Roteamento de Veículo Guiado Autonomamente para armazéns inteligentes

Wesley Marques Lima
Graduando em Engenharia Física
Instituto de Engenharia, Ciência e Tecnologia
Universidade Federal dos Vales do Jequitinhonha e Mucuri
Janaúba MG, Brasil
Email: wesley.marques@ufvjm.edu.br

Honovan Paz Rocha
Instituto de Engenharia, Ciência e Tecnologia
Universidade Federal dos Vales do Jequitinhonha e Mucuri
Janaúba MG, Brasil
Email: honovan.rocha@ufvjm.edu.br

***Resumo*—A área de logística, presente em grande parte da indústria, comércio e serviços, é uma das que mais necessitam de investimento em tecnologia para otimização de recursos, de maneira que o produto ou serviço final da empresa se mantenha no mercado de maneira estável e mais competitiva. Com o intuito de aperfeiçoar a logística vários trabalhos da literatura trazem estudos com Veículos Guiados Autonomamente (*Automated Guided Vehicles*-AGVs) com a intenção de tornar autônomo o transporte interno de mercadorias e consequentemente mais eficiente. Um dos desafios dos *AGVs* é a roteirização, que é a busca por caminhos mínimos entre um estado inicial e os objetivos. Com o objetivo de estudar e solucionar este problema, o presente trabalho busca modelar computacionalmente um projeto real de armazém inteligente e implementa 3 algoritmos, Busca de Custo Uniforme, A* e Colônia de formigas, que são aplicados ao modelo representado através de um grafo. Eles são implementados para várias instâncias do problema, e em seguida seus resultados são avaliados e comparados.**

Index Terms*—Busca de Custo Uniforme, A*, Colônia de formigas, Roteirização, Armazém inteligente, *AGVs

I. Introdução

Considerando-se o rápido avanço tecnológico e a constante busca por redução de custos, a necessidade de tornar os grandes centros de armazenamento e distribuição de produtos mais autônomos é cada vez maior. Embora o Brasil caminhe a passos curtos nesta direção, países da Europa, Ásia e América do Norte já possuem um alto nível de automação aliado à utilização das mais recentes tecnologias para a otimização [1].

O setor industrial no Brasil possui atualmente certa defasagem tecnológica, e devido a isso as companhias brasileiras tendem a perder competitividade frente às concorrentes estrangeiras e podem até deixar de existir em um tempo não tão distante. Em um estudo em 2020 foram realizadas entrevistas em várias organizações em que foi possível constatar que 78% delas ainda estão no estágio 1 e 2 da indústria, enquanto alguns países caminham para a quarta revolução industrial [2].

A logística na indústria abrange vários processos, como a escolha do local pra se armazenar determinado produto, o fluxo de materiais, dentre outros. A geração de um fluxo otimizado de produtos é de grande importância para que a logística seja eficiente. Desta maneira, investimentos em melhorias operacionais que visam aumentar a velocidade dos processos através de um fluxo eficiente de materiais, são cada vez mais frequentes [3].

Assim, pode-se perceber que otimizar o transporte de materiais é essencial para uma logística eficiente. O ganho em eficiência ocorre porque uma quantidade significativa de tempo é despendida com o transporte desses materiais, além disso é necessário o uso de mão-de-obra nesta tarefa, o que dificulta a utilização de alguns recursos baseados no uso de computadores, como em sistemas de monitoramento e controle [1]. Nas tarefas de transporte de materiais em ambientes como armazéns, portos e distribuidores de produtos a mão de obra é sobrecarregada com operações de carga e descarga, monitoração de recursos e planejamento de rotas de transporte. Essa sobrecarga somada ao uso de maquinário pesado nas atividades e a necessidade de concentração nas tarefas, leva os funcionários ao cansaço físico e mental, diminuindo seu desempenho e aumentando a chance de ocorrerem acidentes [4].

A automação do transporte de materiais em ambientes internos é essencial para aumentar a produtividade e reduzir custos. Por isso a utilização de Veículos Guiados Autonomamente (*Automated Guided Vehicles-AGVs*) tem sido cada vez mais frequente. Visando aprimorar sistemas que utilizam *AGVs* para o transporte de materiais, vários trabalhos tem sido desenvolvidos nessa área com o objetivo de melhorar o sistema de roteamento desses veículos. Nesses sistemas são usados diversos métodos para encontrar a melhor rota para cada *AGV*, desde algoritmos clássicos na resolução de problemas por meio de busca até meta-heurísticas inspiradas no comportamento de elementos da natureza.

Tendo em vista que automatizar o transporte de materiais é fundamental para aumentar a produção e reduzir custos, o presente trabalho tem como objetivo modelar computacionalmente um projeto real de armazém inteligente e implementar métodos da literatura para encontrar rotas mínimas para um *AGV* que atua nesse armazém, com o intuito de analisar o funcionamento desses métodos em um ambiente real.

Este trabalho é organizado da seguinte forma: na Seção II são apresentados alguns trabalhos que resolvem problemas semelhantes ao problema proposto. Na Seção III os algoritmos usados para encontrar rotas mínimas são descritos. Na seção

IV é descrito como o armazém inteligente é modelado e representado através de um grafo, mostra também como os algoritmos usados para a busca de rotas mínimas são implementados. A Seção V apresenta e discute os principais resultados obtidos. Por fim a Seção VI apresenta as considerações finais.

II. Trabalhos relacionados

O problema da roteirização de veículos em ambientes com *AGVs* é objeto de estudo em diversos trabalhos nas últimas décadas, alguns deles são brevemente descritos a seguir.

Em [5] os autores propõem a otimização de caminhos no transporte interno de ferramentas. Para resolver o problema o armazém é representado através de um grafo e os algoritmos Busca de Custo Uniforme, A*, Floyd-Warshall e Bellman-Ford são utilizados para otimização das rotas dos veículos autônomos. Os autores apontam que os métodos geram resultados similares, com o Busca de Custo Uniforme sendo ineficiente quanto ao custo computacional.

