Faculdade de Economia, Administração e Contabilidade

Machine Learning

Aplicado a Negócios

Laboratório de Análise de Dados

Algoritmo Genético - GA

Prof. Antonio Geraldo da Rocha Vidal

2023

Sumário

Você deve entregar um relatório com os resultados das etapas elaboradas neste laboratório no **e-Disciplinas**, para formatá-lo siga estas orientações:

1. Crie um documento Word e identifique-o com o nome do laboratório, data de elaboração e o seu nome ou do grupo que o elaborou;
2. Crie um tópico para cada resultado que você considerar relevante (manipulação de dados ou resultado de algum processamento) identificando-o com um título e uma breve explicação. Os resultados podem ser imagens de gráficos gerados ou de listas de valores ou dados de resultados obtidos. Não devem ser incluídos os *scripts* ou instruções de processamento utilizados, inclua apenas os resultados que você considerar relevantes.
3. No final do relatório crie um último tópico denominado “Conclusões” e elabore comentários, sugestões e conclusões sobre o que você pode aprender com a elaboração deste laboratório.

Esta apostila introdutória pode conter erros, falhas ou imprecisões. Se você identificar algum problema por favor informe através do e-mail vidal@usp.br para que a correção possa ser providenciada.

Esta apostila não é autoral, tem objetivo estritamente didático como material de apoio para a disciplina. Foi desenvolvida através da compilação dos diversos textos e materiais citados na bibliografia.

 Obrigado!

Laboratório GA – O Problema da Mochila

Introdução

Algoritmo genético (GA) é uma busca heurística. Os GAs podem gerar um grande número possíveis soluções e usá-las para evoluir para uma aproximação do melhor modelo de solução. Por meio deste processo, imita a evolução que ocorre na natureza.

O GA gera uma população, os indivíduos nessa população (muitas vezes chamados de cromossomos) têm um determinado estado. Uma vez gerada a população, o estado desses indivíduos é avaliado em seu valor. Os melhores indivíduos são então tomados e combinados (ou cruzados) – a fim de, esperançosamente, gerar filhos "melhores" – para formar a nova população. Em alguns casos, os melhores indivíduos da população são preservados para garantir "bons indivíduos" na nova geração (isso é chamado de elitismo).

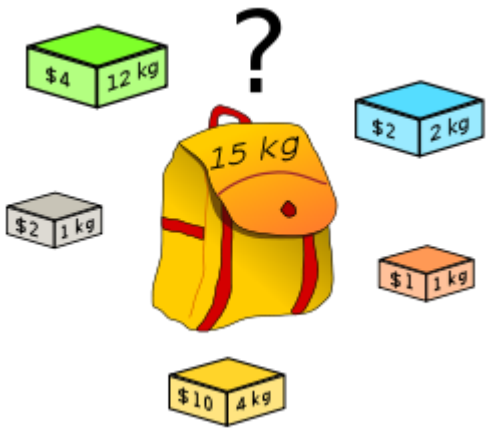
O site GA de Marek Obitko tem um ótimo tutorial para pessoas sem conhecimento prévio sobre este assunto.

Para explicar com um o exemplo, usaremos uma versão do problema da Mochila.

O problema da mochila é um problema de otimização combinatória: Dado um conjunto de itens, cada um com um peso e um valor, determine o número de cada item a incluir em uma coleção para que o peso total seja menor ou igual a um determinado limite e o valor total seja o maior possível.

O problema da mochila deriva seu nome do problema enfrentado por alguém que possui uma mochila de tamanho fixo e deve preenchê-la com os itens mais valiosos (mais úteis) para realizar uma viagem. Este tipo de problema muitas vezes surge na alocação de recursos, onde os tomadores de decisão têm que escolher entre um conjunto de projetos ou tarefas não divisíveis sob um orçamento fixo ou restrição de tempo, respectivamente.

O problema da mochila tem sido estudado por mais de um século, com trabalhos iniciais datando de 1897. O nome "problema da mochila" remonta aos primeiros trabalhos do matemático Tobias Dantzig (1884-1956), e refere-se ao problema comum de embalar os itens mais valiosos ou úteis sem sobrecarregar a bagagem.



Problemas de mochila aparecem em processos de tomada de decisão do mundo real em uma ampla variedade de campos, como encontrar a maneira reduzir o desperdício matérias-primas, seleção de investimentos e portfólios, seleção de ativos para securitização apoiada por ativos, e geração de chaves para sistemas de criptografia.

Uma aplicação inicial de algoritmos de mochila foi na construção e pontuação de testes nos quais os respondentes do teste devem escolher quais questões devem ser respondidas em um teste para obter o maior resultado. Para pequenos exemplos, é um processo bastante simples para fornecer aos respondentes tal escolha. Por exemplo, se um exame contém 10 questões, cada uma no valor de 10 pontos, o examinado precisa apenas responder 10 perguntas para alcançar a pontuação máxima possível de 100 pontos. No entanto, em testes com uma distribuição heterogênea de valores de pontos por questão, é mais difícil fornecer escolhas. Feuerman e Weiss propuseram um sistema no qual os alunos recebem um teste heterogêneo com um total de 125 pontos possíveis. Os alunos são convidados a

responder todas as perguntas com o melhor de suas habilidades. Dos possíveis subconjuntos de problemas cujos valores totais de pontos somam 100, um algoritmo de mochila determinaria qual subconjunto dá a cada aluno a maior pontuação possível.

Um estudo de 1999 do Repositório de Algoritmos da Universidade de Stony Brook mostrou que, de 75 problemas algorítmicos, o problema da mochila foi o 19º mais popular.

O Problema da Mochila é um dos famosos problemas de otimização, especificamente na otimização combinatória. A motivação desse problema vem quando alguém precisa maximizar sua capacidade da mochila — daí o nome — com os itens mais valiosos possíveis. Existem muitas abordagens para resolver esse problema, mas neste texto, vamos dar-lhe um exemplo para resolver este problema usando a abordagem algoritmo genético em R.

