas para se resolver problemas por aproximação da solução ideal. Essas técnicas são usadas em situações nas quais algoritmos tradicionais são muito lentos ou complexos.

O pacote *genalg* possui funções que implementam algoritmos genéticos em R.

```
library(genalg)
```



Problema da mochila

Uma pessoa vai passar um mês em uma floresta e pode carregar, no máximo, 20kg em sua mochila. Qual combinação de itens maximiza os pontos de sobrevivência dessa pessoa?

```
item <- c("canivete", "feijões", "batatas", "cebolas",
          "saco de dormir", "corda", "compasso")
pontos <- c(10, 20, 15, 2, 30, 10, 30)
peso <- c(1, 5, 10, 1, 7, 5, 1)
dados <- data.frame(item, pontos, peso)
```

item	pontos	peso
canivete	10	1
feijões	20	5
batatas	15	10
cebolas	2	1
saco de dormir	30	7
corda	10	5
compasso	30	1

Cromossomos

As possíveis soluções para o problema são representadas por cromossomos.

No exemplo em questão, os cromossomos são vetores com 7 valores (genes) binários, onde 1 representa a escolha do item e 0 a não escolha.

Obs: também existem problemas nos quais os genes assumem valores int out float.

```
cromossomo <- c(1, 0, 0, 1, 0, 1, 1)
```

	item	pontos	peso
1	canivete	10	1
4	cebolas	2	1
6	corda	10	5
7	compasso	30	1

```
cat('pontos:', cromossomo %*% dados$pontos,
    'peso:', cromossomo %*% dados$peso)
```

```
## pontos: 52 peso: 8
```

Função de avaliação

Para maximizar os pontos de sobrevivência, criamos uma função de avaliação que retorna os pontos de uma dada combinação. Como a função *rbga.bin* (R-based genetic algorithm binary, do pacote *genalg*) minimiza a função de avaliação, o output é multiplicado por -1. Se o peso da combinação excede o peso limite, o output é 0.

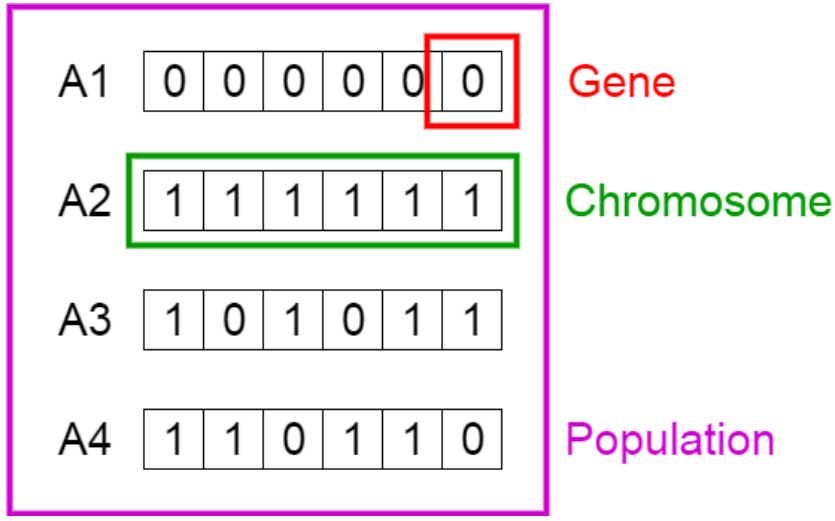
```
peso_limite <- 20

avaliacao <- function(cromossomo) {
    pontos <- cromossomo %*% dados$pontos
    peso <- cromossomo %*% dados$peso

    if (peso > peso_limite)
      return(0)
    else
      return(pontos*(-1))
}
```

O algoritmo

1. **Início** Gerar uma população aleatória de n cromossomos.

