Planejamento como Busca Heurística no Espaço de Estados

Felipe de Sousa Santos, Lucas Ferreira Lima
Universidade Federal do Ceará
Campus Quixadá
Quixadá, Brasil
felipess.dev@gmail.com, lucasferreirapacoti@gmail.com

***Resumo*—Planejamento Automatizado é a sub-área da IA que estuda o processo de raciocínio por meio da computação. 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

Uma das mais importantes metas da IA é o desenvolvimento de um Resolvedor Geral de Problemas (*General Problem Solver*) [(*Newell Simon, 1961*]. A ideia central é voltada da seguinte maneira: problemas são descritos em uma linguagem de alto-nível de abstração e já são resolvidos automaticamente. O objetivo desse método é facilitar a modelagem de problemas com prejuízo mínimo em termos de desempenho. Um sistema de planejamento pode ser considerado um solucionador geral de problemas [(*Fikes, 1971*]. O planejamento automatizado consiste no processo deliberativo de escolha de ações para que um agente inteligente alcance suas metas. Esses agentes devem controlar a evolução do ambiente ao seu redor de uma maneira desejada. Além disso, devem ser capazes de construir planos, que são uma sequência de ações escolhidas com base nos objetivos a serem atingidos pelos agentes. A complexidade de se construir planos depende de uma variedade de propriedades do ambiente e do agente. Para se construir um modelo conceitual é necessário uma Descrição do sistema((*Ambiente*), a especificação de estados iniciais e metas e Plano de ações que são gerados pelo planejador, também chamado de programa de controle. Para realizar um planejamento, requer-se um modelo geral para descrever a dinâmica do Sistema. A maioria das abordagens de planejamento são baseados em sistema de transição de estados.

A. A Linguagem PDDL

A linguagem PDDL - "The Planning Domain Definition Language"[(*McD98b*] foi desenvolvida em 1998 com o objetivo de unificar a definição de problemas para sistemas de planejamento, permitindo assim compará-los mais facilmente a partir de um mesmo conjunto de problemas. Essa linguagem é uma combinação das linguagens STRIPS e ADL e é capaz de representar: efeitos condicionais; quantificadores universais; universos dinâmicos, permitindo a criação e a destruição de objetos; axiomas de domínio; especificação de ações hierárquicas compostas de sub-ações e sub-metas e gerenciamento de múltiplos problemas em múltiplos domínios.

B. Busca Progressiva no Espaço de Estados

A busca progressiva começa do estado inicial e progride para seus estados sucessores, depois ela progride para o sucessores dos sucessores, repetindo essa progressão a cada estado encontrado, excluindo estados que já foram encontrados antes, até achar um estado que satisfaz a meta. Quando a busca chega em um estado que satisfaz a meta, ela retorna a sequência de ações que foram executadas para chegar nesse estado. Se não encontrar o problema não tem solução. No mundo de blocos a busca progressiva ocorreria da seguinte maneira:

Initial state: [(on-table A), (on-table B),(on-table C)]
Busca Progressiva:
(pick-up A),
(stack A, B),
(unstack A, B),
(put-down A),
(pick-up B),
(stack B, C),
(pikc-up A),
(stack A, B)

II. Busca Heurística para Planejamento

A busca heurística, ou busca informada, utiliza um conhecimento específico do problema na escolha do próximo estado a ser expandido. Esse conhecimento pode ser representado por uma função heurística que estima o custo de se chegar ao estado meta a partir de um estado qualquer. Essa estimativa pode ser obtida por meio de uma versão relaxada do problema ou seja, uma solução que contém menos restrições do que uma solução ideal. Uma heurística h(s) serve para estimar a distância de um estado s até a meta, onde algoritmos de busca priorizam expandir

estados que tem menor valor de h, ou seja, estados que acredita-se estarem mais próximos à meta para que a busca seja executada o mais rápido possível.

A. Heurísticas Independentes de Domínio

B. Busca Gulosa de Melhor Escolha (BFS)

Busca Gulosa de Melhor Escolha (BFS)

Utiliza-se de informações baseadas em heurísticas para escolher os nós mais próximos do objetivo, partindo da premissa que esse formato de busca vai levar à solução mais rapidamente. **Função de avaliação:** f(n) = h(n) = custo de n até o objetivo.
A busca gulosa expande o nó que parece mais próximo ao objetivo de acordo com a função heurística. Todavia, a **BFS** possui uma caraterística de não-completude, ou seja, é possível que durante a expansão o algoritmo chegue a um nó folha que não possui transições para outros nós, e a solução não seja encontrada. Isso ocorre quando esse nó sem-saída está próximo à solução mas não faz parte do caminho que leva da raiz até o nó solução. Em termos de complexidade, tanto para complexidade de tempo como para complexidade de espaço, a Busca Gulosa de Melhor Escolha possui $O(b^m)$.

C. Busca A*

Busca A* evita expandir caminhos que são caros, para isso ele soma o custo dos nós até o objetivo. **Função de avaliação:** f(n) = g(n) + h(n)

- g(n) = custo até o momento para alcançar n;
- h(n) = custo estimado de n até o objetivo;
- f(n) = custo total estimado do caminho através de n até o objetivo.