O Problema da Mochila

Neste texto, o problema da mochila que tentaremos resolver é o problema da mochila 0-1. Dado um conjunto de **n** itens numerados de 1 a n, cada um com **w_i** de peso e um valor **v_i**. Suponha que cada item seja restrito a 1, ou seja, o item está incluído na mochila ou não. Aqui queremos maximizar a função objetivo (ou seja, os valores dos itens na mochila):

$$\sum_{i=1}^{n} v_i x_i$$

Onde a função objetiva está sujeita à função de restrição:

$$\sum_{i=1}^{n} w_i x_i, x_i \in \{0, 1\}$$

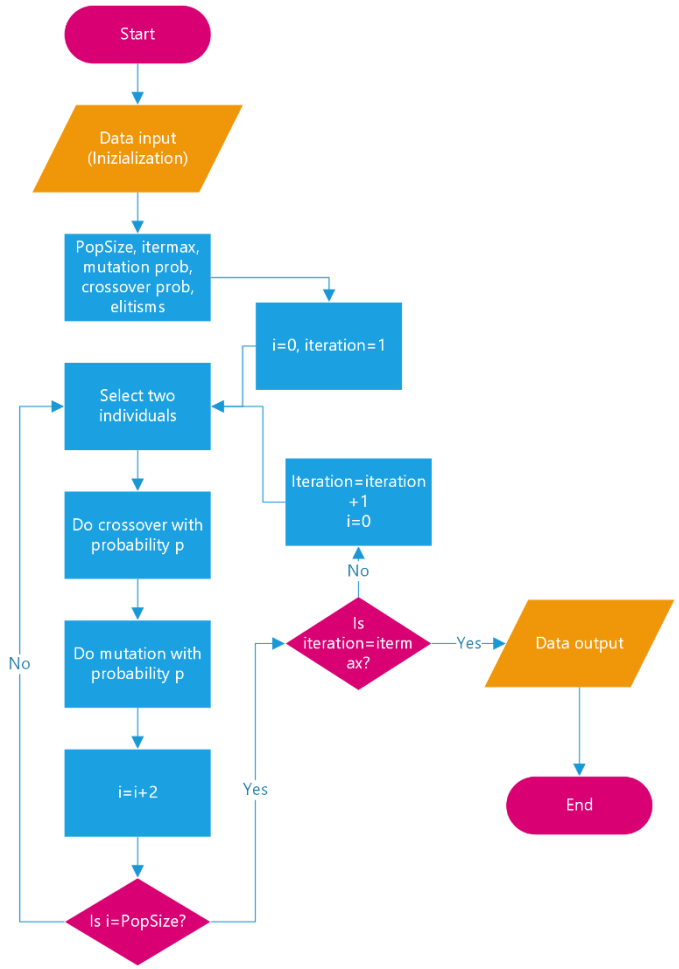
Conceitos de Algoritmo Genético

O Algoritmo Genético é um dos algoritmos de otimização baseados no conceito de evolução por seleção natural. Como proposto por Charles Darwin, a evolução por seleção natural é o mecanismo de quantas variedades de seres vivos se adaptarão ao ambiente para sobreviver através de dois princípios: seleção natural e mutação. Com base nesse conceito, o objetivo do Algoritmo Genético é obter as soluções ideais da função objetivo selecionando a melhor ou mais apta solução ao lado da rara e aleatória ocorrência de mutação. O algoritmo em si pode ser explicado da seguinte forma.

Fluxograma do algoritmo genético

1. Inicialize os dados e/ou a função que vamos otimizar.
2. Inicialize o tamanho populacional, o número máximo de iteração (número de gerações), probabilidade de crossover, probabilidade de mutação e o número de elitismo (o indivíduo melhor ou mais apto que não sofrerá mutação).
3. Da população, selecione dois indivíduos e realize o cruzamento entre dois indivíduos com probabilidade p.
4. Em seguida, realize mutação entre dois indivíduos com probabilidade p (geralmente a probabilidade de mutação é realmente baixa).
5. Repita os passos 3 e 4 até que todos os indivíduos de uma geração sejam treinados. Esses todos os indivíduos serão usados para treinar a próxima geração até que o número de gerações esteja chegando ao limite.

Para dar-lhe uma melhor compreensão do algoritmo genético, vamos pular para o estudo de caso onde usaremos o algoritmo genético para resolver o problema da mochila em R.



Resolvendo problema de mochila usando Algoritmo Genético

Suponha que você queira caminhar com seus amigos, e você tem os itens que você pode usar para caminhadas, com o peso (em quilogramas) e ponto de sobrevivência de cada item, respectivamente, da seguinte forma.

Item para Mochila	Peso (kg)	Pontos (sobrevivência)
Casaco para Chuva	2	5
Canivete	1	3
Água Mineral	6	15
Luvas	1	5
Saco de Dormir	4	6
Barraca	9	18
Fogão Portátil	5	8
Comida Enlatada	8	20
Lanches	3	8

A imagem disponível, com o peso e o ponto de sobrevivência para cada item, respectivamente.

Suponha também que você tenha uma mochila que possa conter itens com capacidade máxima de 25 kg, onde você só pode levar uma cópia para cada item. O objetivo é: você deseja maximizar sua capacidade

de mochila, maximizando seus pontos de sobrevivência também. A partir da instrução do problema, podemos definir o peso dos itens como a função de restrição, enquanto os pontos de sobrevivência acumulados dos itens na mochila como a função objetivo.

Para a implementação da solução, aqui usamos a biblioteca em R criada por Luca Scrucca [2]. Primeiro, precisamos inserir os dados e os parâmetros que usamos escrevendo essas linhas de código.`GA`

```
# Instalação do Pacote
install.packages("GA")
library(GA)
```

Dados do problema:

```
# Dados do Problema
item <- c('casaco de chuva','canivete','água
mineral','luvas','saco de dormir','barraca','fogão
portátil','comida enlatada','lanches')
peso <- c(2,1,6,1,4,9,5,8,3)
sobrevivencia <- c(5,3,15,5,6,18,8,20,8)
dados=data.frame(item,peso,sobrevivencia)
peso_maximo = 25
View(dados)
```

	item	peso	sobrevivencia
1	casaco de chuva	2	5
2	canivete	1	3
3	água mineral	6	15
4	luvas	1	5
5	saco de dormir	4	6
6	barraca	9	18
7	fogão portátil	5	8
8	comida enlatada	8	20
9	lanches	3	8

