Uso do Algoritmo de Busca A* em um Sistema de Navegação de um Robô para Manutenção de Unidades Fabris

Ediana da Silva de Souza[1], **Êrica Peters do Carmo**[1]

[1]Departamento de Ciência da Computação – Universidade do Estado de Santa Catarina (UDESC)
Joinville – SC – Brasil

{edianadasilvadesouza,ericapetersc}@gmail.com

Resumo. *O uso de algoritmo de busca tem sido amplamente utilizado em diferentes áreas, visto que através destes é possível reduzir o custo de recurso e de tempo para solucionar um problema. Para isso, é necessário saber qual classe dos algoritmos de busca melhor resolveria o problema em questão. Nesse sentido, este artigo tem como objetivo relatar o desenvolvimento do algoritmo de busca heurístico A* no cálculo do custo da navegação de um robô que realiza manutenção de unidades fabris. O artigo também estuda a diferença de comportamento do algoritmo A* quando alterado suas principais variáveis de atuação, bem como apresenta a comparação do algoritmo com os algoritmos de Dijkstra e Busca Gulosa. O estudo da literatura realizado indica que o método de busca heurístico A* é conhecido pela sua simplicidade e aplicabilidade. Como resultado deste trabalho, conclui-se a eficácia e eficiência do algoritmo A* como um solucionador do problema das rotas realizadas pelo robô, bem como seu melhor desempenho quando comparados aos algoritmos de Dijkstra e Busca Gulosa.*

1. Introdução

A resolução de problemas por meio dos algoritmos de busca é dado como uma importante técnica da área de Inteligência Artificial [Udemy 2018]. A escolha da estratégia de busca determina a ordem em que os nós de um grafo serão expandidos [de Miranda Alves 2007]. Neste sentido, o que deseja-se são rotas do espaço de busca que resultarão no caminho de custo mínimo. Dado este escopo, [Zanchin 2018] descreve que os algoritmos heurísticos permitem uma grande redução de custo e tempo na resolução de problemas, visto que utilizam um conhecimento específico em questão. O modelo de problema abordado neste trabalho descreve um robô com navegação automática que possui como função manutenir unidades fabris através de ferramentas encontradas no cenário, sendo este cenário dados por diferentes tipos de terrenos. Deseja-se então minimizar o custo do caminho que o robô trilhará até uma ferramenta, bem como até uma fábrica. Como a complexidade e o uso do algoritmo de busca heurístico A* visa ser enxuto e simples, de maneira que possa ser implementado nos mais diversos tipos de aplicações [Zanchin 2018], este trabalho descreve a implementação e avaliação do algoritmo A* no sistema de navegação do robô.

1.1. Justificativa e Motivação

O algoritmo de busca heurístico A* está presente em várias aplicações práticas utilizadas em nosso dia a dia, tal como em jogos e tecnologia GPS [Udemy 2018]. Este trabalho, então, justifica-se pela aplicabilidade do algoritmo, a garantia da obtenção do caminho de custo mínimo e simplicidade da implementação.

1.2. Objetivo

O objetivo deste trabalho foi implementar o sistema de navegação automática de um robô que realiza diferentes manutenções em unidades fabris. Para isto, utilizou-se e avaliou-se o algoritmo de busca heurística A* para o cálculo do custo da rota feita pelo agente durante sua atuação. O ambiente é composto por uma grade bidimensional, onde cada célula possui um tipo de terreno, bem como por unidades fabris e ferramentas dispostos aleatoriamente no cenário. Ao final da execução do algoritmo esperava-se encontrar todas as fábricas consertadas.

1.3. Algoritmo A*

O problema relacionado à busca do menor caminho entre dois pontos tem sido abordado em diversas áreas devido à sua variedade de aplicações. Nesse sentido, a definição de algoritmos de busca mais rápidos e eficientes tem sido sempre um motivo de discussão científica. Um desses algoritmos é o A*, introduzido em [Hart et al. 1968], e conhecido amplamente por sua performance na busca da melhor escolha.

