

# Estudo de Caso 03: Comparação de desempenho de duas configurações de um algoritmo de otimização

Diego Pontes, Elias Vieira, Matheus Bitarães

Março, 2021

## Descrição do problema

Suponha que um pesquisador está interessado em investigar o efeito de duas configurações distintas de um algoritmo em seu desempenho para uma dada classe de problemas de otimização. Como forma de análise deste problema, foi proposta a tarefa de efetuar a comparação experimental de duas configurações em uma classe de problemas, representada por um conjunto de instâncias de teste fornecida. O objetivo deste estudo é responder às seguintes perguntas:

- Há alguma diferença no desempenho médio do algoritmo quando equipado com estas diferentes configurações, para a classe de problemas de interesse?
- Caso haja, qual a melhor configuração em termos de desempenho médio (atenção: quanto menor o valor retornado, melhor o algoritmo), e qual a magnitude das diferenças encontradas?
- Há alguma configuração que deva ser recomendada em relação à outra?

## Introdução

Algoritmos baseados em populações são uma alternativa comum para a solução de problemas de otimização em engenharia. Tais algoritmos normalmente consistem de um ciclo iterativo, no qual um conjunto de soluções-candidatas ao problema são repetidamente sujeitas a operadores de variação e seleção, de forma a promover uma exploração do espaço de variáveis do problema em busca de um ponto ótimo (máximo ou mínimo) de uma dada função-objetivo. Dentre estes algoritmos, um método que tem sido bastante utilizado nos últimos anos é conhecido como evolução diferencial (DE, do inglês differential evolution) [1].

A Evolução Diferencial (ED) é um algoritmo evolutivo que se baseia nos mecanismos de seleção natural e na genética de populações, e utiliza operadores de mutação, cruzamento e seleção para gerar novos indivíduos em busca do mais adaptado [2]. Inicialmente é realizada uma escolha aleatória com distribuição uniforme de uma população composta por  $N_p$  indivíduos, que são denominados vetores, os quais devem cobrir todo o espaço de busca [3].

Após isso, ocorre a mutação, onde estes vetores sofrem modificações, o que faz surgir novos indivíduos, denominados vetores doadores, pela adição da diferença ponderada entre dois indivíduos escolhidos aleatoriamente da população inicial a um terceiro indivíduo que também é escolhido de forma aleatória com distribuição uniforme da população original [4].

Posteriormente ocorre a operação de cruzamento, onde os vetores doadores são combinados com os componentes de um outro vetor escolhido aleatoriamente, chamado de vetor alvo, a fim de gerar o vetor denominado experimental. Este processo é conhecido por cruzamento [5].

Por fim, a seleção é feita comparando o valor de custo do vetor experimental e do vetor alvo, sendo assim, se o custo do vetor experimental for menor que o custo do vetor alvo, o vetor alvo da geração seguinte será o vetor experimental, caso contrário, o vetor alvo da geração seguinte será o vetor alvo da geração atual [6].

O procedimento é finalizado por meio de algum critério de parada, o qual pode ser: um número determinado de iterações consecutivas, um tempo computacional determinado, um número máximo de iterações ou ainda, quando um número máximo de avaliações de indivíduos é atingido [7].

As principais etapas de um algoritmo de evolução diferencial clássico estão mostradas na Figura 1.