Para lhe dar uma melhor compreensão do algoritmo genético neste problema, suponha que só trazemos um canivete, água mineral e lanches em sua mochila, inicialmente. Podemos escrevê-lo como o "cromossomo", e como o problema que queremos resolver é um problema de mochila de 0 a 1, então a partir dessas linhas de códigos abaixo, 1 significa que trazemos o item, enquanto 0 significa que deixamos o item. Podemos ver o resultado da seguinte forma:

```
# Conteúdo da Mochila: 1 significa que colocamos o item
# Conteúdo da Mochila: 0 significa que não colocamos o item
cromossomas <- c(0,1,1,0,0,0,0,0,1)
dados[cromossomas==1,]
```

```
          item peso sobrevivencia
2     canivete    1            3
3 água mineral    6           15
9      lanches    3            8
```

Em seguida, criamos a função objetivo que queremos otimizar com a função de restrição escrevendo essas linhas de código abaixo. A função que usaremos na função da biblioteca GA será **fitness**.

```
# Criação da Função de Otimização
fitness=function(x)
{
  pontos=x%*%dados$sobrevivencia
  pesos=x%*%dados$peso
  if(pesos>peso_maximo)
  {
    return(0)
  }
  else
  {
    return(pontos)
  }
}
```

Agora aqui está a parte interessante: o processo de otimização usando o algoritmo genético. Suponhamos que queremos criar no máximo 30 gerações e 50 indivíduos para o processo de otimização. Para reprodutibilidade, escrevemos o argumento e mantemos a melhor solução.

```
GA=ga(type='binary',fitness=fitness,nBits=nrow(dados),maxiter=30,popSi
ze=50,seed=1234,keepBest=TRUE)
```

Ao executar a linha de código acima, podemos alcançar o resultado da seguinte forma.

```
GA=ga(type='binary',fitness=fitness,nBits=nrow(data),maxiter=
30,popSize=50,seed=1234,keepBest=TRUE)
summary(GA)
plot(GA)
```

Ao executar as linhas de código acima, podemos alcançar o resultado da seguinte forma.

```
> GA=ga(type='binary',fitness=fitness,nBits=nrow(dados),maxiter=30,popSize=50,seed=1234,keepBest=TRUE)
GA | iter = 1 | Mean = 31.92 | Best = 61.00
GA | iter = 2 | Mean = 31.32 | Best = 61.00
GA | iter = 3 | Mean = 33.08 | Best = 61.00
GA | iter = 4 | Mean = 36.14 | Best = 61.00
GA | iter = 5 | Mean = 42.42 | Best = 61.00
GA | iter = 6 | Mean = 36.56 | Best = 61.00
GA | iter = 7 | Mean = 37.32 | Best = 61.00
GA | iter = 8 | Mean = 38.18 | Best = 61.00
GA | iter = 9 | Mean = 39.02 | Best = 61.00
GA | iter = 10 | Mean = 38.92 | Best = 61.00
GA | iter = 11 | Mean = 37.54 | Best = 61.00
GA | iter = 12 | Mean = 35.14 | Best = 61.00
GA | iter = 13 | Mean = 36.28 | Best = 61.00
GA | iter = 14 | Mean = 40.82 | Best = 61.00
GA | iter = 15 | Mean = 44.26 | Best = 61.00
GA | iter = 16 | Mean = 41.62 | Best = 61.00
GA | iter = 17 | Mean = 38.66 | Best = 61.00
GA | iter = 18 | Mean = 36.24 | Best = 61.00
GA | iter = 19 | Mean = 43 | Best = 61
GA | iter = 20 | Mean = 43.48 | Best = 61.00
GA | iter = 21 | Mean = 43.08 | Best = 61.00
GA | iter = 22 | Mean = 44.88 | Best = 61.00
GA | iter = 23 | Mean = 46.84 | Best = 61.00
GA | iter = 24 | Mean = 46.8 | Best = 61.0
GA | iter = 25 | Mean = 42.62 | Best = 61.00
GA | iter = 26 | Mean = 46.52 | Best = 61.00
GA | iter = 27 | Mean = 46.14 | Best = 61.00
GA | iter = 28 | Mean = 43.8 | Best = 61.0
GA | iter = 29 | Mean = 46.16 | Best = 61.00
GA | iter = 30 | Mean = 42.6 | Best = 61.0
```

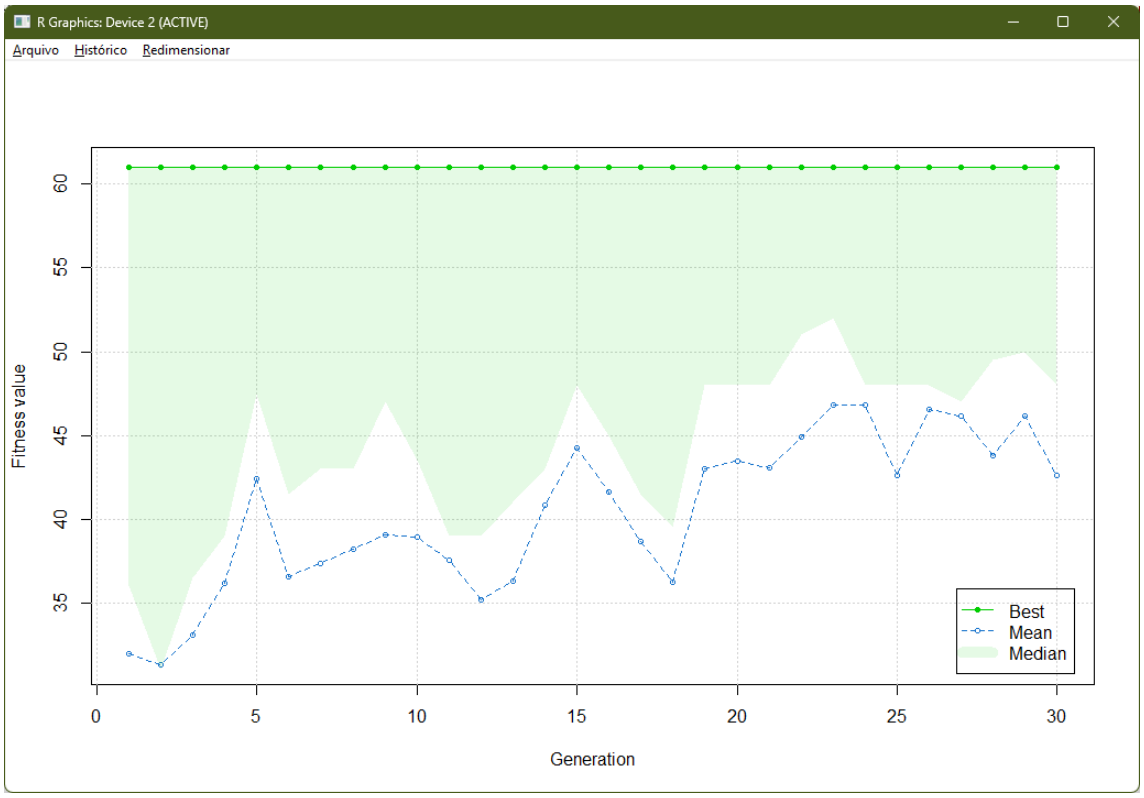
```
summary(GA)
```

```
> summary(GA)
-- Genetic Algorithm -------------------

GA settings:
Type                  =  binary
Population size       =  50
Number of generations =  30
Elitism               =  2
Crossover probability =  0.8
Mutation probability  =  0.1

GA results:
Iterations             = 30
Fitness function value = 61
Solution =
     x1 x2 x3 x4 x5 x6 x7 x8 x9
[1,]  1  0  1  1  0  0  1  1  1
```

```
plot(GA)
```