Um problema semelhante é abordado em [1], onde um armazém também é representado como um grafo e o algoritmo de Dijkstra é usado para encontrar o caminho mínimo para os veículos autônomos. Outros algoritmos como A* e Campos Potenciais são utilizados para diminuição do número de curvas do caminho encontrado. O trabalho mostrou a viabilidade dos algoritmos, que com custo computacional baixo se mostraram uma boa opção para resolução deste tipo de problema.

No trabalho [6] o problema é formulado como um sistema multiagente. Nessa abordagem o principal problema é a associação de tarefas, que foi resolvida utilizando quatro estratégias diferentes: CNET, *Fuzzy*, DynCNET e FiTA. Os autores utilizaram um Algoritmo Genético para o ajuste de parâmetros dos métodos e os resultados obtidos mostraram que o DynCNET obteve o melhor desempenho, seguido de perto pela estratégia *Fuzzy*.

Para gerar rotas livres de colisões para um veículo autônomo, em [7] os autores utilizam um algoritmo de planejamento de caminho livre de colisões, o algoritmo foi testado para vários *layouts*, pontos iniciais e objetivos. Os resultados mostram que o algoritmo obtém rotas livres de colisões em todos os casos. O trabalho considera um único robô móvel, necessitando de adaptação para ambientes com múltiplos *AGVs*.

O algoritmo A* é utilizado para determinar o caminho mínimo para dois *AGVs* no trabalho [8]. Para o tratamento de colisões, um sistema dinâmico com janelas de tempo é utilizado, assim quando houver colisão um *AGV* espera para que o outro passe, de maneira que a colisão seja evitada.

Em [9] o sistema utilizado armazena informações de varias fontes em nuvem, e com isso é possível a detecção antecipada de obstáculos. Após a detecção, uma estratégia conhecida como *Local Path Planning* é utilizada para criação de desvios para as rotas com obstáculos. Nos sistemas atuais de *AGVs* desvios de rota normalmente não são permitidos, sendo mais comum a espera pela desobstrução do caminho, entretanto, a funcionalidade *Local Path Planning* permite ao *AGV* continuar sua tarefa mesmo com obstáculos, diminuindo uma possível dependência temporal no sistema.

Para resolver o problema de roteamento de veículos, o trabalho [10] utiliza o algoritmo de Dijkstra com uma modificação que permite ao método armazenar não apenas a primeira solução ótima encontrada, mas também as demais soluções, de maneira que estas são utilizadas para resolver problemas com colisões entre os *AGVs*. Ao detectar conflitos em uma rota, é analisada a viabilidade de se usar uma das demais soluções encontradas pelo Dijkstra modificado. Caso nenhuma das rotas encontradas pelo Dijkstra modificado seja livre de conflitos, outras formas para tratar o problema são: os *AGVs* aguardam a desobstrução do caminho por um período de tempo, as rotas são refeitas considerando as mudanças no ambiente, e por fim, caso tudo tenha falhado, as tarefas são reenviadas. Esta abordagem mostrou mais eficiência no quesito tempo do que o método tradicional de tempo de espera por colisão.

No trabalho [11] o problema de otimização de rotas para veículos autônomos em armazém inteligente é resolvido utilizando algoritmos como o A* para encontrar caminhos mínimos entre os veículos e as tarefas. Devido a possíveis congestionamentos e obstáculos, essa rota é constantemente otimizada, assim as rotas são replanejadas dinamicamente de acordo com o tráfego local, condições e paradas devido a eventos inesperados. O sistema melhora a eficiência de uma frota de veículos autônomos em termos de entrega e tem custo computacional viável para aplicações no setor industrial, isso foi mostrado através de testes em ambiente real de pequena escala.

Uma estratégia de despacho dinâmico é utilizada em [12], ela se baseia em alocar tarefas aos veículos autônomos mais próximos, porém pode ser muito custoso encontrar um caminho mais curto livre de conflitos e de loops, por isso é proposta uma estratégia de roteamento que adota o algoritmo k-ésimo caminho mais curto, assim se obtendo uma rota ideal e viável computacionalmente. O trabalho mostra a criação de um sistema para encontrar rotas para veículos autônomos sem conflitos ou loops. Os algoritmos são executados em tempo polinomial e podem ser aplicados em sistemas de larga escala, os experimentos foram simulados e mostram a eficiência dos algoritmos.

No trabalho [13] o problema de possíveis conflitos é resolvido usando um método de roteamento dinâmico para controle de múltiplos veículos autônomos. Os conflitos são evitados usando espaço de estados discretos em mapa tridimensional para identificar e resolver os conflitos. O desempenho do algoritmo é avaliado comparando-o com o algoritmo BFS. O algoritmo tem baixo esforço computacional se comparado a métodos convencionais e é muito bom para ambientes dinâmicos onde a flexibilidade dos veículos autônomos tenha que ser alta. Os testes de validação foram feitos via simulação mostrando a aplicabilidade do algoritmo.

A abordagem usada em [14] é bem diferente das citadas anteriormente, pois utiliza em seu trabalho a meta-heurística colônia de formigas para resolver o problema de roteamento de veículos com janela de tempo. Para resolução do problema

o trabalho utiliza o *Multiple Ant Colony System (MACS)* que é a utilização de dois *Ant Colony System (ACS)*, onde cada um otimiza um objetivo, porém os resultados de um dos *ACS* influencia na solução do outro, assim as duas colônias trabalham de forma distinta, mas simultânea na busca de melhores soluções. O *MACS* foi desenvolvido em *Matlab* e foi aplicado na resolução de problemas em grafos com 25 e 50 nós, os resultados para esses problemas foram satisfatórios, pois as soluções foram próximas das ótimas. Para problemas com mais de 100 nós o algoritmo se mostrou ineficiente pois o tempo de execução foi multo alto, isso devido aos *scripts* criados no trabalho não terem sido otimizados no quesito performance.

