PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Rodrigo Neumann

Proposta de sistema inteligente de definição de rotas para ambiente de armazenagem

Proposal for a smart route definition system for storage environment

Trabalho de Graduação

Departamento de Engenharia Mecânica

Programa de Graduação em Engenharia Mecânica

Rio de Janeiro, 03 de Dezembro de 2015

PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Rodrigo Neumann

Proposta de sistema inteligente de definição de rotas para ambiente de armazenagem

Proposal for a smart route definition system for storage environment

Trabalho apresentado como requisito parcial para obtenção do grau de Engenheira Mecânica Departamento de Engenharia Mecânica do Centro Técnico Científico da PUC-Rio.

Prof. Mauro Speranza

Orientador

Prof. Jorge Fontanella

Co-orientador

Departamento de Engenharia Mecânica - PUC-Rio

Prof. Marcelo, de Andrade Dreux

Coordenador do Departamento de Engenharia Mecânica - PUC-Rio

Rio de Janeiro, 03 de Dezembro de 2015

Dedicado à Thalis Gilaberte,

e a todos que foram embora cedo demais.

Agradecimentos

Primeiramente gostaria de agradecer ao meu orientador, Mauro Speranza, por acreditar no projeto, pela paciência e pela ajuda oferecida durante as inúmeras visitas a sua sala. Ao meu co-orientador, Jorge Fontanella, pelo apoio durante a elaboração deste trabalho.

Agradeço a minha família sem a qual eu não seria a pessoa que sou. Em especial a minha mãe, Lucia, pelo apoio incondicional que sempre pude contar.

Obrigado aos meus amigos, por toda a força dada ao longo da faculdade. Gabriela Martins, João Bastos, Mariana Osborne, Tamires Lucas, agradeço a todos os momentos de felicidade e frustação divididos, sem os quais o período de cinco anos que passei nesta instituição seriam com certeza menos felizes.

Um obrigado em especial a Leticia Nicolino, pelo apoio nos momentos mais difíceis, por me manter sano apesar de todos os problemas a minha volta, por me deixar feliz em momentos de tristeza, pelos tantos trabalhos que eu não faria sem sua ajuda, por todos os momentos que eu teria falhado ou desistido, obrigado.

Agradeço também a todos os meus professores, pelos ensinamentos passados, acadêmicos ou não. Estas são as pessoas que tornam a PUC no que é, sem eles seria apenas um prédio.

Resumo

Neste trabalho foram apresentados conceitos utilizados na produção de um sistema para guiar *AGVs (Automated Guided Vehicles*) para armazéns inteligentes. Foi elaborado um sistema de roteamento, utilizando o algoritmo de *pathfinding, A**, capaz definir o menor caminho em um mapa representado por um grafo. Foi realizado também o desenvolvimento de um sistema de controle de tráfego, para evitar a colisão entre veículos. Ambos os conceitos foram aplicados em uma simulação desenvolvida no programa MATLAB, onde foram executados testes para validar o trabalho desenvolvido.

Palavras-chaves: AGV, Planejamento de trajetória, Desvio de obstáculos, A*

Abstract

In this paper was presented concepts used in the development of a system to guide AGVs (Automated Guided Vehicles) for use in smart warehouses. Was elaborated a routing system, using the pathfinding algorithm A*, capable of defining the shortest path in a map represented by a graph. Also was developed a traffic control sistem, to avoid the collision between vehicles. Both concepts were implemented on a simulation developed in the program MATLAB, where testes were executed to validate the work done.

Keywords: AGV, Trajectory planning, Obstacle avoidance, A*

Sumário

Índice de Figura

1 Introdução

Na situação atual do mercado, há um grande esforço na minimização de custos e aumento de produtividade. Representando uma parte significativa dos gastos, as operações de armazenagem e distribuição são alvos comuns para projetos de melhorias. No cenário atual de rápido desenvolvimento tecnológico a automação de centros de armazenagens e distribuição de produtos é uma das possíveis melhorias.

A automação de sistemas de produção vem sendo cada vez mais presente no ambiente da manufatura. Sua presença está relacionada as vantagens que a automação traz ao ambiente de produção. O aumento de eficiência e repetibilidade, redução nos custos de operação e a minimização da exposição dos trabalhadores a condições de trabalho perigosas, evitando assim acidentes.

Uma solução que vem ganhando cada vez mais espaço, e a opção que será foco deste trabalho, é o uso de *AGVs* (*Automatic Guided Vehicle*). Este é definido como um veículo que se movimenta de forma autônoma dispensando o auxílio de operadores. Este tipo de equipamento pode ser aplicado na maioria das atividades de transporte e armazenamento de carga. Suas vantagens dentre as demais opções de automação de armazéns são sua flexibilidade, relativamente baixo custo inicial e a facilidade de adaptação de ambientes previamente não automatizados.

Na implementação de um sistema de *AGVs*, um dos fatores críticos na sua eficiência é a definição das rotas executadas. Devido a sua importância é necessário o desenvolvimento de algoritmos capazes de executar devidamente as operações, de modo efetivo e sem acidentes. Algoritmos de definição de rotas, ou *pathfinding*, são usados para garantir o menor caminho,

respeitando condições definidas pelo programador, como evitar a colisão de veículos, essencial para o funcionamento ideal do sistema.

2 Rotas

Para um uso efetivo e inteligente dos *AGVs* é indispensável um sistema capaz de definir inteligentemente o caminho a ser tomado pelo veículo. Um algoritmo de *pathfinding* para ser utilizado satisfatoriamente nesta aplicação deve ser capaz de definir o caminho mais curto, ultrapassar obstáculos e evitar colisões.

2.1 Mapeamento

Comumente o problema de *pathfinding* é modelado na estrutura de grafo. Esta abordagem simplifica o trabalho de definir a rota como um problema de menor caminho entre vértices de um grafo, trabalho que já foi amplamente estudado na literatura.

Grafo é definido como um conjunto {V,A} onde V é um conjunto arbitrário, seus elementos são chamados de vértices. A é formado por pares não ordenados de elementos de V, seus elementos são denominados arestas. Exemplificando o conceito explicado, na Figura 1 está a representação do grafo com os vértices V= {1,2,3,4,5,6,7,8,9}, e arestas A= {(1,3), (2,3), (3,4), (3,6), (5,6), (6,7), (6,9)}

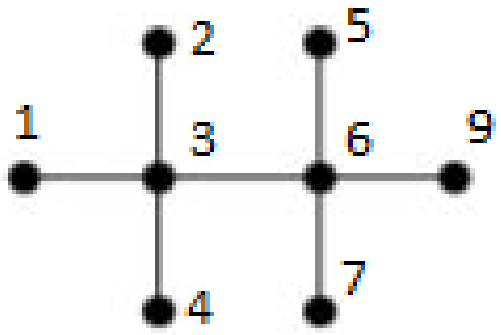

Figura 1- Exemplo de Grafo

Para utilizar-se grafos como a representação do ambiente, uma proposta comumente utilizada é definir a ambiente em um grid, como demostrado na Figura 2.

Assim pode-se converter o formato de grid para grafo localizando os vértices no centro de cada casa.

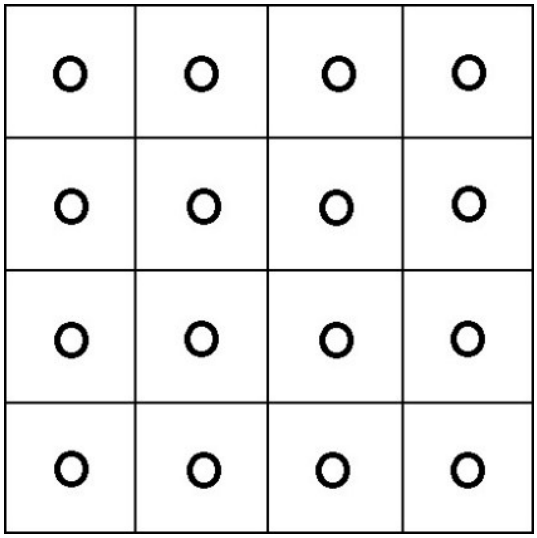

Figura 2 – Vértices do grid

Para definir as arestas deve-se primeiro definir os movimentos possíveis pelo veículo. A definição mais simples seria a possibilidade de apenas movimento nas coordenadas cartesianas, ou seja, pode-se andar para leste, oeste, norte e sul. Embora seja perfeitamente utilizável, esta representação é considerada muito distante da realidade, além de aumentar o caminho a ser tomado. Neste trabalho será permitindo movimento diagonal além dos citados acima (No

capitulo de trabalhos futuros será explorada outras opções de movimentos e suas implicações). Assim é obtido o grafo representado na Figura 3.

