Trabalho 1 - Inteligência Artificial

Gabriel de Souza Alves - 726515
Isadora Eliziário Gallerani - 726542

1. Problema do Quebra-Cabeça de Oito Peças

1.1. Introdução

Para a solução deste problema, foi optada pela representação de estados através do uso de um vetor de posições do tabuleiro em que cada movimentação gera um novo estado, considerando que o 0 é a posição vazia do tabuleiro. O custo de movimentação é unitário, utilizando como heurística a Distância de Manhattan, dada pela seguinte fórmula:

$$\sum((estadoAtual.X - estadoObjetivo.X) + (estadoAtual.Y - estadoObjetivo.Y))$$

Onde a posição X é referente à linha e posição Y referente à coluna do tabuleiro, calculando a soma das distâncias de cada peça fora do lugar para a sua posição correta. Para cada peça, é possível realizar a movimentação em todos os sentidos - direita, esquerda, cima, baixo.
Para a solução do problema, deve-se aplicar a heurística considerando que o estado objetivo/final seria encontrar uma matriz da forma:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \end{bmatrix}$$

A partir de uma matriz proposta pelo grupo, cujo estado inicial era:

$$\begin{bmatrix} 1 & 2 & 3 \\ & 4 & 5 \\ 7 & 8 & 6 \end{bmatrix}$$

1.2. Desenvolvimento

A seguir apresentamos partes da implementação da solução do primeiro problema em R:

```
inicial <- QuebraCabecas(desc = c(1,2,3,0,4,5,7,8,6), objetivo = c(1,2,3,4,5,6,7,8,0))
objetivo <- QuebraCabecas(desc = c(1,2,3,4,5,6,7,8,0), objetivo = c(1,2,3,4,5,6,7,8,0))
```

Imagem 1 - Instanciação dos estados inicial e objetivo do problema 1.

```
## Sobrecarga da função genérica "heuristica", definida por Estado.R
heuristica.QuebraCabecas <- function(atual){

  if(is.null(atual$desc) || is.null(atual$objetivo))
    return(Inf)
  ## h(obj) = soma((atual.X[i] - objetivo.X[i]) + (atual.Y[i] - objetivo.Y[i]))
  dist <- 0
  posAtual <- t(matrix(atual$desc,nrow=3,ncol=3))
  posObj <- t(matrix(atual$objetivo,nrow=3,ncol=3))

  for(x in 1:3) {
    for(y in 1:3) {
      ## Gerando posição objetivo
      obj <- which(posObj == posAtual[x,y], arr.ind = TRUE)
      ## Cálculo da distância de Manhattan para a posição pos(x,y)
      dist <- dist + abs(x - obj[[1]]) + abs(y - obj[[2]])
    }
  }

  return(dist)
}
```

Imagem 2 - Implementação da função heurística para o problema 1.

Como dito anteriormente, para a nossa implementação cada movimento do tabuleiro é representado como um estado. Os operadores são representados como um movimento (de custo unitário) da posição vazia para as direções diretamente acima, abaixo e aos lados no tabuleiro, gerando no máximo 4 nós filhos a partir de cada estado. Os filhos gerados em posições inválidas do tabuleiro após aplicar os operadores às coordenadas atuais da posição vazia são descartados, e então somamos 1 ao custo total de movimentação, visto que todas as operações são de custo unitário. Toda a árvore de transições do problema é gerada desta forma.

1.3. Resultados

Obtivemos êxito em nossa implementação ao executar dois dos três algoritmos de Busca Desinformada (Busca em Largura, Busca de Custo Uniforme) e todos os de Busca Informada de forma correta. Estes foram os resultados:

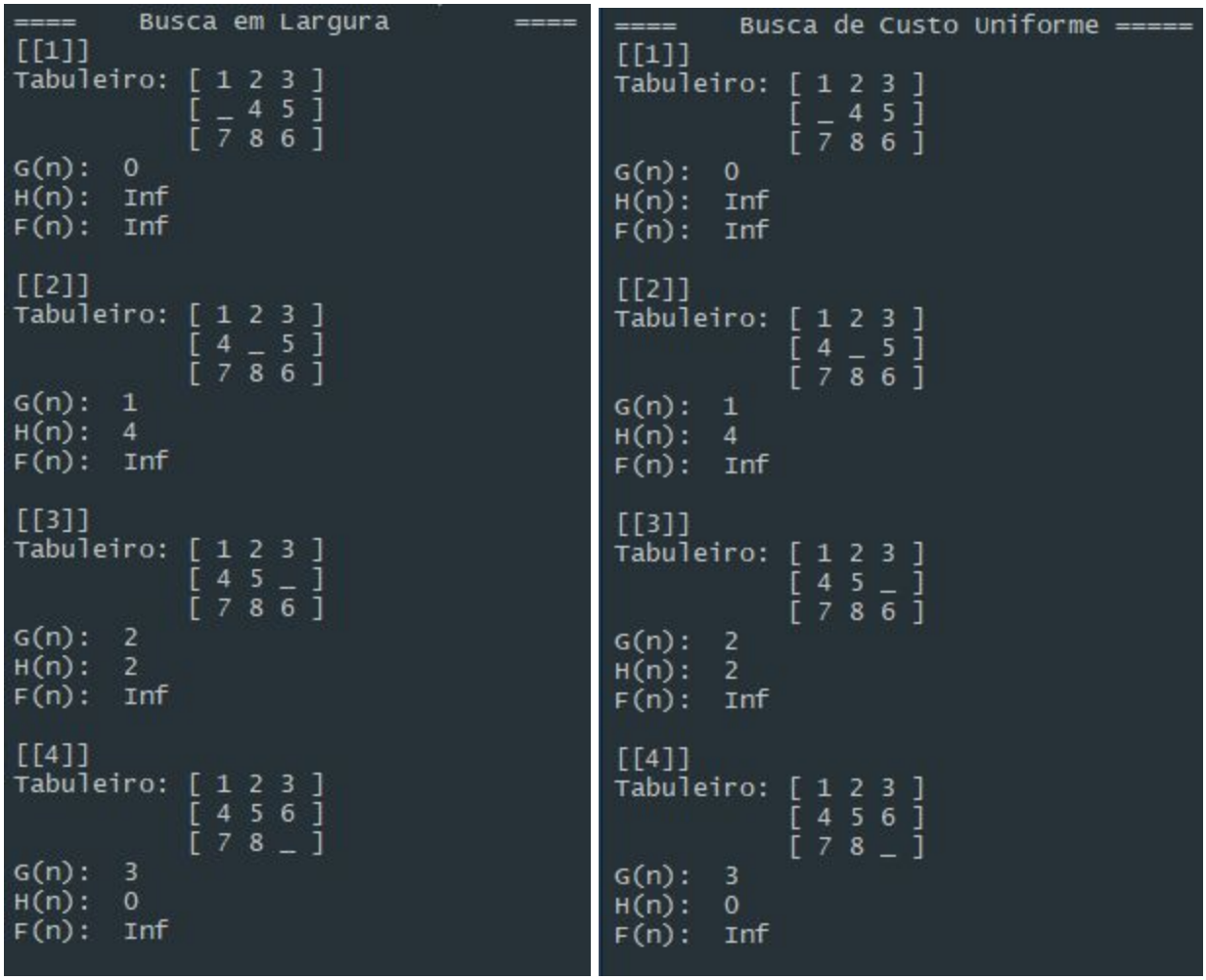

Imagem 3 - Resultados da Busca Desinformada no problema 1.

