Amanda Moreira e Gustavo Faria

Relatório Trabalho de implementação de algoritmos de busca

Uberlândia

2020

1 Introdução

Os Agentes reativos não funcionam em ambientes para quais o número de regras condição-ação é grande demais para armazenar. Nesse caso um temos o agente de resolução de problemas, é um tipo de agente baseado em objetivo. Um agente com várias opções imediatas pode decidir o que fazer comparando diferentes sequências de ações possíveis. A procura pela melhor sequência é chamado de busca.(RUSSELL, 2013)

Os algoritmos de busca sem informação — tem apenas a definição do problema como informação. Ainda que alguns desses algoritmos resolvam qualquer problema que tem solução, nenhum deles o faz de forma eficiente. As buscas sem informação apenas geram sucessores e verificam se o estado objetivo foi atingido e as estratégias de busca sem informação se distinguem pela ordem em que os nós são expandidos.

Já os algoritmos de busca informada tem sucesso obtendo orientação sobre onde procurar soluções.Utilizando conhecimento específico, além da definição do problema, para encontrar soluções de forma mais eficiente. Abordagem geral: busca pela melhor escolha. Utiliza uma função de avaliação f(n) para cada nó e expande o nó que tem a menor f(n). A estratégia muda conforme a fução de avaliação.

Um problema é definido por quatro itens: Estado inicial, Ações ou função sucessor S(x) = conjunto de pares estado-ação, Teste de objetivo, pode ser explícito ou implícito e Custo de caminho (aditivo). Uma solução é uma sequência de ações que levam do estado inicial para o estado objetivo. Uma solução ótima é uma solução com o menor custo de caminho.

1.1 Buscas

1.1.1 Buscas Sem Informação

Idéia: Percorrer o espaço de estados a partir de uma árvore de busca. Expandir o estado atual aplicando a função sucessor, gerando novos estados. A busca irá seguir um caminho, deixando os outros para depois. A estratégia determina qual caminho seguir.

Espaço de estados é o conjunto de todos os estados acessíveis a partir de um estado inicial. Os estados acessíveis são aqueles dados pela função sucessora.

Uma estratégia de busca é definida pela escolha da ordem da expansão de nós e são avaliadas de acordo com os seguintes critérios:

1. completeza: sempre encontra a solução (se existe)

2. complexidade de tempo: número de nós gerados
3. complexidade de espaço: número máximo de nós na memória
4. otimização: a solução encontrada é ótima?

Complexidade de tempo e espaço são medidas em termos de:

1. b: máximo fator de ramificação da árvore (número máximo de sucessores de qualquer nó)
2. d: profundidade do nó objetivo menos profundo
3. o comprimento máximo de qualquer caminho no espaço de estados (pode ser infinita)

1.1.2 Buscas Informada ou Heurística

Idéia: usar uma função de avaliação f(n) para cada nó (estimativa do quanto aquele nó é desejável) e expandir nó mais desejável que ainda não foi expandido.

Implementação: ordenar nós na borda em ordem decrescente de acordo com a função de avaliação.

Uma heurística h(n) é admissível se para cada nó n, h(n) menor igual h*(n), onde h*(n) é o custo verdadeiro de alcançar o estado objetivo a partir de n. Nunca superestima o custo de alcançar o objetivo, isto é, ela é ótima.

Uma heurística é consistente (ou monotônica) se para cada nó n, cada sucessor n' de n gerado por qualquer ação a. Além disso f(n) é não-decrescente ao longo de qualquer caminho.

1.1.2.1 Busca Local

Em muitos problemas de otimização o caminho para o objetivo é irrelevante, isto é, não importando a seqüência de ações e este mantêm apenas o estado atual, sem a necessidade de manter a árvore de busca.

1.1.2.2 Heurística

Nesse trabalho usaremos a **distância de Manhattan** do estado atual até o objetivo como função heurística.