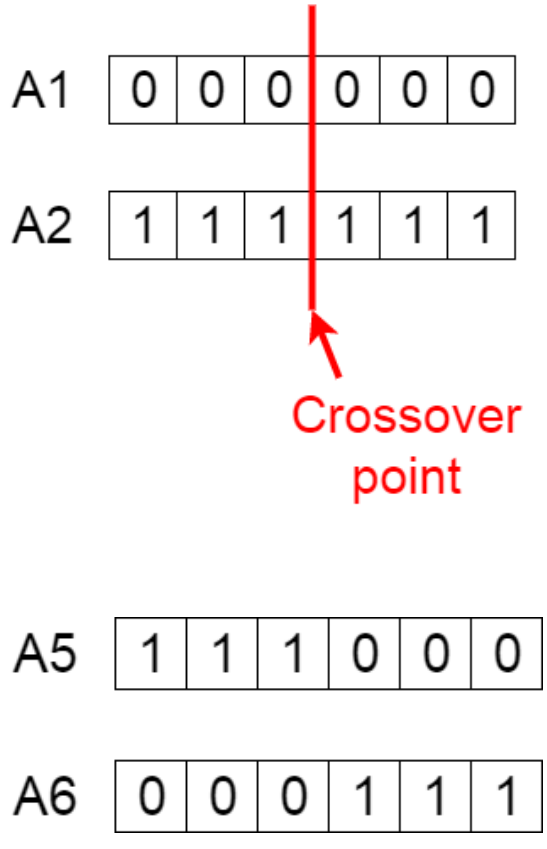


2. **Avaliação** Avaliar cada cromossomo com a função *avaliacao(cromossomo)*.

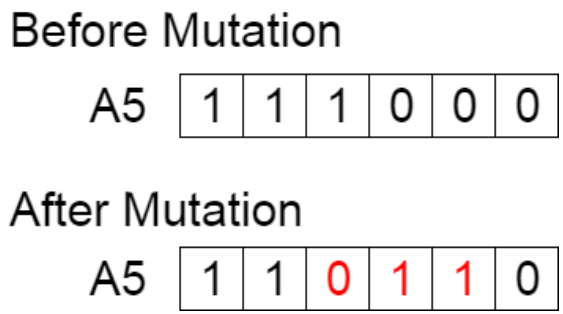
3. **Nova população** Criar uma nova população de n cromossomos repetindo os passos:

- **Seleção** Selecionar dois pares de cromossomos da população original. A chance de escolha de um cromossomo é ponderada pela sua avaliação.
  - O parâmetro **elitismo** faz com que as melhores soluções de uma população ascendente sejam sempre copiadas de forma idêntica para população descendente. Assim, a melhor solução encontrada é preservada até o fim da execução do algoritmo.

- **Cruzamento** Determinar, aleatoriamente, um ponto de cruzamento entre os pares. Criar cromossomos descendentes dos originais por meio da troca dos genes até o ponto de cruzamento.



- **Mutação** Modificar o valor dos genes de cada cromossomo de acordo com um parâmetro de probabilidade.



- **Aceitação** Adicionar os novos cromossomos à nova população.

4. **Teste** Se a condição de encerramento for satisfeita, terminar o algoritmo. O cromossomo melhor avaliado da população final é o output.

- Na função *rbga.bin*, a condição de encerramento é o número de iterações do algoritmo. Outras condições podem ser a convergência das populações (o momento em que as populações descendentes não diferem significativamente das ascendentes) ou um valor mínimo de avaliação.

5. **Loop** Se a condição de encerramento não for satisfeita, voltar ao passo 2.

Solucionando o problema da mochila

```
modelo <- rbga.bin(size = 7,
                   popSize = 200,
                   iters = 100,
                   mutationChance = 0.01,
                   elitism = T,
                   evalFunc = avaliacao)
```

```r
cat(summary(modelo))
```

