Analisando problemas clássicos da IA e suas resoluções utilizando algoritmos de busca.

Lucas Ferreira Mateus
lucas.mateus@utp.com.br

10 de Outubro de 2020

1 Introdução

O artigo deverá analisar a natureza dos problemas:

- Cubo de Rubik;
- Missionários e canibais;
- Problema das n rainhas;
- Sudoku;

Quais algoritmos de busca são mais indicados para cada problema. Além disso, o artigo deverá descrever a modelagem do problema para sua resolução por meio de busca, informando: (1) estado inicial, (2) ações, (3) modelo de transição, (4) custo do caminho, e (5) teste do objetivo.

2 Algoritmos de Busca

2.1 Busca Local

Algoritmo utilizado em problemas de otimização, no qual o caminho e sequências de acoes tomadas pelo agente eh irrelevante para o objetivo. Apenas o estado atual é mantido, assim não gerando a necessidade de armazenamento de uma arvore de busca, o que acaba ajudando a diminuir a quantidade de memoria utilizada.

A Estrategia do algoritmo consiste em manter apenas o estado atual e realizar tentativas me melhorias o movimentando para seus estados vizinhos.

2.1.1 *Hill Climbing* (subida da encosta)

Nesse algoritmo nós iniciamos de um ponto aleatório X e fazemos a sua avaliação, apos isso nós nos movemos do nosso ponto X original para um novo ponto vizinho ao que estamos o X'. Assim são gerados pequenos movimentos que melhoram seu resultado de maneira progressiva. Ponto negativo do *Hill Climbing* é o fato em que algum estado não gerar um vizinho não seja uma melhor solução para o problema a execução é parada e o estado atual é tomado como resultado.

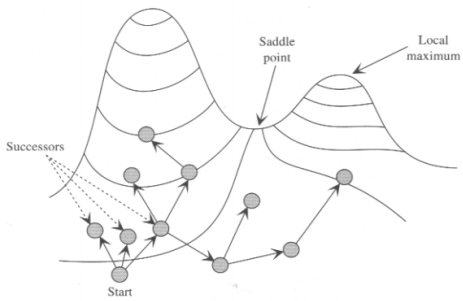


FIGURA 1 Stuart Russel [jun. 2007]

2.1.2 *Simulated Annealing* (recozimento simulado)

De maneira igual ao *Hill Climbing* iniciamos de um ponto aleatório X e realizamos sua avaliação, apos é realizado um movimento até um dos seus vizinhos X' e avalia esse novo ponto. Se os resultados melhoraram no nosso ponto X' então nos movemos até ele e refazemos o processo anterior, todavia se o ponto X' for inferior nós só iremos nos mover para ele caso nossa probabilidade de ir para um ponto negativo seja superior a um número aleatório. Sendo assim tem a vantagem

sobre o *Hill Climbing*, podendo aceitar valores negativos, probabilidade calculada através da formula

$$f(p) = Exp(X' - X/T),$$

X é o ponto que nos encontrávamos anteriormente, X' nosso ponto vizinho e T seria o valor gerado aleatoriamente que diminui seu valor a cada iteração.

2.2 Busca com informação (ou heurística)

Utiliza conhecimento específico e definição do problema para encontrar soluções. Tem como objetivo encontrar a melhor escolha, utilizando uma função de avaliação por nó e expandindo o nó com menor resultado, mas dependendo da função de avaliação a estrategia de busca é alterada.

Na busca pela melhor escolha, usar a função de avaliação para cada nó tem o intuito de fazer uma estimativa de quanto aquele mesmo nó é desejável para a resolução do problema.

2.2.1 Busca Gulosa

Busca gulosa pela melhor escolha expande o nó que parece mais próximo ao objetivo de acordo com a função heurística

$$f(n) = h(n).$$

Busca gulosa não é ótima, pois ela segue apenas o melhor passo considerando apenas o estado atual e pode haver resoluções melhores segundo de nós piores em algum ponto da árvore. O Tempo de execução depende de uma boa função heurística, além de todos os nós serem mantidos na memoria.

