Inteligência Artificial - Lista de Exercícios Teóricos extendido - 01

Roberto Sérgio Ribeiro de Meneses - 520403

30/11/2024

1. Diferença entre busca informada e busca não informada

- **Busca informada**
  - Faz uso de heurísticas (dados prévios utilizados no momento da implementação que aprimoram a solução).
- **Busca não informada**
  - Executam buscas sem aproveitar o conhecimento prévio sobre o problema em questão.

2. Problema dos missionários e canibais utilizando busca em largura

```
// Realiza o (BFS) para encontrar o caminho do estado inicial até o estado destino.
Função BFS(estadoInicial):
    estadoDestino <- (0, 0, 0)
    filaDeEstados <- [(estadoInicial, caminhoPercorrido=[])]
    estadosVisitados <- Conjunto vazio
    Enquanto filaDeEstados NÃO estiver vazia:
        (estadoAtual, caminhoPercorrido) <- filaDeEstados.desenfileirar()
        Se estadoAtual em estadosVisitados:
            Continuar para o próximo estado
        Adicionar estadoAtual em estadosVisitados
        novoCaminho <- caminhoPercorrido + [estadoAtual]
        Se estadoAtual == estadoDestino:
            Retornar novoCaminho
        Para cada proximoEstado em gerarEstadosVizinhos(estadoAtual):
            Enfileirar (proximoEstado, novoCaminho)
    Retornar NULO

// Gera os possíveis estados vizinhos para um estado atual.
Função gerarEstadosVizinhos(estado):
    (missRestante, caniRestantes, posicaoBarco) <- estado
    movimentosPossiveis <- [(1, 0), (0, 1), (1, 1), (2, 0), (0, 2)]
    estadosGerados <- Lista vazia
    Para cada (deltaM, deltaC) em movimentosPossiveis:
        Se posicaoBarco == 1:
            novoEstado <- (missRestante - deltaM, caniRestantes - deltaC, 0)
        Caso contrário:
            novoEstado <- (missRestante + deltaM, caniRestantes + deltaC, 1)
        Se estadoSeguro(novoEstado):
            Adicionar novoEstado em estadosGerados
```

```
    Retornar estadosGerados

// Verifica se um estado é seguro.
Função estadoSeguro(estado):
    (missRestante, caniRestantes, posicaoBarco) <- estado
    Retornar missRestante >= 0 e caniRestantes >= 0 e
           missRestante <= 3 e caniRestantes <= 3 e
           (missRestante == 0 OU missRestante >= caniRestantes) e
           ((3 - missRestante) == 0 OU (3 - missRestante) >= (3 - caniRestantes))

// main
estadoInicial <- (3, 3, 1)
solucao <- BFS(estadoInicial)
Se solucao NÃO for NULO:
    Imprimir "Solução encontrada:"
    Para cada passo em solucao:
        Imprimir passo
Caso contrário:
    Imprimir "Não existe solução"
```

3. Determinação de nós visitados em busca em largura e profundidade

- **Busca em largura:**
  - Expande a fronteira para os nós B e C, depois para D, E, F e G; então expande novamente para H, e por fim para I. Portanto, todos os nós são visitados.
- **Busca em profundidade:**
  - Desce pela esquerda em B ao expandir a fronteira, olhando B e D; então pela direita de B, olhando E e H; por fim, expande o lado direito de E e encontra I.
- **Conclusão:** O DFS foi mais eficiente neste caso, realizando 6 checagens, pois encontrou o nó objetivo rapidamente explorando da esquerda para a direita. O BFS fez 9 checagens devido à sua abordagem exaustiva, mas garente o caminho mais curto.

4. Heurística composta no quebra-cabeça das 8 peças

Para resolver o quebra-cabeça das 8 peças, foram utilizadas duas heurísticas como parte do algoritmo A*:

- **Heurística 1 (h1):** Distância de Manhattan entre o local da peça e o local correto.
- **Heurística 2 (h2):** Número de peças na posição incorreta.

