Atividade Avaliativa I - Uma análise comparativa entre métodos de busca

Giovanni F. S. Rebouças[1], Felipe A. da S. Biazatti[2], Pedro A. B. Bispo[3]
[1,2,3]Institute of Technological Sciences, Universidade Federal de Itajubá, Itabira, Minas Gerais, Brazil
RA: 34270[1], 34452[2], 2016002449[3]
giovannireboucas@hotmail.com, felipeamorimsbiazatti@gmail.com, pedrobispo@unifei.edu.br

***Resumo*—Esse trabalho visa fazer uma análise comparativa entre métodos de busca. O objetivo é, através da análise de parâmetros dos algoritmos como completude, otimalidade e complexidade, possamos traçar e fazer uma análise de desempenho em cima de um espaço de busca em comum, o jogo da velha. Além das comparações por parâmetros, este trabalho irá fazer uma análise discursiva sobre uma outra implementação de busca heurística, a Programação Dinâmica e seus exemplos de aplicação.**

***Palavras-chave*—Métodos de Busca, Busca em Profundidade, Minmax, Espaço de Busca, Jogo da Velha.**

I. Introdução

Análise das possibilidades por exaustão, quando confrontado com uma decisão, ou problema é elemento natural, intrínseco e primordial do pensamento humano. Essa capacidade de análise por exaustão possui um limiar superior prático definido pela capacidade cerebral/cognitiva, tempo, complexidade ou tamanho, isto é, existem problemas que não são passíveis de serem analisados por exaustão da maneira clássica por um ser humano. Nesses casos, dizemos que o **espaço de busca** é demasiadamente extenso.

Com o constante crescimento dos espaços de busca à medida que a complexidade dos problemas cresce, foi necessário, a estipulação de métodos formais de busca baseado ou não em heurísticas para que os resultados desejados pudessem ser alcançados de maneira mais eficazes. Esses métodos tem como objetivo a melhoria de algum parâmetro relacionado à busca de um resultado em um espaço de busca, como por exemplo, reduzir o tempo para encontrar a primeira solução de um problema.

Esses algoritmos são frequentemente desenvolvidos com objetivos e casos de uso muito claros e bem estabelecidos, isto é, nem todo método de busca serve para todo espaço de busca, devido à natureza das heurísticas, condições de contorno, objetivo de otimização e parâmetros utilizados para propor o algoritmo. Por sua vez, estão intrinsecamente relacionados com os problemas que motivaram as suas respectivas concepções. Desta maneira, uma análise aprofundada destes algoritmos é etapa sumariamente importante durante o processo de seleção para a solução de um problema.

II. Revisão de Literatura

Como citado anteriormente, existe uma extensa lista de algoritmos de busca e estes são passíveis de serem analisados com base em parâmetros que podem ser extraídos de todos, para que haja comunalidade e um forte relacionamento em relação às análises. Os parâmetros que iremos usar e suas definições são:

- **Completude** - Sempre encontra uma solução (se existir)?
- **Otimalidade** - Sempre encontra a solução de custo mais baixo?
- **Complexidade (Tempo)** - Números de nós (estados) explorados.
- **Complexidade (Espaço)** - Número de nós (estados) armazenados.

Os parâmetros de Complexidade serão medidos com base na notação *Big O*, inventada por Paul Bachmann e Edmund Landau, [1] enquanto completude e otimalidade serão respondidos de forma discursiva, de acordo com os aspectos de cada algoritmo.

A. Algoritmo de Busca em Profundidade

Dada uma estrutura de dados, a busca em profundidade *(Depth-first Search)* ou *busca DFS* é um algoritmo generalizado que consiste em buscar um nó alvo que pode estar presente ou não em um conjunto de nós conectados por arestas seja em um grafo ou outra de estrutura de dados que seja possível aplicar a busca.

A ideia é que na busca pelo nó alvo, a partir de um nó inicial S, seja priorizada a busca pela aresta mais recente descoberta de S, ainda que existam arestas não exploradas também saindo dele, e assim sucessivamente para os próximos nós filhos e os descendentes de S. Caso não seja encontrado o nó alvo em determinada parte do caminho percorrido pela busca e caso o nó não tenha mais filhos, é feito o retorno ao último explorado o que chamamos de *backtracking*, até que se encontre ou não o nó alvo diante de todo o espaço de busca.

A implementação em *Python* referente ao funcionamento do algoritmo de busca em profundidade feita pelos autores pode ser encontrada no arquivo em anexo `tictactoe_dfs.py`.

B. Algoritmo de Min-max

Em teoria dos jogos, algoritmo de Min-max (ou Minimax) é um método para minimizar a possível perda máxima, dada uma decisão. Esse algoritmo utiliza uma técnica de força bruta em conjunto com uma função de avaliação. O algoritmo avalia

todas as possibilidades, a partir de cada estado, em busca da melhor jogada possível. A função de avaliação determina um bom movimento válido para um jogador pela avaliação dos nós, assim, ela gera um número que é associado a um nó e não considera situações prévias nem possibilidades futuras, avalia apenas com base cenário atual para ambos jogadores, para ao final de sua avaliação, decidir qual é a melhor jogada.

