# Desenvolvimento do algoritmo de A* para solução de planos PDDL

Guilherme Miranda Rossato
Pontifícia Universidade Católica do Rio Grande do Sul
Escola Politécnica – Engenharia da Computação
Av. Ipiranga, 6681 Prédio 32 Sala 505 – Bairro Partenon – CEP 90619-900 – Porto Alegre – RS
guilherme.rossato@acad.pucrs.br

**Abstract**

Este documento tem como objetivo explicar o desenvolver do algoritmo de pesquisa conhecido como A* para a solução de problemas descritos em PDDL (*Planning Domain Definition Language*), sua heuristica e também uma função que valida se um plano é uma solução para um problema.

## I. Algoritmo de busca A* (A star)

A* é um algorimo de busca ou roteamento informado completo entre pontos ou nodos de um grafo que utiliza uma heuristica para obter uma boa performance e precisão na obtenção do menor caminho entre dois pontos.

O princípio do algoritmo de busca é extender seus nodos adjacentes um a um até encontrar o seu destino. A principal caracteristica do A* é a forma com que ele decide, a cada etapa, qual dos nodos expandir através de uma função heuristica e o custo de cada nodo.

## II. BASE DE DESENVOLVIMENTO

O início do desenvolvimento foi feito a partir de um conjunto de scripts preparadas para interpretar e normalizar os estados iniciais e seus predicados, todas as ações possíveis e suas dependências e os objetivos de um determinado plano.

Existem também rotinas de testes já implementadas no sistema, o que significa que o desenvolver do projeto terá uma característica de desenvolvimento orientado a testes, o que facilita a implementação e a validação dos algoritmos.

O projeto foi então desenvolvido sobre o Jupyter Notebook, uma ferramenta que roda um servidor local para disponibilizar uma aplicação web para então criar e colaborar com documentos que também abriga código que pode ser executado, provendo um sistema que facilita o acompanhamento de um código.

A linguagem de programação utilizada é a linguagem interpretada conhecida como Python, versão 3.

## III. HEURÍSTICA

A primeira implementação foi de uma função capaz de retornar o número exato de passos entre o estado inicial e o estado final. Isso logo se mostrou incorreto, tanto para os testes desenvolvidos na base de desenvolvimento quanto a complexidade computacional que o método exigia (era baseado na busca exaustiva ou por força bruta). Acredito que esse tipo de erro não aconteceria se o desenvolvimento houvesse começado na etapa do solucionador de planos uma vez que fica evidente a necessidade de um "relaxamento" do problema devido a frequência de execução da função de heurística.

A função de heurística foi re-implementada e, na segunda abordagem, ela foi feita com base no algoritmo do Graphplan, onde existem listas de possíveis estados conectados por ações, tais ações criam o próximo estado e estes estados podem ser aplicados em outras ações que geram uma nova lista de possíveis estados e assim por diante.

Para encontrar a máxima heurística foi feito uma simplificação do problema inicial através desse algoritmo, relaxando-o para que seja possível identificar quantas iterações do algoritmo são necessárias para atingir todos os objetivos individualmente.

De forma detalhada, a função criada uma lista de possíveis estados com todos os predicados "atingíveis", o número de expansões desse para estados cada vez mais abrangentes de predicados possíveis será a nossa heurística. Além de ser uma abordagem extremamente rápida, ela bate com os testes implementados e portanto foi possível validar a sua implementação com sucesso.

## IV. VALIDAÇÃO DE PLANO

Na etapa de validação de plano é necessário desenvolver uma função capaz de validar se uma lista de ações é capaz de solucionar um plano.

Essa etapa é relativamente simples, porém, a lista de estados vem com ações que tem propriedades faltando, isto é, o plano é enviado de forma incompleta para a função de validação e é preciso popular ela corretamente uma vez que a lista de ações do planos em pddl não deixam evidentes as ações, já que isso faz parte da etapa de descrição do problema.

A lista de todas as ações possíveis (com todos os parâmetros possíveis) é um parâmetro da função de validação, então bastou criar uma função capaz de iterar por cada ação até encontrar o correspondente na lista de plano (a mesma ação com os mesmos parâmetros) e substituir esta ação do plano incompleta pela instância corretamente populada.