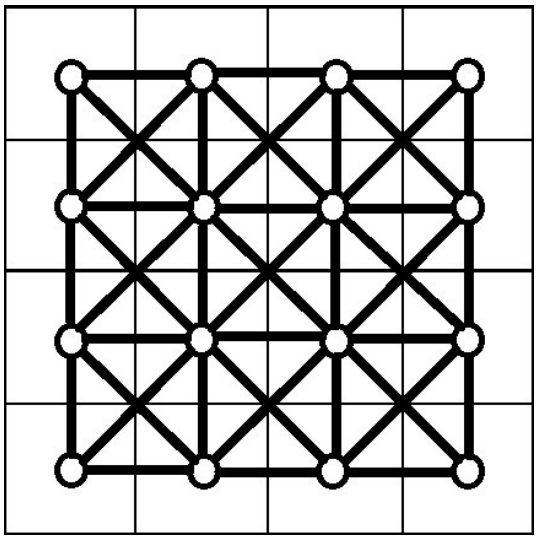

Figura 3 – Grafo do grid com vértices e arestas

A próxima etapa na produção do grafo é definir pesos para as arestas. Pesos serão utilizadas na determinação das rotas, eles representam o "custo" do veículo de andar entre vértices seguindo uma determinada aresta. Uma definição comum é usar o a distância física entre os pontos representados no grafo como o peso.

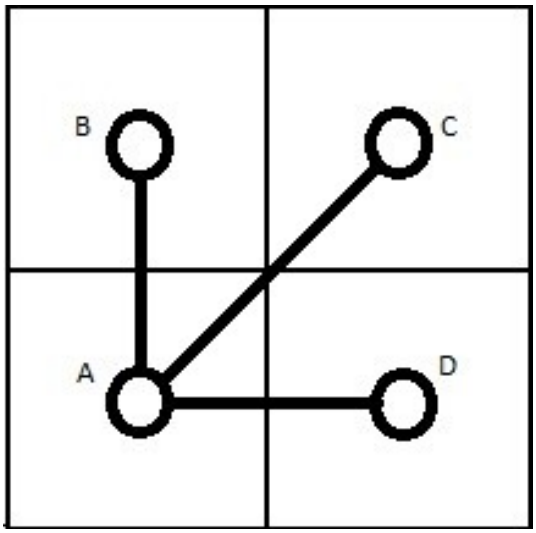

Figura 4 – Exemplo de grafo com pesos

Assim o peso associado as arestas (A,B) e (A,D) seria 1, enquanto o da aresta (A,C) seria$\sqrt{2}$, aproximadamente 1.4. Também podem ser utilizados de ouras maneiras, é possível os associar ao tempo necessário para realizar o caminho entre um vértice ao outro. Isto nos permitiria usar pesos maiores em áreas que requerem transito em velocidade reduzida.

2.2 Obstáculos

Tão importante como definir o mapa é definir os obstáculos. Obstáculos são parte da complexidade no problema de definição de rotas, sem sua existência este seria uma tarefa trivial. Comumente ambientes de armazenagem contem obstáculos que dificultam a locomoção, como: colunas, paredes, estantes, maquinário, etc. Algumas estratégias podem ser utilizadas na representação de obstáculos nos grafos. Pode-se remover vértices, remover arestas ou usar pesos punitivos.

A primeira estratégia é a remoção de vértices, esta é a mais direta e simples das estratégias para representar um obstáculo. Exemplificando este método será representado o mapa

no grid na Figura 5, considerando os círculos azuis como regiões que não podem ser ocupadas. Simplesmente deve-se retirar do grafo os vértices e as arestas associadas.

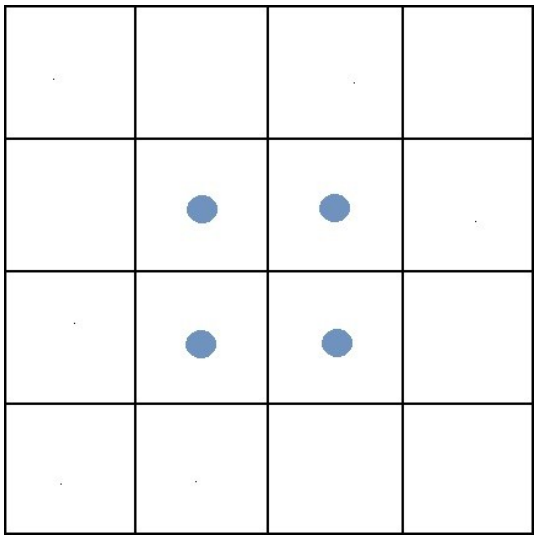

Figura 5 – Grid com obstáculo quadrado no centro

Para se obter o grafo representativo da Figura 5 simplesmente deve-se retirar do grafo os vértices e as arestas associadas aos obstáculos. Pode-se observar na Figura 6 que o grafo não permite ocupar ou passar pelas regiões marcadas como obstáculos. Esta representação é muito útil para representar obstáculos, como colunas, estantes ou maquinários fixos.

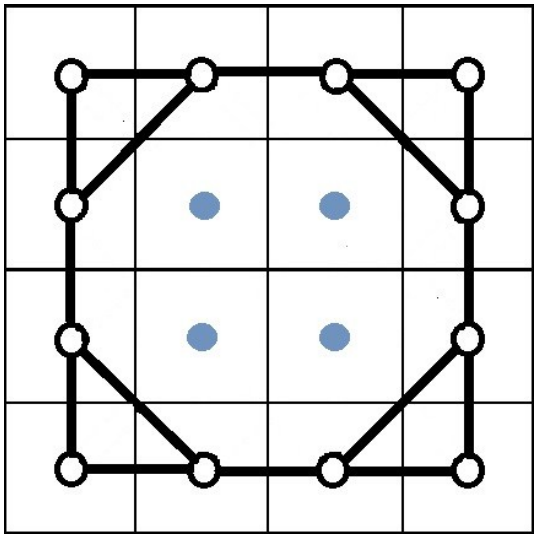

Figura 6 – Grafo do grid com obstáculo quadrado no centro

A segunda opção para a representação de obstáculos é eliminar as arestas representando o caminho bloqueado. Considerando a linha azul na Figura 7 como um obstáculo, para construir o grafo representando o mapa deve-se retirar as arestas que o obstáculo obstruiu.

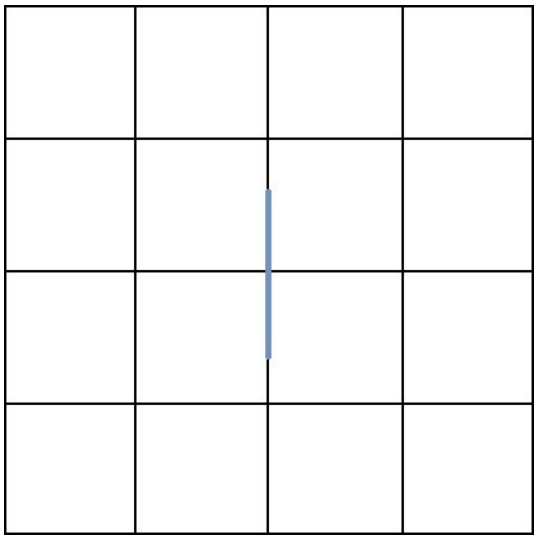

Figura 7 – Grid com obstáculo em forma de linha no centro

Analisando o grafo resultante, na Figura 8, percebe-se que as arestas que representavam caminhos passando pelo obstáculo foram retiradas. É perceptível também, que de modo muito simples é possível extrapolar esse tipo de obstáculos para o primeiro tipo exemplificado, bloqueando todas as arestas para um determinado vértice. Este tipo de representação é útil na representação de paredes, além de apresentarem a flexibilidade de servir a mesma função da remoção de vértices.

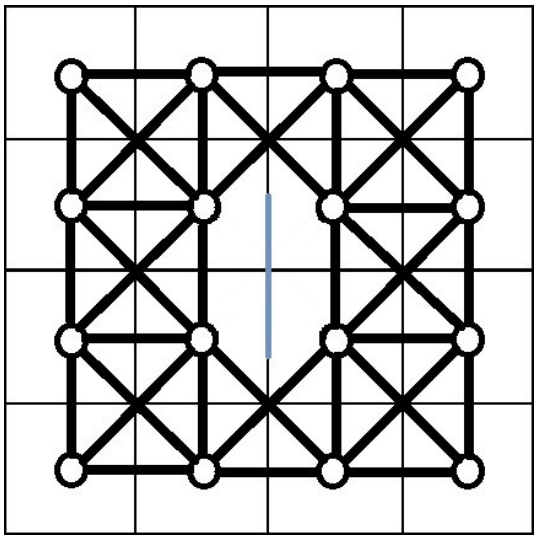

Figura 8 – Grafo do grid com obstáculo em forma de linha no centro

A última estratégia a ser apresentada é o uso dos pesos para representar obstáculos. Esta é a representação mais versátil das três. Usar pesos como maneira de punir certos movimentos apresenta uma opção de grande flexibilidade. Para representar obstáculos que obstruem arestas, deve-se simplesmente punindo as arestas com o uso de um peso infinito.

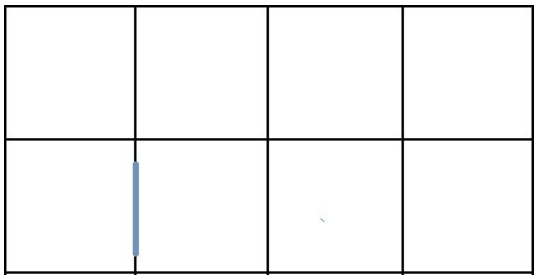

Figura 9 - Grid com obstáculo em forma de linha para pesos punitivos

Pode-se observar que o grafo, representado na Figura 10, é idêntico ao qual seria obtido se não houvesse um obstáculo. Porem como o peso, ou seja, o custo do movimento pela aresta (5,6) seria infinito, o torna assim a opção inviável como caminho. Sendo possível utiliza-lo para representar um obstáculo de remoção de aresta, pode-se deduzir que o processo pode ser extrapolado para representar um obstáculo de remoção de vértice.