O A* implementa uma função heurística, ou seja, caracteriza uma estratégia de busca que utiliza informações do problema para encontrar a solução mais eficiente. Assim, esse algoritmo calcula o menor caminho para ir de um nó n para outro combinando a função $g(n)$, que representa o custo para alcançar o nó n a partir do nó inicial, e a função $h(n)$, que representa o custo para ir do nó n até o nó objetivo [Russell and Norvig 2004]. Desse modo, pode-se descrever a função heurística do algoritmo como:

$$f(n) = g(n) + h(n)$$

Quando aplicados à busca em árvores, uma característica importante de algoritmos de busca é o caráter ótimo. Em [Nilsson 1980], o autor prova que a heurística do algoritmo A* é admissível desde que $h(n)$ respeite a seguinte propriedade: para um nó n e seu sucessor n', o custo de alcançar o objetivo a partir de n não é maior que o custo de ir de n para n' somado ao custo de alcançar o objetivo a partir de n'.

Ao admitir essa propriedade, temos que $h(n)$ tem o requisito da monotonicidade e é admissível. Logo, pode-se afirmar que a busca A* é ótima, ou seja, se há um ou mais caminhos entre um nó qualquer e um nó objetivo, a busca sempre retornará o menor caminho entre esses nós. Além disso, destaca-se também o fato da busca A* ser completa, se há um caminho entre um nó s e o objetivo, o algoritmo sempre retornará uma resposta [Russell and Norvig 2004].

Destaca-se ainda que a função $h(n)$ pode ser calculada com diversas fórmulas de distância. No algoritmo implementado neste trabalho, optou-se pela utilização da distância de Manhattan que define a distância entre dois pontos cartesianos $P = (x1, y1)$ e $P = (x2, y2)$ como:

$$d = |x1 - x2| + |y1 - y2|$$

Essa distância é a soma dos comprimentos das projeções das retas que ligam os pontos P e Q nos eixos X e Y. Logo, a distância entre dois pontos não é mais representada apenas por um segmento de reta, mas sim por um conjunto de segmentos que podem ser paralelos ao eixo X ou Y. Assim, a distância de Manhattan apresenta um cálculo mais fiel à situação do agente inteligente implementado nesse trabalho, que pode movimentar-se apenas vertical ou horizontalmente.

Um pseudocódigo para o A*, utilizado como base para a implementação do presente trabalho, é apresentado abaixo:

```
Adicionar nó_inicial na lista de nós para expandir.

Enquanto verdade:

    Definir q como o nó com menor f(n) na lista de nós para expandir.

    Para cada nó n vizinho de q:

        Definir q como nó pai de n.

        Se n é o nó_objetivo:

            Retornar caminho traçado até n.

        Calcular f(n) = g(n) + h(n).

        Se n está na lista de nós para expandir ou já expandidos com custo menor que f(n):

            Ignorar n.

        Se não:

            Adicionar n na lista de nós para expandir.

    Excluir q da lista de nós para expandir.

    Adicionar q na lista de nós já expandidos.
```

1.4. Organização do Texto

Este trabalho está estruturado da da seguinte forma: a seção 2 apresenta os métodos e justificativas de desenvolvimento do algoritmo implementado. A seção 3 fornece descrições sobre os experimentos realizados no ambiente e os resultados alcançados. A seção 4 estuda os resultados obtidos. Finalmente, conclusões e direções de pesquisas futuras são apresentadas na seção 5.

2. Metodologia de Desenvolvimento

Conforme descrito na seção 1.2, o objetivo do programa desenvolvido é a implementação de um agente inteligente capaz de perceber o estado (consertada ou em manutenção) de diversas fábricas ao seu redor, bem como a posição das ferramentas necessárias para o conserto dessas fábricas. Para pegar cada ferramenta e para locomover-se até cada fábrica, o agente deve calcular o menor caminho com o algoritmo A*. Assim, essa seção descreve a implementação da solução proposta para esse problema.

O programa (disponível em: `https://drive.google.com/open?id=1jKFyvkxanig0TvB9SqyA9QivEB23myjq`) foi implementado com Python e utiliza a biblioteca PyGame na criação da sua interface gráfica.

Em sua execução, o programa lê o arquivo 'ambiente.csv' do diretório em que está sendo executado. Esse arquivo pode ser configurado pelo usuário e é utilizado para descrever a configuração do ambiente que será percorrido pelo robô. O 'ambiente.csv' deve ser formado por uma matriz 42x42 em que cada célula contém um valor representando um tipo de terreno. Nota-se que cada tipo de terreno contém um custo relacionado à movimentação do robô sobre essa célula. Uma mudança adicionada ao trabalho foi que as células onde há fábricas dispostas possuem custo 1.