## V. SOLUCIONADOR DE PLANOS

A solução de planos é onde a heurística e o algoritmo de busca são conectados. A base da implementação é que as ações possìveis em um estado sejam agrupadas e ordenadas de acordo com a heurística do estado resultante da execução desse, essa ordenação definirá qual a ação que deve ser expandida e explorada mais a fundo.

Para iniciar o desenvolvimento foi criado um código simples de busca informada e completa, onde cada ação é explorada por ordem em que foi definida (também conhecida como busca *breadth-first*), para validar a lógica da função em si.

Depois de validar o algoritmo de busca de caminho sem ordenação de ações (breadth-first), foi implementado um sistema de fila de prioridade nas ações para implementar o algoritmo A* por completo.

Para desenvolver a fila de prioridade, foi criado um código separado para realizar os testes necessários para validar a classe de lista de prioridade disponibilizada no Python e em seguida ela foi implementada no código de solucionador de planos para guardar o estado e o custo de chegar aquele estado (basicamente, ele é o custo unitário que cada troca de estado tem) . Essa função funciona mantendo uma lista organizada toda vez que um elemento seja adicionado, garantindo que a complexidade da operação de retirada do elemento com menor prioridade seja linear.

Como há a necessidade de gerar o plano resultante, isto é, a sequência de ações que solucionam o plano, foi criado uma lista simplesmente encadeada de retrospectiva, com todas as ações tomadas e suas respectivas origens. Com isso, gerar a sequência de ações só envolve percorrer essa lista em ordem reversa, adicionando cada ação a uma lista e, no final, revertendo-a, já que ela está de trás pra frente.

Para finalizar a implementação do algoritmo A*, a definição de prioridade é feita com base nos passos tomados (a soma unitária dos custos de cada ação) somados com a função de máxima heurística implementada anteriormente.

## VI. CONCLUSÃO

Para medir as diferenças dos algoritmos, foi utilizado o problema do robô empilhador, onde além de mover-se entre localizações o robô é capaz de pegar ou soltar caixas nestes lugares, este problema é descrito no arquivo *examples/dwr/dwr.pddl* do repositório.

Neste exemplo, foram necessárias 5871 expansões e uma média de 8 segundos para encontrar a solução de 17 passos em breadth-first.

A implementação com a máxima heurística e o custo unitário das ações utiliza, como mencionado na seção anterior, uma fila de prioridades para expandir as ações que levam a estados com menor heurística, ou seja, a implementação do algoritmo A* faz com que a expansão de ações priorize ações de forma a minimizar a distância entre o estado resultante e a heurística implementada. No mesmo exemplo, isso resultou em apenas 781 expansões de estado e em média 5.5 segundos para encontrar a mesma solução do exemplo anteriormente mencionado.

Embora as expansões tenham caído em quase 7 vezes, o tempo de execução não melhorou nem 2 vezes, isso pode parecer contra-intuitivo, porém, o ganho em tempo não é muito significativo devido ao impacto do cálculo da heurística que é certamente menor do que expandir tantos nodos desnecessários (o que a ausência desta heuristica causaria). O impacto computacional da implementação da heuristica e lista de prioridade é justificado pois a implementação diminui o fator de ramificação da busca, restringindo as ações mais convenientes para a solução e diminuindo a complexidade computacional do problema que, por consequência, faz esta abordagem cada vez mais eficiente conforme o tamanho do problema aumenta quando comparado ao método sem prioridade e sem heuristica (breadth-first).

## VII. REFERÊNCIAS

O projeto foi desenvolvido sobre a base de pesquisa por heuristica disponibilizada no Github para os alunos da disciplina de Inteligencia Artifical do segundo semetre de 2018.

“Base code for the heuristic planning assignment” - Autores: Felipe Meneguzzi (Prof.), Mauricio Magnaguagno (PhD Student), Leonardo Rosa Amado (PhD Student).

Os conhecimentos de Python 3 vieram de diversos experimentos e, principalmente, da documentação oficial, disponibilizada em:

https://docs.python.org/3/

Todos os conhecimentos teóricos apresentados aqui e necessários para o desenvolver desse projeto foi desenvolvido em aula e, já que este não é o foco deste documento, a teoria não foi expandida e, portanto, não há referências externas para a teoria deste projeto, já que essa não é o foco do documento.