2.2.2 Busca A*

Algoritmo ótimo que toma como estrategia evitar a expansão por caminhos caros. Sua função de avaliação é determinada por

$$f(n) = g(n) + h(n)$$

, g(n) sendo o custo do caminho até o momento para alcançar o estado atual, h(n) o custo de caminho estimado para o objetivo e f(n) custo total estimado do caminho através do estado atual até o objetivo.

2.3 Busca sem informação (Cega)

A Busca sem informação ao contrario da busca com informação, para resolução do problema é utilizado apenas as informações disponíveis na definição do problema. As estrategias de buscar sem informação se distinguem pela ordem que os nos são expandidos.

2.3.1 Busca em largura

Uma lista de vértice implementada com o a politica FIFO (*first in first out*), fazendo com que a lista se comporte como uma pilha, onde o vértice que se encontra mais tempo na lista é aquele que irá ser retirado. Feofiloff [22 ago. 2019]

2.3.2 Busca em profundidade

O Algoritmo visita todos os vértices da arvore em uma determinada sequencia, a cada vértice visitado é marcado como já percorrido e é passado para o próximo ate todos os vizinhos do vértice sejam visitados, assim vértices que não estão com demarcados como percorridos são vértices validos. Assim implementado uma lista com politica LIFO (*last in first out*).

2.3.3 Busca em profundidade limitada

Realiza a busca em profundidade até determinada altura do grafo.

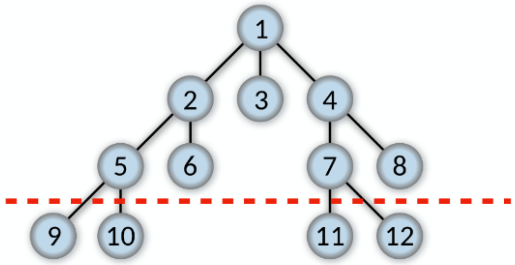


FIGURA 2 Queirolo [11 sep. 2007]

2.3.4 Busca em aprofundamento iterativo

Realiza a busca em profundidade em determinadas altura pré-definidas do grafo.

2.3.5 Busca de custo uniforme

O algoritmo de busca uniforme funciona basicamente da mesma maneira que a busca em profundidade só que ao invés de pegar o primeiro nó expandido que está na lista aguardando processamento, o nó que sera escolhido é o que possui menor custo. Isso garante que a primeira solução encontrada sera a com menor custo.

3 Cubo de Rubik

Junto com seu surgimento o cubo de Rubik, trouxe uma duvida que afligiu cientistas e cubistas. Qual seria a resolução que precisaria de menos movimentos realizados para alcançar o estado solução?

Uma solução possível seria a utilização de um algoritmo de busca em largura, mas devido aos números de estados gerados, acaba impossibilitando essa execução devido aos poderes de processamento de um computador atual. Usando uma árvore em que cada cubo ocupe apenas um bit, uma busca em largura com vinte e quatro níveis de profundidade seriam obtidos 161245862142086000000000000 bits. Esteves [2015]

Sendo assim o Cubo de Rubik é utilizado na IA para teste de eficiência de novas técnicas. Acredita-se que 20 movimentos para chegar no seu estado final seja o máximo para que a solução utilizada seja viável, sendo a media é de 18 movimentos. Algoritmos aplicando a busca heurística são aplicados na resolução do problema.

O Estado inicial do cubo de Rubik, seria um cubo embaralhado, possuindo 12 possibilidade de ações diferentes (girar faces), e seu teste objetivo sendo todos os quadrados de uma mesma face da mesma cor.