Inspirado pelos trabalhos apresentados, no presente trabalho serão implementados alguns dos métodos citados acima, são eles: Busca de Custo Uniforme, A* e *ACS*.

III. Algoritmos implementados

Nesta seção detalharemos os métodos utilizados neste trabalho para resolução do problema de busca pela rota mínima para um *AGV*.

A. Busca de Custo Uniforme

A Busca de Custo Uniforme é um algoritmo de busca sem informação, ou seja, ele só tem acesso às informações contidas no grafo. A estratégia de busca adotada aqui é expandir sempre o nó com menor custo de caminho $g(n)$ na árvore de busca, por isso sua borda é uma fila de prioridades ordenada segundo o custo g.

Usando o problema de buscar o caminho mais curto entre dois vértices de um grafo como exemplo, o funcionamento do algoritmo ocorre da seguinte forma: a lista de prioridades é inicializada com o nó que corresponde ao estado inicial, caso ele não seja o objetivo é retirado da lista de prioridades, expandido e colocado na lista de explorados. Para isso todas as ações possíveis são consideradas (cada ação leva a um vértice adjacente ao que foi expandido), e todos os nós filhos exceto os presentes no conjunto de explorados são colocados na lista de prioridades de acordo com o custo do caminho até o momento. Os nós cujo estado já existe em algum outro nó da lista de prioridades, será substituído pelo nó com menor custo. O algoritmo é finalizado quando a lista de prioridades não possuir mais nenhum nó [15].

O algoritmo é completo e ótimo, ou seja, sempre que houver solução para um dado problema, a solução ótima será encontrada. Ao observar o funcionamento dessa busca é fácil perceber essa característica, pois o nó expandido é sempre o que tem caminho de menor custo até esse nó, sendo assim, quando o objetivo for encontrado o caminho que chegou a ele será o menor existente.

Outros aspectos importantes do algoritmo são as complexidades de tempo e espaço, que para o pior caso é $O(b^{1+\lfloor C^*/\epsilon \rfloor})$ para ambos, C^* representa o custo da solução ótima, ϵ é uma constante positiva que delimita o custo minimo de uma ação, e b é o fator de ramificação. Complexidade de tempo e espaço são os grandes problemas enfrentados por esse método, apesar de sempre achar a solução ótima, é inviável para problemas grandes [15].

Na seção seguinte um algoritmo um pouco mais eficiente será apresentado, ele funciona de forma muito semelhante ao Busca de Custo Uniforme, o que os difere é a função de custo.

B. A*

O A* (pronuncia-se "A estrela") é um algoritmo de busca informada, ou seja, uma informação heurística $h(n)$ é utilizada para direcionar a busca. Assim como na Busca de Custo Uniforme, no A* a borda também é uma lista de prioridades, porém a mesma é ordenada de acordo com uma função de custo $f(n) = g(n) + h(n)$.

O funcionamento do algoritmo é semelhante ao da Busca de Custo Uniforme, já que a única diferença entre eles é a função de custo que ordena a lista de prioridades.

A busca A* é completa, porém sua otimalidade depende da heurística adotada. Uma das condições para que o algoritmo seja ótimo é a heurística ser admissível, que significa que ela nunca deve superestimar o custo até o objetivo. Podemos supor um problema onde se queira encontrar o menor caminho entre dois vértices quaisquer de um grafo que represente um mapa, uma heurística admissível nesse caso seria a distância em linha reta entre os vértices e o vértice objetivo. É perceptível que $h(n)$ nesse caso não superestima o custo até o objetivo, pois a menor distância entre dois pontos é sempre uma reta.

A segunda condição é apenas para problemas de busca em grafos, e diz que a heurística deve ser consistente, essa sendo inclusive uma condição mais forte que a primeira, pois toda heurística consistente é também admissível. Ela será consistente se, o valor de $h(n)$ não for maior que $f(n')-g(n)$, sendo n' um nó gerado por n, ou seja, $h(n) \leq g(n') - g(n) + h(n')$.

As complexidades de tempo e espaço são um problema também para o A*, com o uso de uma boa heurística ele ganha eficiência com relação ao tempo se comparado com algoritmos de busca cegos. Para esse algoritmo a complexidade de espaço é o maior problema, pois geralmente atinge um limite de memória bem antes de atingir um dado limite de tempo, isso torna o algoritmo impraticável pra problemas grandes, heurísticas não consistentes podem ser utilizadas tornando o A* mais eficiente, porém não ótimo.

Em termos dos erros relativo e absoluto da heurística podemos estimar a complexidade deste algoritmo. O erro absoluto é dado por $\Delta \equiv h^* - h$, onde h^* é o custo real a partir da raiz para o objetivo, e o erro relativo é definido como $\epsilon \equiv (h^* - h)/h^*$. Para problemas simples com apenas um objetivo a complexidade de tempo do A* é exponencial no erro máximo absoluto, ou seja, $O(b^{\Delta})$, que para os custos de passo constante, pode ser escrito como $O(b^{\epsilon} d)$, onde d é a profundidade da solução [15].

Para próxima seção será apresentando um algoritmo bem diferente dos vistos até então, o Colônia de formigas, que é um algoritmo estocástico muito utilizado na resolução de problemas combinatórios.

C. Colônia de formigas

O algoritmo Colônia de formigas usado no trabalho é o *Ant Colony System (ACS)*, ele foi proposto por Dorigo e Gambardella (1997) [16], e aplicado inicialmente para a resolução do problema do caixeiro viajante (TSP), que é um problema clássico da matemática e faz parte da classe dos NP-difíceis, que são uma série de problemas que não podem ser resolvidos em tempo polinomial por algoritmos exatos.