Para a solução encontrada pela Busca A* seja ótima, a heurística precisa ser admissível(sempre menor ou igual ao custo real). Esse algoritmo é melhor que a abordagem gulosa, para casos onde existem nós "sem-saída". Como aqui existem pelo menos dois fatores para avaliar se um nó deve ser expandido ou não, então o risco de chegar a um nó que não pertence ao caminho é bem menor e, o mais importante, é possível voltar para um nó anterior e considerar as outras opções, diferentemente do algoritmo guloso que nessas situações entra em laço infinito e já não consegue encontrar a solução. Contudo, o algoritmo também possui desvantagens, especialmente relacionadas à complexidade de espaço. Como a expansão sempre procura pelo melhor estado seguinte para ser visitado, em uma situação com vários nós solução é possível que o algoritmo acabe por escolher as de menor custo, já que também são soluções, mesmo que isso signifique não encontrar as soluções ótimas.
Busca A* é ruim pois consome muita memória.

III. Implementação

O algoritmo A* também é do tipo Best-First, ou seja, ele também busca pela melhor opção primeiro, portanto as funções de busca são as mesmas. A diferança se encontra na execução, pois o A* não sai de forma "gulosa"mas sim utilizando uma combinação de custo e heurística descrita em sua função de avaliação.
Função de avaliação: f(n) = g(n) + h(n)
Portanto, iremos alterar a função que realiaza o comparação heurística. Mais espeficicamente alterando o condicional. Ao invés de verificar ambos valores separadamente como é feito na **BFS**, verificamos ambos juntos armazenando na variável BigDecimal o custo estimado de n mais o custo atual para alcançar n.

```
State s = (State) obj1;
BigDecimal d = s.getHValue().add(s.getGValue());
```

IV. Experimentos

A. Descrição do Domínio

O domínio do Robô de Marte tem sua modelagem motivada pelas missões executadas pelo MSL (Mars Science Laboratory). O domínio do Robô de Marte modela uma tarefa de um ou mais agentes autônomos que realizam exploração planetária. Os robôs navegam entre pontos de apoio no planeta, coletando amostras de solo e rocha para serem analisadas. Cada robô possui um número determinado de câmeras para obter imagens de certos objetos. Dependendo da câmera que o robô possui, é possível obter imagens coloridas, em alta resolução ou baixa resolução. As imagens e os dados resultantes das análises de rocha e solo devem ser transmitidos para a estação espacial que serve de suporte para o robô.

B. Resultados

ALUNOS	Felipe de Sousa Santos Lucas Ferreira Lima			
DOMÍNIO - MARS ROVER				
Problema	Busca Gulosa (*Best First Search*)		Busca A*	
	Tempo	*Tamanho do Plano*	*Tempo*	*Tamanho do Plano*
rovers-01	**0.531sec**	**10**	**2.381sec**	**10**
rovers-02	**0.784sec**	**9**	**1.075sec**	**8**
rovers-03	**2.008sec**	**15**	**1.361sec**	**11**
rovers-04	**1.069sec**	**8**	**1.232sec**	**8**
rovers-05	**1.573sec**	**22**		*Out of Memory*
rovers-15	**146.424sec**	**43**		*Out of Memory*
rovers-16	**472.232sec**	**44**		*Out of Memory*
rovers-17		*Out of Memory*		*Out of Memory*
rovers-18		*Out of Memory*		*Out of Memory*
rovers-19		*Out of Memory*		*Out of Memory*
rovers-20		*Out of Memory*		*Out of Memory*

Os testes foram feitos em uma máquina Intel(R)Pentium(R) CPU G3250 @ 3.20GHz 3.20GHz com uma memória RAM 4,00 GB. A **BFS** executou até o caso 16 estourando o limite de 5 minutos pré-determinado e atingindo 7,8 minutos. A busca A* por ser mais complexa e consumir mais memória só conseguiu executar até o rovers-04.

V. Conclusão

Esse trabalho propôs os principais conceitos sobre heurísticas para planejamento como busca no espaço de estados. Apresentando um subtipo de planejamento, o *Planejamento Automatizado* com testes realizados utilizando agentes inteligentes. Tendo os domínios e problemas implementados em PDDL, linguagem padrão para descrever domínios de planejamento. Que permite incluir: tipos, funções, variáveis numéricas, ações durativas, etc.
Dentre as buscas apresentadas, concluímos com base nos testes realizado no domínio *Robô de Marte*, que a *Busca Gulosa de Melhor Escolha*, mesmo com o problema de não completude tem um desempenho melhor na hora de realizar as operações no domínio, já a $A^{\star}$ consome mais memória para armazenar os valores que ela visita, desse modo deixando uma execução mais lenta(também de levar em conta a baixa memória da máquina em que foram executados os testes, onde esse problema fica bem evidente).

Referências

[1] RUSSELL, S.; NORVIG, P. ‘‘*Inteligencia Artificial*”. 3 ed. Campus, 2013.

[2] M. Ghallab, Dana Nau, and Paulo Traverso. ‘‘*Automated Planning: Theory and Practice*”. Morgan Kaufmann Publishers 2004.

[3] Silvio do Lago Pereira. ‘‘*Planejamento no Cálculo de Eventos*”. Dissertação de Mestrado em Ciência da Computação, IME-USP, 2002.