Caio Ferreira Bernardo – 9276936
Caique Honório Cardoso – 8910222
Matheus Aparecido do Carmo Alves – 9791114
Rafael Rodrigues Santana – 7594375
William Luis Alves Ferreira – 9847599

Projeto 1 da Disciplina Inteligência Artificial:
OS 3 CANIBAIS E OS 3 MISSIONÁRIOS

São Carlos, SP

22 de abril de 2019

1 Introdução

Quando se estuda a área de *IA (Inteligência Artificial)* em Ciência da Computação, esbarra-se inevitavelmente com o que se define como *Algoritmos de Busca*. Responsáveis por apresentar uma das base de pesquisa do que conhecemos da IA atualmente, define-se em termos gerais que esses algoritmos tomam um problema como entrada e buscam encontrar uma solução após a execução de um número possível de ações diferentes dentro do problema dado, isto é, eles tentam encontrar uma solução de um problema modelado após realizar um número viável de *simulações* deste.

Assim, para aplicação de um Algoritmo de Busca Inteligente é necessário que: **(i)** exista um modelo que represente o problema de estudo; **(ii)** exista uma estrutura de dados que implemente e comporte computacionalmente a modelagem proposta, e; **(iii)** exista um objetivo final claro dado um cenário inicial (queira a solução exista ou não dado o cenário inicial). Formalmente, pode-se descrever que:

- o **modelo** é composto por um conjunto de estados $\mathbf{S} = \{\mathbf{s_1}, ..., \mathbf{s_n}\}$, um conjunto de estados terminais $\mathbf{F} \in \mathbf{S}$, uma conjunto de ações $\mathbf{A} = \{\mathbf{a_1}, ...\mathbf{a_k}\}$ e uma função de transição $T : \mathbf{S} \rightarrow A' \in \mathbf{A}$.
- a **estrutura de dados** usualmente é apresentada como um **grafo** ou uma **árvore de busca** que comporte os requisitos do modelo.
- o objetivo é uma **função de maximação** e análise de busca que segue a modelagem e atribui um **valor** à estados seguindo: $f(s) = g(s) + h(s) + ...$, onde geralmente f é uma função de *busca cega* ou *heurística*.

Dado esta breve introdução, este projeto possui como principal proposito cumprir a modelagem e resolução de um problema de Busca pela implementação uma abordagem *cega* e outra *heurística* para comparação e análise de desempenho.

Neste capítulo, será apresentado o problema escolhido para estudo (Seção 1.1) e será discutido brevemente a complexidade para resolução deste problema, justificando a necessidade da aplicação dos conhecimentos de IA (Seção 1.2).

1.1 Descrição do Problema: Os 3 Canibais e os 3 Missionários

Descrito como um tradicional *Problema de Travessia*, o problema dos "3 Canibais e 3 Missionários" é ilustrado pela Figura[1] 1 segue o seguinte enunciado :

> *"Três canibais e três missionários estão viajando juntos e chegam à margem de um rio. Eles desejam atravessar para a outra margem para, desta forma, continuar a viagem. O único meio de transporte disponível é um barco que comporta no máximo duas pessoas. Porém há um problema: em nenhum momento o número de canibais pode ser superior ao número de missionários em um dos lados do rio, pois desta forma os missionários estariam em grande perigo de vida. Como administrar a travessia de forma a não colocar os missionários em risco e garantir que todos atravessem o rio?"*

Figura 1 – Ilustração do Problema da Travessia dos "3 Canibais e 3 Missionários".

Exposto o problema, tem-se como objetivo do projeto modelar e realizar uma implementação computacionalmente viável da situação que possibilite a busca de uma solução em um espaço de busca gerado.

1.2 Discussão sobre o Problema da Travessia proposto em IA

O problema proposto mesmo em um primeiro momento podendo parecer trivial para resolução (dado que só existem um único estado inicial e um único estado final), devido a quantidade possível de *combinações* para travessia, *restrições* de análise para cada movimento e *movimentos* possíveis para se realizar em uma etapa sem direta avaliação de valor, este problema se tornou um desafio nos fundamentos da Busca para a Inteligencia Artificial.

Abordando e encarando o problema de maneira lógica, ao tentar-se tomar a primeira decisão de "qual é o melhor grupo para se atravessar" (dado o primeiro estado) incertezas surgem na tomada desta decisão, pois:

[1] Disponível em Wikipedia - Problema dos canibais e missionários. Acessado: 22 de abril de 2019.

- se 1 canibal e 1 missionário atravessam o rio ou 2 canibais atravessam o rio, tem-se ao final da primeira travessia que 2 pessoas já se encontram do lado correto do rio;
- agora: seria melhor que o barco retorne com 1 canibal ou 1 missionário para solucionar o problema? Qual das duas opções garante maior sucesso?

