MAC 5788 - Planejamento em Inteligência Artificial

Diego de Araújo Martinez Camarinha
7157092

25 de Março de 2015

1 Introdução

Este relatório diz respeito à implementação de três estratégias para buscas em grafo, com o intuito de resolver o problema dos Missionários e Canibais. Ele está dividido da seguinte maneira: na seção 2, enunciamos o problema; na seção 3, cada uma das estratégias utilizadas é descrita; na seção 4, é explicado como rodar o programa e o que cada uma das classes implementadas faz e, por fim, na seção 5 são apresentados alguns resultados.

2 Enunciado

No problema dos missionários e canibais, três missionários e três canibais devem atravessar um rio com um barco que pode transportar no máximo duas pessoas. Em cada margem, se há missionários presentes, eles devem estar em número maior ou igual ao número de canibais. Caso contrário, os canibais comem os missionários. O barco não pode atravessar o rio por si só, sem pessoas a bordo. O objetivo é que todos os missionários e canibais cheguem na outra margem.

3 Estratégias

Neste exercício, foram implementadas 3 estratégias para a seleção do próximo nó a ser visitado, para o algoritmo de busca em grafos visto em aula. Duas delas, FIFO e LIFO, escolhem o próximo nó sem considerar informações sobre o problema. No nosso caso, significa que elas não usam dados sobre os custos dos caminhos e nem heurísticas para estimar a distância até a solução. A outra, A*, usa informação do problema e faz estimativas da distância até a meta.

3.1 Estratégias FIFO e LIFO

Na estratégia FIFO (**First In, First Out**), incluímos os nós da borda em um vetor e, para escolhermos qual deles será o próximo a ser explorado, seguimos a regra de uma **fila**: *o primeiro que chegou é o primeiro a sair*. Ou seja, o primeiro nó colocado no vetor será o primeiro a ser explorado. Buscas em grafos com essa estratégia são conhecidas como **buscas em largura**.

Na estratégia LIFO (**Last in, first out**), também adicionamos os nós da borda em um vetor. Porém, para escolhermos qual deles será o próximo a ser explorado seguimos a regra de uma **pilha**: *o primeiro que entrou é o último a sair*. Nesse caso, o primeiro nó colocado no vetor será o último a ser

explorado. Buscas em grafos que usam essa estratégia são conhecidas como **buscas em profundidade**.

3.2 A*

Ao contrário das anteriores, aqui usaremos informações sobre o problema para realizar a busca. Em particular, levaremos em consideração o custo de cada ação. Também usaremos heurísticas para fornecer uma estimativa da distância até a meta.

Foram implementadas duas heurísticas. Ambas foram criadas a partir do relaxamento de uma restrição do problema: não consideramos que canibais podem comer missionários. Assim, podemos computar quantas viagens são necessárias para que todos atravessem o rio. O barco tem capacidade para duas pessoas, mas depois de cada viagem para a margem destino, o barco tem que voltar para a margem de origem. Portanto, pelo menos uma pessoa precisa voltar também. Isso nos leva a seguinte heurística:

$$h_1(n) = (NumeroDePessoasNaMargemInicial) - 1$$

Essa heurística é admissível porque todas as viagens (menos a última) resultam na trasferência de, no máximo, uma pessoa para a margem destino.

Outra heurística resultante desse relaxamento é:

$$h_2(n) = \frac{NumeroDePessoasNaMargemInicial}{CapacidadeDoBarco}$$

Essa heurística também é admissível e é englobada pela primeira, já que $\forall$ n, n/2 $\geq$ n - 1.

4 Detalhes de Implementação

Para rodar o programa, é preciso ter o Ruby instalado. Depois, no diretório raíz do programa, execute:

ruby missionaries_cannibals.rb <strategy>

Onde <strategy> pode ser uma das seguintes opções: LIFO, FIFO ou Astar.

Cada arquivo funciona da seguinte maneira:

- **missionaries_cannibals.rb**: É o arquivo que roda o problema. Ele prepara o nó inicial, a meta, roda o algoritmo e imprime os resultados.
- **requirements.rb**: Arquivo que importa todos os outros.
- **state.rb**: Define um estado do problema. Um estado pode calcular seus sucessores por meio do método *successors*. Cada estado possui um dicionário com duas chaves, *left* e *right*. Cada uma dessas chaves possui como valor outro dicionário, com chaves *missionaries*, *cannibals* e *boat*, que representam o número de missionários, canibais e um booleano que indica se o barco está daquele lado, respectivamente. Por exemplo, para o estado inicial temos:

  {left: {missionaries: 3, cannibals: 3, boat: true}, right: {missionaries: 0, cannibals: 0, boat: false}}

- **action.rb**: Modela as possíveis ações como um dicionário. As chaves são os nomes da ações e os valores são as pessoas que elas movimentam. Por exemplo, a ação de transferir dois canibais de uma margem para outra é:

  cc: {missionaries: 0, cannibals: 2}

  Também faz o cálculo das ações aplicáveis a um determinado estado, retornando uma matriz na qual cada linha possui o nome da ação e o estado resultante.

