UNIVERSIDADE ESTADUAL PAULISTA “JÚLIO DE MESQUITA FILHO”
FACULDADE DE ENGENHARIA - CÂMPUS DE SÃO JOÃO DA BOA VISTA
GRADUAÇÃO EM ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

LUCAS CARDINAL DANTAS

ALGORITMO A* PARA PLANEJAMENTO DE ARQUITETURA DE REDES EM CENÁRIOS HETEROGÊNEOS

SÃO JOÃO DA BOA VISTA
2022

LUCAS CARDINAL DANTAS

ALGORITMO A* PARA PLANEJAMENTO DE ARQUITETURA DE REDES EM CENÁRIOS HETEROGÊNEOS

Trabalho de Conclusão de Curso apresentado à Universidade Estadual Paulista "Júlio de Mesquita Filho" como requisito para obtenção de título de Bacharel em Engenharia Eletrônica e de Telecomunicações

Orientador: Prof. Dr. Ivan Aritz Aldaya Garde

SÃO JOÃO DA BOA VISTA

2022

D192a

Dantas, Lucas Cardinal
Algoritmo A* para planejamento de arquitetura de redes em cenários heterogêneos / Lucas Cardinal Dantas. -- São João da Boa Vista, 2022
54 p.

Trabalho de conclusão de curso (Bacharelado - Engenharia de Telecomunicações) - Universidade Estadual Paulista (Unesp), Faculdade de Engenharia, São João da Boa Vista
Orientador: Ivan Aritz Aldaya Garde

1. Análise por agrupamento. 2. Telecomunicações. 3. Fibras ópticas. 4. Sistemas de comunicação sem fio. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Faculdade de Engenharia, São João da Boa Vista. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE ENGENHARIA - CÂMPUS DE SÃO JOÃO DA BOA VISTA
GRADUAÇÃO EM ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

TRABALHO DE CONCLUSÃO DE CURSO

ALGORITMO A* PARA PLANEJAMENTO DE ARQUITETURA DE REDES EM CENÁRIOS HETEROGÊNEOS

Aluno: Lucas Cardinal Dantas
Orientador: Prof. Dr. Ivan Aritz Aldaya Garde

Banca Examinadora:

- Ivan Aritz Aldaya Garde (Orientador)
- José Augusto de Oliveira (Examinador)
- Rafael Abrantes Penchel (Examinador)

A ata da defesa com as respectivas assinaturas dos membros encontra-se no prontuário do aluno (Expediente nº 094/2021)

São João da Boa Vista, 15 de março de 2022

Dedico este trabalho à minha família.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me guiado até este ponto e por todas as coisas boas que me proporcionou até o momento.

ao meu pai Luís por sempre ter me presenteado com livros e ter me incentivado muito a ingressar no ensino superior, à minha mãe Márcia que me ajudou de todas as formas possíveis, ao meu irmão Matheus e sua esposa Tamires que acompanharam de perto minha trajetória até aqui e também a todos os meus familiares que me incentivaram nos estudos e me apoiaram durante todos estes anos.

ao meu orientador Prof. Dr. Ivan Aritz Aldaya Garde por ser um excelente professor e pelo imenso apoio, paciência e auxílio no desenvolvimento deste trabalho e também pelos valiosos conselhos neste último ano de graduação.

aos membros da banca Prof. Dr. Rafael Abrantes Penchel e Prof. Dr. José Augusto de Oliveira por aceitarem avaliar este trabalho e por serem ótimos professores que tiveram grande participação e influência em minha formação.

a todos os professores do campus da UNESP de São João da Boa Vista pelos ensinamentos que com certeza utilizarei em minha vida profissional e pessoal nos anos vindouros.

aos meus amigos e colegas de estudos de Espírito Santo do Pinhal e de São João da Boa Vista pelos bons momentos durante estes importantes anos da minha vida e aos colegas da Cracto pelo bom convívio durante o estágio.

à UNESP e a todos os seus funcionários responsáveis pelo ambiente agradável de estudos, convivência e pelo aprendizado imensurável que me proporcionaram durante estes anos, aprendizado este que levarei para o resto de minha vida com imensa gratidão.