Sobre esta perspectiva, tem-se que, para discernir qual a melhor ação a se tomar, é necessário "olhar-se à frente" e analisar os efeitos de cada ação no futuro. Esta característica torna o problema mais complexo, sendo pouco trivial resolvê-lo sem apoio computacional ou treino sobre as possíveis consequências que cada ação gera - como ocorre similarmente no xadrez ou em *"jogos de economia e barganha"* ligados à área de Teoria dos Jogos em computação.

Ademais, note que tem-se na descrição deste problema um **estado inicial** (todos de um lado do rio), um **objetivo/estado final** (atravessar todos para o outro lado do rio), **conjunto de ações** (atravessar 1 canibal, 1 canibal e 1 missionário, etc) e **restrições** (o número de missionários deve ser maior ou igual ao de canibais de ambos os lados do rio) dados, encaixando-se perfeitamente nas condições para modelagem de um problema de busca em IA e justificando a proposta deste problema para este projeto da disciplina.

2 Metodologia

Neste capítulo, será apresentada toda a metodologia utilizada para desenvolvimento do projeto. Primeiramente será introduzido na Seção 2.1 o *modelo* criado para o problema proposto. Na Seção 2.2, será discutido e apresentado de maneira sucinta a *estrutura de dados* utilizada para implementar a modelagem sugerida. Por fim na Seção 2.3, a definição da *função heurística* será explorada para exibir o principal ponto diferencial da implementação de algoritmos de busca heurísticos e cegos.

2.1 Modelo

Para a modelagem do problema, os elementos necessários (**estados**, **estado inicial**, **estado final**, **operadores** (ou conjunto de ações) e **restrições**) foram representados por atributos, valores numéricos, operações matemáticas e verificações, da seguinte forma:

1. Os **estados** foram representados pelos seguintes atributos e valores:
   - **side: +1 ou -1**, representa em qual lado do rio o barco se encontra;
   - **status: (canibais,missionários)**, representa quantos canibais e quantos missionários se encontram do outro lado do rio.
2. O **estado inicial** é definido pelos atributos com os seguintes valores:
   - **side = 1** - O barco se encontra no lado inicial do rio;
   - **status = (0,0)** - Não há nenhum missionário ou canibal do outro lado do rio.
3. O **estado final** é definido pelo atributo status com os seguintes valores:
   - **side = -1** - O barco se encontra no lado final do rio;
   - **status = (3,3)** - Todos os missionários e canibais se encontram do outro lado do rio.

Os **operadores** são o movimento de uma ou duas pessoas através do rio e são representados pela tupla **action = (canibais,missionários)**, que indica quantos canibais e quantos missionários estão se deslocando durante essa ação. As possíveis tuplas para *action* são: {(2,0),(0,2),(1,1),(1,0),(0,1)}.

A cada operação, o atributo side do próximo estado é alternado entre +1 e -1 (simbolizando a ida e a volta do barco), e o atributo status é atualizado de acordo com

o lado em que se encontrava o barco e a quantidade de missionários que se deslocaram, seguindo a função:

$$newstatus = status + side * action. \tag{2.1}$$

Porém antes de uma ação ser executada e completa, i.e, um novo estado ser gerado, sua validade deve ser verificada de acordo com as restrições do problema. As **restrições** do problema podem ser reogranizados da seguinte forma:

1. O número de canibais deve ser menor ou igual ao número de missionários em cada lado do rio;
2. O número de canibais e de missionários deve ser maior ou igual a 0 em cada lado do rio.

2.2 Estrutura de Dados

Para a resolução desse problema, a estrutura de dados escolhida foi a modelagem computacional em Árvore.

Nessa estrutura, os nós representam os estados possíveis do problema e as arestas representam as possíveis operações realizadas em cada nó, ou estado (Figura 2).

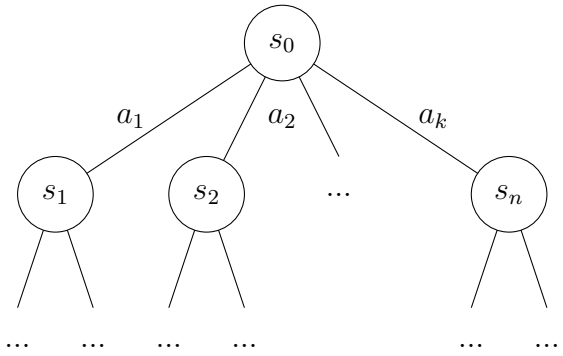

Figura 2 – Ilustração da Árvore de Busca utilizada.

A árvore é gerada a partir do nó raiz, ou estado inicial. Nele são aplicadas todas as operações possíveis (no caso desse problema, as 5 diferentes tuplas possíveis) e os resultados são verificados de acordo com as restrições. Caso o resultado dessa operação atenda às restrições, um novo nó, chamado de nó-filho, é gerado na árvore.