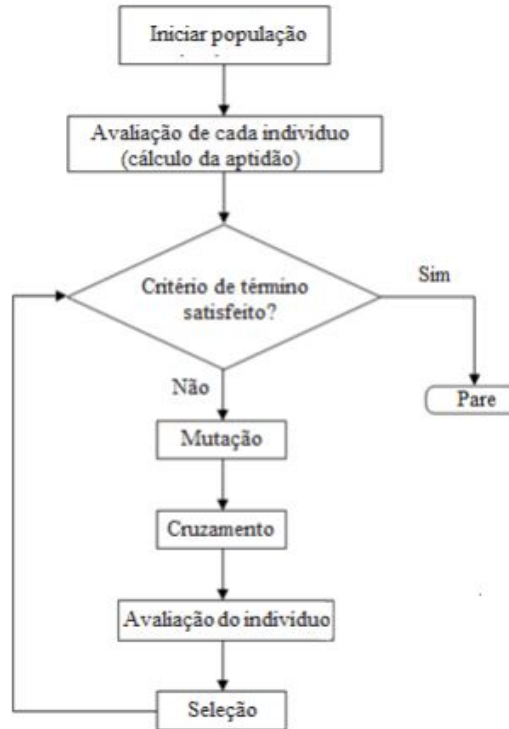


Figure 1: Fluxograma do algoritmo de evolução diferencial clássico

## Design do Experimento

A tarefa para este estudo de caso é a comparação experimental de duas configurações em uma classe de problemas, representada por um conjunto de instâncias de teste. As duas configurações são apresentadas a seguir:

```

## Configuração 1
recpars1 <- list(name = "recombination_blxAlphaBeta", alpha = 0, beta = 0)
mutpars1 <- list(name = "mutation_rand", f = 4)

## Configuração 2
recpars2 <- list(name = "recombination_linear")
mutpars2 <- list(name = "mutation_rand", f = 1.5)
  
```

Além disto, os seguintes parâmetros experimentais foram dados para este estudo:

- Mínima diferença de importância prática (padronizada):  $(d^* = \delta^*/\sigma) = 0.5$
- Significância desejada:  $\alpha = 0.05$
- Potência mínima desejada (para o caso  $d = d^*$ ):  $\pi = 1 - \beta = 0.8$

Para a execução dos experimentos, foram utilizados os pacotes ExpDE e smooF. A classe de funções de interesse para este teste é composta por funções Rosenbrock de dimensão entre 2 e 150 que podem ser .

Para a correta comparação entre duas configurações de algoritmos dentro de uma classe de problemas é necessário que sejam executados os algoritmos em uma quantidade de dimensões representativas, além de que cada instância seja executada um número de vezes suficiente para que se consiga alcançar a potência, significância e importância prática desejadas. Realizar um alto número de execuções em cada dimensão e realizar a coleta para todas as dimensões pode ser computacionalmente pesado ou infactível em alguns casos. Este é um desses casos. Portanto será utilizada a abordagem criada por Campelo e Takahashi, que propõe a estimativa do número mínimo de instâncias (neste caso, dimensões) a serem avaliadas e também propõe uma estratégia para se obter o número mínimo de execuções em cada instância, através de uma avaliação iterativa, enquanto as execuções forem sendo coletadas [8].

Como primeiro passo, deve-se estimar o número de instâncias de acordo com os parâmetros experimentais fornecidos:

O método `calc_instances` do pacote CAISER em linguagem R [9] permite estimar o número de instâncias mínimas necessárias para se comparar múltiplos algoritmos, de modo que os requisitos experimentais sejam atingidos. O parâmetro número de comparações `ncomparisons` é dado pela seguinte equação, onde  $K$  é o número de algoritmos que se deseja comparar:

$$\text{ncomparisons} = \frac{K \times (K - 1)}{2} \quad (1)$$

Como  $K = 2$  para o problema em questão, tem-se que `ncomparisons = 1`.

```
# dados do trabalho
alpha <- 0.05
delta <- 0.5
beta <- 0.2

out <- calc_instances(ncomparisons = 1,
                      d = delta,
                      power = 1 - beta,
                      sig.level = alpha,
                      alternative.side = "two.sided",
                      power.target = "mean")
cat('Número de instâncias necessárias:', out$ninstances)
```

```
## Número de instâncias necessárias: 34
```

Como pode-se concluir pelo resultado da execução da função, o número de instâncias necessárias para se realizar o experimento descrito é 34.