Tabela 1. Tipos de terrenos que podem ser definidos no arquivo 'ambiente.csv'.

Tipo de terreno	Número	Custo	Cor na Interface Gráfica
Sólido e plano	0	1	Verde
Montanhoso	1	5	Marrom
Pântano	2	10	Azul
Árido	3	15	Vermelho
Obstáculo - Intransponível	-1	-	Preto

Outro arquivo que também é lido pelo programa é o 'posicoes.csv'. As primeiras 5 linhas do arquivo definem as posições das fábricas no ambiente e a última linha define a posição inicial do robô. As linhas são do tipo: índice, x, y.

A partir desses arquivos, o programa define as configurações do ambiente que serão apresentadas na interface. Em seguida, a posição das ferramentas são geradas aleatoriamente, sendo que elas só podem ser dispostas em células com terreno do tipo sólido e plano. A tabela a seguir apresenta a relação entre as fábricas, as ferramentas que cada fábrica necessita e a quantidade de ferramentas necessárias/dispostas no ambiente.

Tabela 2. Fábricas e ferramentas dispostas no ambiente.

Fábricas		Ferramentas		
Índice	Nome	Tipo	Nº disponível no ambiente	Nº necessário para a fábrica
0	Indústria de grãos	Baterias de carga elétrica	20	8
1	Empresa de manutenção de embarcações	Braços de solda	10	5
2	Indústria petrolífera	Bombas de sucção	8	2
3	Fábrica de fundição	Dispositivos de refrigeração	6	5
4	Indústria de vigas de aço	Braços pneumáticos	4	2

Sobre as características do agente - robô - ele possui um radar capaz de detectar ferramentas à até quatro células de distância em qualquer direção, movimenta-se apenas na horizontal e vertical, e sabe previamente a localização das fábricas. A função $robot\_behaviour()$ define seu comportamento e é executada até todas as fábricas serem consertadas, sua implementação estrutura-se da seguinte forma:

Se não há nenhuma ferramenta no seu radar de visão:

    Movimenta-se aleatoriamente para uma célula adjacente (cima, baixo ou lados).

Se há uma ou mais ferramentas no seu radar de visão:

    Identifica a ferramenta mais próxima a partir da distância de Manhattan.

    Se ele ainda não tem todas as ferramentas deste tipo disponíveis:

        Calcula com A* o menor caminho até a ferramenta e move-se.

        Pega a ferramenta.

        Se já possui o número de ferramentas necessárias para consertar a fábrica que utiliza esse tipo:

            Calcula com A* o menor caminho até a fábrica e move-se.

            Conserta a fábrica.

    Se não:

        Movimenta-se aleatoriamente para uma célula adjacente (cima, baixo ou lados).

A partir da função acima entende-se que o robô coleta todas as ferramentas de determinado tipo necessárias antes de ir para a fábrica. Optou-se por essa abordagem pois garante uma maior movimentação do robô no ambiente, desde que as fábricas estejam distantes uma das outras, já que ele as visita em diferentes momentos da execução. Além disso, quando mais de uma ferramenta entra no seu radar, o agente calcula a melhor ordem para coletá-las de acordo com a sua proximidade dada pela distância de Manhattan.

Além do ambiente e dos elementos nele dispostos, a interface gráfica apresenta o número de ferramentas de cada tipo coletadas, o número total de ferramenta que ainda precisam ser coletadas e o estado de cada fábrica. Ainda, quando o agente calcula o caminho para uma ferramenta ou uma fábrica, os nós expandidos são apresentados (na cor branca) e o caminho escolhido é destacado (na cor amarela), enquanto o I e F marcam a posição inicial e final do caminho.