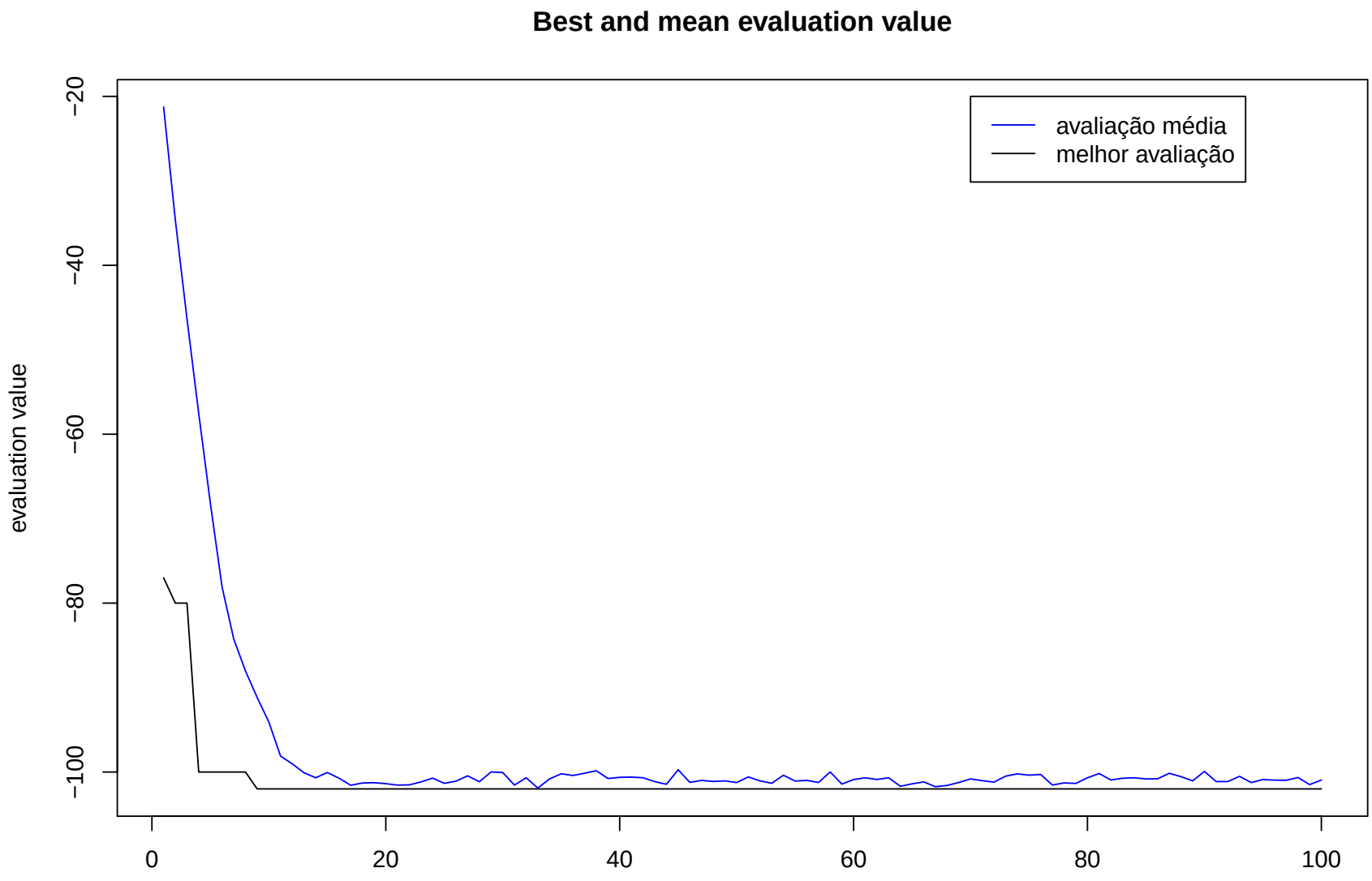
```
## GA Settings
##   Type                  = binary chromosome
##   Population size       = 200
##   Number of Generations = 100
##   Elitism               = TRUE
##   Mutation Chance       = 0.01
##
## Search Domain
##   Var 1 = [,]
##   Var 0 = [,]
##
## GA Results
##   Best Solution : 1 1 0 1 1 1 1
```

```
solucao <- c(1, 1, 0, 1, 1, 1, 1)
```

	item	pontos	peso
1	canivete	10	1
2	feijões	20	5
4	cebolas	2	1
5	saco de dormir	30	7
6	corda	10	5
7	compasso	30	1

```
## pontos: 102 peso: 20
```