## Coleta dos Dados

Para que seja possível utilizar todo o intervalo de valores possíveis de  $D$  ( $D \in [2, 150]$ ), espaçou-se uniformemente as 34 instâncias:

```
num_dims <- 34
dims = round(linspace(2, 150, num_dims), digits = 0)
print(dims)
```

```
## [1] 2 6 11 15 20 24 29 33 38 42 47 51 56 60 65 69 74 78 83
## [20] 87 92 96 101 105 110 114 119 123 128 132 137 141 146 150
```

Seguindo o Algoritmo 2 proposto por Campelo e Takahashi (2018), deve-se realizar um conjunto de execuções piloto para que se possa obter os dados de média e variância das instâncias e, dessa forma, iniciar-se a execução iterativa até que se obtenha o erro padrão (a estimativa da diferença em performance entre os dois algoritmos) com a precisão pretendida (Algoritmo 1 proposto por Campelo e Takahashi (2018)) [8].

Execução piloto com escolha inicial de 10 execuções e com os 34 blocos, obtidos anteriormente.

```
suppressPackageStartupMessages(library(ExpDE))
suppressPackageStartupMessages(library(smoof))

file_name = "pilot_execution_backup.csv"
num_exec <- 10

if (file.exists(file_name)) {

  # se ja existir uma arquivo com o dados da execução, não é necessário executar novamente. Apenas será
  print("Arquivo de backup encontrado. Recuperando dados ao inves de realizar uma nova execução")
  initial_data <- read.csv(file=file_name, header = TRUE, sep=",")
  dims <- initial_data$dim
  mean_y1 <- initial_data$mean.instance1
  mean_y2 <- initial_data$mean.instance2
  sd_y1 <- initial_data$sd.instance1
  sd_y2 <- initial_data$sd.instance2
} else {

  # Função para execução de n funções de rosenbrock para determinada dimensao
  executeRosenbrock <- function(dim, num_exec) {
    # Função de ronsebrock para determinada dimensao
    fn <- function(X) {
      if(!is.matrix(X)) X <- matrix(X, nrow = 1) # <- if a single vector is passed as X
      Y <- apply(X, MARGIN = 1,
        FUN = smoof::makeRosenbrockFunction(dimensions = dim))
      return(Y)
    }

    # definições dadas no enunciado
    selpars <- list(name = "selection_standard")
    stopcrit <- list(names = "stop_maxeval", maxevals = 5000*dim, maxiter = 100*dim)
    probpars <- list(name = "fn", xmin = rep(-5, dim), xmax = rep(10, dim))
    popsize <- 5*dim

    y1 <- vector(,num_exec)
    y2 <- vector(,num_exec)
    for (i in 1:num_exec){

      # rodando problema para instancia 1
      out <- ExpDE(mutpars = mutpars1,
        recpars = recpars1,
        popsize = popsize,
        selpars = selpars,
        stopcrit = stopcrit,
```

```

        probpars = probpars,
        showpars = list(show.its = "dots", showevery = 20))
y1[i] <- out$Fbest

# rodando problema para instancia 2
out <- ExpDE(mutpars = mutpars2,
            recpars = recpars2,
            popsize = popsize,
            selpars = selpars,
            stopcrit = stopcrit,
            probpars = probpars,
            showpars = list(show.its = "dots", showevery = 20))
y2[i] <- out$Fbest
cat("\n y1", y1[i])
cat("\n y2", y2[i])
}

return(
  data.frame(
    mean1 = mean(y1),
    mean2 = mean(y2),
    sd1 = sd(y1),
    sd2 = sd(y2))
)

# Função para execução de n funções de rosenbrock para uma lista de dimensões
executeRosenbrockForDims <- function(dims, num_exec){
  num_dims <- length(dims)
  mean_y1 <- vector(), num_dims
  mean_y2 <- vector(), num_dims
  sd_y1 <- vector(), num_dims
  sd_y2 <- vector(), num_dims

  for (d in 1:num_dims){
    dim <- dims[d]

    # multiplas execuções
    Y = executeRosenbrock(dim, num_exec)

    cat("dim:", dim, " \n")
    cat("mu1:", Y$mean1, " \n")
    cat("sd1:", Y$sd1, " \n")
    cat("mu2:", Y$mean2, " \n")
    cat("sd2:", Y$sd2, " \n")

    mean_y1[d] <- Y$mean1
    mean_y2[d] <- Y$mean2
    sd_y1[d] <- Y$sd1
    sd_y2[d] <- Y$sd2
  }

  data <- data.frame(
    dim = dims,