A Distância Manhattan tem a seguinte definição: a soma das diferenças entre x e y, em módulo, em cada dimensão.

$$|x1 \breve{} x2| + |y1 \breve{} y2|$$

1.2 Buscas Implementadas

Este trabalho consiste em implementar na linguagem Python (v.3) os algoritmos descritos abaixo para simular o comportamento de um agente cruzando um labirinto.(CODE...,)

Os programas bases disponibilizados permitem criar objetos em Python que representam labirintos aleatórios 2D a partir do algoritmo de Kruskal e apresentam uma interface simples para explorar esses labirintos. (PYMAZE...,)

Esses programas foram ligeiramente modificados, de modo a simplificar os labirintos e a interface para o teste de algoritmos de busca.

1.2.1 Profundidade

Pesquisa em profundidade (DFS) é utilizado para pesquisar um grafo ou estrutura de dados em árvore. A busca começa no nó raiz (topo) de uma árvore e segue até o nó folha de um determinado ramo (caminho), então retrocede até encontrar um caminho ainda não percorrido e então o explora. Esse processo é realizado até que todo o grafo tenha sido explorado.

A busca em profundidade é útil para solução de problemas como: analisar redes, mapear rotas, programar e encontrar árvores geradoras são problemas de gráfico.

1.2.2 Profundidade Iterativa

A busca em profundidade iterativa, ou busca limitada, é usado para realizar a busca em profundidade onde temos uma árvore ilimitada no algoritmo e e isso é resolvido estabelecendo um limite à profundidade do domínio de pesquisa.

Esse limite de profundidade torna a estratégia de pesquisa DFS mais refinada. O limite é denotado por l, fornecendo a solução para o problema do caminho infinito que se originou anteriormente no algoritmo de busca em profundidade.

O limite de profundidade é obrigatório para a execução deste algoritmo. O nó objetivo pode não existir no limite de profundidade definido anteriormente, o que levará o usuário a iterar ainda mais, adicionando tempo de execução.

1.2.3 Descida de Encosta

A descida de encosta, ou escalada, é um tipo de algoritmo de busca local. Nele, os estados vizinhos são comparados ao estado atual e, se algum deles for melhor, mudamos o nó atual para o estado vizinho. O que o qualifica como melhor é definido pela função

objetivo, que prefere um valor mais alto, ou no caso de uma função decrescente um valor mais baixo.

Neste algoritmo, começamos com um estado atual. Em alguns problemas, saberemos qual é o estado atual, enquanto, em outros, teremos que começar selecionando um aleatoriamente. Em seguida, repetimos as seguintes ações: avaliamos os vizinhos, selecionando aquele com o melhor valor. Em seguida, comparamos o valor desse vizinho com o valor do estado atual. Se o vizinho for melhor, mudamos o estado atual para o estado vizinho e repetimos o processo. O processo termina quando comparamos o melhor vizinho ao estado atual, e o estado atual é melhor.

1.2.4 Tempera Simulada

A tempera simulada permite que o algoritmo saia da posição atual caso essa for um máximo local.

A tempera é o processo de aquecer o metal e permitir que ele resfrie lentamente, o que serve para endurecer o metal. Isso é usado como uma metáfora para o algoritmo de tempera simulada, que começa com uma temperatura alta, tornando-a mais provável para tomar decisões aleatórias e, conforme a temperatura diminui, torna-se menos provável, tornando-se mais "firme".

Este mecanismo permite que o algoritmo mude seu estado para um vizinho que é pior do que o estado atual, que é como ele pode escapar dos máximos locais.

