

Trabalho Individual II

Exercício realizado no âmbito da Unidade Curricular de Optimização
Heurística do 2º ano da Licenciatura em Ciência de Dados

André Plancha, 105289

Andre_Plancha@iscte-iul.pt

27 maio 2023

Versão 1.0.0

Índice

a)	2
b)	2
c)	2
d)	3
e)	4
f)	4
g)	5
h)	5
i)	5

a)

Uma solução S diz-se admissível se satisfaz todas as condições do problema. Para este problema, uma solução admissível é uma solução que: admite a cada cientista C_i um projeto P_j , cada projeto tem apenas um cientista, e todos os projetos têm um cientista, $i, j \in \{1 \dots 10\}$. Ou seja, sendo A o conjunto de soluções admissíveis, U o conjunto de soluções, e $C_\alpha \mapsto P_\beta$ o alocamento do cientista α ao projeto β :

$$\forall S \in U : S \in A \Leftarrow S = \{C_1 \mapsto P_a, C_2 \mapsto P_b, \dots, C_{10} \mapsto P_j\}$$

$$, a \dots j, \alpha, \beta \in \{1 \dots 10\} \wedge a \neq b \neq \dots \neq j, A \subset U.$$

Como há apenas 10 cientistas e apenas 10 projetos, todos os cientistas vão ter projetos e todos os projetos vão ter cientistas.

b)

Uma possível heurística construtiva seria admitir como líder do projeto P_i o cientista C_i , $i \in \{1 \dots 10\}$. Ou seja,

$$S = \{C_1 \mapsto P_1, C_2 \mapsto P_2, \dots, C_{10} \mapsto P_{10}\}$$

Esta heurística, embora simples e produtora de uma solução admissível, não é interessante para **Lusa_med**, sendo que é equivalente a uma heurística que escolhe os líderes de forma aleatória (sem reposição). Desta forma, uma heurística construtiva alternativa seria alocar para o cientista C_i o projeto P_j que tenha a melhor aptidão, entre as ainda não alocadas, $j \in \{1 \dots 10\}$. Ou seja, sendo $a(C_i, P_j)$ a aptidão do cientista C_i ao projeto P_j , e P os projetos:

$$S = \left\{ \begin{array}{l} C_1 \mapsto P_{\text{argmax}\{a(C_1, p) : p \in P\}}, \\ C_2 \mapsto P_{\text{argmax}\{a(C_2, p) : p \in P \setminus \{p \leftarrow C_1\}\}}, \\ C_3 \mapsto P_{\text{argmax}\{a(C_3, p) : p \in P \setminus \{p \leftarrow C_1, p \leftarrow C_2\}\}}, \\ \dots, \\ C_{10} \mapsto P_{\text{argmax}\{a(C_{10}, p) : p \in P \setminus \{p \leftarrow C_1, p \leftarrow C_2, \dots, p \leftarrow C_9\}\}} \end{array} \right\}$$

c)

Para construir a solução, o Algoritmo 1 pode ser usado.

Recebe: Cientistas C , Projetos P , função aptidão $a(C, P)$
 Devolve: Solução S

```

1: para todos  $C_i$  em  $C$  fazer
2:    $j \leftarrow \operatorname{argmax}\{a(C_i, p) : p \in P\}$ 
3:   adicionar  $(C_i \mapsto P_j)$  a  $S$ 
4:   remover  $P_j$  de  $P$ 

```

Algoritmo 1: Heurística construtiva

A solução resultante do algoritmo neste problema é a seguinte:

$$\begin{aligned}
 S_0 = \{ \\
 & C_1 \mapsto P_9, \\
 & C_2 \mapsto P_5, \\
 & C_3 \mapsto P_2, \\
 & C_4 \mapsto P_8, \\
 & C_5 \mapsto P_1, \\
 & C_6 \mapsto P_3, \\
 & C_7 \mapsto P_7, \\
 & C_8 \mapsto P_{10}, \\
 & C_9 \mapsto P_6, \\
 & C_{10} \mapsto P_4 \\
 & \}
 \end{aligned}$$

, para uma aptidão total de 832.

d)

Uma estrutura de vizinhança para este problema pode ser definida como a troca de projetos entre dois cientistas, para todos os cientistas. Ou seja,

$$\begin{aligned}
 V(S) = \{ S' \in A : S' = (S \setminus \{C_\alpha \mapsto P_i, C_\beta \mapsto P_j\}) \cup \{C_\alpha \mapsto P_j, C_\beta \mapsto P_i\} \\
 \wedge \{C_\alpha \mapsto P_i, C_\beta \mapsto P_j\} \subset S \\
 \}
 \end{aligned}$$

Isto pode ser implementado com o Algoritmo 2.