- **node.rb**: Modela um nó do grafo. Possui os seguintes atributos: um estado, um pai, a ação que foi executada no pai para alcançá-lo, o custo do caminho para alcançá-lo e sua profundidade na busca. O método *is_goal_state?* verifica se o nó tem o estado meta. O método *expand* expande o nó, retornando os nós que são alcançáveis por ele através de uma ação.
- **search.rb**: Possui apenas métodos estáticos. Faz a busca no método *graph_search*. Imprime o caminho até um nó no método *path_to*. Verifica se um nó já foi visitado e se ele já está na borda com custo maior nos métodos *already_visited?* e *insert_on*, respectivamente.
- **strategy.rb**: Fábrica abstrata de estratégias. O método estático *strategies* imprime as estratégias disponíveis.

- **a_star.rb**: Implementa a estratégia que usa informação do problema. Possui métodos para as duas heurísticas citadas. O nó com menor valor calculado pela *evaluation_function* e a borda atualizada são retornados pelo método *select_node_from*. Ruby não possui um limite máximo para inteiros, portanto, para a comparação inicial, foi estipulado um valor alto o suficiente que fosse maior que qualquer valor que possa ser retornado pela função de avaliação. Para problemas que possam gerar um número grande de nós na borda, o recomendado é que seja implementado uma estrutura de *heap*. Para trocar a heurística utilizada, é preciso mudar o código da função de avaliação.

- **fifo.rb**: Implementa a estratégia de FIFO. Remove o primeiro elemento da borda. Retorna esse elemento e a borda atualizada.

- **lifo.rb**: Implementa a estratégia de LIFO. Remove o último elemento da borda. Retorna esse elemento e a borda atualizada.

5 Resultados

Os resultados para cada estratégia são apresentado a seguir:

5.1 A*

```
ruby missionaries_cannibals.rb Astar
Initial state:  {:left=>{:missionaries=>3, :cannibals=>3, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>0, :boat=>false}}
Goal state:     {:left=>{:missionaries=>0, :cannibals=>0, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>3, :boat=>true}}

================ Results ================
Path cost: 11
Depth: 11
Visited nodes: 14
        Action  | State
        cc      | {:left=>{:missionaries=>0, :cannibals=>0, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>3, :boat=>true}}
        c       | {:left=>{:missionaries=>0, :cannibals=>2, :boat=>true}, :right=>{:missionaries=>3, :cannibals=>1, :boat=>false}}
        cc      | {:left=>{:missionaries=>0, :cannibals=>1, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>2, :boat=>true}}
        c       | {:left=>{:missionaries=>0, :cannibals=>3, :boat=>true}, :right=>{:missionaries=>3, :cannibals=>0, :boat=>false}}
        mm      | {:left=>{:missionaries=>0, :cannibals=>2, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>1, :boat=>true}}
        mc      | {:left=>{:missionaries=>2, :cannibals=>2, :boat=>true}, :right=>{:missionaries=>1, :cannibals=>1, :boat=>false}}
        mm      | {:left=>{:missionaries=>1, :cannibals=>1, :boat=>false}, :right=>{:missionaries=>2, :cannibals=>2, :boat=>true}}
        c       | {:left=>{:missionaries=>3, :cannibals=>1, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>2, :boat=>false}}
        cc      | {:left=>{:missionaries=>3, :cannibals=>0, :boat=>false}, :right=>{:missionaries=>0, :cannibals=>3, :boat=>true}}
        c       | {:left=>{:missionaries=>3, :cannibals=>2, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>1, :boat=>false}}
        cc      | {:left=>{:missionaries=>3, :cannibals=>1, :boat=>false}, :right=>{:missionaries=>0, :cannibals=>2, :boat=>true}}
                | {:left=>{:missionaries=>3, :cannibals=>3, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>0, :boat=>false}}

Running time: 0.002122873
```