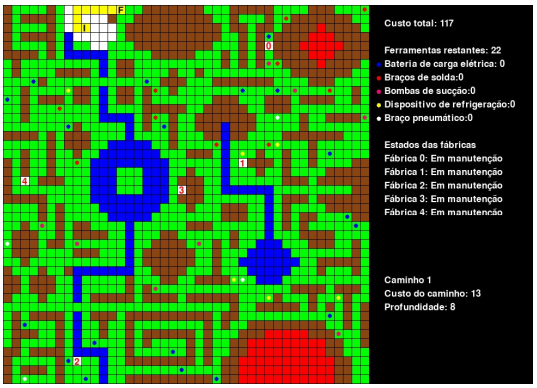
O critério de sucesso do agente é a minimização do custo da sua movimentação. Assim, as medidas de desempenho implementadas no programa são: o custo total da movimentação do robô no ambiente, o custo do caminho, o número de nós expandidos e a profundidade da árvore de cada caminho percorrido. Todas essas informações são apresentadas na tela durante a execução.

Destaca-se ainda que o programa salva as imagens referentes à árvore de expansão do A* e ao estado do ambiente cada vez que um caminho é calculado, mostrando assim todos os nós expandidos e a sequência de nós de menor custo. Algumas dessas imagens serão apresentadas na próxima seção, juntamente com detalhes das simulações realizadas e dos resultados obtidos.

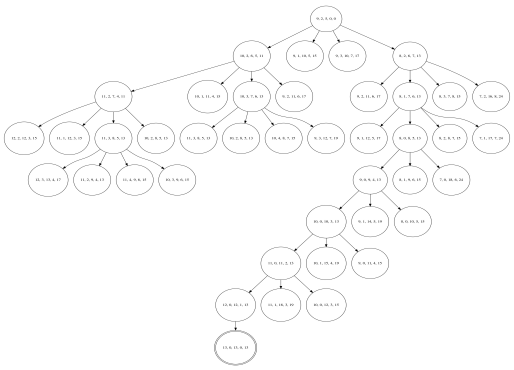
3. Descrição das Simulações e Resultados Obtidos

3.1. Simulações com o algoritmo padrão - A*

Como forma de analisar a solução desenvolvida, foram realizadas simulações dando diferentes enfoques para os algoritmos que formam o A* - através da aplicação de fatores multiplicadores - bem como testando o desempenho do programa para diferentes configurações do ambiente - com ou sem obstáculos.

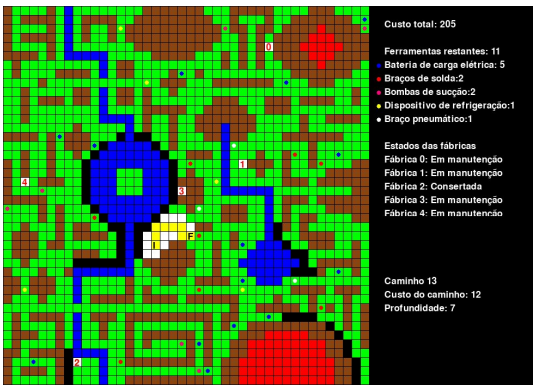


(a) Ambiente exibindo os nós expandidos, bem como o caminho de custo mínimo

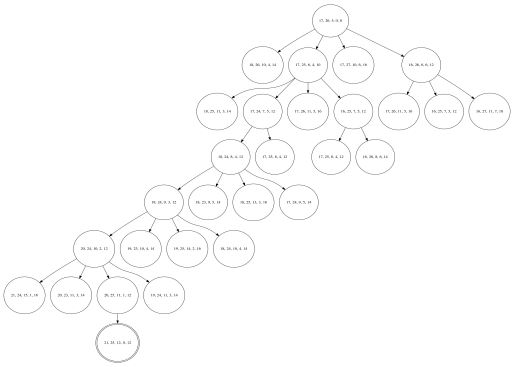


(b) Árvore de expansão

Figura 1. Execução com ambiente não dispondo de obstáculos. Fonte: as autoras, 2019.



(a) Ambiente exibindo os nós expandidos, bem como o caminho de custo mínimo



(b) Árvore de expansão

Figura 2. Execução com ambiente dispondo de obstáculos. Fonte: as autoras, 2019.