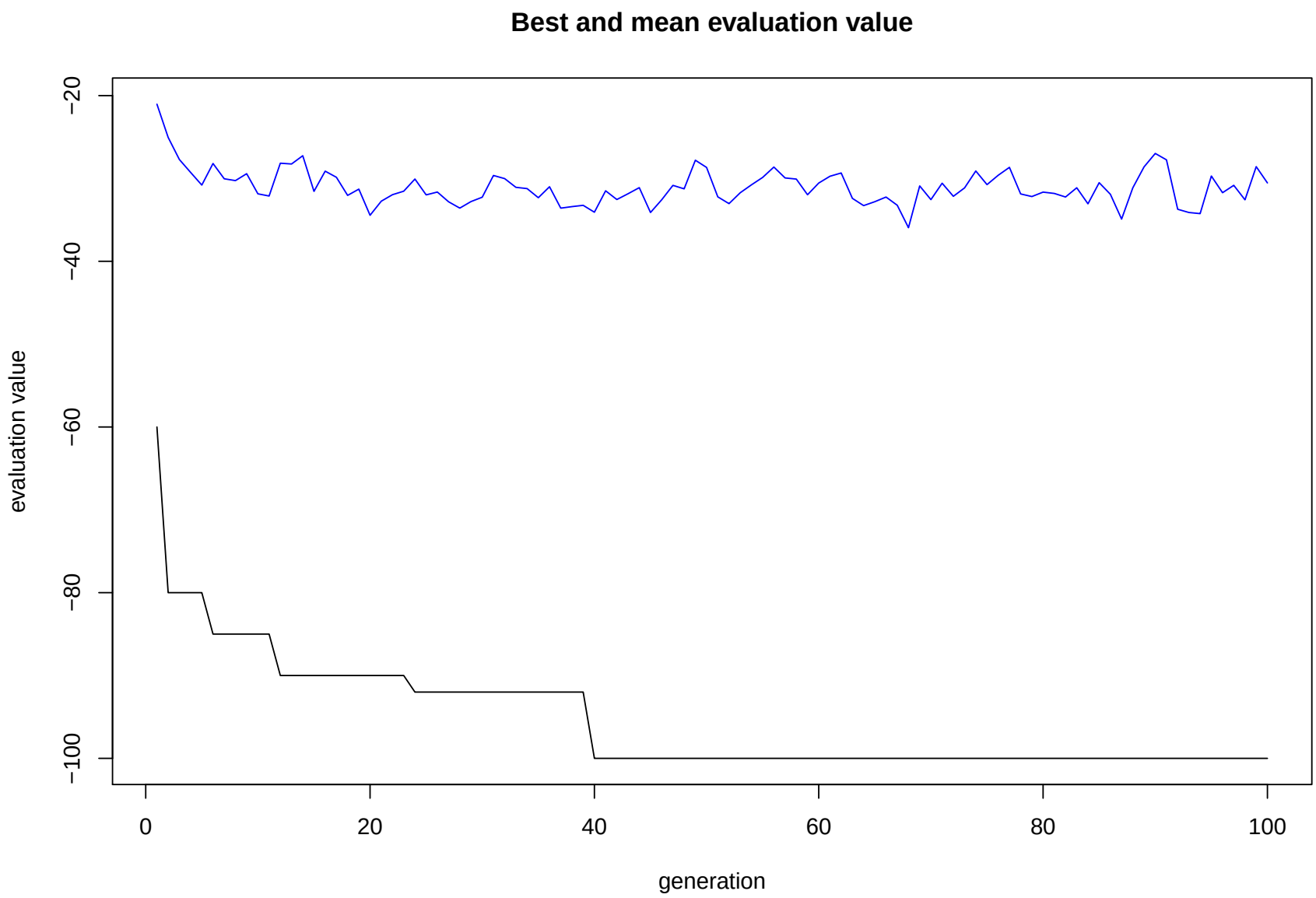
```
plot(modelo, type = "vars")
legend(70, -20,
       c('avaliação média', 'melhor avaliação'),
       col = c('blue', 'black'), lty = c(1, 1))
```



50% de chance de mutação

```
modelo <- rbga.bin(size = 7,
                   popSize = 200,
                   iters = 100,
                   mutationChance = 0.5,
                   elitism = T,
                   evalFunc = avaliacao)
```