Pelo resultado acima, podemos ver que o desempenho de cada indivíduo está aumentando a cada geração. Podemos vê-lo a partir do valor de aptidão (ou seja, os pontos de sobrevivência neste caso) média e mediana que tende a aumentar a cada geração. Vamos tentar treiná-lo novamente, mas com mais gerações. Inicialmente GA2 com 50 gerações e depois GA3 com 100 gerações.

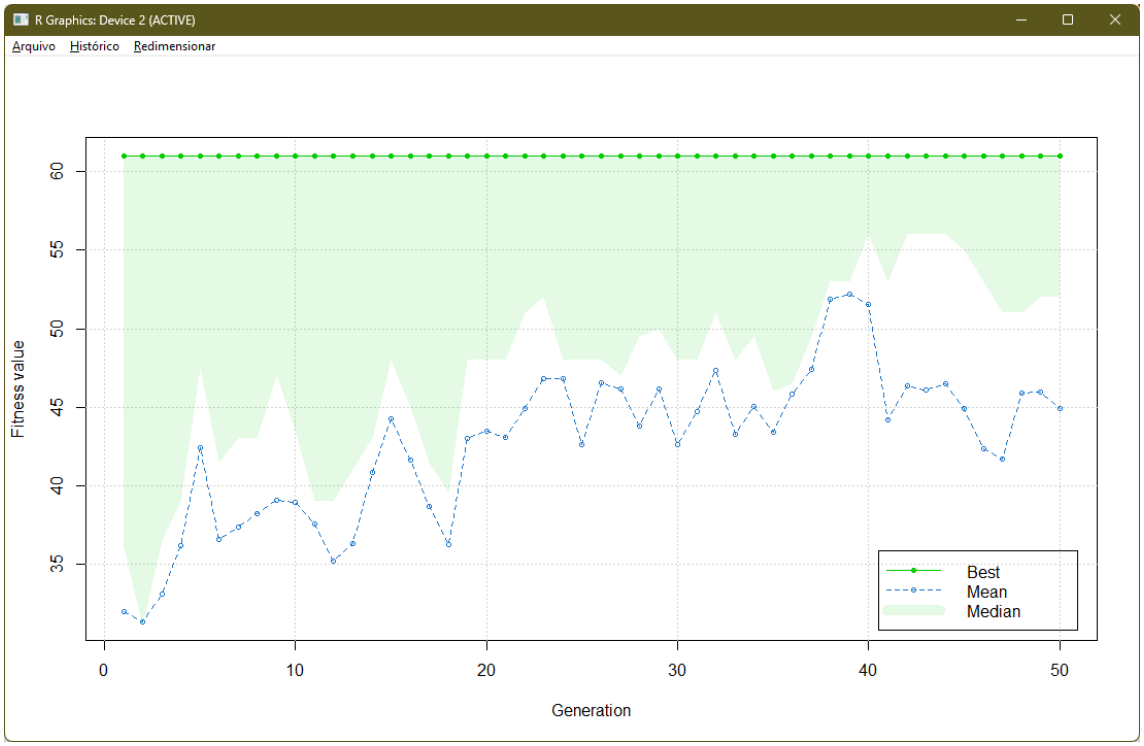
```
GA2=ga(type='binary',fitness=fitness,nBits=nrow(dados),maxiter=50,popSize
=50,seed=1234,keepBest=TRUE)
GA3=ga(type='binary',fitness=fitness,nBits=nrow(dados),maxiter=100,popSiz
e=50,seed=1234,keepBest=TRUE)
```

Vamos agora visualizar os resultados através do gráfico do GA:

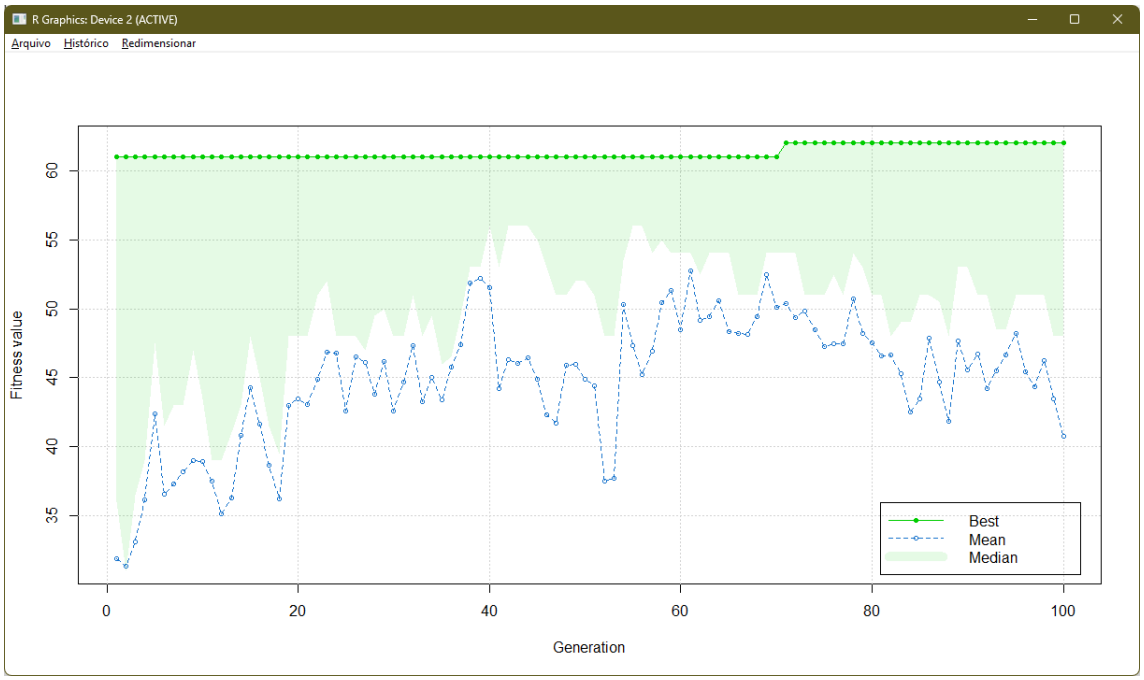
```
# Primeiro o GA2, para 50 gerações

plot(GA2)
```

```
# Depois para o GA3, para 100 gerações
plot(GA3)
```



O resultado de otimização do Algoritmo Genético — GA2



O resultado de otimização do Algoritmo Genético — GA3

A partir de GA3, podemos ver que o resultado de otimização para cada indivíduo está no seu melhor na geração 40 e 60, de acordo com a média e mediana do valor fitness sobre essa geração. Também podemos ver que o melhor valor fitness está aumentando para 62 da 72ª geração em diante.

Como mantemos o melhor resultado em cada processo de otimização, queremos descobrir os itens que podemos trazer para caminhadas com base no melhor resultado da otimização do algoritmo genético. Podemos ver o resumo das gerações GA3.

```
summary(GA3
```

```
> summary(GA3)
-- Genetic Algorithm ------------------

GA settings:
Type                  =  binary
Population size       =  50
Number of generations =  100
Elitism               =  2
Crossover probability =  0.8
Mutation probability  =  0.1

GA results:
Iterations             = 100
Fitness function value = 62
Solution =
     x1 x2 x3 x4 x5 x6 x7 x8 x9
[1,]  1  1  1  1  1  0  0  1  1
```

A partir do resultado acima, podemos concluir que os itens que podemos incluir na mochila são uma capa de chuva, canivete, água mineral, luvas, saco de dormir, comida enlatada e lanches. Podemos calcular o peso da mochila, para ter certeza de que a mochila não está superlotada.

```
cromossomas_final=c(1,1,1,1,1,0,0,1,1)

cat(cromossomas_final%*%dados$peso)

25

dados[cromossomas_final==1,]
```

```
> cromossomas_final=c(1,1,1,1,1,0,0,1,1)
> cat(cromossomas_final%*%dados$peso)
25
> dados[cromossomas_final==1,]
             item peso sobrevivencia
1 casaco de chuva    2             5
2        canivete    1             3
3    água mineral    6            15
4           luvas    1             5
5  saco de dormir    4             6
8 comida enlatada    8            20
9         lanches    3             8
```

Conseguimos! Podemos ver que o peso dos itens é o mesmo que a capacidade da mochila e a sobrevivência será “máxima”!

Conclusão

Pronto! Você pode caminhar com seus amigos com os pontos de sobrevivência e capacidade máximas implementando o algoritmo genético em R. Na verdade, você pode resolver o problema da mochila usando algoritmo genético em muitas aplicações do mundo real, como escolher o portfólio de melhor desempenho em aplicações financeiras, programação de produção em fábricas e muito mais.

Referências

1. G.B. Matthews, Sobre a Partição dos Números (1897), Proceedings of the London Mathematical Society.
2. Luca Scrucca, GA: A Package for Genetic Algorithms in R (2013), Journal of Statistical Software.
3. https://www.rdocumentation.org/packages/GA/versions/3.2/topics/ga
4. Harvey M. Salkin e Cornelis A. De Kluyver, O problema da mochila: Uma pesquisa (1975), Naval Research Logistics Quarterly.
5. Krzysztof Dudziński e Stanisław Walukiewicz, métodos exatos para o problema da mochila e suas generalizações (1987), European Journal of Operational Research.
6. Sami Khuri, Thomas Bäck e Jörg Heitkötter, O problema da mochila múltipla zero/um e algoritmos genéticos (1994), SAC '94: Proceedings of the 1994 ACM symposium on Applied computing.
7. https://rpubs.com/Argaadya/550805
8. Algoritmo Genético em R: O Problema da Mochila | por Raden Aurelius Andhika Viadinugroho | Rumo à Ciência de Dados (towardsdatascience.com)

Laboratório GA – O Problema do Caixeiro Viajante

Introdução

O Problema do Caixeiro Viajante (TSP) é um problema de otimização combinatória complexo, bastante conhecido e extensamente estudado. Embora seja um problema complexo, existem muitos algoritmos heurísticos disponíveis que funcionam bem com alguns milhares de cidades ou locais. E uma ampla variedade de pesquisas foi conduzida para explorar a aplicação do TSP no planejamento, logística e fabricação de microchips.

Neste laboratório focaremos a aplicação do TSP em Logística para a indústria de e-commerce. Primeiramente, discutiremos brevemente sobre o crescimento do e-commerce nos últimos anos e a importância de se ter uma logística realmente eficiente para redução de custos e uma melhor experiência do cliente. E então discutiremos sobre TSP, Algoritmo Genético (GA) e veremos "como encontrar a rota ideal para um Problema do Caixeiro Viajante usando GA com a linguagem R". Vamos mostrar, por meio de um exemplo clássico de solução de TSP usando GA, uma alternativa para tornar o processo de entrega no setor de comércio eletrônico mais eficiente e econômico.