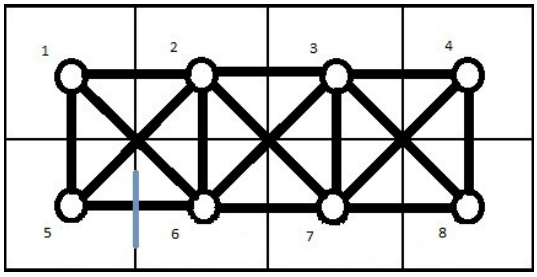

Figura 10 – Grafo do grid com obstáculo em forma de linha para pesos punitivos

Esta estratégia é muito útil em definir áreas restritas como regiões de produção ou de onde funcionários podem estar presentes ou áreas prioritárias como vias expressas para a movimentação do veículo.

Por exemplo, no grafo da Figura 11 deseja-se desencorajar o movimento na região interna do retângulo verde. Assim multiplica-se o peso de cada aresta por uma constante maior que 1, por exemplo 2. Isto faria que o custo de andar nesta área seria o dobro do normal. Se o objetivo fosse encorajar a movimentação na área basta utilizar uma constante menor que 1.

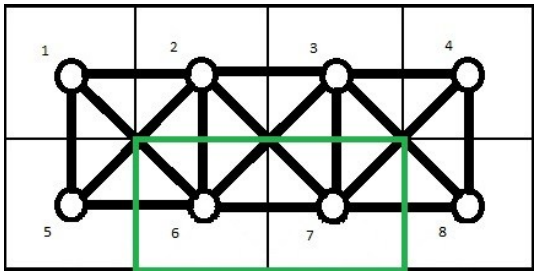

Figura 11 – Grafo do grid com região restrita

2.3 Algoritmos

A tarefa de escolher o algoritmo de *pathfinding* a ser usado na aplicação é uma tarefa complexa, com impacto direto na eficiência do sistema. É necessário balancear a velocidade no

qual o caminho é calculado e a qualidade do caminho encontrado. A variedade de opções quando se trata nos tipos de algoritmos a serem escolhidos é vasta e está fora de as ambições deste trabalho analisar todas as opções. Nesta sessão serão analisadas três opções de algoritmos: Algoritmo de Dijkstra, *Greedy Best-First-Search* e o algoritmo A*.

2.3.1 Algoritmo de Dijkstra

Em 1959 o pesquisador holandês Edsger Wybe Dijkstra, publicou o artigo "*A note on two problems in connexion with graphs*", propondo o um algoritmo para encontrar o menor caminho entre dois vértices de um grafo. Seu trabalho foi um dos trabalhos mais influentes no campo de *pathfinding* e é a inspiração para muitos dos algoritmos atuais inclusive os abordados neste trabalho.

O algoritmo é capaz de sempre encontrar um caminho de tamanho mínimo, logo é óbvio porque este algoritmo é de interesse para este trabalho. O algoritmo segue o seguinte procedimento

Primeiro é necessário definir a variável G. G é a distância entre o vértice atual e o vértice inicial seguindo um determinado caminho.

A primeira etapa é analisar todos os vértices alcançáveis a partir do ponto de partida. Registrando a distância entre o ponto de partida até o vértice em questão e marcando o vértice inicial como origem do caminho calculado. Supondo o ponto vermelho, na Figura 12, como o ponto de partida para o algoritmo e o azul com destino desejado.

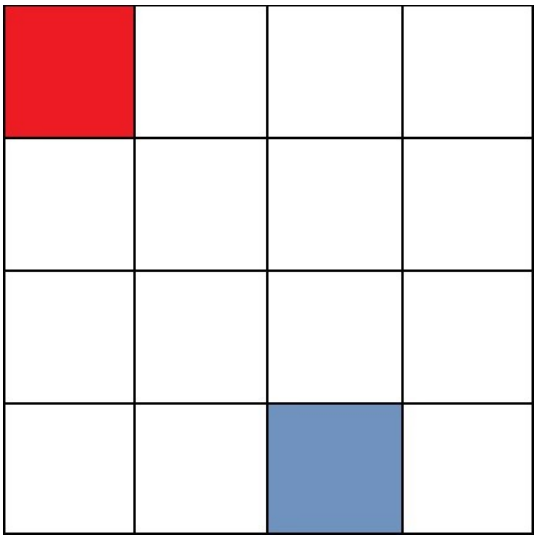

Figura 12 – Mapa representativo do ambiente

Realizando a primeira etapa, na Figura 13, registrando o custo de movimento até os vértices vizinhos.

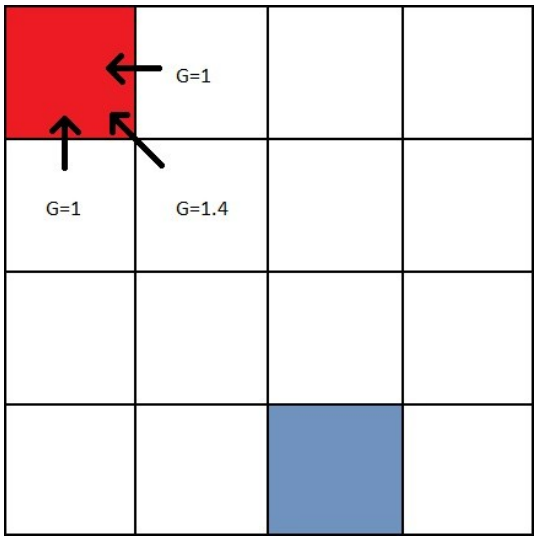

Figura 13 - Mapa representativo do ambiente após a primeira etapa do algoritmo de Dijkstra

Na próxima etapa deve-se marcar o vértice origem como um vértice visitado. Vértices visitados não serão revisitados. Além disto os vértices em volta são marcados como vértices à serem investigados. Na Figura 14 é demonstrado este passo marcando de azul os vértices visitados e de amarelo vértices a serem visitados.

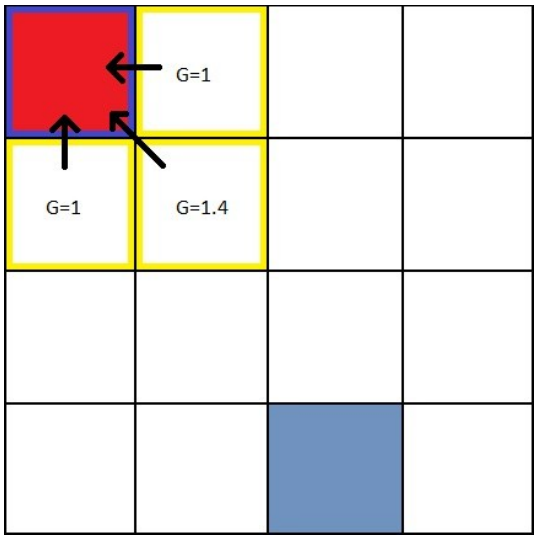

Figura 14 - Mapa representativo do ambiente após a segunda etapa do algoritmo de Dijkstra

A terceira etapa é repetir a primeira etapa em um dos vértices na lista de serem visitados, porem só será atualizado um vértice caso o valor de G tendo este vértice como origem seja menor que o anteriormente calculado. Esta etapa será demonstrada no vértice ao sul do vértice inicial, como pode-se ver na Figura 15.

É visível na Figura 15 que apenas os vértices mais ao sul foram atualizados. O vértice a leste não foi atualizado pois o valor G seria maior que o valor atual. Como G é a distância entre vértice inicial e o atual, pode-se definir G de um ponto como o G do seu vértice de origem mais o custo de movimento entre eles.

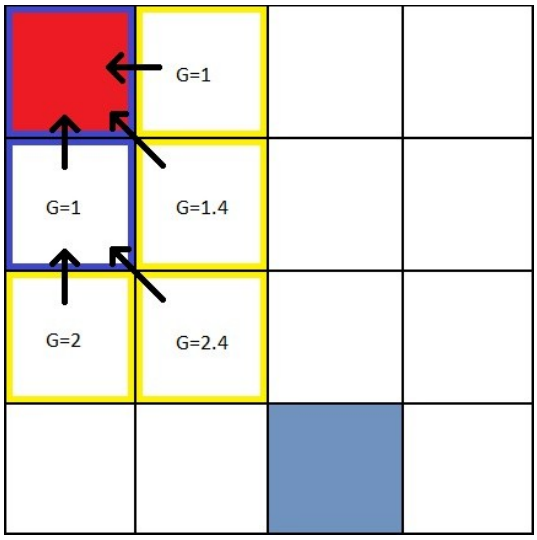

Figura 15 - Mapa representativo do ambiente no início da primeira interação na terceira etapa do algoritmo de Dijkstra

Eventualmente todos os vértices originalmente na lista de vértices a serem visitados serão marcados como visitados e novos vértices deverão ser visitados.