```

```

    mean.instance1 = mean_y1,
    mean.instance2 = mean_y2,
    sd.instance1 = sd_y1,
    sd.instance2 = sd_y2)

  return(data)
}

# execução das funções de rosenbrock
initial_data = executeRosenbrockForDims(dims, num_exec)

# escreve no arquivo de backup os resultados coletados
write.csv(initial_data, file = file_name)
}

```

## [1] "Arquivo de backup encontrado. Recuperando dados ao inves de realizar uma nova execução"

```

initial_data$n1 <- rep(num_exec, num_dims)
initial_data$n2 <- rep(num_exec, num_dims)

```

Após esta execução, pode-se prosseguir com o Algoritmo 1, disponível em [8]. O algoritmo proposto irá executar as instâncias até que o limite de precisão  $se^*$  seja atingido **ou** que o número máximo de execuções  $n_{max}$  seja alcançado. O valor de  $n_{max}$  foi definido como 50 (máximo de 25 execuções de cada instância a ser comparada).

```

file_name = "execution_backup.csv"
num_exec <- 10

if (file.exists(file_name)) {

  # se ja existir uma arquivo com o dados da execução, não é necessário executar novamente. Apenas será
  print("Arquivo de backup encontrado. Recuperando dados ao inves de realizar uma nova execução")
  data <- read.csv(file=file_name, header = TRUE, sep=",")
} else {
  data <- initial_data
  se_opt <- 0.05 # se pretendido
  n_max <- 50 # máximo número de execuções

  # loop de dimensões
  for (d in 1:num_dims) {
    dim <- data$dim[d]
    mu1 <- data$mean.instance1[d]
    mu2 <- data$mean.instance2[d]
    sd1 <- data$sd.instance1[d]
    sd2 <- data$sd.instance2[d]
    n1 <- 10 # número de execuções que já foram realizadas
    n2 <- 10

    # Função de rosenbrock para determinada dimensao
    fn <- function(X) {
      if(!is.matrix(X)) X <- matrix(X, nrow = 1) # <- if a single vector is passed as X
      Y <- apply(X, MARGIN = 1,
        FUN = smooof::makeRosenbrockFunction(dimensions = dim))
    }
  }
}

```

```

    return(Y)
}

# definições dadas no enunciado
selpars <- list(name = "selection_standard")
stopcrit <- list(names = "stop_maxeval", maxevals = 5000*dim, maxiter = 100*dim)
probpars <- list(name = "fn", xmin = rep(-5, dim), xmax = rep(10, dim))
popsize <- 5*dim

# calculo do se
se <- sqrt(sd1^2/n1 + sd2^2/n2)

cat("\n se inicial:", se)

# loop de execuções
while (se > se_opt && n1 + n2 < n_max){
  ropt <- sd1/sd2
  if (n1/n2 < ropt){

    # executa algoritmo 1
    out <- ExpDE(mutpars = mutpars1,
      recpars = recpars1,
      popsize = popsize,
      selpars = selpars,
      stopcrit = stopcrit,
      probpars = probpars,
      showpars = list(show.its = "dots", showevery = 20))
    y1 <- out$Fbest

    # atualiza parâmetros
    mu_ <- (mu1*n1 + y1)/(n1+1)
    sd_ <- sqrt(((n1-1)*sd1^2 + (y1 - mu_)*(y1 - mu1))/n1)
    mu1 <- mu_
    sd1 <- sd_
    n1 <- n1 + 1
  } else {

    # executa algoritmo 2
    out <- ExpDE(mutpars = mutpars2,
      recpars = recpars2,
      popsize = popsize,
      selpars = selpars,
      stopcrit = stopcrit,
      probpars = probpars,
      showpars = list(show.its = "dots", showevery = 20))
    y2 <- out$Fbest

    # atualiza parâmetros
    mu_ <- (mu2*n2 + y2)/(n2+1)
    sd_ <- sqrt(((n2-1)*sd2^2 + (y2 - mu_)*(y2 - mu2))/n2)
    mu2 <- mu_
    sd2 <- sd_
    n2 <- n2 + 1
  }
}