Recebe: Solução S
 Devolve: Vizinhança V

```

1: para todos  $C_\alpha \mapsto P_i$  em  $S$  fazer:
2:   para todos  $C_\beta \mapsto P_j$  em  $(S \setminus C_\alpha \mapsto P_i)$  fazer:
3:      $S' \leftarrow S$ 
4:     tirar  $C_\alpha \mapsto P_i$  e  $C_\beta \mapsto P_j$  de  $S'$ 
5:     adicionar  $C_\alpha \mapsto P_j$  e  $C_\beta \mapsto P_i$  a  $S'$ 
6:     adicionar  $S'$  a  $V$ 

```

Algoritmo 2: Estrutura de vizinhança

e)

Uma solução vizinha pode ser a solução que troca os projetos de $C_1 \rightleftharpoons C_2$, sendo essa:

$$\begin{aligned}
 S_1 = \{ & \\
 & C_1 \mapsto P_5, \\
 & C_2 \mapsto P_9, \\
 & C_3 \mapsto P_2, \\
 & C_4 \mapsto P_8, \\
 & C_5 \mapsto P_1, \\
 & C_6 \mapsto P_3, \\
 & C_7 \mapsto P_7, \\
 & C_8 \mapsto P_{10}, \\
 & C_9 \mapsto P_6, \\
 & C_{10} \mapsto P_4 \\
 & \}
 \end{aligned}$$

, para uma aptidão total de 804.

f)

A lista tabu é uma lista de movimentos que o algoritmo tabu tem que evitar em cada iteração, de forma ao algoritmo não ficar preso num máximo local não global. Esta lista começa vazia no início do algoritmo, aumentando o seu tamanho a cada iteração, até um máximo t , dimensão da lista tabu, ou tempo de permanência ($t \in \mathbb{N}_0$). Este parâmetro do algoritmo é definido *a priori*. A cada iteração, o algoritmo adiciona na lista tabu uma representação do movimento da solução anterior para o vizinho escolhido, e retira o movimento mais velho se a dimensão da lista exceder t .

Desta forma, o algoritmo será forçado a potencialmente explorar vizinhos com pior aptidão, em vez de vizinhos previamente visitados, de forma a sair do máximo local e explorar outro máximo.

Uma outra forma de interpretar o parâmetro é vê-lo como *tempo de permanência*, ou seja, t representa o número de iterações que o movimento fica inválido. Este conceito é equivalente ao conceito anterior, sendo que uma dimensão t significa que o movimento m_i irá sair da lista em $i + t$ iterações porque m_i será o mais velho.

Um t demasiado alto pode fazer com que o algoritmo não visite soluções que poderiam ter maior aptidão total (nos mesmos vizinhos mas por caminhos alternativos), e um t demasiado baixo pode fazer com que o algoritmo não saia do máximo local. Devido a esse facto, se o algoritmo não conseguir chegar ao mínimo de aptidão desejado, serão tentados tempos diferentes. Para a primeira tentativa, admite-se que $t = 4$.

g)

À lista tabu T vai ser adicionada uma representação da diferença entre a solução anterior e o vizinho escolhido, independentemente do valor de aptidão da solução anterior. Aqui, vai ser escolhido representar a diferença com a troca do projeto que os cientistas fizeram, independentemente do projeto que trocaram. Ou seja, usando o vizinho definido na alínea e), e sendo i a iteração do algoritmo,

$$\forall t > 0 : T_1 = [C_1 \rightleftharpoons C_2]^T$$

Neste caso, a lista tabu é uma matriz de dimensão $t \times 1$, sendo que cada movimento tem apenas uma (1) troca de projetos.

h)

Os movimentos tabu são as tais representações da diferença entre a solução anterior e o vizinho escolhido, aqui representado como $C_i \rightleftharpoons C_j$. Ou seja, na escolha do vizinho durante o algoritmo tabu, um movimento não é válido se o vizinho tiver os projetos de C_i e C_j trocados. Por exemplo, se a lista tabu for a apresentada na alínea g), e a solução for a apresentada na alínea e), o movimento criado pelo vizinho S_0 e movimentos equivalentes (qualquer movimento $C_1 \rightleftharpoons C_2$), seria um movimento inválido, independente dos projetos trocados, mesmo se esses tivessem uma aptidão maior; logo o algoritmo teria que escolher outro vizinho na iteração em causa.

i)

O algoritmo tabu com os critérios de paragem especificados é implementado com o Algoritmo 3, $m = 850$, $M = 100$, $t = 4$.