É importante que ao ser construída, a função de avaliação seja simples, pois ela será calculada diversas vezes. A implementação em *Python* referente ao funcionamento do algoritmo de min-max feita pelos autores pode ser encontrada no arquivo em anexo `tictactoe_minmax.py`.

C. Programação Dinâmica

Muitos algoritmos são baseados no método de programação dinâmica. Esse método, ou estratégia de projeto de algoritmos, funciona como tradução iterativa de uma recursão e pode ser definido como recursão com o apoio de uma tabela.

A característica distintiva da programação dinâmica é o armazenamento das soluções dos vários sub problemas. Por conta disso, é necessário que o problema tenha natureza recursiva, visto que, a programação dinâmica se baseia em eliminar os recálculos inerentes de algoritmos recursivos, ao invés, optando por armazenar os resultados ou soluções dos sub problemas que compõe o problema maior como um todo.

Programação dinâmica é uma técnica que só pode ser usada quando o problema possuí natureza recursiva (como exposto anteriormente) e atenda 2 características primordiais:

- Possuir sub-estrutura ótima (*optimal substructure*) - Característica de quando a solução ótima de um problema contém nela própria soluções ótimas para sub-problemas do mesmo tipo; e
- Possuir sub-problemas coincidentes - Característica de quando um espaço de sub-problemas é pequeno, isto é, não existem muitos sub-problemas a resolver pois muitos deles são repetidos.

E mesmo quando um problema atende todos os requisitos citados, existe a possibilidade de que ele não seja resolvível através de Programação Dinâmica.

A metodologia da Programação Dinâmica pode ser resumida nos seguintes passos e subpassos:

- Caracterizar a solução ótima do problema;
  - Verificar se o problema obedece as 2 características de otimalidade e subproblemas coincidentes; e
  - Dividir o problema em sub-problemas do mesmo tipo.
- Definir de maneira recursiva a solução ótima, através das soluções ótimas dos subproblemas;
- Calcular as soluções dos subproblemas *bottom-up* ou *top-down* com algum artifício de memorização;
- Reconstruir a solução ótima, baseada nas soluções armazenadas dos subproblemas.

Programação dinâmica possui uma aplicabilidade imensa, apesar dos inúmeros requisitos que deve atender para ser útil. Dentre as possíveis aplicações, podemos citar problemas matemáticos inerentemente recursivos como cálculo do n-ésimo elemento da sequência de Fibonacci e multiplicação encadeada de matrizes. Ambos problemas que atendem todos os requisitos pois são problemas inerentemente recursivos, dependentes dos resultados provenientes de seus sub-problemas, cujos sub-problemas são do mesmo tipo que o problema principal (ou seja, contem sub-estrutura ótima) e possuem um espaço de sub-problemas pequeno.

D. Algoritmo de Busca em Largura

A busca em largura *(breadth-first search)* ou *busca BFS* é um método exaustivo que examina todos os nós presentes em um grafo direcionado ou não-direcionado. Sendo assim, o algoritmo deve garantir que nenhum nó ou aresta está sendo visitado mais de uma vez. [2]

A *busca BFS* começa por um nó, digamos *S*, especificado pelo usuário, assim como ocorre em *busca DFS*. O algoritmo visita S, depois os vizinhos de *S*, depois todos os vizinhos dos vizinhos, e assim por diante. Quando não houver mais vizinhos a serem explorados em um determinado nível é que o algoritmo se moverá para o próximo mais profundo repetindo o processo.

E. Algoritmo de Djikstra

O algoritmo de Djikstra visa resolver o problema que dado um grafo G com arestas de valores positivos, encontrar o menor caminho entre o vértice S e o objetivo.

O algoritmo que Edsger Djikstra descobriu demonstra que fazendo crescer uma subárvore T em G, a partir de S, até que englobe todos os vértices que estão ao alcance de S nos permite encontrar o menor caminho percorrendo a árvore gerada.

A lógica consiste em cada iteração de T, com raiz em S e um vetor distância L[]. Na primeira iteração, S é o único vértice de T portanto L[S] vale zero. Nas seguintes iterações podemos abstrair o seguinte processo:

- Escolha a aresta com ponto inicial em T e final fora de T. que minimize L[x] + x-y.
- Acrescente a aresta x-y e o vértice y a T.
- Faça L[y] = L[X] + x-y.

Isso garante que iremos encontrar o caminho mínimo entre 2 vértices desconhecidos interligados.

F. Algoritmo de Poda $\alpha - \beta$

É uma variação do algoritmo de min-max que busca otimizar o consumo de memória comparado a este, ao armazenar menos estados possíveis através da simplificação de estados. Ao encontrar uma maneira comprovada de que um movimento cujo valor associado seja pior que um movimento previamente analisado.