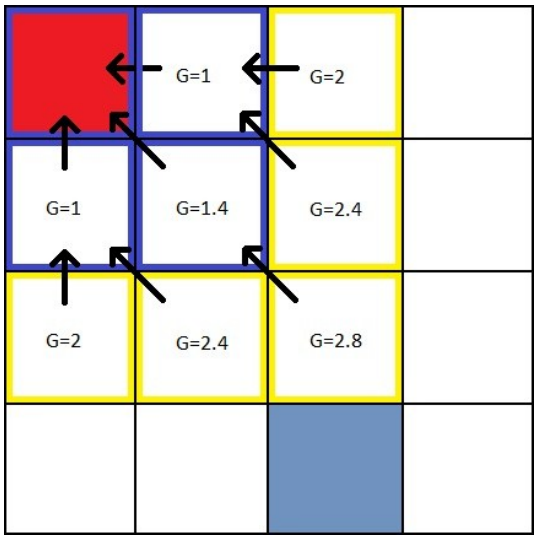

Figura 16 - Mapa representativo do ambiente no final da primeira interação na terceira etapa do algoritmo de Dijkstra

Este processo deve ser repetido até que o vértice destino seja visitado ou não haja mais vértices a visitar, o que significaria que não há nenhum caminho possível.

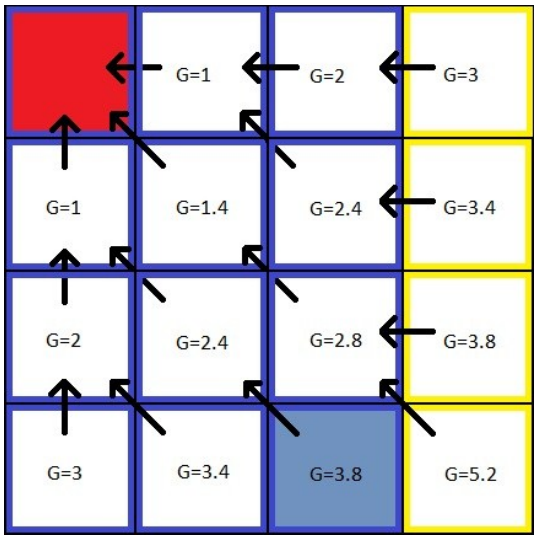

Figura 17 - Mapa representativo do ambiente na ultima interação na terceira etapa do algoritmo de Dijkstra

Tendo visitado o ponto de destino, basta realizar um último passo. Para extrair o caminho mínimo basta seguir as setas, a partir do destino até o vértice inicial. Assim será encontrado um caminho de tamanho G=3.8, como apresentado na Figura 18

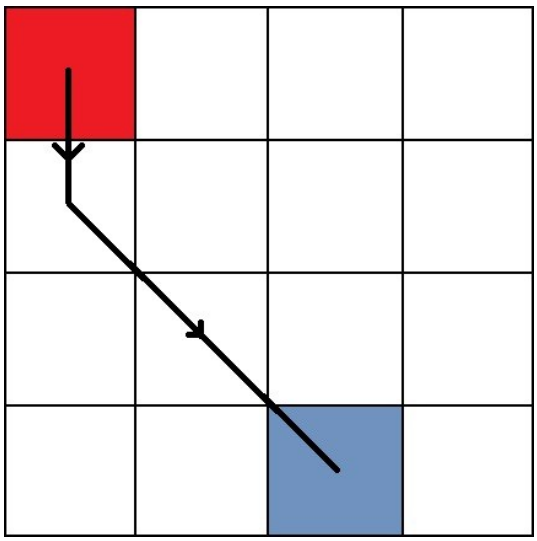

Figura 18 - Mapa representativo do ambiente na última etapa do algoritmo de Dijkstra

O algoritmo, como prometido, nos resultou no menor caminho ao destino. O algoritmo de Dijkstra é uma ótima opção para *pathfinding*, sua capacidade de sempre encontrar o menor caminho o torna muito confiável, porém não é a resposta ideal.

No processo para encontrar o caminho acima foram necessários verificar 16 vértices (12 visitados) antes de encontrar um caminho que continha apenas 4. Embora os números pareçam razoáveis deve-se ressaltar a pequena escala do exemplo apresentado acima.

Na Figura 19 está um exemplo do algoritmo em um ambiente maior. O exemplo apresentado é uma implementação do algoritmo, permitindo apenas movimento cartesiano. Os quadrados azuis representam áreas exploradas. Neste exemplo se torna gritante a necessidade de tornar a busca pelo caminho mais eficiente. O formato circular formado envolta do ponto inicial ocorre, pois, o algoritmo procura igualmente em todas as direções até encontrar o destino.

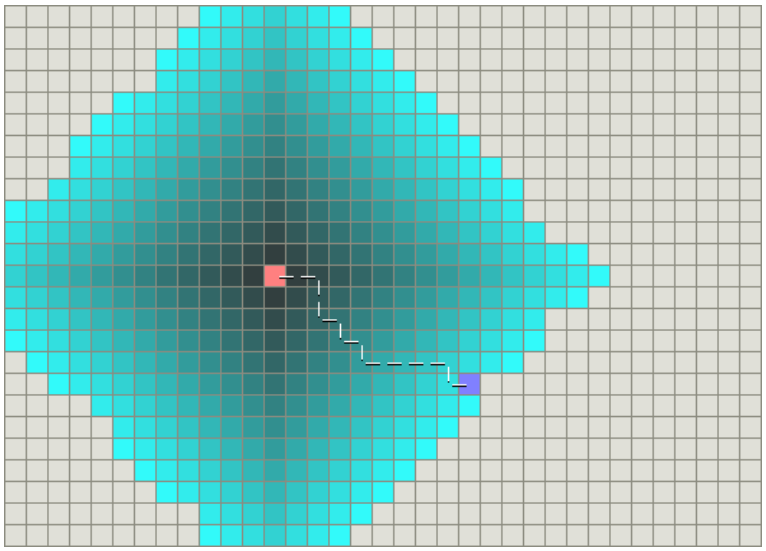

Figura 19 – Exemplo em larga escala da aplicação do algoritmo de Dijkstra em ambiente livre de obstáculos

2.3.2 Greedy Best-First-Search

Greedy Best-First-Search, ou busca puramente heurística, é uma alternativa mais rápida ao algoritmo de Dijkstra, seu processamento é acelerado devido ao seu uso de heurísticos.

Heurísticos foram definidos pelo engenheiro palestino Judea Pearl. Eles os definiu como uma estimativa de qual será o vértice mais promissor a ser seguido para completar a busca. No contexto da busca pelo caminho mínimo, esta estimativa é distância do vértice ao destino. Neste trabalho o termo heurístico será representado como a letra H.

Aplicando o algoritmo ao exemplo utilizado anteriormente. Como anteriormente o começo do algoritmo é no vértice inicial. Olhando para os vértices a sua volta são registrados o valor de H.

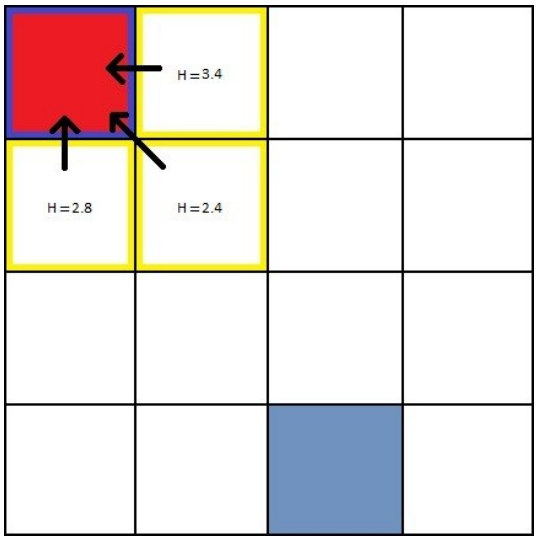

Figura 20 – Mapa representativo do ambiente após a primeira etapa do algoritmo Greedy Best-First-Search

Para dar continuidade ao algoritmo, o processo deve ser repetido no vértice que tiver o menor valor de H. Como a distância de um ponto ao destino não muda não há necessidade de atualizar o valor.

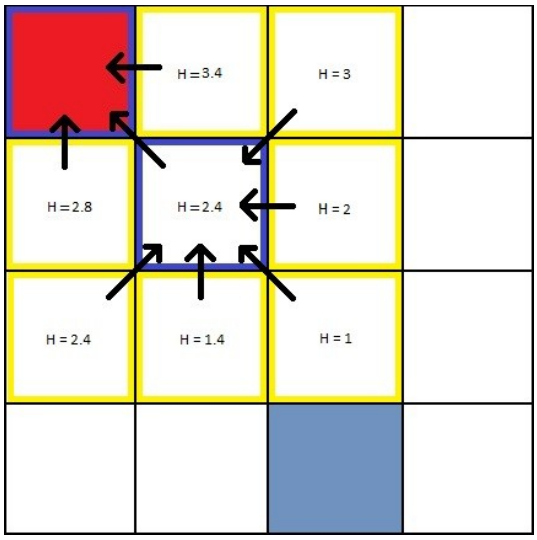

Figura 21 - Mapa representativo do ambiente após a primeira interação da segunda etapa do algoritmo Greedy Best-First-Search

Deve-se simplesmente reproduzir o este passo até atingir o vértice destino. Como o valor H não é atualizado deve-se parar assim que o termo for determinado para o vértice destino.