```

```

    }

    # atualiza se
    se <- sqrt(sd1^2/n1 + sd2^2/n2)
    cat("se:", se, " \n")
    cat("dim:",dim, " \n")
    cat("se:",se, " \n")
    cat("mu1:",mu1, " \n")
    cat("sd1:",sd1, " \n")
    cat("mu2:",mu2, " \n")
    cat("sd2:",sd2, " \n")
    cat("n1:",n1, " \n")
    cat("n2:",n2, " \n")
  }

  data$mean.instance1[d] <- mu1
  data$sd.instance1[d] <- sd1
  data$n1[d] <- n1
  data$mean.instance2[d] <- mu2
  data$sd.instance2[d] <- sd2
  data$n2[d] <- n2
}

# escreve dados no arquivo de backup
write.csv(data, file = file_name)
}

```

## [1] "Arquivo de backup encontrado. Recuperando dados ao inves de realizar uma nova execução"

Abaixo pode ser visto os dados coletados bem como o número de execuções de cada instância:

data

```

##      X dim mean.instance1 mean.instance2 sd.instance1 sd.instance2 n1 n2
## 1    1  2  4.801098e-03  3.155444e-31 7.030550e-03 9.978389e-31 10 10
## 2    2  6  1.556532e+02  1.036112e-09 7.658966e+01 3.127060e-09 40 10
## 3    3 11  2.859544e+04  1.822307e+00 1.237948e+04 1.358104e+00 40 10
## 4    4 15  1.150719e+05  1.013441e+01 2.877621e+04 6.471559e-01 40 10
## 5    5 20  2.758080e+05  2.590763e+01 4.312319e+04 2.147535e+00 40 10
## 6    6 24  4.131276e+05  9.529037e+01 7.175148e+04 2.003774e+01 40 10
## 7    7 29  5.785252e+05  4.393029e+02 6.851799e+04 1.306340e+02 40 10
## 8    8 33  7.468094e+05  1.507542e+03 7.125151e+04 3.606764e+02 40 10
## 9    9 38  9.641466e+05  5.266483e+03 9.524623e+04 1.343672e+03 40 10
## 10  10 42  1.118477e+06  1.317862e+04 1.334866e+05 3.233785e+03 40 10
## 11  11 47  1.348605e+06  3.173632e+04 1.344165e+05 8.002613e+03 40 10
## 12  12 51  1.565964e+06  5.195229e+04 1.660361e+05 1.055687e+04 40 10
## 13  13 56  2.214173e+06  8.669291e+04 6.134398e+05 2.023011e+04 40 10
## 14  14 60  2.649036e+06  1.655832e+05 6.822616e+05 6.491125e+04 40 10
## 15  15 65  3.588845e+06  2.408687e+05 6.535566e+05 7.678247e+04 40 10
## 16  16 69  3.975597e+06  2.937040e+05 4.908136e+05 6.173127e+04 40 10
## 17  17 74  4.546692e+06  4.720125e+05 5.415539e+05 7.663386e+04 40 10
## 18  18 78  4.942110e+06  5.920831e+05 4.598277e+05 1.381192e+05 38 12