3.1 Algoritmo de quatro fases

O método de busca empregado funciona com base na divisão da solução em quatro etapas em que cada uma possuía cada vez menos cubos e movimentos possíveis, o que gerava buscas mais ágeis e em conjuntos restritos de Cubos. A sequencia de buscas era dividida nos seguintes grupos:

- Grupo 0: É o grupo inicial da busca, o que contém todos os Cubos possíveis. É realizadas rotações de noventa graus, buscando estados que pertençam ao Grupo 1.
- Grupo 1: O grupo um é constituído por cubos que podem ser resolvidos somente com os movimentos de noventa graus das faces *Near, Far, Left e Right*, e o giro de cento e oitenta graus das faces *Up e Down*. Utilizando estes movimentos é procurado algum estado que se enquadre no Grupo 2.
- Grupo 2:O grupo um é constituído por cubos que podem ser resolvidos somente com os movimentos de noventa graus das faces *Near, Far, Left e Right*, e o giro de cento e oitenta graus das faces *Up e Down*. Utilizando estes movimentos é procurado algum estado que se enquadre no Grupo 3.
- Grupo 3: Só são possíveis movimentos de cento e oitenta graus em qualquer face. Utilizando estes movimentos é procurado algum estado que se enquadre no Grupo 4.
- Grupo 4: Grupo que possui apendas o estado final do problema.

Esteves [2015]

TABELA 1: Relação entre o grupo e a quantidade de movimentos do mesmo

Grupo de Estados	**Quantidade de Movimentos**	**Possíveis Movimentos**
Grupo 0	12	U, U2, U, D, D2, D, L, L2, L, R, R2, R, F, F2, F, B, B2, B
Grupo 1	10	U2, D2, L, L2, L, R, R2, R, F, F2, F, B, B2, B
Grupo 2	8	U2, D2, L, L2, L, R, R2, R, F2, B2
Grupo 3	6	U2, D2, L2, R2, F2, B2
Grupo 4	0	Nenhum

O processo de busca, dividido nessas quatro etapas, é mais rápido e barato que uma busca geral pelo cubo.

3.2 Algoritmo de duas fases

Baseado no algoritmo de quatro fases o algoritmo de duas fazes tem como objetivo na sua primeira fase eram gerados milhares de macro-operadores que levariam o estado aleatório ate um grupo específicos de cubos em que somente oito movimentos eram permitidos *Up, Down* girados em 90 graus, e demais faces em 180 graus. No próximo grupo é executada apena operações permitidas, que acabem reduzindo drasticamente o total de nós gerados. Com esse algoritmo é possível solucionar qualquer cubo com 29 movimentos.

3.3 Próximo do Algoritmo de Deus

Através de um banco de dados que armazena todos os estados-alvos. Korf(1997) através de uma busca em profundidade (A*), consegui chegar ao estado final em no máximo de 18 movimentos. O uso de um algoritmo bidirecional diminui drasticamente o número de estados gerados em uma busca em largura (de 10^{27} para 10^{13} estados). O algoritmo gerado por Korf (1997) é até hoje aceito como a melhor solução para qualquer estado do cubo, o seu único fator limitativo é o tempo. Diante dessa impossibilidade, novos algoritmos vêm sendo lançados, aproximando-se cada vez mais do número de Deus.

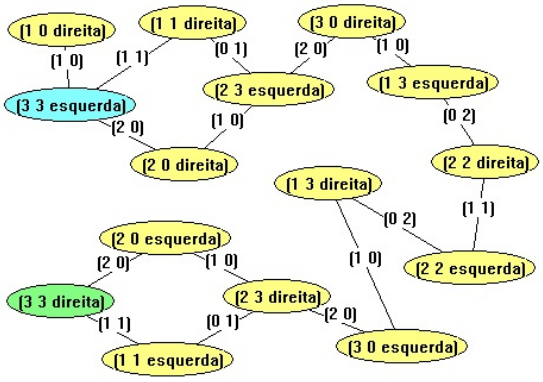
4 Problema dos canibais e dos missionários

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. Há uma outra dificuldade: em nenhum momento o número de canibais pode ser superior ao número de missionários pois desta forma os missionários estariam em grande perigo de vida

Sendo que o barco comporta 2 pessoas por vez no máximo, há 5 possíveis combinações de tribulantes. (1) 1 canibal somente; (2) 2 canibais; (3) 1 missionário; (4) 2 missionários; (5) 1 missionário e 1 canibal;

Problema é facilmente resolvido utilizando uma busca em profundidade como podemos ver na imagem GRAFO 1, busca local neste problema não pode ser utilizada devido ao motivo que o caminho importa devido as regras pré-definidas do problema em que um missionário e um canibal não podem ficar juntos sozinhos.