O método consiste em percorrer a árvore em profundidade, a cada nó que não seja uma folha, é armazenado um valor em uma variável sendo a variável Alfa para os níveis MAX, que é o máximo valor encontrado até então nos descendentes dos nós MAX, e é armazenado um valor na variável Beta para os níveis MIN que é o mínimo valor encontrado até então nos descendentes dos nós MIN. Então é feita a análise na direção das folhas para a raiz, nos nós MAX e nos nós MIN, em que

TABLE I
ALGORITMOS DE BUSCA DE ACORDO COM ALGUNS PARÂMETROS COMPARATIVOS

Algoritmo	Completude	Otimalidade	Complexidade de Tempo	Complexidade de Espaço
Busca em Profundidade	Sim	Não	O(M+N)	O(M+N)
Min-max	Sim	Sim	O(N)	O(M+N)
Programação Dinâmica	Sim	Sim	O(M+N/2)	O(M)
Busca em Largura	Sim	Não	O(M+N)	O(M)
Djikstra	Sim	Sim	O(M + NlogN)	O(M+N)
Poda $\alpha - \beta$	Não	Sim	O($b^{\frac{d}{2}}$)	O($b^{\frac{d}{2}}$)

para os nós MAX se o valor de alfa do nó atual é maior ou igual o valor do beta "ancestral" então o nó que seria buscado abaixo é cortado e para a análise de um nó MIN, se o valor de beta do nó atual é menor ou igual ao valor do alfa ancestral então o nó que seria buscado abaixo é cortado.

III. MATERIAIS E MÉTODOS

Esta seção descreve os algoritmos escolhidos, a linguagem usada e o espaço de busca escolhido para os testes.

A. Espaço de Busca

Para as implementações que serão feitas e para os algoritmos que serão sujeitos à nossas análises, o espaço de busca que foi escolhido é o **Jogo da Velha** padrão de 9 espaços, 2 jogadores ("X" e "O"), condições de vitória padrão (combinação sequencial de 3 símbolos iguais orientado horizontalmente, verticalmente ou diagonalmente) e condição inicial de campo vazio (9 espaços vazios).

B. Linguagem de Programação

A linguagem de programação escolhida para o desenvolvimento dos algoritmos que desejamos testar de maneira prática foi *Python*. Especificamente, estamos usando CPython, que é a implementação **oficial** que pode ser baixada da *Python Foundation*. Ambos os algoritmos foram desenvolvidos na versão 3.7.3 do *Python*. Nenhuma biblioteca ou pacote externo foram utilizados, apenas bibliotecas padrão da linguagem.

C. Algoritmos Escolhidos

Os algoritmos escolhidos para serem implementados foram: **Busca em profundidade** e **Min-max**. Enquanto os algoritmos que não foram implementados, mas que também foram objeto de pesquisa deste trabalho são:

- **Programação Dinâmica**
- **Busca em Largura**
- **Djikstra**
- **Poda** $\alpha - \beta$

Uma descrição de cada um dos algoritmos podem ser encontrados na seção de Revisão de Literatura.

IV. RESULTADOS

A Tabela de comparação e os resultados dos algoritmos escolhidos de acordo com os parâmetros definidos pode ser encontrado na tabela I.

Os parâmetros da notação Big-O são:

- **Busca em Profundidade** - **M** se refere ao número de estados (ou nós ou vértices) e **N** se refere ao número de movimentos (ou bordas);
- **Min-max** - **N** é o número de movimento necessário para chegar à uma vitória e **M** é o máximo número de estados a frente que o algoritmo olha para chegar a uma decisão;
- **Programação Dinâmica** - Mesma definição do algoritmo de Busca em Profundidade;
- **Busca em Largura** - Mesma definição do algoritmo de Busca em Profundidade;
- **Djikstra** - Mesma definição do algoritmo de Busca em Profundidade;
- **Poda** $\alpha - \beta$ - **b** é o número de filhos que cada nó possui e **d** é a profundidade de busca.

V. CONCLUSÃO

Com o inevitável crescimento dos espaços de busca presentes nas diferentes problemáticas de interesse, as buscas por força bruta e por exaustão mostram-se cada vez mais ultrapassadas no que diz respeito à eficiência, complexidade de espaço, complexidade de tempo e otimalidade. Essas ineficácias são o racional pelo desenvolvimento e aplicação de técnicas de buscas cada vez mais especializadas e otimizadas para o problema em questão.

Esses métodos de busca visam resolver os problemas que advém do crescimento exacerbado ou inerente de certas problemáticas, como aqueles vistos nas áreas de *Big Data*, *Data Mining*, *Machine Learning* e outros. Existe um conjunto cada vez maior de técnicas de busca à medida que a necessidade de maior especificidade, contextualização e otimização das complexidades ficam cada vez mais aparentes e, é dever do encarregado de pensar no âmbito da solução, de fazer as devidas análises para que a escolha de algoritmo atinja o ótimo local para a problemática em questão.

REFERÊNCIAS

[1] Paul Bachmann and Edmund Landau. *Die analytische Zahlentheorie*. Leipzig B.G. Teubner, 1894.

[2] Paulo Feofiloff. Busca em largura (bfs). In *Algoritmos para Grafos via Sedgewick*. 2020.

[3] George F. Luger. *Inteligência Artificial*. Pearson, 2013.

[4] Anderson Ferreira. Programação dinâmica.

[5] Mariana Kleina. Programação dinâmica.