Expansão do Comércio Eletrônico

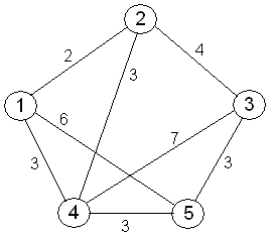
Devido à aplicação global de medidas de isolamento social para controle da pandemia causada pelo COVID-19, o comércio eletrônico teve uma expansão acelerada e sem precedentes.

A forme expansão do *e-commerce* coloca uma enorme pressão nas funções logísticas de suporte. A proposta do e-commerce ao cliente é oferecer uma variedade quase infinita de opções espalhadas por uma enorme área geográfica. As empresas não podem competir apenas com base em volumes absolutos no mundo atual do comércio eletrônico, em constante evolução, com informações simétricas e globalizadas. A competição mudou exigindo entregas com pontualidade, de forma consistente e previsível. Preços de entrega insignificantes ou nulos, soluções de rastreabilidade e logística reversa conveniente se tornaram os elementos mais importantes de diferenciação para os fornecedores. Embora os desafios atuais de logística relacionados à fabricação e distribuição de produtos de consumo e varejo organizado sejam bem conhecidos, as demandas do comércio eletrônico elevam as complexidades associadas a um nível diferente. Os varejistas de comércio eletrônico estão bem cientes desses desafios e da necessidade de investir em ativos de capital e operacionais.

Com toda essa expansão e mudança, você pode perceber o quão importante o planejamento logístico pode se tornar para as empresas tradicionais que estão entrando no e-commerce e para as que já o praticavam a fim de serem cada vez mais eficientes e competitivas.

TSP – O Problema do Caixeiro Viajante

A origem do nome **TSP** "*Travelling Salesman Problem*" é desconhecida. Não parece existir qualquer documento que prove o(a) autor(a) do nome do problema. O TSP, ou problema do caixeiro-viajante em português, representado na figura, consiste na procura de um circuito que possua a menor distância, começando numa cidade (ou local) qualquer, entre várias, visitando cada cidade (local) precisamente uma vez e regressando à cidade (local) inicial. O tamanho do espaço de procura aumenta exponencialmente dependendo de **n**, o número de cidades (locais), uma vez que existem inúmeros circuitos possíveis, a posição inicial é arbitrária e a ordem do circuito pode ser invertida.



Embora tenham sido desenvolvidos bons algoritmos de aproximação para o TSP, o problema continua a oferecer uma grande atração para a aplicação de novos algoritmos, tais como os evolucionários. Isto deve-se, essencialmente, às seguintes razões:

- A problemática do TSP pode ser entendida facilmente, uma vez que se aproxima dos problemas populares do mundo real;
- O TSP demonstra o caso mais simples dos problemas de requisição que são de enorme relevância para a programação de processos industriais e de logística de entrega;
- Existem vários conjuntos de dados sobre o TSP que estão publicamente disponíveis, de tal forma que os resultados são comparáveis mesmo que uma otimização global não seja ainda definitivamente conhecida;
- Relativamente à complexidade computacional, o TSP, como um problema complexo, é conhecido por representar uma larga classe de problemas para os quais não existem algoritmos polinomiais determinísticos.

Os algoritmos genéticos (GA) são um dos vários métodos que se utilizam para a resolução de problemas complexos como o TSP. Este método tem por base um processo iterativo sobre uma determinada população fixa, denominados por indivíduos, que representam as várias soluções do problema. Esta técnica advém do processo de evolução dos seres vivos demonstrada por *Darwin*.

GA - Algoritmo Genético

Algoritmo Genético, ou GA de *Genetic Algorithm*, é uma estratégia evolucionária poderosa inspirada nos princípios básicos da evolução biológica. O fluxograma a seguir, extraído do artigo “GA: Um Pacote para Algoritmos Genéticos em R” (Scrucca, 2013), explica a ideia de uma maneira simples.

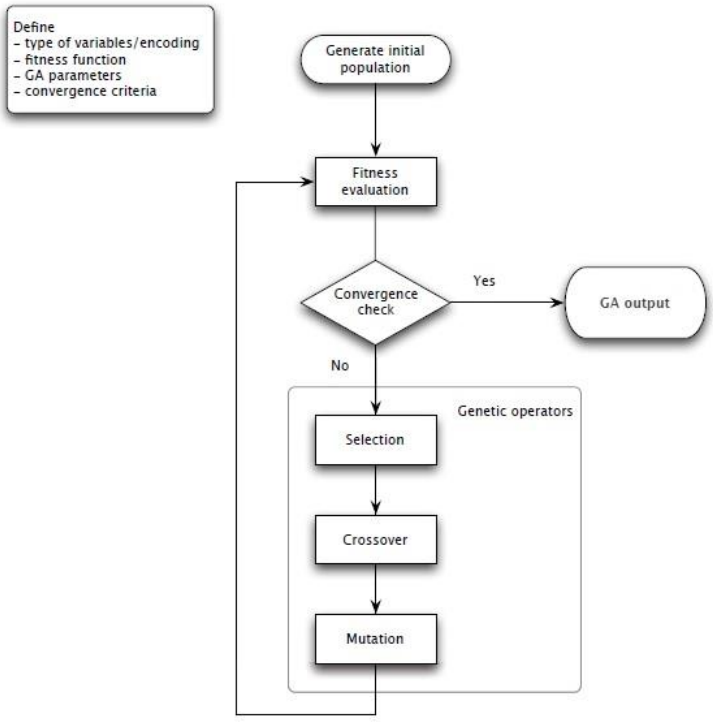


Figure 1: Flow-chart of a genetic algorithm.

Um usuário deve primeiro definir o tipo de variáveis e sua codificação para o problema em questão. Em seguida, uma função de aptidão é definida, em que muitas vezes é simplesmente a função objetivo a ser otimizada. De forma mais geral, pode ser qualquer função que atribua um valor de mérito relativo a um indivíduo. Operadores genéticos, como cruzamento e mutação, são aplicados estocasticamente (aleatoriamente) em cada etapa do processo de evolução, portanto, suas probabilidades de ocorrência devem ser definidas. Finalmente, os critérios de convergência devem ser fornecidos. Com todas essas ideias, agora estamos prontos para fazer algumas práticas.