A heurística composta é definida como:

$$h_{\text{composta}}(n) = h_1(n) + h_2(n)$$

Cada heurística é admissível separadamente:

$$h_1(n) \leq h^*(n) \quad \text{e} \quad h_2(n) \leq h^*(n)$$

Entretanto, a soma pode resultar em:

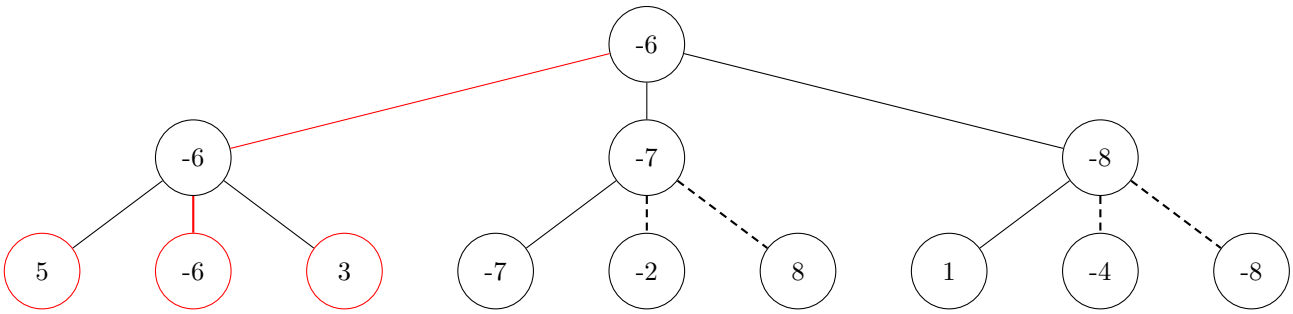
$$h_{\text{composta}}(n) > h^*(n)$$

pois h_1 e h_2 podem contar parcialmente o mesmo custo, levando a uma superestimação.

5.Verifique completude e otimalidade do algoritmo de busca em largura em um problema de busca onde existem ações com dois custos diferentes (c1 = 1 e c2 = 2).

Dado um Grafo G = (V,E) com V = n e E = e, contra exemplo para otimalidade, dado um caminho p composto por $[e_0, e_1, e_2, e_3...]$ onde pelo menos um deles tem custo = 2, e o caminho p_2 onde $\forall$ e Ep tem custo =1.
OBS: p_1 e p_2 tem como f o destino se $p_1 = p_2$ e p_1 começar a mais a esquerda em V_0 então a solução não será mais otima.

6.Considere a seguinte árvore de um jogo de soma 0, no qual as utilidades mostradas nos nós-folha são para o primeiro jogador (A) que é um MAXimizador. Suponha que o jogador (B) é um MINimizador.



a)

Foi realizado na imagem acima

b)

Infelizmente não consegui circular, mas modifiquei a cor para vermelho.

c)

Infelizmente não consegui fazer um x nas conexões podadas, mas coloquei elas como linhas pontilhadas.

7. Seja o problema do jogo da velha. Definimos X_n como o número de linhas, colunas ou diagonais com exatamente n valores de X e nenhum valor de O (análogo para O). A função de utilidade atribui +1 a qualquer posição com X_3= 1 e -1 a qualquer posição com O_3= 1. Todas as outras posições terminais têm utilidade 0. No caso de posições não-terminais, utilizamos uma função de avaliação linear definida como

6.2 Explique a estratégia implementada no algoritmo simulated annealing para fugir de máximos/mínimos locais.

O Simulated Annealing é um algoritmo de otimização que permite aceitar soluções piores de forma controlada para escapar de máximos ou mínimos locais. Isso é feito com base em uma probabilidade que depende de um parâmetro chamado temperatura, que diminui ao longo do tempo.
No início, soluções piores são mais facilmente aceitas, promovendo maior exploração. Com o tempo, a aceitação dessas soluções diminui, permitindo o refinamento da busca e aumentando as chances de encontrar uma solução ótima

7.2 Qual o papel do operador de mutação em um algoritmo genético? O que pode ocorrer se a taxa de mutação for muito alta? E se for muito baixa?

O operador de mutação em um algoritmo genético tem o papel de introduzir diversidade nas soluções da população. Ele altera aleatoriamente uma parte de uma solução (cromossomo), permitindo ao algoritmo explorar novas áreas do espaço de busca e evitar que a busca fique restrita a soluções subótimas ou locais. A mutação é essencial para garantir que o algoritmo não se prenda a uma única região do espaço de soluções.

- **Taxa de mutação muito alta:**
  - o algoritmo pode se tornar aleatório, desfazendo rapidamente boas soluções geradas por cruzamento. Isso pode levar a uma busca ineficaz, pois o processo de mutação pode destruir as soluções promissoras antes que elas tenham a chance de ser refinadas.
- **Taxa de mutação muito baixa**
  - o algoritmo pode se tornar muito conservador. Isso limita a diversidade das soluções e dificulta a exploração de novas possibilidades. Como resultado, o algoritmo pode sofrer de convergência prematura, reduzindo as chances de encontrar a solução ótima.

8. No algoritmo de colônia de formigas, como os fatores α e β influenciam no comportamento do método.

O parâmetro α determina a influência do feromônio, dando maior peso aos caminhos já explorados. Já β controla a importância da distância, priorizando caminhos mais curtos.