Planejamento como Busca Heurística no Espaço de Estados

Elano N. Caitano, João Victor Aquino Correia, Roberto Coutinho
Universidade Federal do Ceará
Campus Quixadá
Quixadá, Brasil
nunes.elano@alu.ufc.br, jvac99@alu.ufc.br, roberto.coutinho@alu.ufc.br

***Resumo*—Planejamento Automatizado é o processo que gera uma sequência de ações necessárias para que um agente inteligente resolva um problema. Este relatório tem por objetivo apresentar os principais conceitos sobre heurísticas para planejamento como busca no espaço de estados, bem como apresentar detalhes da implementação da busca A* no planejador JavaFF e experimentos utilizando o domínio do robô de marte (*Mars Rover Domain*).**

***Index Terms*—Inteligência Artificial, Planejamento Automatizado, Busca Heurística.**

I. Planejamento Automatizado

O planejamento automatizado lida com a geração automática de ações para que um agente inteligente alcance suas metas. Portanto, dado um problema, existirá um conjunto de ações necessárias para que, a partir de um estado inicial, chegue-se uma resolução, que será o estado final.

Sendo assim, dado uma situação inicial, um conjunto de ações e uma situação final desejada, a tarefa de planejamento consiste em determinar uma sequência de ações que soluciona o problema, ou seja, uma sequência que atinja a situação desejada a partir da situação inicial.

A. A Linguagem PDDL

A Planning Domain Definition Language, ou PDDL, é uma linguagem baseada na lógica de predicados que busca permitir comparações de desempenho entre os planejadores em diferentes domínios. As ações são modeladas em termos de pré-condições e efeitos, ou seja, para uma ação qualquer haverão condições pré-definidas que precisam ser satisfeitas para que a ação seja aplicável, e após a sua execução acontecerá efeitos sob os predicados definidos anteriormente, onde é gerado valores para os predicados, sendo verdadeiro se for um efeito positivo ou falso se for um efeito negativo. Ademais, em um problema são definidos o nome, o domínio relacionado, os objetos, o estado inicial e a meta que deseja-se atingir. Nessa configuração, os objetos são os objetos do ”mundo real”, como no mundo dos blocos onde os objetos são os blocos. O estado inicial se caracteriza pelos elementos ”verdadeiros” na configuração. A meta é descrita como o objetivo que se deseja atingir, como por exemplo no mundo dos blocos, onde a meta pode ser que o bloco A esteja sobre o bloco B e ambos estejam sobre o bloco C.

B. Busca Progressiva no Espaço de Estados

Na busca progressiva, a partir do estado inicial serão expandidas todas as ações em busca de uma que seja válida, com objetivo de alcançar uma determinada meta. A expansão dos estados é feita de modo que a partir de um estado, verifica-se entre todas as ações possíveis se alguma ação é aplicável a esse estado, caso exista alguma ação válida, é feita a adição de seus efeitos positivos e a remoção de seus efeitos negativos.

II. Busca Heurística para Planejamento

A. Heurísticas Independentes de Domínio

Na heurística independente de domínio, é usado uma estimativa do número de ações necessárias para atingir a meta. Por exemplo, ao utilizar um plano, a estimativa será a distância euclidiana independente do domínio.

B. Busca Gulosa de Melhor Escolha (BFS)

A aplicação dessa configuração no algoritmo da busca gulosa de melhor escolha se dá através da expansão do nó que está mais próximo do objetivo. Em um grafo que representa um mapa de uma determinada região, os nós serão expandidos de acordo com o valor da menor distância em linha reta entre o nó atual e o nó objetivo. Utilizando este algoritmo não é possível garantir que o melhor caminho será encontrado, visto que só será usado o critério da distância como parâmetro para avaliar o caminho a ser escolhido.

C. Busca A*

Já a busca A* surge para suprir a necessidade da busca gulosa de melhor escolha. Nesse algoritmo de pesquisa, será utilizado uma combinação entre o valor da distância para o nó objetivo e o custo de caminho partindo do estado inicial até a meta.

III. Implementação

Foram criadas duas classes AStarSearch e HValueComparatorAStartSearch dentro do pacote search. AStarSearch é similar a classe BestFirstSearch, apenas com uma modificação na linha 47, onde foi modificado o segundo o argumento do this, antes era new HValueComparator() agora é new HValueComparatorAStartSearch().

De igual modo a classe HValueComparatorAStartSearch é bem semelhante a classe HValueComparator, foram efetuadas

duas modificações, nas linhas 44 e 45. Nas duas linhas temos a adição dos valores getHValue() e getGValue(), a diferença está no state, na linha 44 os gets são chamados pelo state s1, já na linha 45 são chamados pelo state s2.

Às modificações foram necessárias para converter nossa busca BFS para uma busca A*, pois como já citado anteriormente, a busca A* utiliza uma combinação entre o valor da heurística (getHValue()) e o custo de caminho (getGValue()). Enquanto que a BFS utiliza apenas o valor da heurística.

```
29  package javaff.search;
30
31  import javaff.planning.State;
32  import java.util.Comparator;
33  import java.math.BigDecimal;
34
35  public class HValueComparatorAStartSearch implements Comparator
36  {
37      public int compare(Object obj1, Object obj2)
38      {
39          int r = 0;
40          if ((obj1 instanceof State) && (obj2 instanceof State))
41          {
42              State s1 = (State) obj1;
43              State s2 = (State) obj2;
44              BigDecimal d1 = (s1.getHValue()).add(s1.getGValue());
45              BigDecimal d2 = (s2.getHValue()).add(s2.getGValue());
46
47
48              r = d1.compareTo(d2);
49              if (r == 0)
50              {
51                  d1 = s1.getGValue();
52                  d2 = s2.getGValue();
53                  r = d1.compareTo(d2);
54                  if (r == 0)
55                  {
56                      if (s1.hashCode() > s2.hashCode()) r = 1;
57                      else if (s1.hashCode() == s2.hashCode() && s1.equals(s2)) r=0;
58                      else r = -1;
59                  }
60              }
61          }
62          return r;
63      }
64  }
```

Figura 1. Código da implementação.