Resolvendo TSP usando Algoritmo Genético em R

O exemplo que iremos aplicar foi extraído do artigo “GA: Um Pacote para Algoritmos Genéticos em R” (Scrucca, 2013). Dada uma matriz com distâncias entre cada par de origem e destino, o código R abaixo produzirá o caminho mais curto e, subsequentemente, colocará os resultados em um gráfico direcionado simples.

Primeiro, vamos carregar a biblioteca 'GA' e um exemplo de dados chamado “**eurodist**”, que fornece distâncias entre várias cidades na Europa (se você não tiver essa biblioteca, basta executar install.packages (“GA”))

```
install.packages("GA")
library(GA)
data("eurodist", package = "datasets")
D <- as.matrix(eurodist)
```

Vamos visualizar os dados contidos na matriz das cidades e suas distâncias:

```
View(D)
```

	Athens	Barcelona	Brussels	Calais	Cherbourg	Cologne	Copenhagen	Geneva	Gibraltar	Hamburg	Hook of Holland	Lisbon	Lyons	Madrid	Marseilles	Milan	Munich	Paris	Rome
Athens	0	3313	2963	3175	3339	2762	3276	2610	4485	2977	3030	4532	2753	3949	2865	2282	2179	3000	
Barcelona	3313	0	1318	1326	1294	1498	2218	803	1172	2018	1490	1305	645	636	521	1014	1365	1033	
Brussels	2963	1318	0	204	583	206	966	677	2256	597	172	2084	690	1558	1011	925	747	285	
Calais	3175	1326	204	0	460	409	1136	747	2224	714	330	2052	739	1550	1059	1077	977	280	
Cherbourg	3339	1294	583	460	0	785	1545	853	2047	1115	731	1827	789	1347	1101	1209	1160	340	
Cologne	2762	1498	206	409	785	0	760	1662	2436	460	269	2290	714	1764	1035	911	583	465	
Copenhagen	3276	2218	966	1136	1545	760	0	1418	3196	460	269	2971	1458	2498	1778	1537	1104	1176	
Geneva	2610	803	677	747	853	1662	1418	0	1975	1118	895	1936	158	1439	425	328	591	513	
Gibraltar	4485	1172	2256	2224	2047	2436	3196	1975	0	2897	2428	676	1817	698	1693	2185	2565	1971	
Hamburg	2977	2018	597	714	1115	460	460	1118	2897	0	550	2671	1159	2198	1479	1238	805	877	
Hook of Holland	3030	1490	172	330	731	269	269	895	2428	550	0	2280	863	1730	1183	1098	851	457	
Lisbon	4532	1305	2084	2052	1827	2290	2971	1936	676	2671	2280	0	1178	668	1762	2250	2507	1799	
Lyons	2753	645	690	739	789	714	1458	158	1817	1159	863	1178	0	1281	320	328	724	471	
Madrid	3949	636	1558	1550	1347	1764	2498	1439	698	2198	1730	668	1281	0	1157	1724	2010	1273	
Marseilles	2865	521	1011	1059	1101	1035	1778	425	1693	1479	1183	1762	320	1157	0	618	1109	792	

Agora, como nosso objetivo é encontrar o caminho mais curto para nosso entregador, a função de fitness deve maximizar o recíproco da duração do trajeto, o que, por sua vez, minimizará a duração do trajeto. Portanto, a seguir duas funções são definidas para o efeito.

```
# Função para calcular o comprimento do trajeto
tourLength <- function(tour, distMatrix) {
   tour <- c(tour, tour[1])
   route <- embed(tour, 2)[,2:1]
   sum(distMatrix[route])
}
# Função ser maximizada
tspFitness <- function(tour, ...) 1/tourLength(tour, ...)
```

Finalmente, aqui vem a “super função” construída no pacote “GA” que realmente calcula o caminho mais curto.

```
GA <- ga(type = "permutation", fitness = tspFitness, distMatrix = D,
          lower = 1, upper = attr(eurodist, "Size"), popSize = 50,
maxiter = 5000,
          run = 500, pmutation = 0.2)
```

```
Console  Terminal  Jobs
R 4.2.1 · D:/DATALAB/R/GA/
GA | iter = 2264 | Mean = 4.131046e-05 | Best = 7.740537e-05
GA | iter = 2265 | Mean = 3.914108e-05 | Best = 7.740537e-05
GA | iter = 2266 | Mean = 3.900835e-05 | Best = 7.740537e-05
GA | iter = 2267 | Mean = 3.966558e-05 | Best = 7.740537e-05
GA | iter = 2268 | Mean = 3.885377e-05 | Best = 7.740537e-05
GA | iter = 2269 | Mean = 3.934071e-05 | Best = 7.740537e-05
GA | iter = 2270 | Mean = 3.841650e-05 | Best = 7.740537e-05
GA | iter = 2271 | Mean = 3.916729e-05 | Best = 7.740537e-05
GA | iter = 2272 | Mean = 3.872477e-05 | Best = 7.740537e-05
GA | iter = 2273 | Mean = 4.001087e-05 | Best = 7.740537e-05
GA | iter = 2274 | Mean = 3.848230e-05 | Best = 7.740537e-05
GA | iter = 2275 | Mean = 4.208074e-05 | Best = 7.740537e-05
GA | iter = 2276 | Mean = 4.353807e-05 | Best = 7.740537e-05
GA | iter = 2277 | Mean = 4.347141e-05 | Best = 7.740537e-05
GA | iter = 2278 | Mean = 4.475402e-05 | Best = 7.740537e-05
GA | iter = 2279 | Mean = 4.455624e-05 | Best = 7.740537e-05
GA | iter = 2280 | Mean = 4.411039e-05 | Best = 7.740537e-05
GA | iter = 2281 | Mean = 4.304240e-05 | Best = 7.740537e-05
GA | iter = 2282 | Mean = 4.303153e-05 | Best = 7.740537e-05
> 
```