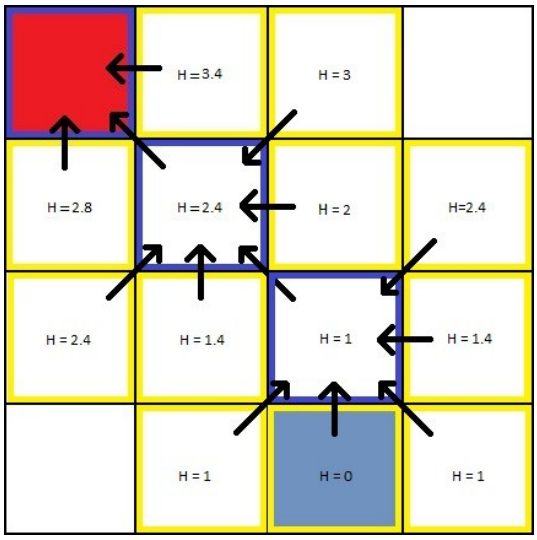

Figura 22 - Mapa representativo do ambiente após última interação da segunda etapa do algoritmo Greedy Best-First-Search

Como no algoritmo de Dijkstra o caminho a ser seguido é obtido simplesmente seguindo as setas de volta ao ponto inicial.

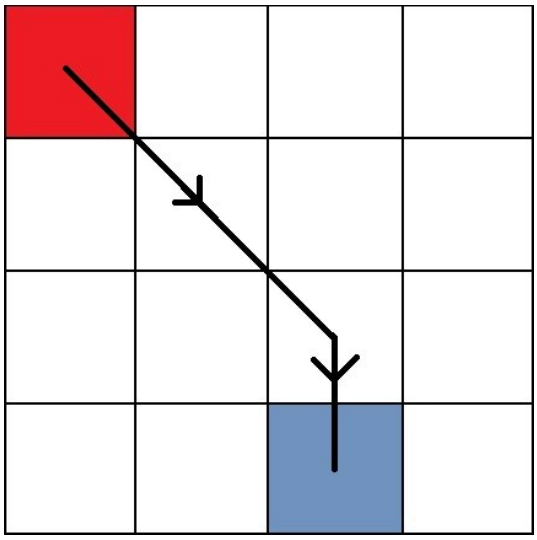

Figura 23 - Mapa representativo do ambiente após a última etapa do algoritmo Greedy Best-First-Search

É perceptível a vantagem do Greedy Best-First-Search sobre o de Dijkstra, a velocidade. Muitos dos pontos não contidos no caminho final não foram checados, isso se deve ao uso do heurístico.

Na Figura 24 o algoritmo foi aplicado no mesmo exemplo utilizado no final do estudo sobre o algoritmo de Dijkstra. Representando os vértices explorados em amarelo. Neste exemplo em maior escala, é ressaltada a vantagem em termos de velocidade. Diferentemente do algoritmo de Dijkstra, este não trata todos vértices da mesma forma, priorizando vértices mais próximos do destino foi evitado grande parte da expansão da área de busca.

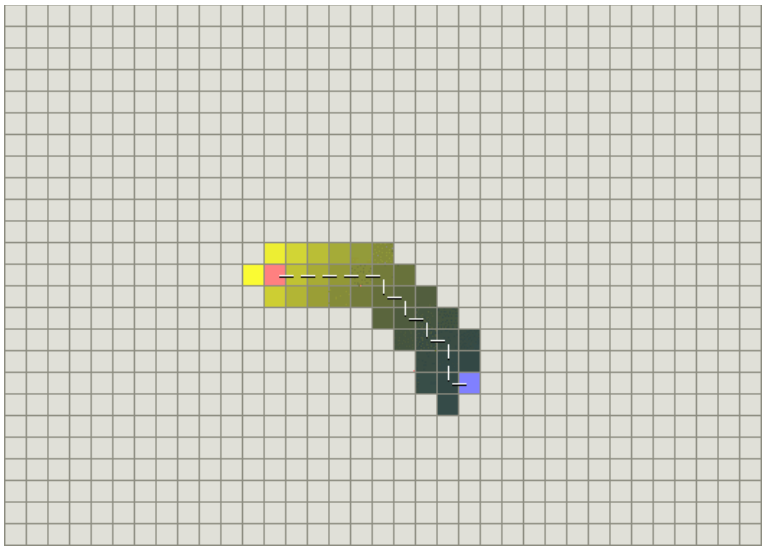

Figura 24 – Exemplo em larga escala do algoritmo Greedy Best-First-Search em ambiente livre de obstáculos

O problema com este tipo de algoritmo é que não existe garantia de se determinar o menor caminho. Por esta característica, este tipo de algoritmo é determinado como "*greedy*" ou "ganancioso". Suas decisões são baseadas no que parece melhor no momento.

Na Figura 25, está representada uma região onde há um obstáculo côncavo. Pode-se perceber a dificuldade do algoritmo. Inicialmente o caminho seguido tenta sempre se aproximar do destino, mas ao encontrar com o obstáculo é necessário expandir a área de busca, além do caminho final encontrado divergir bastante do caminho mínimo.

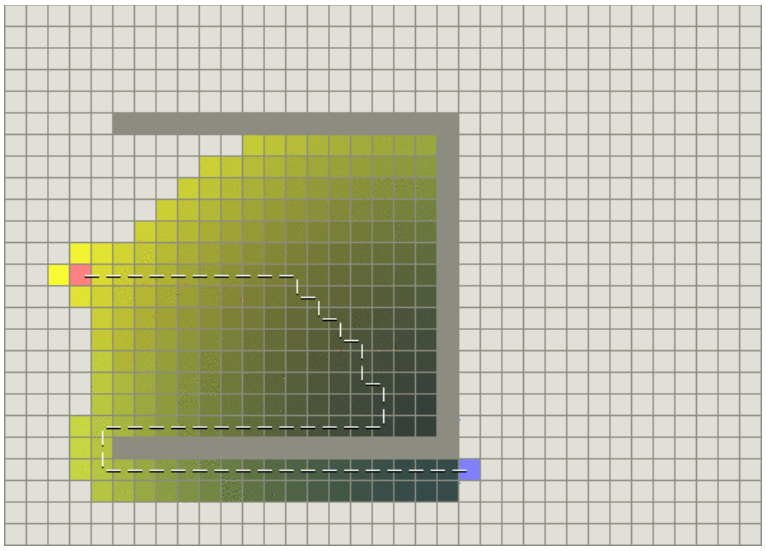

Figura 25 - Exemplo em larga escala do algoritmo Greedy Best-First-Search em ambiente com obstáculo côncavo

Na Figura 26 está representada o mesmo problema, sendo resolvida pelo algoritmo de Dijkstra. Percebe-se que a área de busca foi consideravelmente maior, mas ele foi capaz de achar o caminho mínimo.

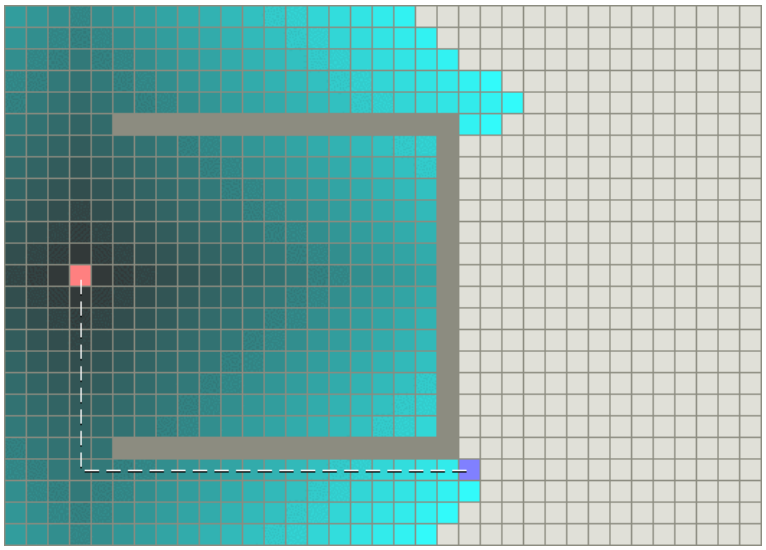

Figura 26 - Exemplo em larga escala do algoritmo de Dijkstra em ambiente com obstáculo côncavo.

2.3.3 A*

O algoritmo A* é uma das soluções mais utilizada em problemas de *pathfinding*. Devido sua velocidade, precisão, lesividade e relativa facilidade de implementação ele se tornou uma solução popular. Baseado no trabalho realizado em 1968 por Peter Hart, Nils Nilsson, e Bertram Raphael, o algoritmo A* propõe integrar heurísticos ao trabalho de Dijkstra. No seu trabalho, eles apresentaram uma variável F definida como a suma de um termo heurístico H e um custo G.