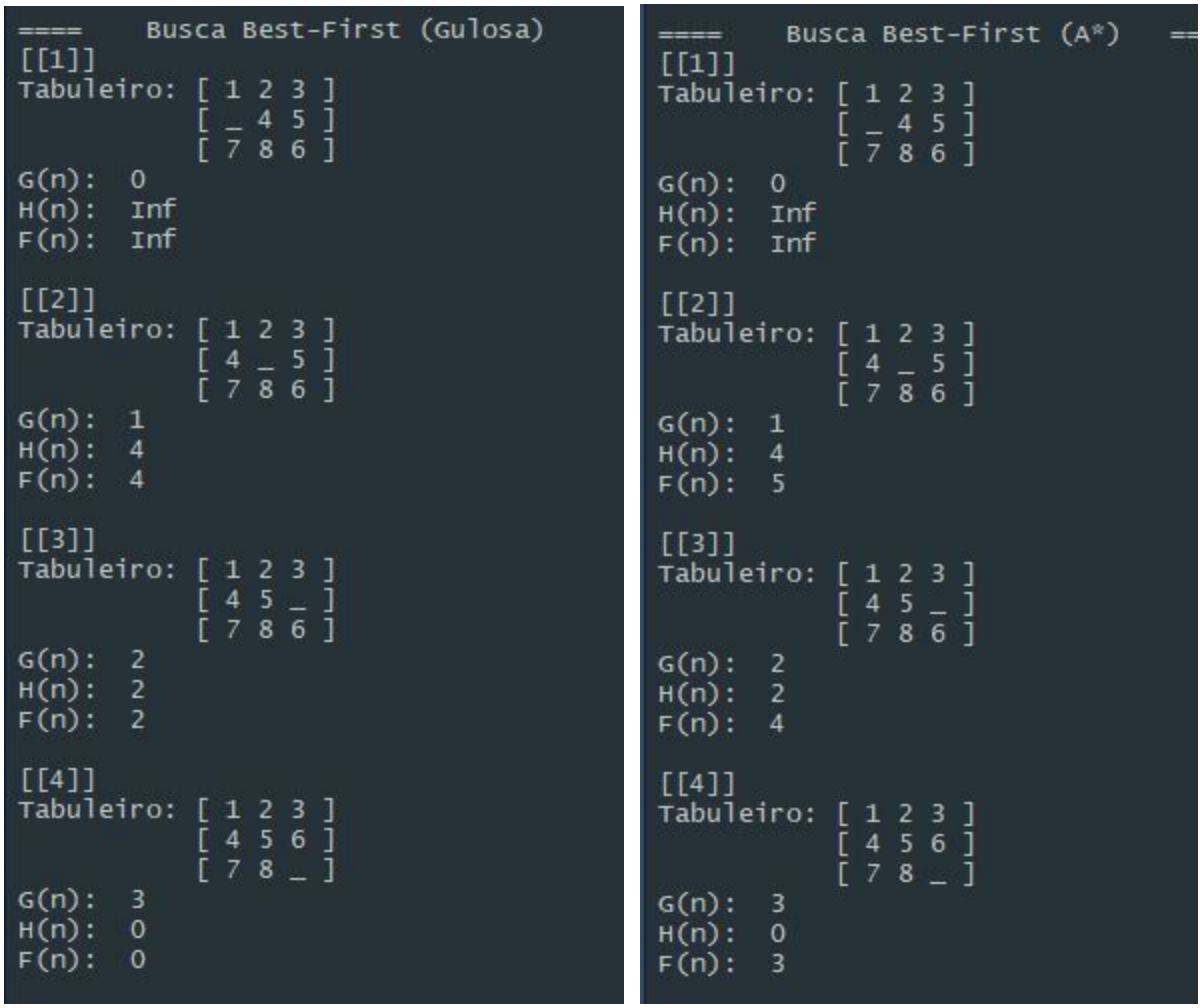

Imagem 4 - Resultados da busca Informada no problema 1.

2. Problema do Trajeto Entre Duas Cidades

2.1. Introdução

Para a solução do segundo problema, foi optada pela representação de cada cidade como um estado, utilizando uma matriz de transições para representar o grafo, que continha os pesos das arestas correspondentes aos da figura a seguir. O custo de movimentação é o valor da aresta que conecta duas cidades no grafo, enquanto a função heurística utilizada foi a distância em linha reta entre cada cidade e a cidade destino/objetivo, dada na tabela a seguir. Para realizar a movimentação entre uma cidade e outra, é levada em consideração o menor valor de distância encontrado.

Para a solução do problema foi aplicada a heurística apresentada anteriormente, levando em consideração também o custo de movimentação dependendo do algoritmo utilizado. O objetivo apresentado foi chegar à cidade B a partir da cidade O.