Através da GRAFO 1 podemos ver que são necessárias um mínimo de 11 travessias para que canibais e missionários cheguem em segurança ao outro lado do rio . de Santa Catarina [04 out. 2020]



GRAFO 1

Para a implementação do problema precisamos estar ciente das regras do problema.

1. Para que t canibais, 1 t 2, atravessem sozinhos o rio, é preciso que haja t canibais na margem de origem

   $$(cx \geq t).$$

   Também é preciso que ao chegarem estes t canibais na margem de destino, o número total de canibais não exceda o número de missionários. Isto é obtido em duas situações: se não houver nenhum missionário na margem oposta antes da travessia do barco, ou seja,

   $$(3 - mx) = 0;$$

   haja pelo menos t missionários a mais do que canibais na margem oposta antes da travessia do barco, ou seja,

   $$(3 - mx) - (3 - cx) \geq t.$$

2. Para que t missionários, 1 t 2, atravessem sozinhos o rio, é preciso que haja t missionários na margem de origem (mx t).

Também é preciso que não restem mais canibais do que missionários na margem de origem, ou seja, que (mx - t) cx, ou que o número de missionários restantes seja zero (mx - t = 0). Quanto à margem de destino, ou ela não tem nenhum canibal antes do início da travessia

$$(3 - cx) = 0$$

, ou o número de canibais não exceda o número de missionários em mais do que t unidades

$$((3 - cx) - (3 - mx) \leq t).$$

3. Para que 1 canibal e 1 missionário atravessem o rio, é preciso que haja pelo menos 1 canibal

$$(cx \geq 1)$$

e 1 missionário

$$(mx \geq 1)$$

na margem de origem. Também é preciso garantir que o número de canibais não exceda o número de missionários na margem de destino, ou seja, que não haja nenhum canibal antes da travessia

$$((3 - cx) = 0)$$

, ou que o número de canibais não seja superior ao número de missionários

$$((3 - cx) \leq (3 - mx)).$$

de Santa Catarina [04 out. 2020]

5 Problema das N-Rainhas

O problema das n rainhas tem objetivo a alocação de N rainhas em um tabuleiro NxN, de maneira que nenhuma se ataque, com se atacar esta subentendido que as mesma não podem compartilhar a mesma linha, coluna ou diagonal.

Sabendo que N não pode assumir os valores 2 e 3, pois seria impossível fazer com que as rainhas não compartilhassem a mesma diagonal, para exemplificação e elaboração do problema iremos tomar N com um valor igual a 8, pois no caso obteremos um tabuleiro padrão de xadrez.

De modo que o objetivo do problema não leva em consideração ordem que essas rainhas são posicionadas no tabuleiro, podemos utilizar o algoritmo de busca local.

Estado inicial do algoritmo se torna um pouco diferente, sendo o tabuleiro já preenchido com as N rainhas de forma aleatória, cada ação de movimentar uma peça pelo tabuleiro tomada pelo agente tem como objetivo otimizar a solução através da função de avaliação do problema

$$f(x) = h(x).$$

O custo da função se torna a quantidade de rainhas se atacando quanto menor o valor, mais ótima a solução se torna, até que o teste abjetivo de nenhuma rainha estar se atacando seja contemplado.