Na análise sobre *Greedy Best-First-Search* percebe-se que seu desempenho é melhor em ambientes não obstruídos. Sua performance ótima neste caso se deve ao caminho ha ser seguido tem o mesmo tamanho da estimativa representada pelo heurístico. Embora a possibilidade de utilizar o tamanho do caminho como o termo heurístico pudesse resolver os problemas da busca puramente heurística, deve-se lembrar que sem conhecer o caminho a ser seguido, descobrir o seu tamanho é uma tarefa de alta complexidade.

Por outro lado, a introdução de um termo heurístico ao algoritmo de Dijkstra reduziria a exploração de áreas menos promissoras. Procurando uma solução para o problema analisa-se a recentemente definida variável F.

$$F = G + H \tag{1}$$

Se for escolhido um valor de H que seja muito pequeno em relação a G o funcionamento será muito próximo ao de Dijkstra. Caso seja escolhido um termo heurístico de valor maior em escala que G o comportamento será "ganancioso", que não garante caminho mínimo.

A solução está em simplesmente escolher ambos termos na mesma escala. Em casos onde não há obstrução, H representará exatamente o caminho ser tomado, porem em um ambiente com a presença de obstáculos, o termo heurístico será sempre menor que o custo G do caminho completo levando à um funcionamento próximo do algoritmo de Dijkstra, garantindo

caminho mínimo. Exemplificando os conceitos apresentados acima, será demonstrado o funcionamento do algoritmo.

Exemplificando os conceitos apresentados acima, será demonstrado o funcionamento do algoritmo. Três variáveis deverão ser calculadas para cada vértice.

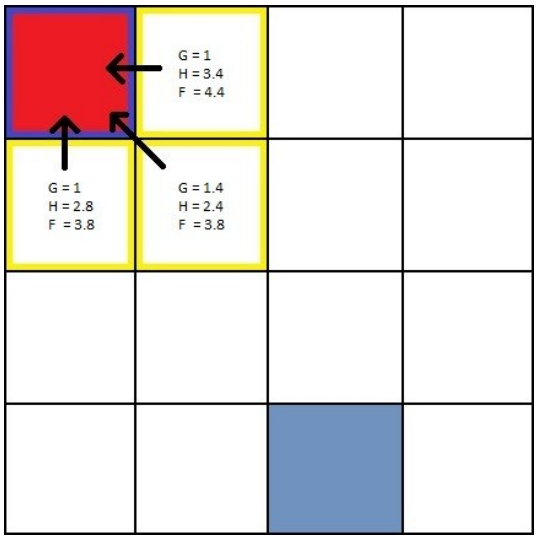

*Figura 27 – Mapa representativo do ambiente após a primeira etapa do A**

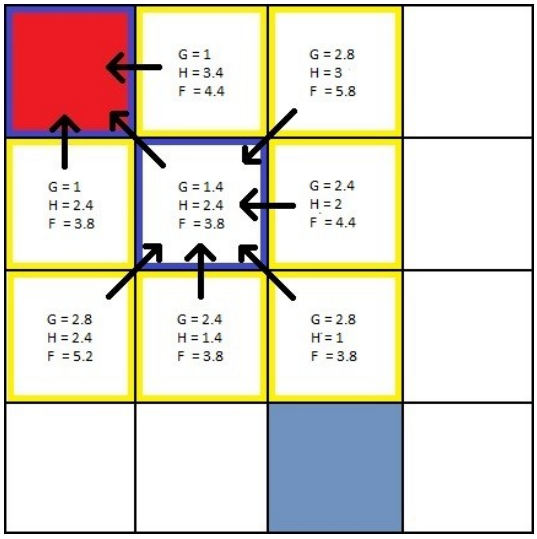

*Figura 28 - Mapa representativo do ambiente após a primeira interação da segunda etapa do A**

Seguindo a lógica apresentada passando rapidamente até o fim do algoritmo. É perceptível que o algoritmo manteve o comportamento parecido com o da busca puramente heurística, como esperado.

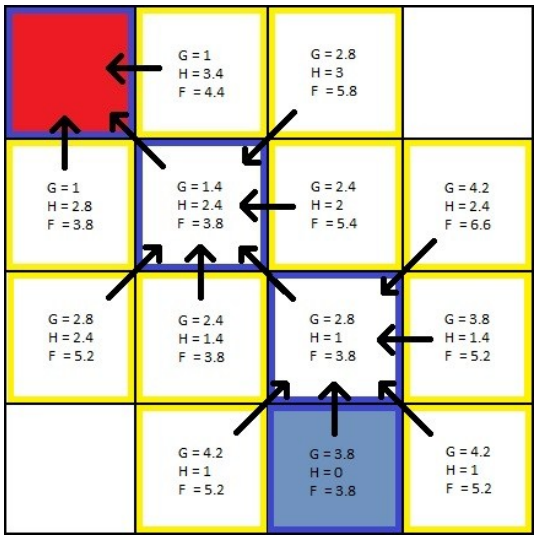

*Figura 29 - Mapa representativo do ambiente após a última interação da segunda etapa do A**

Passando o algoritmo pelos testes de grande escala realizados nos métodos anteriores. A razão da popularidade do A* se torna clara.

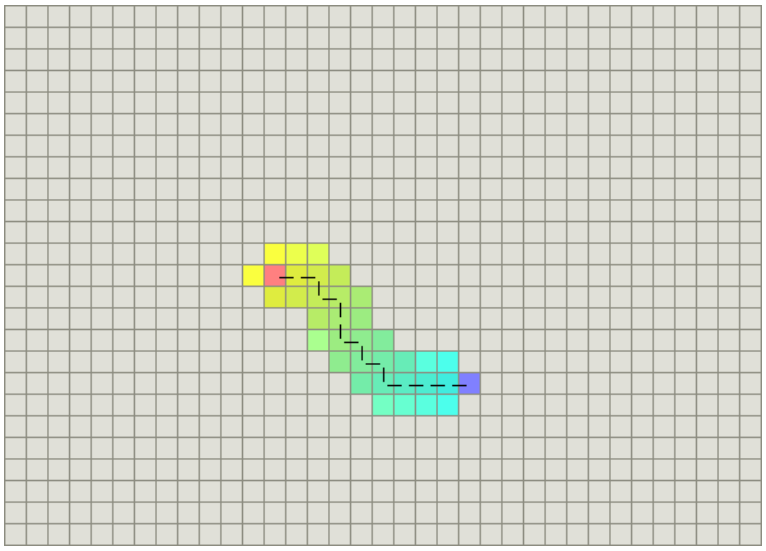

Figura 30 – Exemplo em larga escala do algoritmo A em ambiente livre de obstáculos*

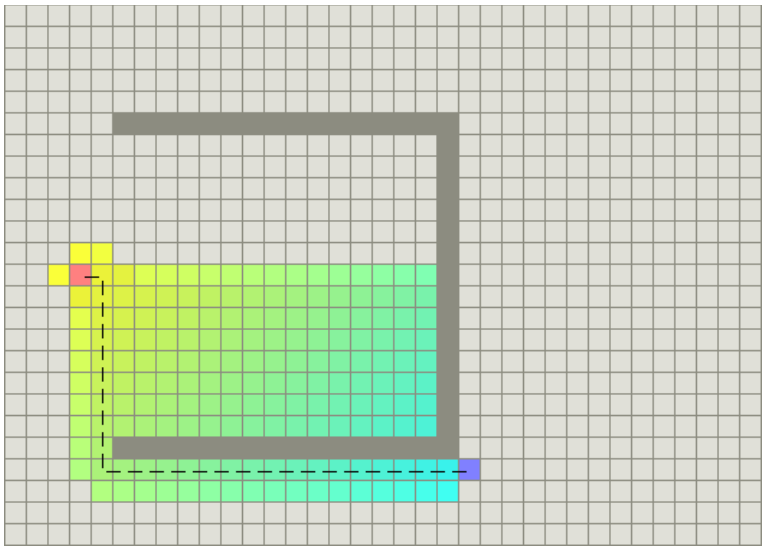

Figura 31 – Exemplo em larga escala do algoritmo A em ambiente com obstáculo côncavo*

Preservando o comportamento de alta velocidade do Greedy Best-First-Search, em conjunto com a precisão do Algoritmo de Dijkstra. O A* é uma ótima opção quando se trata na definição de rotas e será o algoritmo utilizado neste utilizado no resto deste trabalho.

3 Controle de tráfego

No contexto de sistemas de armazenamentos, raramente será utilizado apenas um veículo na realização de as tarefas de movimentação de cargas, assim torna-se de vital importância a elaboração de um sistema de controle de tráfego capaz de evitar colisões entre veículos.

O sistema proposto deve ao localizar uma colisão e definir um caminho alternativo para um dos veículos. Será demonstrado aqui um exemplo simples onde essas duas funções são executadas.

Foi definido um ambiente na forma de um corredor estreito, apenas com dois vértices de largura. Inicialmente os dois veículos se encontram em lados opostos do corredor, denominando o ponto inicial superior de A e o inferior de B.