"Longa é a viagem rumo a si próprio, inesperada é a sua descoberta."
(Thomas Mann)

RESUMO

A crescente demanda por capacidade nos sistemas de comunicação requer uma abordagem holística que otimize o uso dos recursos disponíveis. Podemos solucionar parte dos problemas de capacidade utilizando técnicas de clusterização no planejamento da estrutura das redes (uma estrutura sem hierarquia sofre com problemas de escalabilidade e capacidade, portanto uma estrutura hierarquizada conduz melhorias na performance da rede). Então, por esta razão, usualmente são utilizadas técnicas de clusterização no projeto de redes de telecomunicações. No entanto, tipicamente o processo de clusterização não é capaz de levar em conta a presença de obstáculos presentes no cenário. Assim, neste trabalho integramos os algoritmos de A* e Lloyds como forma de produzir um novo algoritmo que tem a capacidade de efetuar o processo de clusterização em cenários que possuem obstáculos. Cabe mencionar que não encontramos anteriormente na literatura a integração de A* com K-means. Agora, por meio deste novo algoritmo é possível efetuar o processo de clusterização em cenários mais complexos (i.e., cenários com obstáculos). Nosso método proposto consiste em mudar as medidas de distância euclidiana ou distância Manhattan no algoritmo de Lloyds para as distâncias calculadas por meio de uma busca exaustiva com o algoritmo A*. Nossos resultados mostram que o algoritmo que obtivemos como produto final fez a distância média total mudar, de 4,7 para 3,0833 quando comparado ao algoritmo de Lloyds com a presença de obstáculos no cenário de clusterização para o caso específico que consideramos.

Palavras chave: Análise por agrupamento; Telecomunicações; Fibras ópticas; Sistemas de comunicação sem fio.

ABSTRACT

The growing demand for capacity in communication systems requires a holistic approach which optimizes the use of the available resources. We can solve part of the capacity problems by using clustering techniques in network structure planning (a non-hierarchical structure suffers from problems as scalability and capacity, therefore a hierarchical structure conduces improvement in the network performance). Then, for this reason, usually clustering techniques are used in telecommunications network projects. Nevertheless, typically the clustering process is unable to take into account the presence of obstacles in the scenario. Then, in this work we integrated the A* and Lloyd's algorithm as a way of producing a new algorithm which is able to perform the clustering process in a scenario that has obstacles. It is worth mentioning that we couldn't find back in the literature integration of A* and K-means. Now, through this new algorithm it is possible to perform the clustering process in more complex scenarios (i.e., scenarios with obstacles). Our proposed method consists in changing the Euclidian or Manhattan distances in the Lloyds algorithm to the distances calculated through a Brute-force search with A* algorithm. Our results show that the algorithm we attained as final product made the total mean distance change, from 4,7 to 3,0833 when compared with Lloyd's algorithm in the presence of obstacles in the clustering scenario to the specific case we considered.

Keywords: Cluster analysis; Telecommunication; Fiber optics; Wireless Communication Systems.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SÍMBOLOS

k - número de centroides do algoritmo de Lloyds

X - conjunto dos pontos que serão submetidos ao processo de clusterização

x_i - i-ésimo dado que será submetido ao processo de clusterização

$\mathbb{R}$ - conjunto dos números reais

d - dimensão dos dados pertencentes ao conjunto X

C - conjunto dos *clusters* do algoritmo de Lloyds

c_i - i-ésimo centroide

C_i - i-ésimo *cluster*

$J(C_k)$ - função que minimiza a distância até todos os pontos para um *cluster*

$J(C)$ - função que minimiza a distância até todos os pontos para todos os *clusters*