As formigas inicialmente caminham de forma aleatória, porém nas iterações subsequentes tendem a seguir por caminhos com feromônio (o feromônio é a substância que as formigas usam para se comunicarem). Essa substância evapora a uma certa taxa, fazendo com que caminhos onde passam poucas formigas sejam cada vez menos atrativos e assim elas começam a convergir para caminhos onde passam a maior parte das formigas.

No *ACS*, as formigas ao fazerem sua trilha, podem ser mais conservadoras e seguir o melhor caminho conhecido por elas, ou mais desbravadoras se baseando em uma probabilidade que as permite encontrar novos caminhos. Para auxiliar na escolha de uma das opções um parâmetro q_0 que assume um valor entre 0 e 1 foi criado, em seguida um número aleatório q entre 0 e 1 é sorteado, caso ele seja menor que q_0 o caminho escolhido pela formiga será o melhor conhecido por ela, caso contrário ela escolherá seu caminho baseada em uma probabilidade.

Com o parâmetro q_0 podemos deixar as formigas mais desbravadoras ao diminuir o valor de q_0 ou mais conservadoras ao aumentar o valor de q_0, i.e., aumentando a diversidade no algoritmo. Portanto temos que a formiga no *ACS* decide para qual nó seguir de acordo com (1).

$$J = \begin{cases} \underset{j \in N_i^k}{\operatorname{argmax}} [\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}, \text{ se } q \leq q_0 \\ S, \text{ caso contrário} \end{cases} \tag{1}$$

Onde t representa a iteração corrente, τ_{ij} é o feromônio presente na aresta ij, η_{ij} é o inverso da distância entre os pontos ij, N_i^k é um conjunto de nós adjacentes ao nó i que ainda não foram visitados pela formiga k, α e β ditam a relevância do feromônio e da distância respectivamente e S é uma variável discreta com distribuição de probabilidade dada por (2):

$$P_{ij}^k(t) = \frac{[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{il}(t)] \cdot [\eta_{il}]^{\beta}}, \ \forall \ j \in N_i^k \tag{2}$$

Em que P_{ij}^k é a probabilidade da formiga k percorrer a aresta ij. Após a formiga trilhar seu caminho até o objetivo ocorre a atualização do feromônio nas rotas, no *ACS* acontece uma atualização local e uma global. Na local toda vez que uma formiga passa por um arco o valor de seu feromônio é alterado de acordo com (3).

$$\tau_{ij}(t) \longleftarrow (1 - \rho)\tau_{ij}(t) + \rho\tau_0 \tag{3}$$

Em que τ_0 é o feromônio inicial nas rotas e ρ é o coeficiente de evaporação do feromônio. O objetivo dessa atualização é diminuir a influência de rotas iniciais tornando as formigas capazes de fugir de caminhos não ótimos. Já a atualização global do feromônio nas rotas ocorre de acordo com (4).

$$\tau_{ij}(t) \longleftarrow (1 - \rho)\tau_{ij}(t - 1) + \rho\Delta\tau(t) \tag{4}$$

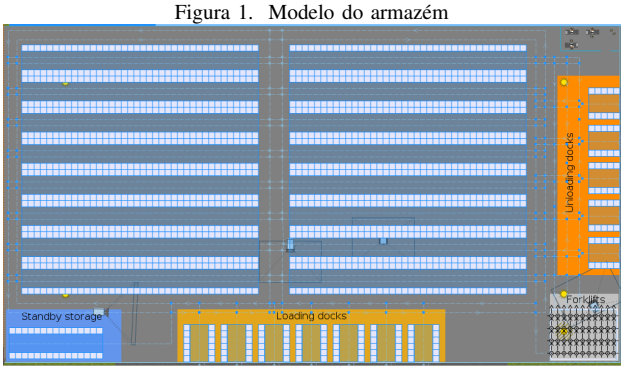
A atualização aqui ocorre em todos os arcos do grafo, porém apenas as arestas que compõem a melhor solução encontrada pelas formigas recebem feromônio, as demais sofrem apenas evaporação, i.e., $\Delta\tau(t)$ é zero para arestas que não fazem parte da melhor solução. A quantidade de feromônio que será depositado depende da qualidade da solução, pois $\Delta\tau(t)$ é o inverso da distância do melhor caminho encontrado na iteração t. O algoritmo foi implementado com uma adaptação, ela é descrita na Seção IV.

IV. Metodologia

Nesta seção é descrito como o trabalho foi desenvolvido, desde a modelagem de um projeto de armazém até a aplicação de algoritmos para encontrar rotas mínimas.

Com o objetivo de simular um armazém de médio porte, foi utilizado o *software AnyLogic*, que possibilita uma grande gama de aplicações, sendo uma delas a modelagem de ambientes reais. Uma versão gratuita do *software* foi utilizada para projetar graficamente o armazém inteligente que servirá de base para o trabalho, ele está representado na Figura 1. Um material didático disponível em [17] foi utilizado como guia no uso do *software*.

Figura 1. Modelo do armazém



Fonte: Autor.

As informações do modelo desenvolvido ficam em um arquivo XML gerado pelo próprio *software*. Essas informações foram usadas para representar o modelo na forma de um grafo, nele os vértices irão representar os compartimentos das prateleiras e as arestas representarão as ligações entre esses compartimentos.

A. Geração do grafo

O grafo para representação do armazém foi gerado utilizando as informações do modelo contidas no XML. Para acessar o xml foi desenvolvido um algoritmo para realização do *parser*, que além de verificar se o documento obedece à sintaxe do XML, identifica se ele está de acordo com as regras definidas por um vocabulário definido pelo software

gerador. O padrão para manipulação de documentos DOM (*Document Object Model*) foi utilizado. O DOM é baseado no modelo *tree-based*, ou seja, oferece uma visão de documento estruturado em árvore.