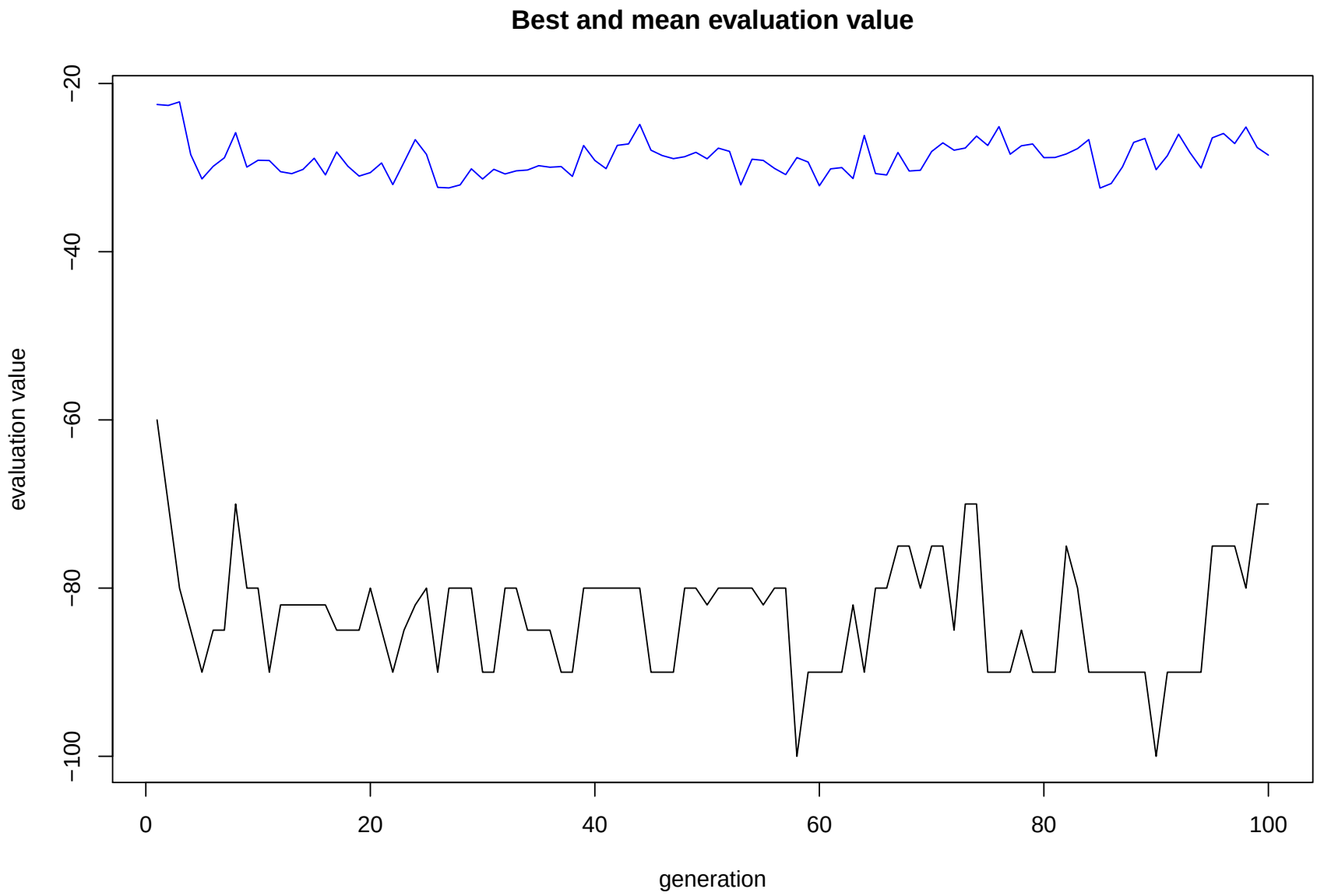
```r
plot(modelo, type = "vars")
```



50% de chance de mutação sem a condição elitismo

```
modelo <- rbga.bin(size = 7,
                   popSize = 200,
                   iters = 100,
                   mutationChance = 0.5,
                   elitism = F,
                   evalFunc = avaliacao)
```

```
plot(modelo, type = "vars")
```