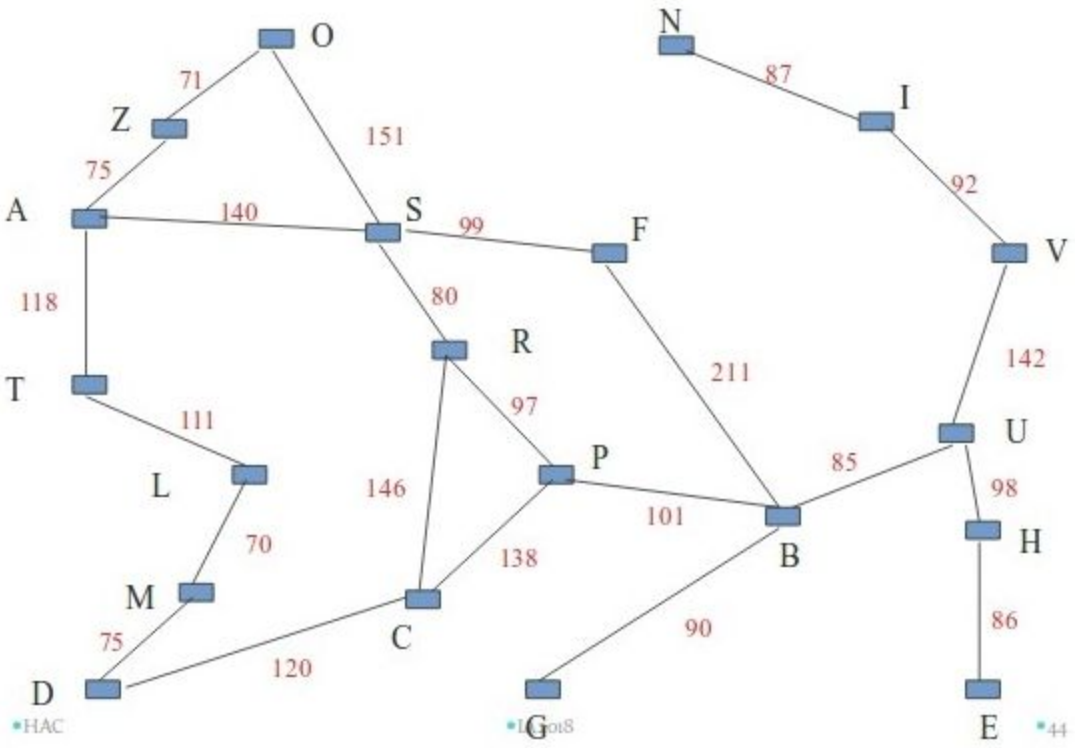

Imagem 5 - Grafo das cidades. Nossa instância começa em O e têm como objetivo chegar à B

Cidade	Distância até a cidade B
A	366
B	0
C	160
D	242
E	161
F	178
G	77
H	151
I	226
L	244
M	241
N	234
O	380
P	98
R	193
S	253
T	329
U	80
V	199
Z	374

Imagem 6: Tabela de função heurística

2.2. Desenvolvimento

```
## Função para calcular as distâncias em linha reta de todas as cidades para o objetivo
geraDistancias <- function(obj) {
  ## Na verdade só temos as distâncias para a cidade B, então obj = "B"
  distancias <- matrix(c(366,0,160,242,178,77,244,241,380,98,193,253,329,80,374))
  rownames(distancias) <- nomeCidades()
  colnames(distancias) <- "B"

  return(distancias[obj,])
}

## Sobrecarga da função genérica "heuristica", definida por Estado.R
heuristica.TrajetoCidades <- function(atual) {
  return(geraDistancias(atual$desc))
}
```

Imagem 7: Implementação da função heurística para o problema 2.

A implementação da geração de nós filhos neste problema é bastante intuitiva, pois consiste apenas de expandir o nó atual. Os filhos então serão todos os nós conectados a ele ao checarmos a matriz de transições, ou seja, as cidades vizinhas.

```r
## Função de geração de estados filhos
geraFilhos <- function(obj) {
    filhosDesc <- list()
    filhos <- list()

    # gera uma lista de vizinhos do nó atual a partir do grafo global
    vizinhos <- obj$cidades[obj$desc,]
    # remove os nós que não tem conexão (custo infinito)
    vizinhos <- vizinhos[vizinhos != Inf]
    # gera lista de nomes dos nós filhos
    filhosDesc <- names(vizinhos)

    ## gera os objetos TrajetoCidades para os filhos
    for(filhoDesc in filhosDesc) {
      filho <- TrajetoCidades(desc = filhoDesc, pai = obj)
      filho$h <- heuristica(filho)
      filho$g <- vizinhos[filho$desc]
      filho$f <- filho$h + filho$g
      filho$cidades <- obj$cidades
      filhos <- c(filhos, list(filho))
    }

    return(filhos)
}
```

Imagem 8: Implementação da função de geração de filhos para o problema 2.

2.3. Resultados

A. Algoritmo Greedy

Ao utilizar o algoritmo Greedy, encontramos como resultado o trajeto **O-S-F-B**. O algoritmo guloso minimiza o custo estimado para atingir um objetivo, de forma que expande o primeiro nó considerado mais perto do objetivo - ou seja, o de menor valor -, tal que sua função de avaliação é a mesma que a heurística. Vale ressaltar que não necessariamente esse é o caminho menos “custoso”.