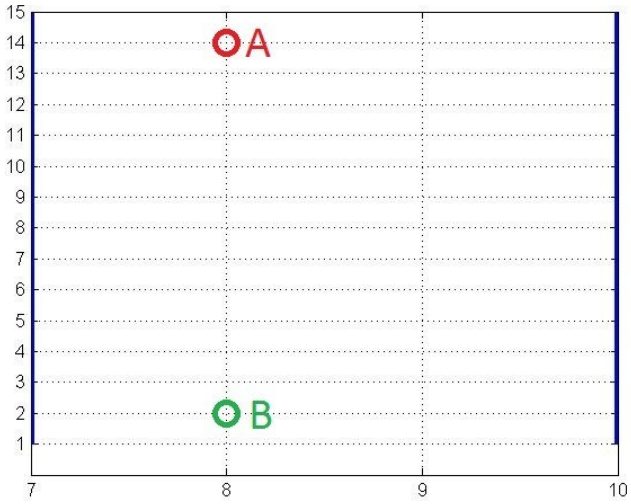

Figura 32 - Mapa representativo do ambiente em forma de corredor

Foi simulado então a troca de posição destes veículos, ou seja, o destino do veículo inicialmente em B será A e vice-e-versa. A Figura 33 demonstra em representação tridimensional, onde a coordenada adicional é o tempo, os resultados da simulação.

O caminho de B até A foi representado em vermelho, de verde está representado o caminho que seria seguido de A até B sem o algoritmo para evitar colisão e finalmente em amarelo está o que deve ser realizado para evitar colisão.

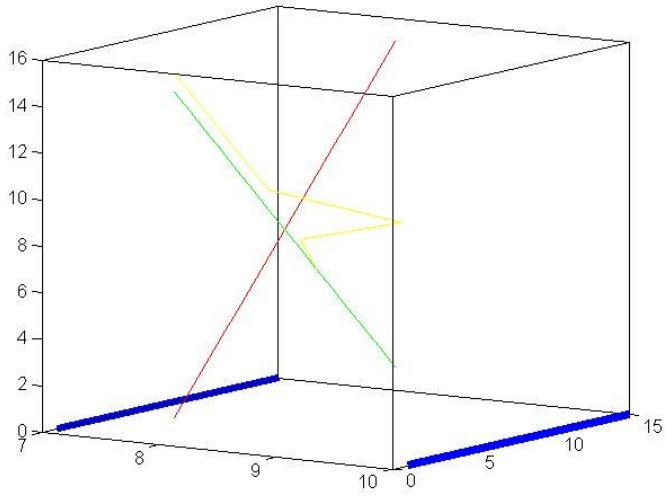

Figura 33 – Grafico tridmencional do caminho a ser realizado

Este exemplo tem a função de exemplificar a importância de levar em conta a colisão com outros veículos no cálculo de rotas. Pode-se ver na Figura 33 que a curva verde, que ignora colisão entre veículos, intercepta a vermelha. Esta negligencia na realidade poderia levar um grave acidente. Mesmo levando em conta sensores de proximidade que são usualmente instalados em *AGVs*, elaborar rotas sem colisões aumenta a eficiência e a segurança do sistema.

4 Sistema proposto

Utilizando os conceitos explorados nos capítulos 2 e 3, foi desenvolvida uma simulação no programa MATLAB. Esta simulação será utilizada para demonstrar os conceitos aprendidos em diferentes ambientes.

Nos ambientes simulados os vértices foram nomeados de forma que um vértice qualquer (a,b), em um ambiente comprimento C e largura L, tem a posição no eixo X, a, entre 1 e L e posição no eixo Y, b, entre 1 e C.

Primeiramente foi criada uma função capaz de aplicar o algoritmo A*. As entradas desta função são o mapa representativo do ambiente, a posição inicial e a posição final. Sua saída é o caminho entre o ponto inicial e final na forma de uma lista dos vértices que devem ser atravessados, caso não haja caminho a resposta será um vetor vazio, além de uma mensagem de alerta.

O primeiro teste a ser realizado será sobre a capacidade do algoritmo de encontrar o caminho mínimo até o seu destino. Para isto foi definido um mapa em formato de labirinto representado pela Figura 34. As linhas azuis representam as paredes do labirinto e os vértices estão localizados nas intersecções das linhas pontilhadas. O ponto inicial será o vértice (2,1) e o ponto final será localizado no vértice (5,1). Assim o veículo terá de atravessar todo o labirinto até para chegar ao destino. Na Figura 34 a rota a ser executada está demarcada pela linha vermelha. A rota definida é perceptivelmente correta, além de ser o caminho mínimo, assim validando este aspecto da simulação.

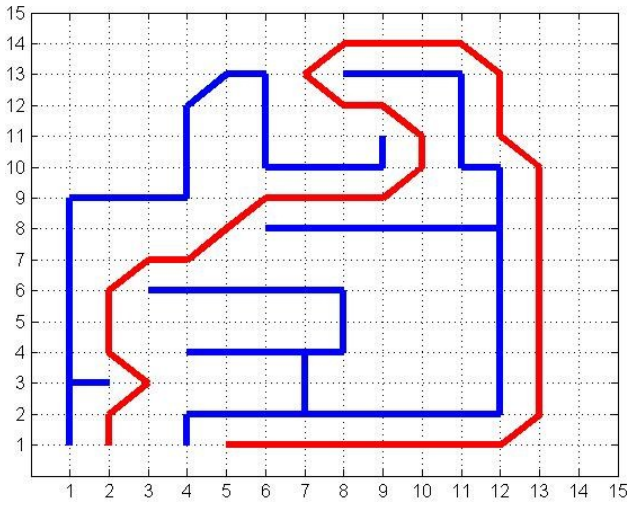

Figura 34 – Teste do sistema de roteamento da simulação

Para o sistema de controle de tráfego foi crida outra função, esta é capaz de detectar e redefinir trajetos das rotas para evitar colisões. Esta função recebe dois vetores representando rotas de dois veículos, estes vetores são compatíveis com as saídas da função que define as rotas. As saídas desta função são os caminhos atualizados sem conflitos.

O segundo teste será sobre a capacidade da função elaborada de evitar colisão entre veículos. O ambiente utilizado terá o mesmo formato que o teste anterior. Neste teste, ilustrado na Figura 35, serão utilizados dois veículos, o primeiro, representado pela linha vermelha, tem posição inicial no vértice (2,1) e o segundo, representado pela linha verde tem sua posição inicial em (7,13), a linha preta representa o desvio necessário a ser executado pelo segundo veículo para se evitar colisão.

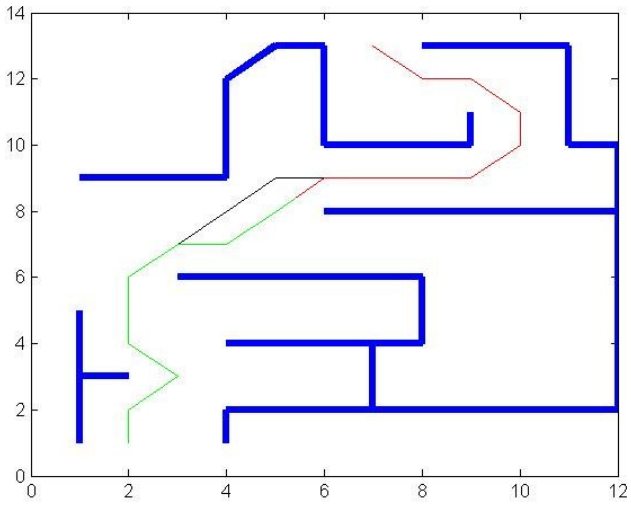

Figura 35 – Imagem 2D do teste do sistema de controle de trafego da simulação

A representação em três dimensões da rota esclarece melhor o processo realizado. Na Figura 36 pode-se ver que as linhas verde e vermelha se interceptam o que representa uma colisão, tendo detectado esta colisão o algoritmo encaminha o segundo veículo por um caminho alternativo, representado pela linha preta, que evita a colisão detectada.

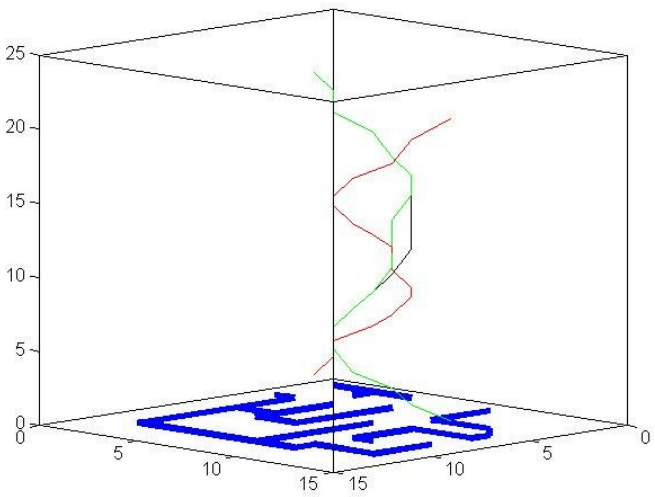

Figura 36 - Imagem 3D do teste do sistema de controle de trafego da simulação

5 Sugestões para Trabalhos Futuros

5.1 Movimentos possíveis.

No capítulo 2, foi escolhido trabalhar com a possibilidade de movimentos cartesianos e diagonais, foi argumentado que o movimento cartesiano não foi considerado próximo o bastante da realidade, porém não foi estudado a possibilidade de abranger uma gama maior de movimentos. É exatamente este tópico que será abordado nesta sessão.