Combina a descida de encosta com um percurso aleatório resultando em eficiência e completeza. A descida de encosta da uma “chacoalhada” nos estados sucessores. Estados com avaliação pior podem ser escolhidos com uma certa probabilidade. Mas esta probabilidade diminui com o tempo

2 Resultados

2.1 Buscas

2.1.1 Experimento 1

Teste dos algoritmos de busca sem informação para 100 labirintos 10x10

Busca	Movimentos realizados na busca	Tamanho do caminho da solução
Profundidade	166.32	29.7
Profundidade Limitada	2047.32	19.8

Figura 1 – Experimento 1

2.1.2 Experimento 2

Teste dos algoritmos de busca que usam a distância de Manhattan como informação para 100 labirintos 10x10 *

Busca	Distância final até o objetivo	Movimentos realizados na busca	Tamanho do caminho da solução
Descida de Encosta	0.0	0.0	0.0
Têmpera Simulada T1(t)	0.0	0.0	0.0
Têmpera Simulada T2(t)	0.0	0.0	0.0

Figura 2 – Experimento 2

Os resultados obtidos foram inconclusivos, mostrando que os algoritmos implementados são ineficientes para resolver o problema em questão.

3 Conclusão

Em geral, o método de busca sem informação é preferido quando existe um espaço de busca grande e a profundidade da solução não é conhecida. A busca em profundidade tem uma grande desvantagem no caso do estado objetivo estar em uma ramificação oposta de onde inicia a busca na árvore.

A busca em profundidade iterativa é melhor do que a busca em profundidade e requer menos tempo e espaço de memória. Porém, o nó de objetivo não será encontrado se não existir no limite desejado.

E o método de busca informada, em geral pode obter soluções mais eficientemente do que uma estratégia de busca sem informação. A função de avaliação utilizada pode ser analisada como uma estimativa de custo, de modo que o nó com a menor avaliação será expandido primeiro.

As funções heurísticas é o conhecimento adicional ao problema transmitido no algoritmo de busca. Além disso, o desempenho dos algoritmos de busca heurística dependem da qualidade da função heurística implementada.

Concluímos através da observação feita nos testes que os resultados obtidos por todos os algoritmos de busca não informada implementados foram satisfatórios para a comprovação dos mesmos através da bibliografia utilizada. Com isso, podemos dizer que esses algoritmos funcionaram da forma esperada e prevista.

Já nos testes dos algoritmos de busca informada implemtados neste trabalho não obtivemos resultados satisfatórios, onde os mesmos se mostraram ineficientes para o problema em questão.