```



```
## 19 19 83 5.281279e+06 9.235232e+05 4.327631e+05 1.645013e+05 35 15
## 20 20 87 5.854144e+06 9.874201e+05 5.183355e+05 2.241751e+05 35 15
## 21 21 92 6.198234e+06 1.256315e+06 5.544042e+05 2.842339e+05 33 17
## 22 22 96 6.707469e+06 1.709252e+06 6.013253e+05 2.142376e+05 37 13
## 23 23 101 6.836463e+06 2.029101e+06 6.324840e+05 3.475469e+05 32 18
## 24 24 105 7.311739e+06 2.186746e+06 6.124760e+05 4.924547e+05 28 22
## 25 25 110 7.650827e+06 2.673647e+06 6.952003e+05 4.757902e+05 29 21
## 26 26 114 7.992897e+06 2.930139e+06 7.183937e+05 6.979160e+05 25 25
## 27 27 119 8.708038e+06 3.477332e+06 5.661883e+05 5.829719e+05 24 26
## 28 28 123 8.719999e+06 3.847995e+06 7.927956e+05 7.631544e+05 25 25
## 29 29 128 9.496021e+06 4.170314e+06 6.091188e+05 6.410488e+05 24 26
## 30 30 132 9.748881e+06 4.467374e+06 7.536504e+05 7.263416e+05 26 24
## 31 31 137 1.017978e+07 5.102232e+06 7.409436e+05 7.697012e+05 24 26
## 32 32 141 1.040743e+07 5.421776e+06 8.120632e+05 1.037953e+06 22 28
## 33 33 146 1.132399e+07 5.986549e+06 6.043259e+05 6.611940e+05 25 25
## 34 34 150 1.164427e+07 5.917618e+06 8.842647e+05 1.077318e+06 23 27
```

Podemos perceber que a maioria dos blocos utilizou o orçamento máximo de execuções  $n_{max}$ , que era de 50 execuções. Mas a distribuição deste número de execuções é balanceada de uma forma que minimize o limite de precisão *se*.

## Teste de Hipóteses

Optou-se por utilizar o ANOVA para realizar a comparação das amostras. Será realizada uma validação das premissas *a posteriori*.

```
d <- data.frame(dim = rep(data$dim,2), config = c(rep(1,num_dims), rep(2,num_dims)), Y = c(data$mean.in

for (i in 1:2){
  d[, i] <- as.factor(d[, i])
}

model <- aov(Y~dim+config, data = d)
summary(model)
```

```
##          Df    Sum Sq  Mean Sq F value    Pr(>F)
## dim       33 5.519e+14 1.672e+13    7.57 3.95e-08 ***
## config     1 1.718e+14 1.718e+14   77.78 3.41e-10 ***
## Residuals  33 7.290e+13 2.209e+12
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Para a realização do teste ANOVA, é necessário que as seguintes premissas sejam cumpridas [10]:

- As amostras devem ser independentes;
- As amostras devem apresentar distribuição normal;
- As variâncias podem ser consideradas iguais (homocedasticidade).

**Verificação da independência:** Por ser dois algoritmos diferentes, tem-se a independência das amostras.

**Verificação de Normalidade:** Para uma avaliação estatística sobre a validação da hipótese de uma distribuição normal para as amostras, pode-se usar o teste Shapiro-Wilk, onde têm-se as seguintes hipóteses [11]:

$$\begin{cases} H_0 : \text{A amostra provém de uma população com distribuição normal} \\ H_1 : \text{A amostra não provém de uma população com distribuição normal} \end{cases}$$

```
shapiro.test(model$residuals)
```

```
##
## Shapiro-Wilk normality test
##
## data: model$residuals
## W = 0.89306, p-value = 2.671e-05
```

Como interpretação do teste, temos que se o p-valor  $< 0.05$  ( $\alpha$ ), deve-se rejeitar a hipótese nula, ou seja, os dados não possuem distribuição normal [11], caso contrário, não há evidências para se rejeitar a hipótese nula. Como pode ser visto pelo teste acima, não se pode assumir normalidade para este modelo e portanto o ANOVA não pode ser utilizado para estes dados.

Pode-se tentar realizar uma transformação logaritmica nos dados a fim de se conseguir resíduos normalizados no ANOVA:

```
model_log <- aov(log(Y)~dim+config, data = d)
summary(model_log)
```

```
##           Df Sum Sq Mean Sq F value    Pr(>F)
## dim         33   6318    191.5     2.891 0.00154 **
## config        1    532    531.6     8.027 0.00780 **
## Residuals    33   2186     66.2
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Validação de normalidade dos resíduos do modelo com escala logaritmica:

```
shapiro.test(model_log$residuals)
```

```
##
## Shapiro-Wilk normality test
##
## data: model_log$residuals
## W = 0.57099, p-value = 8.563e-13
```

Como pode-se observar pelo resultado do teste de Shapiro-Wilk, não podemos assumir normalidade para as amostras em escala logaritmica. O que implica que não podemos realizar o ANOVA.