Primeiro deve-se esclarecer a consequência na escolha de cada movimento, olhando para a Figura 37, vemos a esquerda a possibilidade de movimentos de um veículo usando o sistema cartesiano e a direita vemos a capacidade de movimentação de um veículo com possibilidade de movimento diagonal. A diferença que se deseja ressaltar é o ângulo dos possíveis movimentos, na primeira opção a apenas quatro possibilidades ou é mantido um ângulo de zero graus, ou seja continua-se andando na mesma direção, ou será feito uma curva de 90 graus a direita ou a esquerda, ou se andará na direção oposta realizando uma volta de 180 graus. Adicionando o movimento diagonal foi criada opções extras de movimento, curvas de 45 ou 135 graus para ambos os lados.

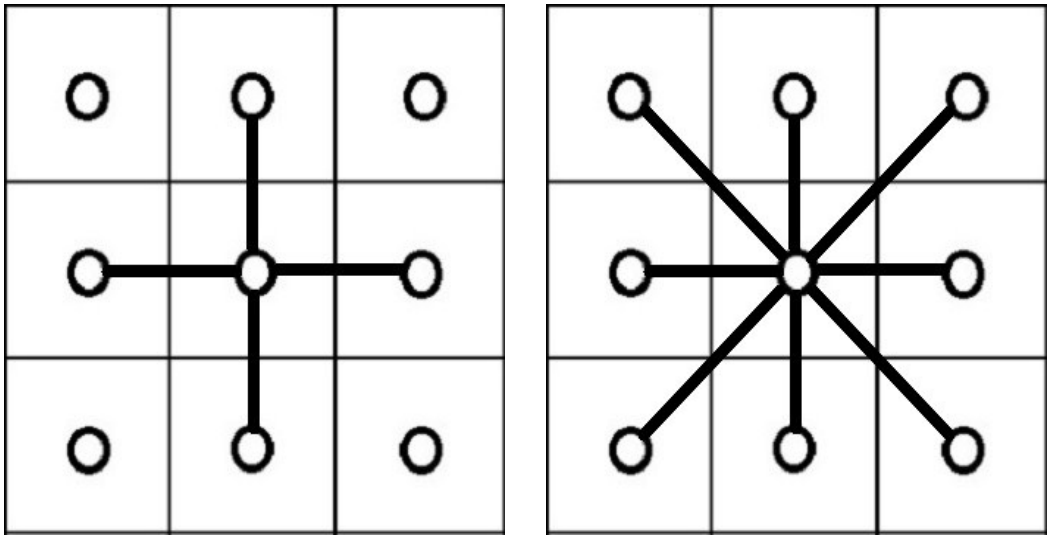

Figura 37 – Representação dos tipos de movimento. (Cartesiano a esquerda, cartesiano e diagonal a direita)

Este exemplo justifica a afirmativa encontrada no capitulo 2. Veículos reais apresentam uma grande gama de opções de movimentos possíveis, podendo andar virtualmente e qualquer direção, assim quanto mais movimentos o veículo puder realizar na simulação mais próxima a mesmas estará da realidade. Tendo realizado esta analise se torna óbvio a possibilidade de se aumentar a precisão na decisão de rotas.

Usando diferentes configurações de movimento, foram realizadas simulações preliminares, ilustradas na Figura 38, demonstrando o aumento na qualidade da rota com o aumento na possibilidade de movimentos. A contrapartida é o aumento de vértices checados por interação aumentando assim o tempo de processamento necessário. Recomenda-se para um trabalho futuro, o estudo na determinação da configuração a ser utilizada, buscando otimizar o tempo total gasto, tempo de definição de rota mais o tempo de execução.

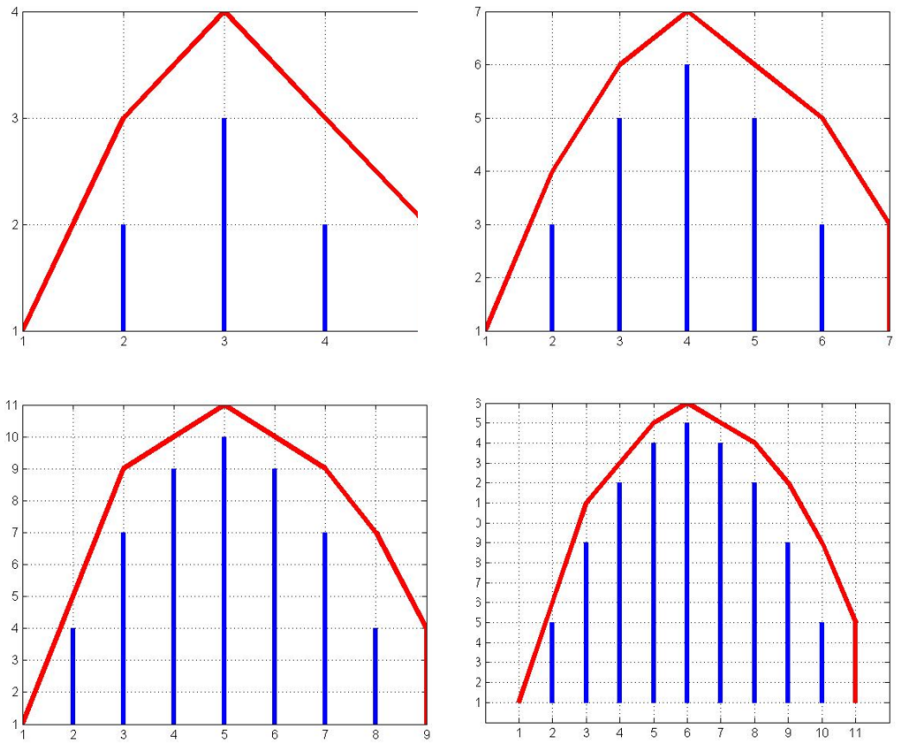

Figura 38 – Exemplos de curvas obtidas aplicando diferentes tipos de movimento

5.2 Modelagem e Controle do Veiculo

Importante aspecto no projeto de robôs é a escolha de um sistema de controle. Sistemas dinâmicos, como os veículos estudados neste trabalho, tem suas propriedades variando no tempo, no exemplo de AGV há variação de velocidade, direção, peso, etc. É necessário definir um, controlador para manter os parâmetros nos valores desejados. Recomenda-se para trabalho futuro a modelagem de veículo, e o desenvolvimento e teste de um controlador junto a simulação deste trabalho. Também é recomendado desenvolvimentos de protótipos para teste do funcionamento dos sistemas desenvolvidos

6 Conclusão

Neste trabalho foram explorados os conceitos por trás do projeto dos sistemas de roteamento e de controle de tráfego, visando a aplicação em AGVs em ambiente de armazenagem. O desenvolvimento da simulação em MATLAB comprovou a tanto a validade dos conceitos, quanto o modo qual estes foram aplicados.

Como pode-se perceber no capitulo 5 há espaço para melhora dos sistemas implementados aqui, além dos tópicos abordados poderem ser utilizados em diversas aplicações em robótica móvel. Ramo o qual há crescente interesse, devido a suas inúmeras aplicações.

Bibliografia

[1] DIJKSTRA,E. W. , “A Note on Two Problems in Connexion with Graphs”. 1959. 3f

[2] FEOFILOFF, Paulo, KOHAYAKAWA, Yoshiharu e WAKABAYASHI Yoshiko, “Uma Introdução Sucinta à Teoria dos Grafos”, 2011, disponível em: <http://www.ime.usp.br/~pf/teoriadosgrafos/>, Acesso em: 20 de outubro de 2015

[3] JOFFE, Thais, “Modelagem, simulação e controle de um triciclo omnidirecional com motores independentes, 2015, 37f, Monografia (Graduação em Engenharia de Controle e Automação) – Pontifícia Universidade Católica do Rio de Janeiro

[4] GROVER, Mikell P. , “Automação industrial e sistema de manufatura”, 3ª ed.

[5] OSBORNE, Mariana, “Concepção de um triciclo omnidirecional”, 2015, 35f, Monografia (Graduação em Engenharia Mecânica) – Pontifícia Universidade Católica do Rio de Janeiro

[6] PATEL,Amit, “Amit’s Thoughts on Pathfinding”, 2015, disponível em: <http://theory.stanford.edu/~amitp/GameProgramming/> , Acesso em: 20 de outubro de 2015

[7] RICHARDS, Hamilton, “Edsger Wybe Dijkstra”, disponível em: <http://amturing.acm.org/award_winners/dijkstra_1053701.cfm> , acesso em: 1 de novembro de 2015

[8] ROCHA, Rui Paulo, “Estado da Arte da Robótica Móvel em Portugal”, 2001, 31f, disponível em <http://ap.isr.uc.pt/archive/37.pdf>, acesso em 10 de novembro de 2015

[9] VIVALDINI, Kelen Cristiane Teixeira, “Roteamento automático de empilhadeiras robóticas em armazém inteligente”, 2010, 91f, Tese de Mestrado em Engenharia Mecânica – Escola de engenharia de São Carlos da universidade de São Paulo, São Carlos