UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

TRABALHO PRÁTICO 1: INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

Aluno: Bruno Marcelino Borges dos Santos

Matrícula: 2019013155

Data: 14/05/2023

Profº: Luiz Chaimowicz

Belo Horizonte/MG

2022

UFMG

SUMÁRIO

I. ESTRUTURAS UTILIZADAS

Todo o trabalho foi desenvolvido utilizando a linguagem Python, onde as estruturas de dados mais utilizadas já estão prontas. Dessa forma, aproveitando as funções que vêm incluídas no pacote inicial da linguagem, foram utilizadas apenas operações com listas de números inteiros para a construção dos algoritmos.

Os componentes da busca foram modelados como:

- Estado : objeto do tipo “lista” que representa a situação, em determinado momento, do vetor a ser ordenado. Nele é aplicada uma ação que retorna outro estado, até que se chegue em uma solução ótima que representa o estado desejado do vetor, que é a ordenação completa.

- Nó : objeto do tipo “lista de listas” contendo os elementos que caracterizam um estado na árvore de busca. São eles: [estado atual, estado-pai, ação realizada, custo, profundidade]. Conforme o algoritmo utilizado, fez-se necessária a adição de mais atributos ao nó, como o valor da função heurística e a função sucessora (casos de busca com informação).

- Ação: também conhecida como “função sucessora”, é um método que recebe um nó como valor de entrada e aplica todas as ações possíveis a ele, para que sejam adicionadas à fronteira. Em resumo, gera todos os seus filhos que são nós a serem explorados.

- Fronteira: objeto do tipo “lista de listas” que contém o conjunto de nós que podem ser acessados mas ainda não foram expandidos pelo algoritmo, e são frutos da aplicação de ações nos estados anteriores ao vetor atual. Caso a fronteira esteja vazia, o algoritmo não é completo, pois não conseguiu encontrar nenhuma solução para o problema a partir dos dados fornecidos.

- Conjunto de nós expandidos: objeto do tipo “lista de listas” que armazena os nós que já foram expandidos pelo algoritmo. Ao expandir um nó, o programa verifica se ele já foi expandido previamente, a fim de evitar loops infinitos.

II. DIFERENÇAS ENTRE OS ALGORITMOS

Algoritmos de busca são agentes que realizam uma busca no ambiente e retornam com uma ação ou sequência de ações que o levem a um estado desejado previamente conhecido. Para a formulação de um problema de busca, deve-se considerar um ambiente estático, observável, discreto e determinístico (só depende do agente e suas ações).

Pode-se dividir os algoritmos de busca em dois grandes grupos: busca sem informação ou com informação. A busca sem informação ocorre quando não há conhecimento de outras informações sobre os estados que não foram definidas no início do problema, cabendo ao algoritmo explorar o ambiente sem saber se está próximo da solução. Dentre os algoritmos de busca sem informação, foram explorados:

- Busca em Largura: algoritmo que busca expandir o nó mais superficial que ainda não foi expandido, até encontrar a solução. Avalia a **menor profundidade** como ordem de prioridade para a expansão dos nós.

- Busca em Profundidade com Aprofundamento Iterativo: a Busca em Profundidade expande o mesmo "galho" da árvore até encontrar seu nó mais profundo, que pode ser definido pelo usuário. Isso significa avaliar a **maior profundidade** como ordem de prioridade para a expansão dos nós. A busca com Aprofundamento Iterativo realiza uma sequência de buscas em profundidade de ordem 1:n, com o objetivo de encontrar a solução ótima na menor profundidade possível e evitar o aprofundamento desnecessário em torno de um galho da árvore que não contém a solução.

- Busca com Custo Uniforme (Dijkstra): expande o nó que apresenta o **menor custo** total do caminho até chegar até ele, sem informação sobre os custos até a solução.

Já os algoritmos de busca com informação (também chamada de busca com função heurística) são orientados por uma função heurística (h(n)) que fornece uma estimativa do melhor caminho para a solução, a fim de orientar o algoritmo de busca. Dentre os algoritmos de busca com informação, foram explorados:

- Busca Gulosa: expande o nó que apresenta o **menor valor para a função heurística** (custo estimado da solução), sem levar em consideração o custo total do caminho até ela.
- Algoritmo A*: expande o nó que apresenta o menor valor para uma função de avaliação, que é calculada a partir da soma entre o valor da função heurística (custo estimado até a solução) e o custo total do caminho. É a estimativa mais precisa que pode ser obtida para o custo total da solução, a depender da escolha de uma boa função heurística.

III. HEURÍSTICA UTILIZADA

Como descrito anteriormente, a heurística é uma função que representa uma estimativa do custo a ser incorrido para chegar na solução do algoritmo de busca. A escolha de uma boa heurística é fundamental, e por isso precisamos garantir que ela seja admissível. Isso significa que todos os seus valores são menores ou iguais ao custo real da solução.