O *parser* desenvolvido, descrito pelo Algoritmo 1, carrega todo o documento para a memória e tem à disposição uma visão de todos os objetos na forma de uma árvore, que é percorrida para extração dos dados de interesse para representação do armazém.

Algoritmo 1 Parser XML

Input: Raiz do XML, TAG do objetivo, Informações
Output: Informações do armazém

```
1: if Raiz do xml é não vazio then
2:   if Informações é não vazio then
3:     return Informações
4:   end if
5:   foreach Raiz do
6:     if Informações é vazio then
7:       if filho da Raiz = TAG do objetivo then
8:         Informações ⟵ conteúdo de filho da Raiz
9:       end if
10:      Informações ⟵ Pega informação xml(filho da
         raiz,TAG do objetivo, Informações)
11:    else
12:      Sair do loop
13:    end if
14:  end for
15: end if
16: return Informações
```

Após a realização do *Parser*, o Algoritmo 2 foi implementado para geração do grafo. As informações extraídas são: posição de início de cada prateleira do armazém, como ela é dividida e a dimensão de cada divisória da mesma. Com acesso a essas informações são estipuladas as posições de cada divisória, e um vértice é criado para cada uma delas, tendo assim uma lista de vértices. Em seguida é criada uma lista de arestas, que guarda as adjacências existentes no grafo. Por fim essas informações são armazenadas em uma estrutura do tipo dicionário, finalizando a representação do grafo com implementação em linguagem *Python* versão 3.7.

Na Figura 2 é representado o grafo gerado, este grafo contém 1380 vértices. Ao compararmos com a Figura 1, pode-se perceber que a Figura 2 está invertida, o que ocorre devido à representação do armazém no *Anylogic* representar o eixo y com seu negativo orientado para cima. O grafo criado será utilizado como base para a resolução do problema proposto.

B. Busca pelos caminhos mínimos

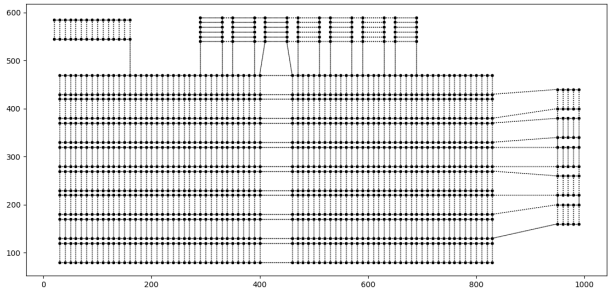
Os algoritmos Busca de Custo Uniforme, A* e *ACS* foram implementados para resolver o problema de busca proposto. Inicialmente foram implementados os 2 algoritmos clássicos na resolução de problemas de busca em grafo e na sequência o algoritmo de inteligência de enxame. Os pseudocódigos desses

Algoritmo 2 Geração do grafo

Input: Informações do armazém
Output: Grafo do armazém

```
1: foreach prateleiras do
2:   crie um vértice para cada divisória da prateleira
3:   adicione o vértice em Lista de vértices
4: end for
5: foreach Lista de Vértices do
6:   crie uma lista de adjacentes para o vértice
7:   adicione à lista de adjacentes os vértices adjacentes a
     vértice
8:   adicione à lista de adjacentes as distâncias entre vértice
     e vértices adjacentes
9:   adicione lista de adjacentes a Lista de arestas
10: end for
11: Grafo dicionário ⟵ Cria dicionário(Lista de
    vértices,Lista de arestas)
12: return Grafo dicionário
```

Figura 2. Ilustração do grafo gerado



algoritmos não são apresentados por serem bem conhecidos da literatura.

Os algoritmos, Busca de Custo Uniforme e A* foram desenvolvidos exatamente como descritos na Seção III, sendo a heurística usada no A* a distância euclidiana entre os vértices do grafo e o vértice objetivo. Para o *ACS* uma alteração foi feita, o η que tradicionalmente é o inverso do tamanho de um arco do grafo, foi mudado para o inverso da distância do vértice ao objetivo, a métrica usada foi a distância de Manhattan. A alteração tem o intuito de aumentar a probabilidade de a formiga ir no sentido do objetivo, fazendo também com que o método chegue à convergência com maior rapidez.

Os parâmetros utilizados no *ACS* foram obtidos através de uma busca em *grid*, onde os selecionados foram: $\alpha = 1,4$, $\beta = 2$, $\tau_0 = 0,001$, $\rho = 0,01$, $Q = 20$, $q_0 = 0,3$ e quantidade de formigas = 50. Os critérios de parada foram, convergência da população ou 500 iterações.

Os 3 algoritmos foram executados sobre o grafo para 10 instâncias diferentes do problema, ou seja, para 10 pares de origem e destino diferentes. Para as 10 instâncias foram analisadas as qualidades das soluções encontradas por cada algoritmo, bem como o tempo gasto para chegar a essas soluções. Para o *ACS* cada instância foi executada 10 vezes,

pois ele é estocástico e diferentemente dos demais pode encontrar um resultado diferente em cada execução.

C. Configurações da máquina

Todos os algoritmos foram implementados em linguagem *Python* utilizando o IDE (ambiente de desenvolvimento integrado) *Spyder* 3.7, sistema operacional Ubuntu 18.04 lts e processados em um computador com a seguintes configurações: Intel(R) Core(TM) i3-4005U CPU 1.7GHz, 4GB de RAM, 64-bits.

V. Resultados e discussão

Esta seção apresenta os resultados da resolução de várias instâncias do problema de busca através da aplicação dos algoritmos, Busca de Custo Uniforme, A* e *ACS*, além da discussão dos resultados apresentados por cada um deles.