Como alternativa, tem-se o teste de Friedman com as seguintes hipóteses:

$$\begin{cases} H_0 : \text{A média das duas populações são iguais} \\ H_1 : \text{A média das duas populações são diferentes} \end{cases}$$

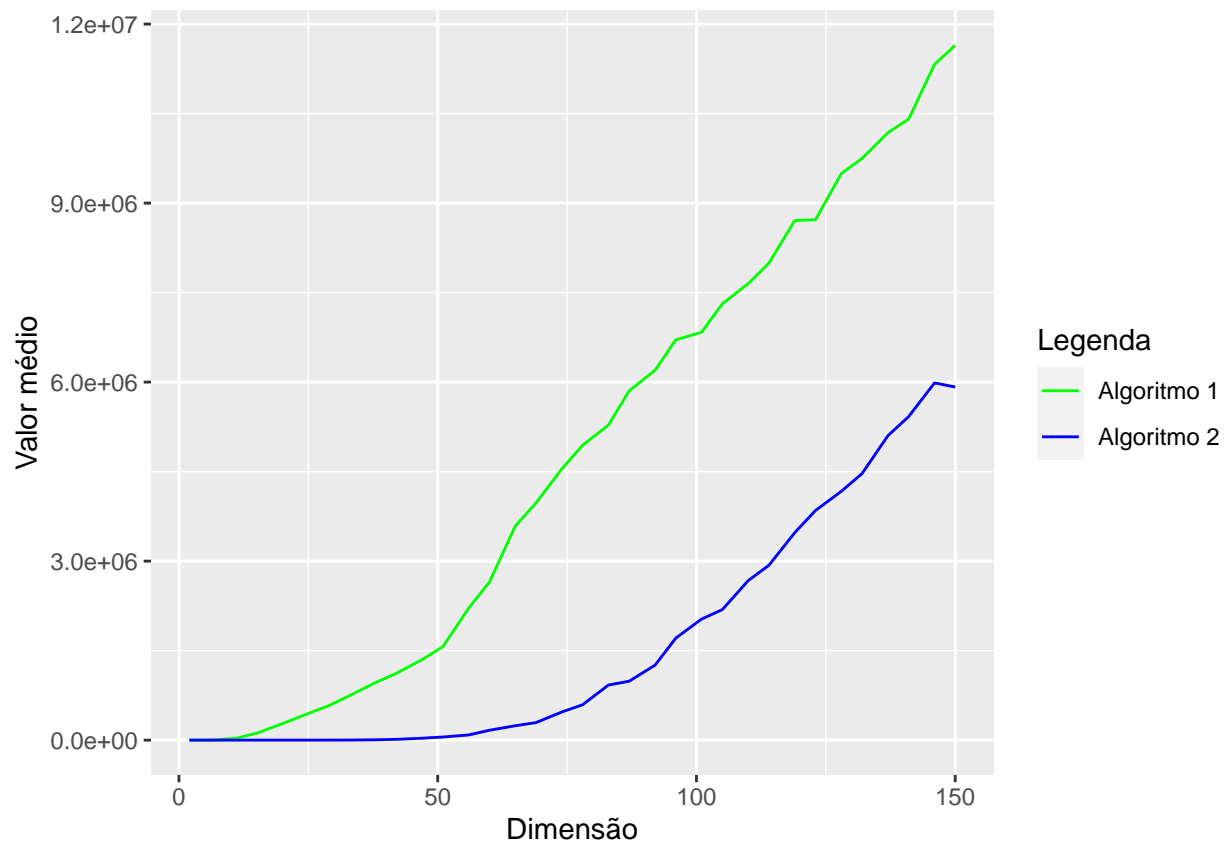
```
y <- c(data$mean.instance1, data$mean.instance2)
group <- as.factor(c(rep(1, length(data$mean.instance1)), rep(2, length(data$mean.instance2))))
datatable = cbind(y, group)
friedman.test(datatable)
```

```
##
## Friedman rank sum test
##
## data: datatable
## Friedman chi-squared = 52.941, df = 1, p-value = 3.437e-13
```