Quando as 5 diferentes aplicações tiverem sido analisadas e todos os nós-filhos adicionados, o foco da análise passa a ser um dos nós-filhos. Primeiramente, é verificado se o nó atende ao critério de parada, ou seja, se o estado que ele representa atende à definição de estado final. Caso ele represente um estado final, a busca é interrompida e o nó em

questão é retornado como solução. Caso ele não represente um estado final, novamente as 5 operações são aplicadas a ele e seus resultados analisados, gerando assim mais possíveis nós filhos.

A sequência de nós a serem analisados depende do tipo de busca, mas sempre que um nó está em foco na análise ele é comparado com os critérios de parada e, caso não os atenda, todas as operações são aplicadas a ele para gerar os nós-filhos.

Caso um nó não tenha filhos, a análise continua no próximo nó da sequência. Caso todos os nós sejam analisados e nenhum corresponda aos critérios de parada, a busca é encerrada, sem solução.

2.3 Função Heurística

Na implementação da busca heurística foi utilizada a seguinte função heurística:

$$f(x) = g(x) + h(x) \tag{2.2}$$

onde g é a função de custo e h é a função de análise.

Dado a metodologia, modelo e estrutura de dados apresentados, abordou-se o problema adotando as funções g e h sendo:

- g é a função de custo obtida pelo calculo da **distância de Manhattan** somado ao **número de passos** entre o *"status"* do estado atual e o *"status"* do estado final;

$$g(x) = Manhattan(x, x') + Depth(x) \tag{2.3}$$

, com x sendo o estado atual e x' sendo o estado final

- h é a função de análise pela aplicação da heurística de **Variável menos Restritiva** para o estados futuros dado o conjunto de ações modelados.

$$h(x^t) = Restrictions(x^{t+1}) \tag{2.4}$$

, com x^t sendo o estado atual no tempo t e x^{t+1} o próximo estado, no tempo $t + 1$.

Aplicando está função f definida, garantiu-se (de maneira empírica) que a heurística é *admissível* para o problema de estudo.

3 Implementação e Desenvolvimento

Neste capítulo serão explicitadas as tecnologias utilizadas para modelagem, implementação, execução e teste da proposta no documento (Seção 3.1). Ademais, será descrito de maneira sucinta o pacotes de códigos desenvolvidos no decorrer do projeto (Seção 3.2).

3.1 Tecnologias Utilizadas

O projeto foi implementado integralmente na linguagem *Python (v2.7)* de programação. Todos os pacotes utilizados na modelagem do problema são de autoria dos integrantes do grupo e foram escritos de maneira conjunta.

O código encontra-se disponível gratuitamente para consulta online em GitHub-Micanga (Acessado em 22 de abril de 2019).

O controle de versões do código, assim como dos requisitos do projeto, foi feito com auxilio da plataforma *GitHub*[1].

Os testes foram executados através de um *"script"* implementado em *Python*. Todos os dados gerados foram analisados e exportados como gráficos utilizando o pacote da linguagem *Matplotlib*[2].

As máquinas utilizadas para testes possuem as configurações presentes na Tabela 1.

Sistema Operacional	Ubuntu 18.04.2 LTS
Memória	3.7 GiB
Processador	Intel Core i3-3110M CPU
Tipo do SO	64-bit
Disco	491.2 GB

Tabela 1 – Configurações das máquinas utilizadas para execução dos testes.

Todos os testes foram executados 50 vezes cada e com processo dedicado.

3.2 Implementação

Foram implementados 4 pacotes principais de código *Python* para execução do projeto. Os pacotes são: **main.py**, **SearchTree.py**, **State.py** e **a_star.py**.

[1] Acesso a plataforma online pelo endereço https://github.com.
[2] Documentação disponível em Matplotlib: Python plotting. Acessado: 22 de abril de 2019.

- O pacote *main.py* possui a implementação da rotina principal do programa que realiza a inicialização do modelo proposto, criação do espaço de busca e execução dos métodos de busca.

- O pacote *SearchTree.py* possui a implementação das duas classes usadas para a busca: a Árvore de Busca e o Nó da Árvore de Busca. A árvore possui 2 parâmetros (nó raiz e a profundidade máxima permitida) e a implementação dos métodos úteis de seu TAD (Tipo Abstrato de Dado). Os nós possuem 4 parâmetros (profundidade, nó pai, nós filhos e o estado associado), assim como os métodos úteis de seu TAD.