B. Algoritmo A*

Ao utilizar o algoritmo A*, encontramos como resultado o trajeto **O-S-R-P-B**. Este algoritmo nos garante a solução ótima. Utiliza-se uma função heurística consistente, tal que para todo nó X e para todo sucessor

X’ gerado por uma dada ação, o custo estimado para se alcançar o objetivo de X não for maior que o custo de chegar a X’ mais o custo estimado para se alcançar o objetivo à partir de X’.

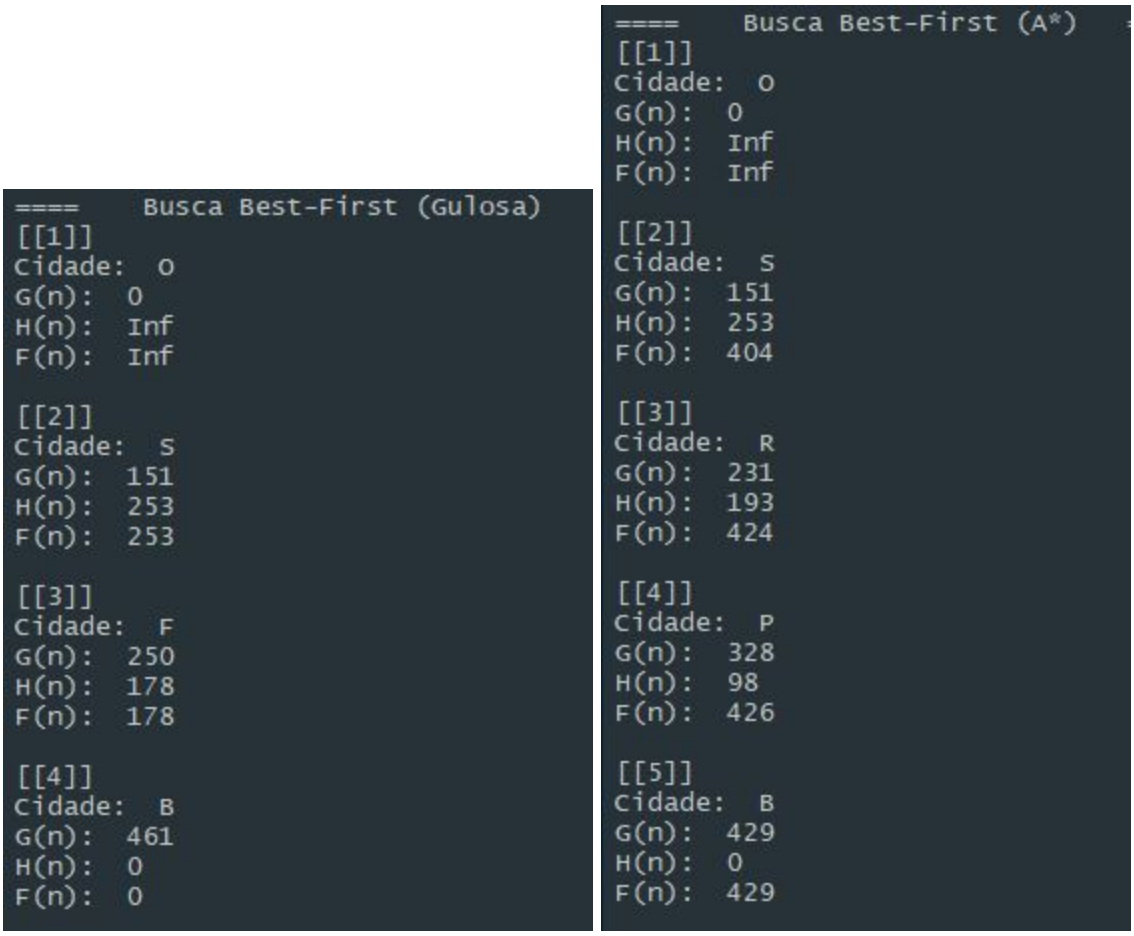

Imagem 9: Resultados dos algoritmos de Busca Informada aplicados na instância proposta.

Por mais que tenhamos passado por menos cidades ao utilizar o Greedy, podemos afirmar com base na matriz de transições e na tabela de heurística que neste exercício encontramos a melhor solução através da execução do algoritmo A*, sendo a solução ótima o caminho **O-S-R-P-B**.