A. Resultados

Foram realizados experimentos para simular um *AGV* com um ponto de origem e um ponto de destino para cada instância do problema, sendo que estes pontos foram gerados aleatoriamente. Os resultados obtidos consistem nas rotas encontradas em cada simulação pelos 3 métodos, bem como a qualidade desse caminho e o tempo gasto por cada um deles para resolver o problema. O algoritmo *ACS* foi executado 10 vezes para cada instância, enquanto os demais apenas uma vez, por estes se tratarem de métodos determinísticos.

As Tabelas I e II contêm os resultados obtidos ao executar os algoritmos Busca de Custo Uniforme e A*. Nelas são mostrados o tempo de execução e o tamanho da solução encontrada em cada instância. A solução encontrada é ótima, pois o Busca de Custo Uniforme e o A* com heurística adequada são ótimos, i.e., sempre encontram a solução ótima para o problema.

Tabela I
Resultados obtidos pelo Busca de Custo Uniforme

Instâncias	Tamanho(m)	Tempo(s)	N^o de nós expandidos
1	1070	$8,995 \times 10^{-2}$	1348
2	400	$3,137 \times 10^{-2}$	480
3	740	$4,894 \times 10^{-2}$	775
4	750	$9,692 \times 10^{-2}$	1204
5	650	$7,281 \times 10^{-2}$	838
6	1335	$9,818 \times 10^{-2}$	1375
7	810	$9,931 \times 10^{-2}$	1272
8	1375	$1,033 \times 10^{-1}$	1378
9	570	$1,546 \times 10^{-1}$	1167
10	620	$4,314 \times 10^{-2}$	638

Figura 3. Resultado obtido pelo Busca de Custo Uniforme para a instância 1

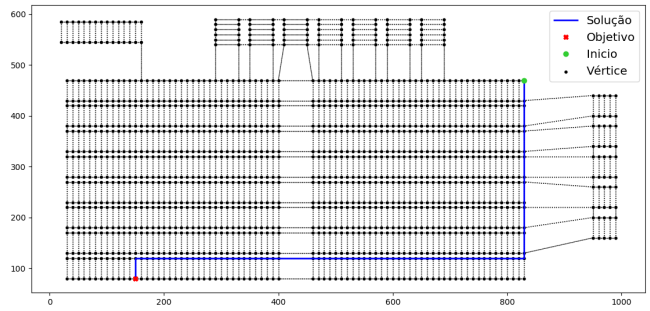


Tabela II
Resultados obtidos pelo A*

Instâncias	Tamanho(m)	Tempo(s)	N^o de nós expandidos
1	1070	$8,532 \times 10^{-2}$	1061
2	400	$3,214 \times 10^{-3}$	33
3	740	$6,039 \times 10^{-3}$	58
4	750	$2,695 \times 10^{-2}$	550
5	650	$9,682 \times 10^{-3}$	164
6	1335	$1,163 \times 10^{-1}$	1251
7	810	$2,956 \times 10^{-2}$	395
8	1375	$6,451 \times 10^{-2}$	757
9	570	$2,589 \times 10^{-2}$	328
10	620	$5,114 \times 10^{-3}$	105

As Tabelas III e IV mostram os resultados obtidos pelo algoritmo *ACS*. Na Tabela III é mostrado para cada instância, o tempo médio gasto em 10 execuções, o desvio padrão, o tempo gasto na execução mais demorada e o tempo gasto na execução mais rápida. A Tabela IV apresenta para cada instância, o tamanho médio das soluções em 10 execuções, o desvio padrão, o tamanho da pior e da melhor solução encontradas nessas execuções.

As Tabelas V e VI mostram para cada algoritmo os valores médios de tempo de execução e tamanho da solução, respectivamente. Os resultados apresentados nessas tabelas são relativos a 10 instâncias do problema, onde podemos notar que para os algoritmos determinísticos os valores médios de tamanho da solução são idênticos.

As figuras geradas na instância 1 podem ser observadas nas Figuras 3, 4 e 5. As demais figuras geradas foram omitidas, considerando-se as limitações de espaço no trabalho. Os resultados mostrados nessas imagens demonstram bem os resultados obtidos nas demais instâncias, onde podemos verificar que o Busca de Custo Uniforme e o A* encontram a solução ótima e o *ACS* converge para uma solução sub-ótima.

B. Discussão

Ao comparar os resultados apresentados nas Tabelas I, II, III e IV, podemos perceber que o Busca de Custo Uniforme e o A* foram os métodos que se saíram melhor na resolução do problema, ambos encontraram o caminho mínimo e apresentaram baixo tempo de execução. O A* foi mais eficiente no quesito tempo em 9 das 10 instâncias, e na única onde não foi a diferença foi pouco significativa, resultado esperado, pois devido a busca no A* ser direcionada por uma informação

Tabela III
RESULTADOS DE TEMPO DE EXECUÇÃO PARA O *ACS*

Instâncias	Tempo médio(s)	Desvio padrão(s)	Menor tempo(s)	Maior tempo(s)
1	$6,012 \times 10^3$	$1,184 \times 10^3$	$4,653 \times 10^3$	$7,954 \times 10^3$
2	$3,847 \times 10^2$	$2,138 \times 10^2$	$1,484 \times 10^2$	$7,843 \times 10^2$
3	$4,937 \times 10^3$	$1,533 \times 10^3$	$3,482 \times 10^3$	$8,975 \times 10^3$
4	$2,148 \times 10^3$	$6,557 \times 10^2$	$1,354 \times 10^3$	$3,568 \times 10^3$
5	$1,808 \times 10^3$	$7,693 \times 10^2$	$1,161 \times 10^3$	$3,823 \times 10^3$
6	$5,319 \times 10^3$	$8,593 \times 10^2$	$4,258 \times 10^3$	$6,514 \times 10^3$
7	$3,134 \times 10^3$	$6,709 \times 10^2$	$2,594 \times 10^3$	$4,814 \times 10^3$
8	$4,201 \times 10^3$	$1,063 \times 10^3$	$3,156 \times 10^3$	$6,631 \times 10^3$
9	$2,740 \times 10^3$	$3,388 \times 10^2$	$2,210 \times 10^3$	$3,325 \times 10^3$
10	$1,218 \times 10^3$	$3,793 \times 10^2$	$7,753 \times 10^2$	$2,091 \times 10^3$