- O pacote *State.py* possui a implementação da classe do estado definido pela modelagem sendo composto assim por 4 parâmetros: número total de canibais, número total de missionários, o lado do rio que estamos analisando e o *"status"* associado. Além disso, o objeto dessa classe possui 3 métodos:

  - **move(self,action)**: método por é responsável de retornar o estado gerado dado o estado atual e a ação realizada (retornando *None* caso o estado gerado seja invalido);
  - **other_side(self)**: método que retorna o valor do indicador ao lado contrário do rio associado ao estado objeto;
  - **is_equal(self,state)**: método responsável por comparar o estado objeto com outro estado argumento. Caso eles sejam iguais, o método retorna *True*. Caso contrário, *False*.

- O pacote *a_start.py* possui a implementação da classe responsável por realizar a busca A*. Sendo muito similar a árvore de busca usada para os algoritmos de busca cega, esta classe possui como diferencial os parâmetros de ações e processos ativos. Ademais e como era de se esperar, a classe implementa a busca A* para o problema modelado.

4 Resultados

Neste capítulo, serão apresentados os resultados colhidos para os algoritmos de Busca em Profundidade (Seção 4.1), de Busca em Largura (Seção 4.2) e de Busca Heurística utilizando o A* (Seção 4.3. Seguindo um padrão de teste, coletou-se resultados de Tempo de Execução, número de Nós Visitados, Passos necessários para se alcançar a Solução e o nível de Solução Encontrada utilizando profundidades variáveis. Todos os experimentos foram repetidos 50 vezes para coleta estatística significante.

4.1 Depth-First Search (DFS)

Para a Busca Cega em Profundidade, foram coletados os seguintes resultados:

Profundidade Máxima	12	13	15	18
Tempo de Execução (s)	0.017 ± 0.003	0.101 ± 0.001	0.007 ± 0.000	0.0116 ± 0.011
Nós Visitados	9607	4554	4554	4554
Passos para Solução	11	13	15	17
Solução Encontrada	Ótima	Não-Ótima	Não-Ótima	Não-Ótima

Tabela 2 – Resultados dos testes para o algoritmo de Busca em Profundidade.

4.2 Breadth-First Search (BFS)

Para a Busca Cega em Largura, foram coletados os seguintes resultados:

Profundidade Máxima	12	15	18
Tempo de Execução (s)	0.097 ± 0.024	0.082 ± 0.008	-
Nós Visitados	10444	10444	-
Passos para Solução	11	11	-
Solução Encontrada	Ótima	Ótima	-

Tabela 3 – Resultados dos testes para o algoritmo de Busca em Largura.

Para profundidade máxima 18, o processamento do computador não foi suficiente para obter resultados.

4.3 Busca Heurística - A*

Para a Busca Heurística A*, foram coletados os seguintes resultados:

Profundidade Máxima	12	15	18
Tempo de Execução (s)	4.09	4.91	4.53
Nós Visitados	740	740	740
Passos para Solução	11	11	11
Solução Encontrada	Ótima	Ótima	Ótima

Tabela 4 – Resultados dos testes para o algoritmo de Busca A*.

5 Conclusão

Frente toda discussão e a apresentação dos resultados neste documento, comprova-se a eficiência esperada para a aplicação dos métodos de Busca em Inteligência Artificial para resoluções de problemas modelados.

No documento, foi estudado um Problema de Travessia conhecido como “os 3 missionários e os 3 canibais”. Implementando o problema utilizando a modelagem proposta, analisou-se a eficiência de *dois algoritmos de busca cega* (Busca em Largura e Busca em Profundidade) e *um algoritmo de busca heurística* (Busca A*).

Pela análise dos resultados, a **busca heurística** é a que apresenta o **melhor desempenho** quanto a número de visitas necessárias para resolução, capacidade de encontrar as respostas ótimas (ou sub-ótimas) e escalabilidade. Devido à capacidade de poda do espaço de busca garantido pela aplicação de uma **heurística admissível**, temos que a escalabilidade do problema se comporta de melhor maneira em comparação a métodos de busca cega, garantindo a visita de um número menor de nós para se encontrar uma resposta ótima ao problema.

Realizando a comparação dos métodos de **busca cega**, temos que o desempenho deles está diretamente relacionado com o **tamanho de espaço de busca gerado**. A *Busca em Profundidade* para um espaço de busca cuja a profundidade máxima é próxima ao número de passos necessários para a solução ótima possui um desempenho superior ao método de *Busca em Largura* para maioria dos casos, uma vez que ele realiza a exploração de um número menor de nós que o método em Largura e encontra uma resposta ótima. Contudo para cenários onde o *espaço de busca é maior*, mesmo a Busca em Profundidade apresentando um melhor resultado quanto ao número necessário de visitas para encontrar uma resposta, esta não é a resposta ótima para o problema - enquanto a Busca em Largura sempre apresentará uma resposta ótima (ou sub-ótima) para o problema. Ambos métodos possuem desempenhos semelhantes quanto a escalabilidade.