Vamos olhar os resultados e uma visualização para ver como as cidades se parecem em um gráfico,

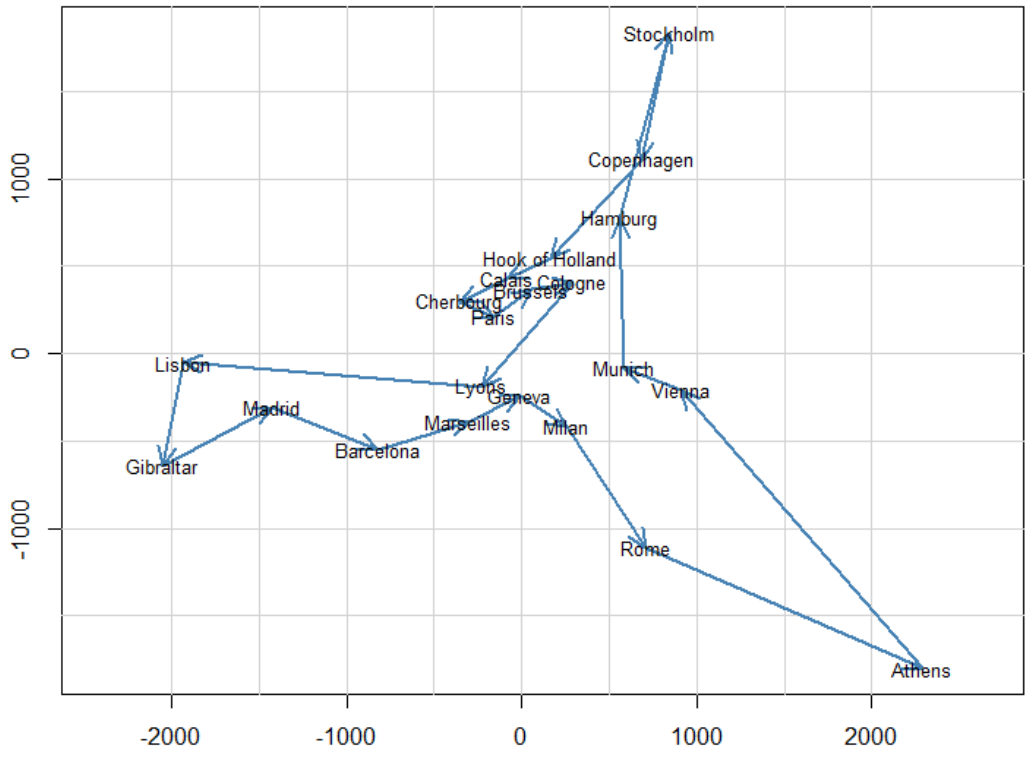
```
# Sumário
summary(GA)
```

```
Console  Terminal  Jobs
R 4.2.1 · D:/DATALAB/R/GA/
> # Sumario
> summary(GA)
-- Genetic Algorithm -------------------

GA settings:
Type                  =  permutation
Population size       =  50
Number of generations =  5000
Elitism               =  2
Crossover probability =  0.8
Mutation probability  =  0.2

GA results:
Iterations             = 2282
Fitness function value = 7.740537e-05
Solutions =
     x1 x2 x3 x4 x5 x6 x7 x8 x9 x10  ...  x20 x21
[1,] 21 17  6 10 20  7 11  3  4  18        19   1
[2,] 10 20  7 11  3  4 18  5 14  12        17   6
> 
```

```
# Visualização
mds <- cmdscale(eurodist)
x <- mds[, 1]
y <- -mds[, 2]
plot(x, y, type = "n", asp = 1, xlab = "", ylab = "")
abline(h = pretty(range(x), 10), v = pretty(range(y), 10),
           col = "light gray")
tour <- GA@solution[1, ]
tour <- c(tour, tour[1])
n <- length(tour)
arrows(x[tour[-n]], y[tour[-n]], x[tour[-1]], y[tour[-1]],
           length = 0.15, angle = 25, col = "steelblue", lwd = 2)
text(x, y, labels(eurodist), cex=0.8)
```



Conclusão

O modelo exemplo simples que vimos neste laboratório não considerou muitos fatores que podem desempenhar um grande papel em tornar este exemplo adaptável à realidade da indústria de e-commerce. Pode haver muitas outras variáveis importantes como:

- Número de pedidos por cidade e localidade;
- Número de entregadores disponíveis em um centro de distribuição;

- Tamanho / tipo de pacote a ser distribuído (isso decidirá se você deseja uma bicicleta ou carro para entrega);
- Capacidade dos veículos para transporte das encomendas;
- Etc.

No mundo real, o modelo simples que construímos, baseado no conceito do TSP com a utilização do GA, não servirá diretamente para as empresas de e-commerce otimizarem o seu negócio, mas a ideia por trás deste exemplo foi fornecer um ponto de partida que irá gerar o estímulo para que uma mente curiosa e brilhante como a sua explore mais esta técnica para poder aplicá-la em casos reais.

Referências

- https://rstudio-pubs-static.s3.amazonaws.com/132872_620c10f340f348b88453d75ec99960ff.html
- GA: A Package for Genetic Algorithms in R, Scrucca, Luca – Journal of Statistical Software, April 2013, Volume 53, Issue 4. (http:/www.jstatsoft.org/)

Etapa Final: Relatório de Elaboração do Laboratório

Você deve entregar um relatório com os resultados das etapas elaboradas neste laboratório no e-Disciplinas, para formatá-lo siga estas orientações:

1. Crie um documento Word e identifique-o com o nome do laboratório, data de elaboração e o seu nome ou da dupla que o elaborou;
2. Crie um tópico para cada resultado que você considerar relevante (manipulação de dados ou resultado de algum processamento) identificando-o com um título e uma breve explicação. Os resultados podem ser imagens de gráficos gerados ou de listas de valores ou dados de resultados obtidos. Não devem ser incluídos os scripts ou instruções de processamento utilizados, inclua apenas os resultados que você considerar relevantes.
3. No final do relatório crie um último tópico denominado “Conclusões” e elabore comentários, sugestões e conclusões sobre o que você pode aprender com a elaboração deste laboratório.

Parabéns! Você concluiu com sucesso o Laboratório!