Tabela IV
RESULTADOS DOS TAMANHOS DAS ROTAS OBTIDAS PELO *ACS*

Instâncias	Tamanho médio(m)	Desvio padrão(m)	Menor tamanho(m)	Maior tamanho(m)	Tamanho da solução ótima(m)
1	1341	46,14	1270	1410	1070
2	400	0,00	400	400	400
3	822	74,00	740	980	740
4	804	57,31	750	930	750
5	734	51,22	650	810	650
6	1773	103,71	1635	1955	1335
7	1000	76,55	890	1190	810
8	1615	77,97	1475	1735	1375
9	738	66,45	570	810	570
10	696	43,63	640	760	620

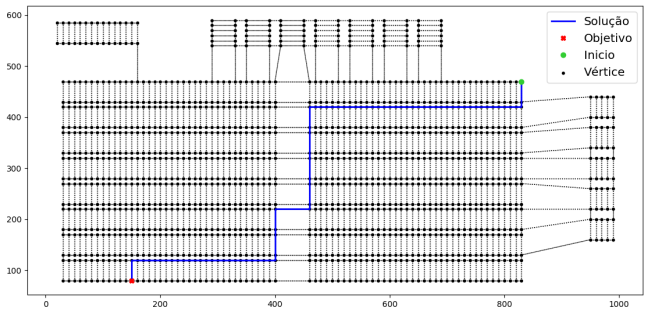
Tabela V
TEMPO MÉDIO DAS 10 INSTÂNCIAS

Métodos	Tempo médio(s)
Busca de Custo Uniforme	$8,385 \times 10^{-2}$
A*	$3,725 \times 10^{-2}$
ACS	$3,190 \times 10^3$

Tabela VI
TAMANHO MÉDIO DAS SOLUÇÕES DOS ALGORITMOS DETERMINÍSTICOS E ESTOCÁSTICOS PARA AS 10 INSTÂNCIAS

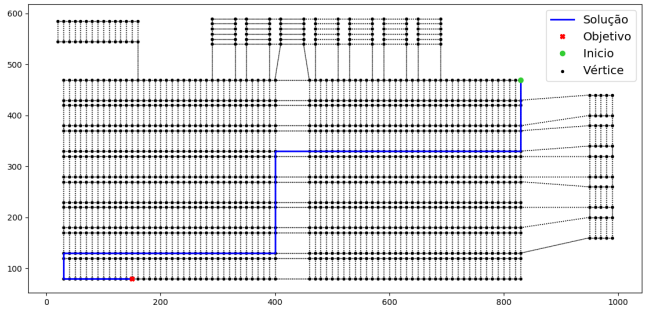
Métodos	Tamanho médio(m)
Determinísticos (Busca de Custo Uniforme e A*)	832
Estocástico (*ACS*)	992,3

Figura 4. Resultado obtido pelo A* para a instância 1



heurística sua árvore de busca acaba sendo menor que a do Busca de Custo Uniforme. Podemos ver que a árvore de busca

Figura 5. Resultado obtido pelo *ACS* na instância 1



do A* é menor ao observar nas Tabelas I e II o N^o de nós expandidos.

O que fez com que o Busca de Custo Uniforme fosse superior em uma das instâncias foram as características da mesma. Ao analisar a instância 6 observamos que o Busca de Custo Uniforme se saiu melhor, isso ocorre pois o vértice de origem não é rodeado por outros vértices, e como o Busca de Custo Uniforme expande os estados a sua volta num espécime de círculo, ele acabou expandindo menos estados do que se a origem fosse um vértice do grafo com mais vértices em volta. O A* não se beneficia dessa configuração, pois a forma como explora o ambiente é diferente.

Na Tabela V podemos ver que na média o A* foi mais eficiente que o Busca de Custo Uniforme, considerando as 10 instâncias o tempo de execução do Busca de Custo Uniforme foi mais que 2 vezes maior quando comparado ao A*. Sendo assim foi observado que o A* como algoritmo é mais eficiente

que o Busca de Custo Uniforme.

Ao comparar os resultados dos algoritmos determinísticos das Tabelas I e II, com os resultados do algoritmo estocástico mostrado nas Tabelas III e IV, podemos perceber que o mesmo não resolveu bem o problema visto que o tempo de execução foi muito superior e os caminhos encontrados muitas vezes ficaram presos em mínimos locais. Essa má eficiência em relação ao tempo pode ser atribuída a uma implementação que não foi voltada para tal. O *ACS*, devido a suas características facilita a programação paralela, logo ao implementá-lo dessa forma o tempo gasto seria reduzido em muitas vezes. Além disso mais formigas e iterações se fariam viáveis podendo melhorar e muito a qualidade das soluções.

Outro aspecto importante a se ressaltar é que devido ao tamanho do problema os métodos determinísticos se mostraram suficientemente bons, fazendo com que a utilização de um estocástico não seja tão interessante, ou seja, para esse problema os determinísticos se mostram muito superiores, porém para problemas maiores e com uma implementação mais voltada pra eficiência o *ACS* tem grande potencial, já que algoritmos determinísticos não se saem bem em problemas de maiores dimensões devido a terem tempo de execução exponencial.