Foram realizadas três execuções para as simulações onde o ambiente se alterava pela presença ou não de obstáculos. Nelas, examinou-se somente o custo total das execuções, não examinando o custo de cada caminho pois, pela aleatoriedade, em cada caminho gerado, o objetivo do robô e sua posição poderia ser diferente nas execuções. Na Figura 1 (a) é mostrado um caminho gerado no ambiente quando o mesmo não possuía obstáculos e a Figura 1 (b) mostra a árvore de expansão. O mesmo é mostrado na Figura 2, mas quando o ambiente dispõe de obstáculos. O título de cada nó da árvore

de expansão possui 5 campos, sendo eles: *x, y, g(n), h(n), f(n)* onde, x e y representam a posição da célula expandidas, g(x,y) é o custo de (x,y) até o nó inicial, h(x,y) é a distância estimada (manhattan) de (x,y) até o nó de objetivo e f(n) é a função de avaliação do algoritmo A* dada por f(x,y) = g(x,y) + h(x,y). As execuções podem ser vistas de maneira mais completa através do link: `https://drive.google.com/drive/folders/16K-7LvUKZuApl602FCts1gp8hkSVXyor?usp=sharing`, onde o arquivo denominado como *inicio.jpeg* mostra a posição inicial do robô, os arquivos denominados como *captura* representam o espaço de estados e o caminho gerado no ambiente e os arquivos denominados como *caminho* representam a árvore de expansão. Os números dados aos arquivos são sequenciais, ou seja, o arquivo *caminho1* representa a árvore de expansão de *captura1*. Os valores dos custos totais de cada execução são mostrados na Tabela 3.

Tabela 3. Resultados das execuções do ambiente dispondo ou não de barreiras. Fonte: as autoras, 2019.

Execução	Obstáculo	Custo Total
1ª	Não	5042
2ª	Não	6677
3ª	Não	3481
1ª	Sim	2029
2ª	Sim	2027
3ª	Sim	7700

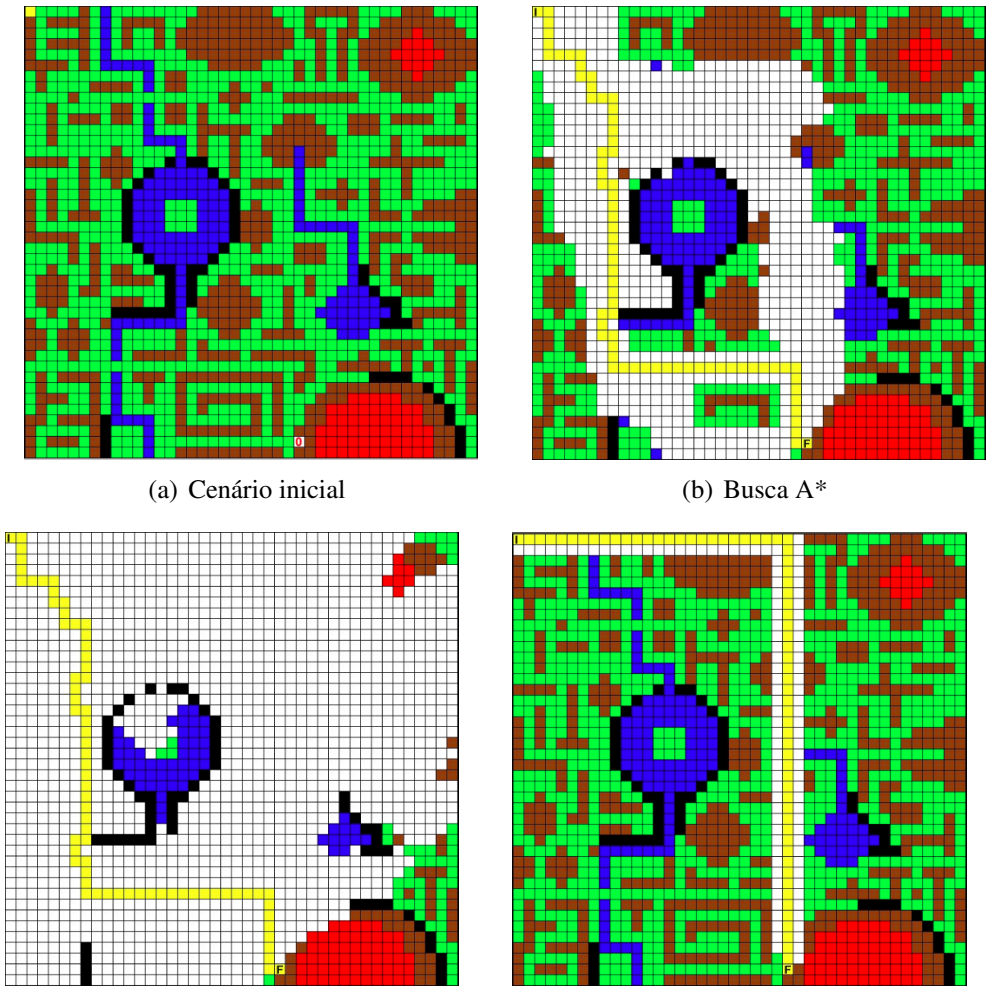
3.2. Simulações com variação na função heurística