IV. Experimentos

Foram executados 11 arquivos de entrada que se diferenciam nos problemas que são usados no domínio Rover, de modo que foram executados em duas buscas heurísticas, a busca gulosa de melhor escolha e a busca A*. Após a execução, os resultados foram colocados em uma tabela onde é possível comparar o tamanho dos planos de cada resposta e o tempo de execução de cada algoritmo.

A. Descrição do Domínio

O domínio Rover tem em seus requisitos a palavra-chave typing, que indica que os objetos do domínio devem ser declarados com o seu respectivo tipo, por consequência na seção types, são declaradas os tipos do domínio. Na seção predicates são declarados cada predicado do domínio, onde os objetos presentes representam valores booleanos, logo, na referida seção do arquivo do domínio se encontra a sentença *"(equipped_for_rock_analysis ?r - rover)"*, que indica se o Rover está equipado para análise de solo, além de outros predicados que indicam outras configurações.

As seções "action" são declaradas de forma indívidual, uma vez que cada ação é diferente. Uma ação possui um nome, uma lista dos parâmetros, as pré-condições e efeitos. Um exemplo do domínio Rover é a ação "take_image", que possui como parãmetros o Rover, o ponto de parada (Waypoint), o objetivo (objective), a camêra e o modo. Este parâmetros são necessários para que sejam verificadas as pré condições, que são específicadas na seção dos predicados, sendo elas: calibrated (se está calibrado), on_board (se está na borda), equipped_for_imaging (se está equipado para imagens), supports (se está com suporte para câmera), visible_from (se o objeto está vísivel do ponto de parada) e at (em que lugar está se referindo). Os efeitos são a aplicação de um valor booleano para o predicado "have_image"e a negação do valor booleano do predicado "calibrated".

B. Resultados

Posterior a realização dos experimentos, é possível comparar os resultados de ambas as buscas no domínio determinado. A busca gulosa de melhor escolha oferece resultados com menor tempo em relação aos resultados oferecidos pela busca A*, contudo, o tamanho dos planos na busca A* apresentaram ser menores ou iguais aos da busca gulosa. Os resultados para comparação estão disponíveis na tabela a seguir:

ALUNOS	Roberto, Elano, João victor			
DOMÍNIO - MARS ROVER				
Problema	Busca Gulosa (*Best First Search*)		Busca A*	
	Tempo	*Tamanho do Plano*	*Tempo*	*Tamanho do Plano*
rovers-01	0.4sec	10	0.785sec	10
rovers-02	0.259sec	8	0.651sec	8
rovers-03	0.345sec	12	1.046sec	11
rovers-04	0.432sec	8	0.53sec	8
rovers-05	0.985sec	22	timeout	timeout
rovers-15	167.022sec	43	timeout	timeout
rovers-16	44.118sec	43	timeout	timeout
rovers-17	timeout	timeout	timeout	timeout
rovers-18	timeout	timeout	timeout	timeout
rovers-19	timeout	timeout	timeout	timeout
rovers-20	timeout	timeout	timeout	timeout

Figura 2. Tabela com resultados do Experimento.

V. Conclusão

O planejamento consiste em gerar uma sequência de ações que leva até uma solução a partir de uma configuração inicial de problema.

Durante a disciplina foram visto dois tipos de buscas Heurísticas, a busca gulosa de melhor escolha e a busca A*. Na busca gulosa o nó a ser expandido é baseado em seu valor heurístico ficando na borda o nó que tem o melhor valor, dessa forma essa busca consistisse apenas nos valores heurístico, já na busca A* é feita a soma do valor heurístico com o custo do caminho até o nó, dessa forma o nó que possui o melhor resultado dessa soma fica na borda para ser expandido.

Os experimentos realizados consistem em executar 11 arquivos de entradas com diferentes problemas usando o mesmo domínio, no caso o domínio do robô de Marte, esses arquivos foram executados primeiro usando a busca gulosa e em seguida usando a busca A* e foram registrados em uma planilha o tempo de execução e o tamanho do plano de resposta de cada um dos experimentos.

Após realizados os experimentos percebemos pelos resultados que a busca gulosa consegue chegar a um resultado em um tempo menor que a busca A*, porém a busca A* acha a menor solução, dessa a forma a escolha da busca a ser usada depende do que se está trabalhando, se o objetivo for achar

uma solução em menor tempo pode-se usar a busca gulosa, mas se o objetivo for achar o melhor resultado deve ser usada a busca A*.

REFERÊNCIAS

[1] Lima, Edirlei Soares. ‘‘*Busca Heuristica.*”. Disponível em https://www.inf.ufsc.br/~alexandre.goncalves.silva/courses/14s2/ine5416/trabalhos/t3B/aula.pdf. Acesso em: 08/02/2022.

[2] Russell, S. e Novig, P. ‘‘*P. Artificial Intelligence: a Modern Approach*”. 2nd Edition, Prentice-Hall, 2003.