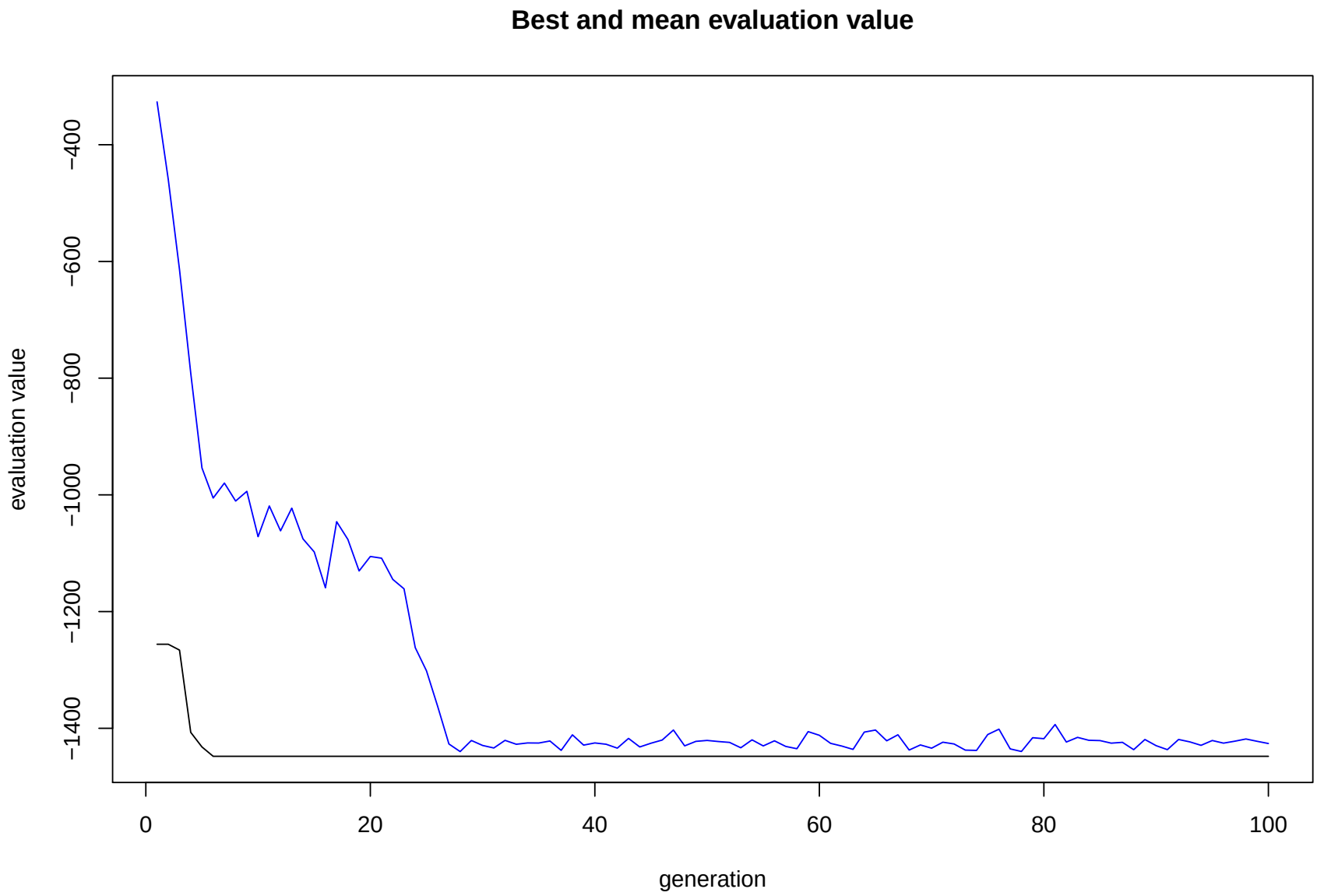
15 itens, capacidade máxima 750

```
pontos <- c(135, 139, 149, 150, 156, 163, 173,
            184, 192, 201, 210, 214, 221, 229, 240)
peso <- c(70, 73, 77, 80, 82, 87, 90, 94, 98,
          106, 110, 113, 115, 118, 120)
dados <- data.frame(pontos, peso)

peso_limite <- 750

modelo <- rbga.bin(size = 15,
                   popSize = 200,
                   iters = 100,
                   mutationChance = 0.01,
                   elitism = T,
                   evalFunc = avaliacao)
```

```r
plot(modelo, type = "vars")
```



Referências

https://www.neuraldesigner.com/blog/genetic_algorithms_for_feature_selection

http://www.obitko.com/tutorials/genetic-algorithms/index.php

https://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html

https://www.r-bloggers.com/genetic-algorithms-a-simple-r-example/

https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-

https://en.wikipedia.org/wiki/Genetic_algorithm

https://en.wikipedia.org/wiki/Knapsack_problem