O experimento também foi realizado variando o peso das função $g(n)$ e $h(n)$ na função heurística $f(n)$ do A*. Quando desconsideramos $h(n)$, tornamos $f(n) = g(n)$ e o algoritmo passa a se comportar como uma busca de custo uniforme (Dijkstra). Já quando desconsideramos $g(n)$, tornamos $f(n) = h(n)$ e o algoritmo torna-se uma busca gulosa. Para verificar a diferença entre as suas performances, foram executadas as três buscas (A*, Dijkstra e Busca Gulosa) para um mesmo cenário: robô posicionado em (0,0) buscando um caminho para chegar até uma fábrica em (25,40). A tabela 4. apresenta os resultados das execuções dessas buscas e a Figura 3 mostra os nós expandidos e os caminhos gerados.

Por fim, foram realizadas execuções mantendo uma das funções constantes, com peso 1, e variando o peso da outra. A Figura 4 (a) mostra os resultados obtidos ao variar o x em $f(n) = x.g(n) + h(n)$ e a Figura 4(b) mostra os resultados obtidos ao variar o x em $f(n) = g(n) + x.h(n)$.

Tabela 4. Resultados das buscas A*, Dijkstra e Busca Gulosa para um mesmo cenário.

Tipo de Busca	Custo do caminho encontrado	Tamanho do caminho encontrado	Nº de nós expandidos	Profundidade da árvore	Caminho ótimo encontrado
A*	74	70	691	69	Sim
Dijkstra	74	70	1484	69	Sim
Busca Gulosa	235	66	167	65	Não



(a) Cenário inicial (b) Busca A*

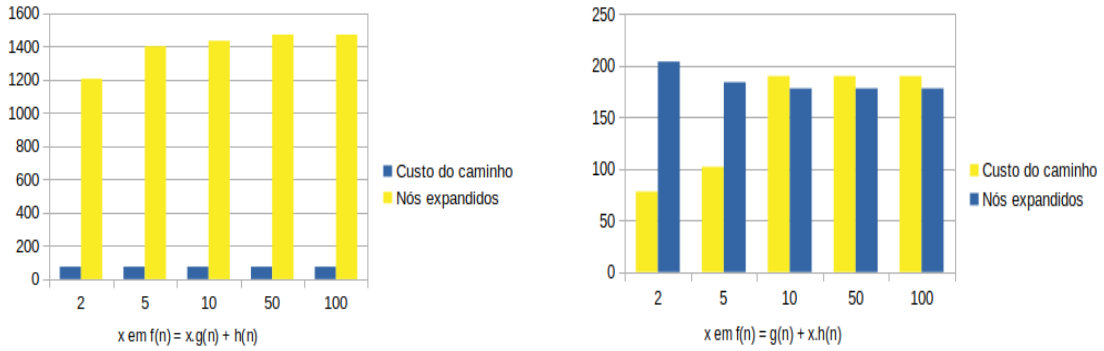
(c) Dijkstra (d) Busca Gulosa

Figura 3. Execuções das buscas A*, Dijkstra e Gulosa em um mesmo cenário. Fonte: as autoras, 2019.

4. Análise dos Resultados Obtidos

Analisando os resultados gerados, pode-se confirmar a otimalidade e completude do algoritmo A*. Desta forma, os caminhos feitos pelo robô sempre tinha como característica ser o caminho de custo mínimo. Sobre o custo total, não é possível declarar um padrão à respeito destes nas simulações feitas, dispondo ou não de obstáculos. Isto está atrelado ao fato de o robô andar aleatoriamente, logo, o mesmo pode ficar se movimentando durante muito tempo até encontrar uma ferramenta necessária no seu raio de visão, aumentando o custo total.