5.2 LIFO

```
ruby missionaries_cannibals.rb LIFO
Initial state:  {:left=>{:missionaries=>3, :cannibals=>3, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>0, :boat=>false}}
Goal state:     {:left=>{:missionaries=>0, :cannibals=>0, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>3, :boat=>true}}

================ Results ================
Path cost: 11
Depth: 11
Visited nodes: 14
        Action  | State
        mc      | {:left=>{:missionaries=>0, :cannibals=>0, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>3, :boat=>true}}
        m       | {:left=>{:missionaries=>1, :cannibals=>1, :boat=>true}, :right=>{:missionaries=>2, :cannibals=>2, :boat=>false}}
        cc      | {:left=>{:missionaries=>0, :cannibals=>1, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>2, :boat=>true}}
        c       | {:left=>{:missionaries=>0, :cannibals=>3, :boat=>true}, :right=>{:missionaries=>3, :cannibals=>0, :boat=>false}}
        mm      | {:left=>{:missionaries=>0, :cannibals=>2, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>1, :boat=>true}}
        mc      | {:left=>{:missionaries=>2, :cannibals=>2, :boat=>true}, :right=>{:missionaries=>1, :cannibals=>1, :boat=>false}}
        mm      | {:left=>{:missionaries=>1, :cannibals=>1, :boat=>false}, :right=>{:missionaries=>2, :cannibals=>2, :boat=>true}}
        c       | {:left=>{:missionaries=>3, :cannibals=>1, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>2, :boat=>false}}
        cc      | {:left=>{:missionaries=>3, :cannibals=>0, :boat=>false}, :right=>{:missionaries=>0, :cannibals=>3, :boat=>true}}
        m       | {:left=>{:missionaries=>3, :cannibals=>2, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>1, :boat=>false}}
        mc      | {:left=>{:missionaries=>2, :cannibals=>2, :boat=>false}, :right=>{:missionaries=>1, :cannibals=>1, :boat=>true}}
                | {:left=>{:missionaries=>3, :cannibals=>3, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>0, :boat=>false}}

Running time: 0.001365186
```

5.3 FIFO

```
ruby missionaries_cannibals.rb FIFO
Initial state:  {:left=>{:missionaries=>3, :cannibals=>3, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>0, :boat=>false}}
Goal state:     {:left=>{:missionaries=>0, :cannibals=>0, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>3, :boat=>true}}

================ Results ================
Path cost: 11
Depth: 11
Visited nodes: 15
        Action  | State
        cc      | {:left=>{:missionaries=>0, :cannibals=>0, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>3, :boat=>true}}
        c       | {:left=>{:missionaries=>0, :cannibals=>2, :boat=>true}, :right=>{:missionaries=>3, :cannibals=>1, :boat=>false}}
        cc      | {:left=>{:missionaries=>0, :cannibals=>1, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>2, :boat=>true}}
        c       | {:left=>{:missionaries=>0, :cannibals=>3, :boat=>true}, :right=>{:missionaries=>3, :cannibals=>0, :boat=>false}}
        mm      | {:left=>{:missionaries=>0, :cannibals=>2, :boat=>false}, :right=>{:missionaries=>3, :cannibals=>1, :boat=>true}}
        mc      | {:left=>{:missionaries=>2, :cannibals=>2, :boat=>true}, :right=>{:missionaries=>1, :cannibals=>1, :boat=>false}}
        mm      | {:left=>{:missionaries=>1, :cannibals=>1, :boat=>false}, :right=>{:missionaries=>2, :cannibals=>2, :boat=>true}}
        c       | {:left=>{:missionaries=>3, :cannibals=>1, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>2, :boat=>false}}
        cc      | {:left=>{:missionaries=>3, :cannibals=>0, :boat=>false}, :right=>{:missionaries=>0, :cannibals=>3, :boat=>true}}
        c       | {:left=>{:missionaries=>3, :cannibals=>2, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>1, :boat=>false}}
        cc      | {:left=>{:missionaries=>3, :cannibals=>1, :boat=>false}, :right=>{:missionaries=>0, :cannibals=>2, :boat=>true}}
                | {:left=>{:missionaries=>3, :cannibals=>3, :boat=>true}, :right=>{:missionaries=>0, :cannibals=>0, :boat=>false}}

Running time: 0.003291261
```

5.4 Comparação

Das figuras anteriores, podemos ver que todas as estratégias encontraram caminhos de custos e profundidades iguais. Porém, a estratégia FIFO visitou mais nós do que as outras. Isso é de se esperar, já que ela corresponde a uma busca em largura. Também podemos ver que a heurística usada conseguiu reduzir o número de nós visitados no A*. Isso indica que ela ajudou a guiar o algoritmo a encontrar uma meta mais diretamente.

Das três estratégias, a que consumiu menos tempo foi a LIFO. Porém, os tempos de todas foram baixos. Para realmente entender a diferença no tempo de execução, precisaríamos rodar esses algoritmos em problemas maiores.

Por fim, é interessante notar que as estratégias FIFO e A* geraram soluções iguais, mas diferentes da solução gerada pela estratégia LIFO.