No caso do algoritmo de ordenação de vetores, temos as seguintes restrições:

A) os vetores realizam uma troca entre dois elementos, por vez;
B) as trocas só são realizadas se o elemento da esquerda for maior que o da direita;
C) trocas entre vizinhos têm custo 2, trocas entre não-vizinhos têm custo 4.

Uma maneira de encontrar heurísticas é encontrar o custo real de uma solução mais "relaxada" do problema, ou seja, ignorando determinadas restrições, pois ele tende a ser sempre menor ou igual ao custo real do problema e garante que a heurística é admissível. A heurística escolhida para ser utilizada neste trabalho foi **a quantidade de números na posição errada**, que ignora as restrições B e C.

Também é possível provar que a heurística escolhida é admissível pois, no caso onde há o menor custo real até a solução ([1,2,4,3] - custo 2), o valor de h(n) é igual a 2 pois **pelo menos dois** elementos devem estar fora de ordem. A partir de expansões deste evento para o caso onde duas trocas devem ser realizadas entre elementos distantes ([4,2,3,1) - custo 4 - h(n) 2) e três elementos estão fora de ordem ([1,3,4,2] - custo 4 - h(n) 3) o custo real sempre é maior do que o valor da heurística, e assim por diante.

IV. EXEMPLOS E ANÁLISE QUANTITATIVA DAS SOLUÇÕES ENCONTRADAS

Foram incluídos casos hipotéticos para vetores de tamanho n = 4 até n = 7

Figura 1 - Estado Inicial: [2,3,1,4]

	Custo Total	Número de Expansões	Tempo de Execução
BFS	6	8	< 0.01 s
IDS	6	2	< 0.01 s
UCS	4	7	< 0.01 s
A*	4	2	< 0.01 s
GS	6	2	< 0.01 s

Figura 2 - Estado Inicial: [3,1,4,5,2]

	Custo Total	Número de Expansões	Tempo de Execução
BFS	12	108	0.01 s
IDS	12	18	0.03 s
UCS	8	68	0.05 s
A*	8	9	< 0.01 s
GS	12	4	< 0.01 s

Figura 3 - Estado Inicial: [5,1,3,4,6,2]

	Custo Total	Número de Expansões	Tempo de Execução
BFS	8	131	0.01 s
IDS	8	6	0.01 s
UCS	8	117	0.09 s
A*	8	14	< 0.01 s
GS	8	3	< 0.01 s

Figura 4 - Estado Inicial: [1,2,5,4,7,3,6]

	Custo Total	Número de Expansões	Tempo de Execução
BFS	8	711	0.10 s
IDS	8	101	0.11 s
UCS	8	236	0.39 s
A*	8	10	< 0.01 s
GS	8	3	< 0.01 s

V. DISCUSSÃO DOS RESULTADOS

Pode-se notar que os algoritmos de Busca sem informação tendem a apresentar um custo computacional extremamente alto no que diz respeito ao tempo de execução do código. Isso se deve ao fato de que, na presença de uma heurística admissível, os algoritmos de busca com informação se permitem ser muito mais eficientes na escolha do nó a ser explorado. Também pode-se notar este fenômeno ao avaliar a quantidade de expansões realizadas por ambos os códigos. Em todos os casos avaliados, a busca gulosa permite encontrar a solução com uma quantidade de 2 a 4 nós expandidos, ao contrário da busca em largura que expandiu +700 nós para encontrar a solução quando o vetor possui 7 elementos.

Dentre os algoritmos de busca sem informação, a busca com aprofundamento iterativo se demonstrou mais eficiente do que os demais, no que tange ao número de expansões realizadas. Quando custo = 1, o aprofundamento iterativo permite encontrar uma solução ótima. Dado que no caso o custo variava conforme o nó explorado, nem sempre esse algoritmo foi ótimo. Porém, expandiu bem menos nós.

A busca em largura é um algoritmo completo por essência, achando a solução caso exista. Entretanto, não leva em consideração os custos dos caminhos e por isso é o algoritmo mais caro de todos, testando todos os nós que estão presentes a uma profundidade d-1 (sendo d a profundidade da solução ótima). A busca por custo uniforme, diferente do que era esperado, não trouxe grandes resultados e se apresentou bastante cara computacionalmente também, pois por buscar sempre os nós de menor custo, demora a achar a solução em cenários onde os números estão bastante distantes de sua posição original (sendo necessária a realização de trocas de maior custo). Ainda assim, em cenários de trocas curtas (custo 2), ele tende a ser bem eficiente.

Dentre os algoritmos de busca com informação, a busca gulosa se provou um algoritmo mais rápido no encontro de uma solução, não necessariamente ótima. Em contrapartida, o algoritmo A* se mostrou eficiente no encontro da solução de menor custo total em todos os casos analisados, por levar em consideração o custo total do caminho até o nó que seria explorado. Por conta disso, A* realizou uma quantidade maior de expansões do que a busca gulosa em todos os casos.