Considerando o cenário de uso onde utilizou-se os algoritmos de Busca Gulosa e Dijsktra para resolução do problema, nota-se que a Busca Gulosa não encontrou o caminho ótimo, apresentando um custo de caminho maior que do A* e Dijsktra. Além disso, apesar do Dijsktra ter encontrado o caminho ótimo como o A*, a quantidade de nós expandidos quando utilizado o A* é menor do que quando usado Dijsktra, resultando assim em um espaço de busca menor.



(a) Variação do peso da função g(n).

(b) Variação do peso da função h(n).

Figura 4. Execuções mantendo uma função constante e variando a outra. Fonte: as autoras, 2019.

Conforme as execuções apresentadas nas figuras 4(a) e 4(b), percebe-se que: ao aumentar o peso da função $g(n)$, o algoritmo comporta-se cada vez mais como o Dijkstra, aumentando o número de nós expandidos, mas ainda garantindo um resultado ótimo pois considera o custo de sair do nó de origem até cada um dos nós expandidos; já ao aumentar o peso da função $h(n)$, o algoritmo comporta-se cada vez mais como a Busca Gulosa, diminuindo o número de nós expandidos e aumentando o custo do caminho, de forma a não apresentar um caminho ótimo pois considera apenas a distância do nó expandido até o nó objetivo.

A partir desses resultados, entende-se a diferença entre os tipos de busca apresentados bem como suas características positivas e negativas. Além disso, observa-se a eficiência do A* no problema proposto como objetivo desse trabalho.

5. Conclusões e Trabalhos Futuros

O presente trabalho teve como objetivo o estudo e a implementação do algoritmo de busca heurístico A* para o cálculo do custo da rota de um robô que atua como manutentor de fábricas. A partir do seu desenvolvimento, foi possível realizar uma análise do comportamento do algoritmo diante de mudanças de configurações de ambiente e de variações na função de avaliação do algoritmo. Além disso, realizou-se a comparação do algoritmo A* com os algoritmos de Dijsktra e Busca Gulosa.

Conclui-se a eficácia e eficiência do algoritmo implementado, dado a comprovação das características de completude e otimalidade. Percebeu-se ainda que ao aumentar o peso da função $g(n)$ o algoritmo comporta-se de maneira semelhante ao Dijkstra, expandindo uma maior quantidade de nós e retornando o caminho ótimo. Diferentemente, ao aumentar o peso da função $h(n)$ o algoritmo comporta-se de maneira semelhante à Busca Gulosa, criando uma árvore de expansão reduzida, porém, não retornando o caminho ótimo.

Identificam-se como trabalhos futuros uma análise do impacto do comportamento do algoritmo A* quando alterado características de movimentação do robô, como somente poder se dirigir as fábricas depois de obtidas todas as ferramentas ou altera-lo para

poder se dirigir à fábrica em qualquer momento da execução, não somente depois de ter encontrado todas as ferramentas necessárias para uma determinada fábrica.

Referências

de Miranda Alves, F. R. (2007). Aplicação de buscas heurísticas ao problema de determinação de rotas para recomposição fluente de sistemas elétricos de potência. http://www.pee.ufrj.br/index.php/pt/producao-academica/teses-de-doutorado/2007/2007041901-2007041901/file. Acessado em: 06/10/2019.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. In *IEEE Transactions on Systems Science and Cybernetics*, volume 4, pages 100–107.

Nilsson, N. J. (1980). Principles of artificial intelligence. Morgan Kaufmann.

Russell, S. and Norvig, P. (2004). Artificial intelligence: A modern approach. Elsevier.

Udemy (2018). Inteligência artificial: Algoritmos inteligentes de busca. https://www.udemy.com/course/inteligencia-artificial-algoritmos-inteligentes-de-busca/. Acessado em: 06/10/2019.

Zanchin, B. C. (2018). Análise do algoritmo a* (a estrela) no planejamento de rotas de veículos autônomos. http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/10221/1/PG_COELE_2018_1_03.pdf. Acessado em: 06/10/2019.