VI. Conclusão

Utilizando um *software* de modelagem e simulação, o *AnyLogic*, foi possível a criação de um modelo gráfico de um armazém inteligente. Foram implementados 2 algoritmos que geram um grafo a partir do modelo gráfico, tornando possível a aplicação de algoritmos de roteirização sobre um ambiente real.

Com um estudo detalhado do problema de roteamento de veículos, foi possível a implementação de 3 algoritmos para a resolução do problema, o Busca de Custo Uniforme, o A* e o *ACS*. Todos foram capazes de resolver o problema proposto, mostrando a viabilidade da aplicação desses algoritmos para problemas reais. Dois deles apresentaram melhores resultados tanto na qualidade da solução encontrada quanto no tempo de execução, o A* e o Busca de Custo Uniforme. Ambos apresentaram tempo de execução viáveis para a resolução do problema, porém o A* foi melhor em quase todas as instâncias, se mostrando superior aos demais métodos para o problema dado.

O *ACS* apresentou tempo de execução relativamente alto e ficou preso em mínimos locais em algumas situações. Porém sua implementação pode ser melhorada, trazendo mais eficiência ao método. Um aspecto importante a se ressaltar é o tamanho do problema resolvido pelos algoritmos, por ser um problema pequeno os métodos determinísticos tiveram desempenho muito bom, porém o tempo cresce exponencialmente para eles, fazendo com que para problemas muito grandes a sua utilização seja inviável computacionalmente, por isso quanto mais o problema cresce melhor o método estocástico é em relação aos determinísticos.

Como propostas de trabalhos futuros, pode-se citar uma implementação mais eficiente do *ACS*, fazendo um código paralelizado, assim o tempo gasto para encontrar soluções seria muito otimizado, tornando possível o acréscimo de formigas e iterações, podendo assim melhorar também a qualidade da solução. Além disso poderia-se incrementar o problema tornando-o mais desafiador, fazendo com que a utilização do *ACS* seja mais interessante, isso pode ser feito por exemplo, aumentando o número de objetivos.

Outra proposta de trabalho seria um sistema com vários *AGVs* no ambiente, trazendo uma situação mais realista, onde problemas como colisões entre os *AGVs* tivessem que ser tratadas.

Agradecimentos

Os autores agradecem à UFVJM por todo o suporte prestado no desenvolvimento deste trabalho.

Referências

[1] K. C. T. Vivaldini, "Roteamento automático de empilhadeiras robóticas em armazém inteligente," Ph.D. dissertation, Universidade de São Paulo, 2010.

[2] M. V. C. d. S. Ordônio *et al.*, "Brasil e china no mundo 4.0: uma visão a partir da política industrial," 2020.

[3] M. Oda, Z. A. I. Miranda, A. Itani, E. Licco, and L. A. Kulay, "Logística sustentável: contribuição a processos de gestão," *INTERFACEHS-Revista de Saúde, Meio Ambiente e Sustentabilidade*, vol. 4, no. 1, 2010.

[4] M. Stürmer, "Sistema de controle e navegação para robôs móveis autônomos em ambientes de armazenagem," *Portuguese, Universidade do Vale do Rio dos Sinos, São Leopoldo*, 2004.

[5] I. Beker, V. Jevtić, and D. Dobrilović, "Shortest-path algorithms as a tools for inner transportation optimization," *International Journal of Industrial Engineering and Management (IJIEM)*, vol. 3, no. 1, pp. 39–45, 2012.

[6] L. B. Branisso *et al.*, "Sistema multiagente para controle de veículos autônomos," 2014.

[7] N. V. Kumar and C. S. Kumar, "Development of collision free path planning algorithm for warehouse mobile robot," *Procedia computer science*, vol. 133, pp. 456–463, 2018.

[8] N. C. Truong, T. G. Dang, and D. A. Nguyen, "Optimizing automated storage and retrieval algorithm in cold warehouse by combining dynamic routing and continuous cluster method," in *International Conference on Advanced Engineering Theory and Applications*. Springer, 2018, pp. 283–293.

[9] E. Cardarelli, V. Digani, L. Sabattini, C. Secchi, and C. Fantuzzi, "Cooperative cloud robotics architecture for the coordination of multi-agv systems in industrial warehouses," *Mechatronics*, vol. 45, pp. 1–13, 2017.

[10] Z. Zhang, Q. Guo, J. Chen, and P. Yuan, "Collision-free route planning for multiple agvs in an automated warehouse based on collision classification," *IEEE Access*, vol. 6, pp. 26 022–26 035, 2018.

[11] C. Secchi, R. Olmi, F. Rocchi, and C. Fantuzzi, "A dynamic routing strategy for the traffic control of agvs in automatic warehouses," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 3292–3297.

[12] X. Li, C. Zhang, W. Yang, and M. Qi, "Multi-agvs conflict-free routing and dynamic dispatching strategies for automated warehouses," in *International Conference on Mobile and Wireless Technology*. Springer, 2018, pp. 277–286.

[13] Y. Noh and Y. Yoon, "Tactical traffic control for multiple agv systems based on three dimensional space," in *AIP Conference Proceedings*, vol. 1790, no. 1. AIP Publishing, 2016, p. 150027.

[14] R. L. Santos, "Uma aplicação de algoritmos de colônias de formigas em problemas de roteirização de veículos com janelas de tempo," 2006.

[15] S. J. Russell and Norvig, *Inteligência Artificial*, 3rd ed. Rio de Janeiro: Elsevier, 2013, resolução de problemas por meio de busca.

[16] M. Dorigo and L. M. Gambardella, "Ant colonies for the travelling salesman problem," *biosystems*, vol. 43, no. 2, pp. 73–81, 1997.

[17] B. Andrei, "Multimethod simulation modeling for business applications," 2017. [Online]. Available: https://www.anylogic.com/