Recebe:

- Solução inicial S_0 ,
- Mínimo de aptidão m ,
- Máximo de iterações M ,
- Tempo de permanência t ,
- Estrutura de vizinhança $V(S)$
- Definição de movimento $\Delta(S_a, S_b)$
- Função de aptidão $a(S)$

Devolve: Melhor solução encontrada S_O

- 1: $T \leftarrow$ uma matriz $t \times 1$
- 2: $h = 0$
- 3: **para** i **de** 1 **a** M **fazer**
- 4: $V_i \leftarrow V(S_i)$
- 5: Remover de V_i todos S onde $\Delta(S, S_i)$ é uma das linhas de T
- 6: $S_i \leftarrow$ o S de V_i com maior aptidão
- 7: **se** $a(S_i) > a(S_O)$ **então**
- 8: $S_O \leftarrow S_i$
- 9: **se** $a(S_O) > m$ **então** acabar Algoritmo
- 10: $h \leftarrow h + 1$
- 11: $T_{h,1} \leftarrow \Delta(S_i, S_{i-1})$
- 12: **se** $h > t$ **então** $h \leftarrow 0$

Algoritmo 3: Algoritmo Tabu

Cada iteração do algoritmo pode ser dividido em cinco partes:

- A construção da vizinhança [4]
- Escolha do melhor vizinho que não esteja na lista tabu [5,6]
- Critério de Aspiração [7,8]
- Critério de paragem [3,9]
- Atualização da lista tabu [10 - 12]

Ao implementar o algoritmo para o nosso S_0 , o algoritmo revelou a seguinte solução:

$$S_O = \{$$

$$C_1 \mapsto P_9$$

$$C_2 \mapsto P_5$$

$$C_3 \mapsto P_2$$

$$C_4 \mapsto P_8$$

$$C_5 \mapsto P_1$$

$$C_6 \mapsto P_{10}$$

$$C_7 \mapsto P_7$$

$$C_8 \mapsto P_3$$

$$C_9 \mapsto P_6$$

$$C_{10} \mapsto P_4$$

$$\}$$

, com uma aptidão de 865, encontrada na primeira iteração ($S_O = S_1$).

Para efeitos ilustrativos, foi feito o mesmo algoritmo, mas com $m = 1001$, uma aptidão impossível de chegar, para deixar o algoritmo correr durante as 100 iterações. O algoritmo encontrou a seguinte solução:

$$S_O = \{$$

$$C_1 \mapsto P_9$$

$$C_2 \mapsto P_2$$

$$C_3 \mapsto P_6$$

$$C_4 \mapsto P_8$$

$$C_5 \mapsto P_7$$

$$C_6 \mapsto P_4$$

$$C_7 \mapsto P_{10}$$

$$C_8 \mapsto P_3$$

$$C_9 \mapsto P_1$$

$$C_{10} \mapsto P_5$$

$$\}$$

, com uma aptidão de 902.

O gráfico da Figura 1 mostra a evolução da aptidão ao longo das iterações.