APÊNDICE A – Buscas não informadas

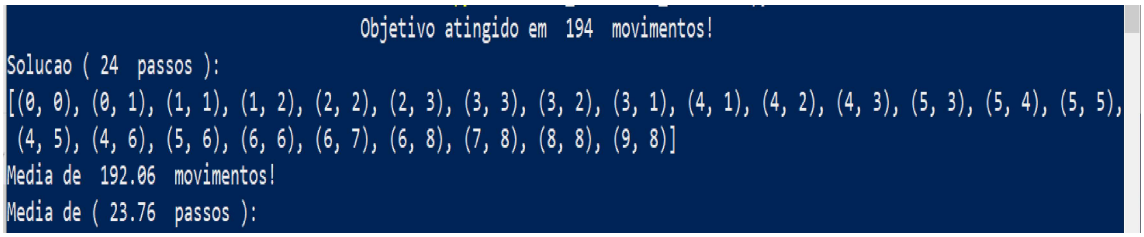

Figura 3 – não visual

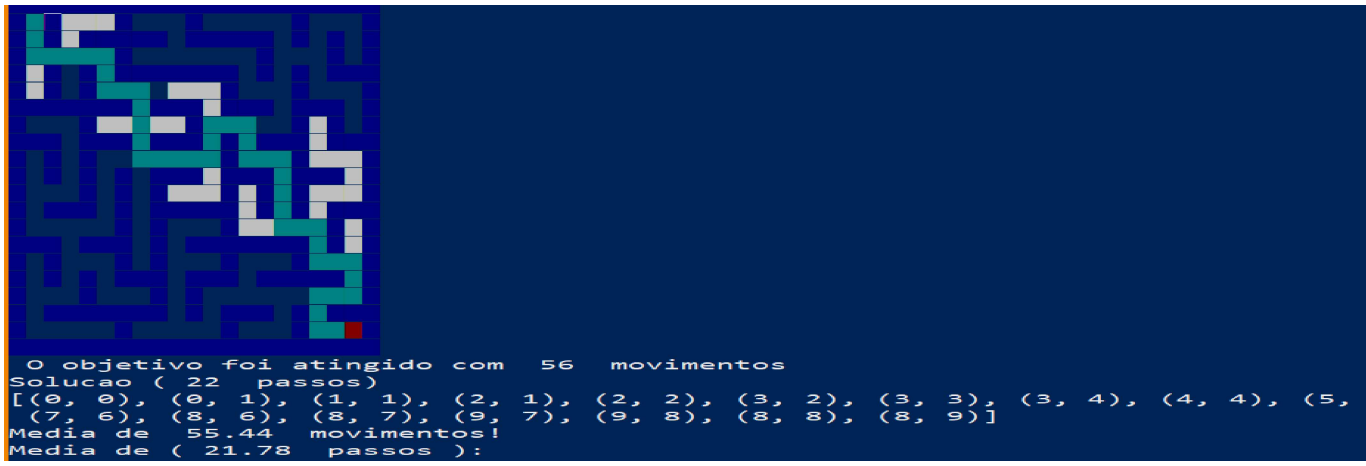

Figura 4 – visual

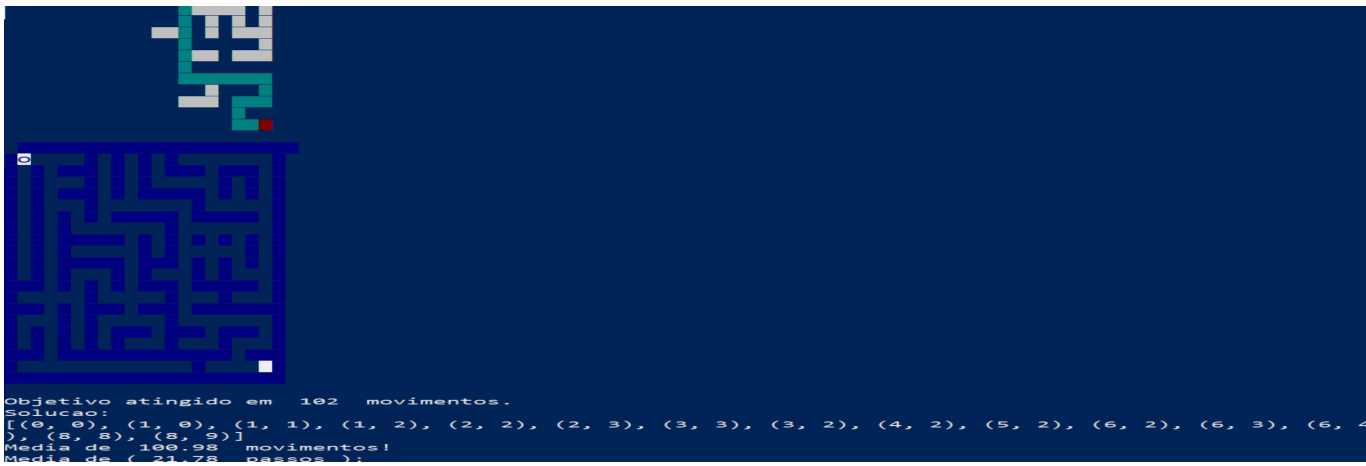

Figura 5 – visual only tracking

APÊNDICE B – Buscas que não atingiram o objetivo

Podemos ver que as buscas termina sem atingir o objetivo.

Figura 6 – Escalada

Referências

CODE for the book "Artificial Intelligence: A Modern Approach". [Online; accessed 27. Ago. 2020]. Disponível em: <https://github.com/aimacode>. Citado na página 3.

PYMAZE Paul Miller. [Online; accessed 27. Ago. 2020]. Disponível em: <https://github.com/138paulmiller/PyMaze>. Citado na página 3.

RUSSELL, P. N. S. *Artificial Intelligence*. [S.l.: s.n.], 2013. ISBN 978-85-352-3701-6. Citado na página 1.