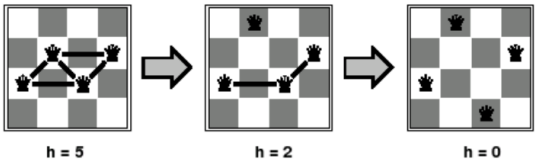


FIGURA 2 Stuart Russel [jun. 2007]

6 Problema do Sudoku

Sudoku e um jogo baseando em uma desatribuição logica de números em um *grid* 9x9 o mesmo contendo *subgrids* 3x3. O quebra cabeça vem com n células preenchidas de maneira a dar pistas da resolução do problema. O objetivo é preencher os restantes das células de maneira que respeitem as seguintes regras:: cada casa deve conter um único numero, e cada linha, coluna e bloco do tabuleiro deve conter, sem repetição, todos os inteiros de 1 a 9 .Wikipédia [04 out. 2020]

6.1 Modelando o problema de resolução com grafos

Podemos realizar o Sudoku através da implementação de um algoritmo de de busca local, devido ao fato do problema ter como interesse apenas o estado final e já possuir conhecimento do mesmo. O estado inicial do problema seria um *grid* 9x9 já começa totalmente preenchido. Cada ação se resume em trocar a posição de dois valores, a função de avaliação consiste na quantidade de valores repetidos nos *subgris*, e linhas,

causo esse valor seja melhor que o valor passado ele é tomando como estado atual do problema.

O algoritmo *Hill Climbing* pode ser aplicado também, mas com outra tratativa e outro estado inicial. O algoritmo ira selecionar a vizinhança ¸a de melhor custo e ira retorna-la. O custo de uma solução e calculado pela soma da quantidade de dıgitos que faltam ser inseridos em todo o tabuleiro. E assim a meta-heurıstica prossegue, guardando sempre a melhor solução encontrada, ate que alguma condição de parada seja verdadeira. A condição de parada pode ser definida como um total de iterações ou quando se encontra a solução é ótima (custo zero). Quanto maior o coeficiente *alpha*, mais aleatório, quanto menor mais guloso. Kevin Takano [2008]

7 Conclusão

Com esse artigo conseguimos ver que o grande limitador para a resolução dos problemas de maneira tradicional é o poder computacional que possuímos atualmente, e por isso necessitamos da implementação de diferentes algoritmos de IA para uma resolução dos problemas de maneiras não custosas. Vemos que umas das mais utilizadas para algoritmos de busca é a busca em profundidade que pode ser representada através de grafos, o objetivo da IA é percorrer o grafo ate chegar no estado final, de maneira que elimine os vértices menos atrativos a partir da implementação de uma determinada logica. Algoritmos de busca tem como objetivo chegar no estado final, mesmo alguns podendo alcançar uma solução ótima o que dependendo da situação pode não ser indicado, ou como no caso dos missionários em que o caminho percorrido tem relevância no resultado final do problema, ou no do cubo de Rubik que precisamos o ótimo é a solução com menor sequencia de movimentos.

Referências

U. F. de Santa Catarina. *Problema dos canibais e dos missionários*, 04 out. 2020. URL `http://www.inf.ufsc.br/grafos/temas/travessia/canibais.htm`.

B. A. Esteves. SoluÇÕes inteligentes para o cubo de rubik. *Universidade Federal de Juiz de Fora*, 1 (1), 2015.

P. Feofiloff. *Busca em largura*, 22 ago. 2019. URL `https://www.ime.usp.br/ pf/algoritmos`$_p ara_g rafos/aulas/bfs.html$

V. G. P. d. S. Kevin Takano, Rosiane de Freitas. O jogo de lógica sudoku: modelagem teórica, np-completude, algoritmos e heurísticas. *Instituto de Matematica – Universidade Federal do Rio de Janeiro*, 1(1), 2008.

C. Queirolo. *Inteligencia artificial - Busca sem informação*, 11 sep. 2007. URL `https://github.com/chaua/inteligencia-artificial/blob/master/Aulas/Aula07 - Busca sem informação.pdf`.

P. N. Stuart Russel. *Busca Local*, jun. 2007. URL `http://www.din.uem.br/ jmpinhei/SI/07BuscaLocal.pdf`.

Wikipédia. *Sudoku*, 04 out. 2020. URL `https://pt.wikipedia.org/wiki/Sudoku`.