Como interpretação do teste, temos que se o p-valor  $< 0.05$  ( $\alpha$ ), deve-se rejeitar a hipótese nula, ou seja, as médias dos grupos de amostras não são homogêneas, caso contrário, não há evidências para se rejeitar a hipótese nula. Portanto, analisando os resultados do teste, pode-se considerar que a média dos grupos não são homogêneas.

### explicação grafica

```
merged <- rbind(graf1, graf2)
p <- ggplot(merged, aes(x = var, y = value, color = label)) + geom_line()
p + labs(x = "Dimensão", y = "Valor médio") +
  guides(color=guide_legend(title="Legenda")) +
  scale_color_manual(values = c("green", "blue"))
```



falta: - coleta e tabulação dos dados - testes das hipoteses - estimacao da magnitude da diferença entre os metodos - verificação das premissas dos testes - conclusoes - possiveis limitações do estudo e sugestoes de melhoria

### Atividades dos membros

Diego

Elias

Matheus

Todos

## Referências Bibliográficas

- [1] Storn R and Price K. Differential Evolution: A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):341–359, 1997.
- [2] N. C. ROCHA and S. F. P SARAMAGO. Estudo de algumas Estratégias da Evolução Diferencial. *Congresso de matemática aplicada e computacional, Uberlandia*, 2011.
- [3] D. D. FIGUEIREDO, L. V. SOUZA, and A. S. ARAÚJO. Algoritmo Evolução Diferencial Adaptado para o Problema das P-Mediana. *Congresso de matemática aplicada e computacional, Curitiba*, 2014.
- [4] A. L. M. PAIVA. Aplicação do método de evolução diferencial à otimização de um ciclo de refrigeração por compressão de vapor de dois estágios através da análise energética. Dissertacao de mestrado, Universidade Federal do Rio de Janeiro, 2011.
- [5] G. T. S. OLIVEIRA. Estudos e aplicações da evolução diferencial. Dissertacao de mestrado, Faculdade de Engenharia Mecânica, 2006.
- [6] A. S. M. LACERDA. Proposta de um algoritmo evolucionário nebuloso para a solução de problemas de otimização multiobjectivo. Dissertacao de mestrado, Universidade Federal de Minas Gerais, 2010.
- [7] R. R. L. ROSÁRIO. Algoritmos evolutivos adaptativos para problemas de programação de pessoal. Tese de doutorado, Universidade Federal de Santa Catarina, 2011.
- [8] Felipe Campelo and Fernanda Takahashi. Sample size estimation for power and accuracy in the experimental comparison of algorithms. *Journal of Heuristics*, 25(2):305–338, 2019.
- [9] Felipe Campelo. CAISer. <https://github.com/fcampelo/CAISer/blob/master/.gitignore>.
- [10] "Amanda C. Reiter, Ana M. Barreto, Clara R. Pires, Daniel Gonçalves-Souza, Danilo S.S. Lima, and Rafaela Lampa". "Tutorial ANOVA". [https://rstudio-pubs-static.s3.amazonaws.com/544630\\_7849ef9441ae454e831295b7e2de2d52.html](https://rstudio-pubs-static.s3.amazonaws.com/544630_7849ef9441ae454e831295b7e2de2d52.html), 2019. Acesso em 7 de Fevereiro de 2021.
- [11] Como realizar teste de normalidade no r? <https://rpubs.com/paternogbc/46768>. Acesso em 18 de Janeiro de 2021.