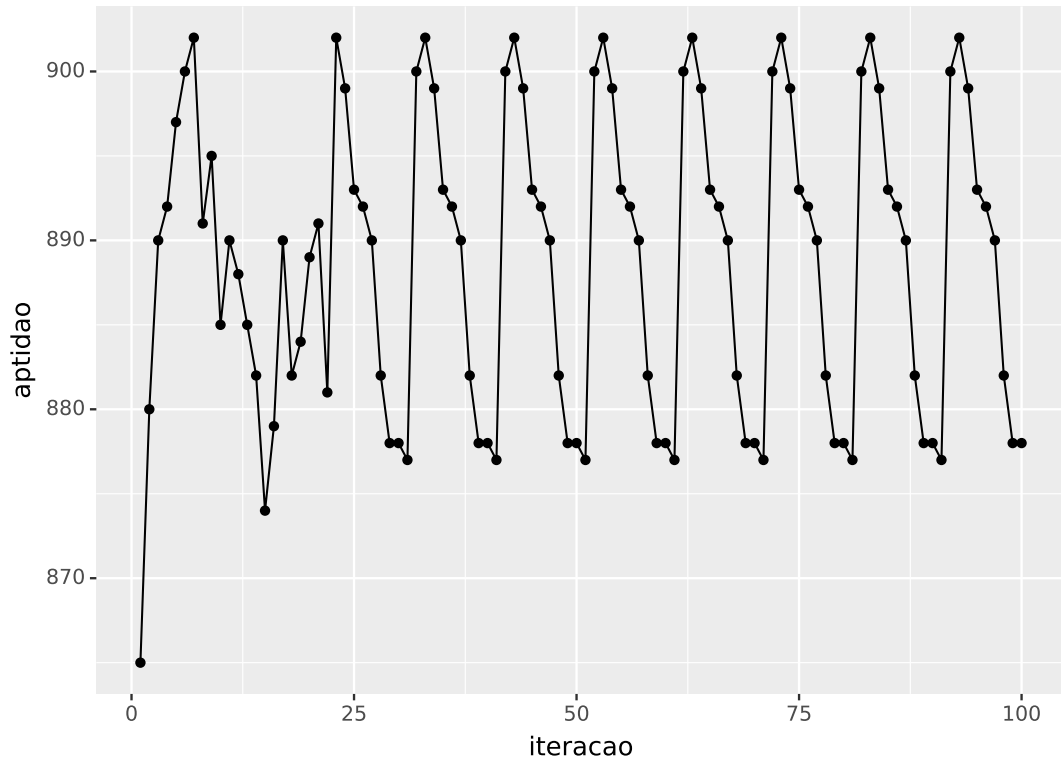


Figura 1: Evolução da aptidão, $t = 4$

Como se pode observar, o algoritmo parece ficar num ciclo de vizinhos na iteração 23, o que significa que o algoritmo não está a explorar outras soluções que podem ser melhores, devido ao facto da dimensão da lista tabu t ser demasiado pequena.

O gráfico da Figura 2 mostra o mesmo algoritmo, mas com $t = 5$.

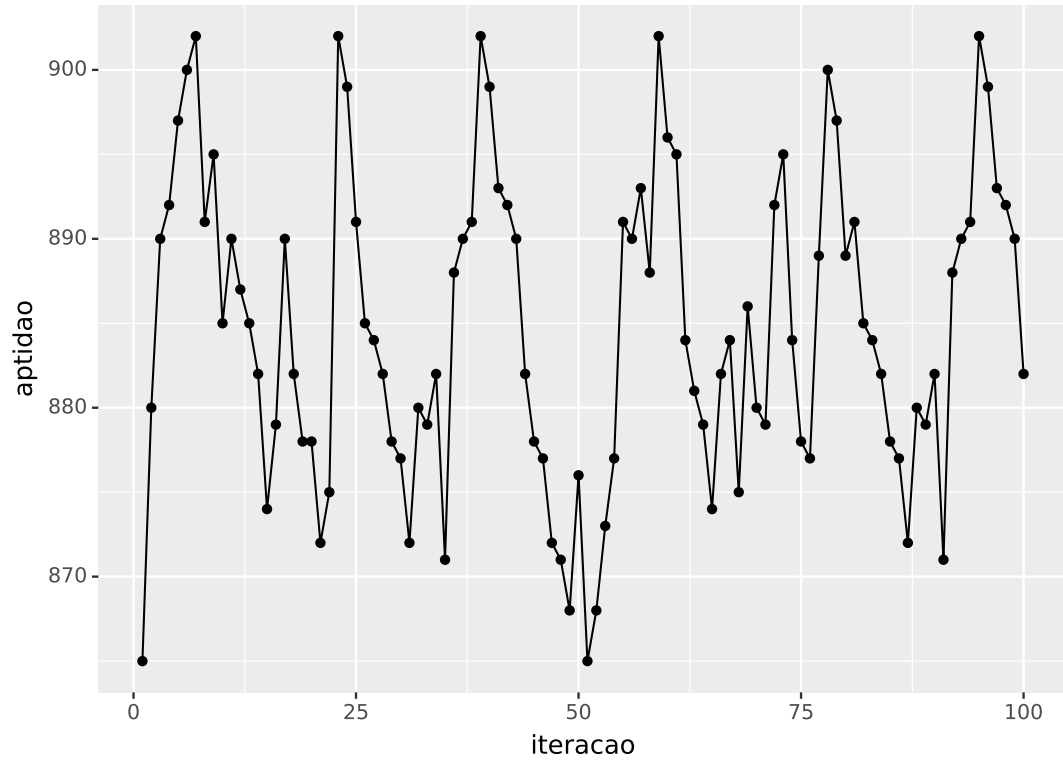


Figura 2: Evolução da aptidão, $t = 5$

Perante este resultado, o algoritmo parece procurar mais soluções, sem ficar preso num ciclo de vizinhos, embora este se comece a formar entre as iterações 25 e 80. No entanto, uma aptidão de 902 deve ser completamente apropriada para a **Lusa_Med**, e baseado nestas duas execuções, como o problema não é muito complexo, será razoável dizer que qualquer solução que tenha uma aptidão de 902 é uma solução ótima. Isto podia ser verificado ao experimentar todas as soluções admissíveis, algo que seria computacionalmente dispendioso, devido ao número demasiado elevado de soluções para analisar.