$f(n)$ - função de custo total do algoritmo A*

$g(n)$ - custo do percurso do nó de origem até o nó n no algoritmo A*

$h(n)$ - função heurística do algoritmo A*

n - nó que está sendo processado pelo algoritmo de A*

m_k - número de dados no *cluster* k

SUMÁRIO

1 INTRODUÇÃO

1.1 AGRUPAMENTO DE USUÁRIOS/NÓS

A rápida evolução dos sistemas de comunicações e o crescimento sem precedentes da demanda por comunicação de alta velocidade tanto em sistemas sem fio quanto por meio de fibra óptica no decorrer desta década tem atraído atenção global. Seguindo as tendências ditadas pelo mundo moderno, um expressivo número de usuários ao redor do planeta utiliza os serviços da *Internet* e de transferência de dados por meio de redes sociais, jogos *online*, comunicações por videoconferência e serviços de *streaming*. Sabemos por meio da Cisco (CISCO, 2020), que quase dois terços da população global vai ter acesso à *Internet* até 2023. Além disso, o número de dispositivos conectados em redes IP (*Internet Protocol*) vai ser mais do que três vezes a população global, com 3,6 dispositivos conectados per capita até este mesmo ano, resultando em 29,3 bilhões de dispositivos interconectados. Como consequência deste crescimento significativo em tão pouco tempo, temos a produção de um grande volume de dados e uma demanda de transmissão muito grande, resultando em desafios para o projeto de sistemas de comunicação modernos. Estes desafios têm levado os engenheiros a repensarem seus métodos de arquitetura de redes e buscarem formas mais eficientes de utilizar os escassos recursos disponíveis.

Uma forma de solucionar parte destes problemas envolve a utilização de técnicas de clusterização no projeto das redes. Sabemos atualmente que em uma estrutura sem hierarquia e sem agrupamento de nós geograficamente próximos, com o aumento do tamanho da rede surgem problemas de escalabilidade e capacidade (AL-HADDAD; ALDABBAGH, 2015) (GUPTA; KUMAR, 2000). Portanto, para alcançar um melhor desempenho em redes sem fio de larga escala, uma estrutura hierarquizada traz grandes benefícios (XU; HONG; GERLA, 2002) (MITRA; NANDY et al., 2012), motivando por esta razão a utilização de algoritmos de clusterização no projeto de redes de telecomunicações. Agora, tipicamente com a clusterização, é possível obter uma redução no consumo de potência dos transmissores no caso de comunicações sem fio e uma minimização do comprimento da fibra para o caso de comunicações ópticas (e uma melhor utilização dos recursos da rede para ambos os casos). A técnica de clusterização consiste em segregar dados em grupos que apresentam maior similaridade entre si do que com dados de outros grupos (RODRIGUEZ et al., 2019). É uma técnica recorrente em análise de dados, estatística, reconhecimento de padrões, bioinformática e aprendizado de máquina. Uma observação importante: neste trabalho focamos na clusterização de redes de comunicações. Portanto, estamos aplicando o conceito de clusterização para um caso específico. Ao invés de "segregar dados em grupos", no processo de clusterização estamos na verdade efetuando a "segregação dos usuários da rede em *clusters*".

A análise por clusterização tem suas origens nos estudos de Driver e Kroeber em 1932 (DRIVER; KROEBER, 1932). Existem diferentes métodos de clusterização. Neste ponto é importante destacarmos dois métodos: a clusterização baseada em protótipo e a clusterização baseada em densidade. Na clusterização baseada em protótipo, um *cluster* é composto por um conjunto de dados que possuem mais similaridade com um protótipo que define um *cluster* do que com um protótipo que define outro *cluster*. Para um dado que possui atributos contínuos o protótipo de um *cluster* geralmente é um

centroide. Por outro lado, na clusterização baseada em densidade, um *cluster* é uma região densa de objetos circundada por uma região de baixa densidade (TAN; STEINBACH; KUMAR, 2016). Um dos métodos de clusterização mais utilizados nos dias atuais é o K-means. O K-means é um método de clusterização baseado em protótipo. Sua popularidade é resultado principalmente da facilidade de interpretação de seus resultados e de sua prática implementação (no entanto, existem também outras técnicas de clusterização). Uma outra técnica muito popular é a "Clusterização hierárquica". Assim como o K-means esta técnica não é moderna, embora seja muito utilizada até os dias atuais devido à sua simplicidade. A abordagem deste método é a seguinte: selecionamos pontos individuais para composição dos *clusters*. Em seguida, os dois clusters mais próximos se unem, formando um único *cluster* e são efetuadas diversas iterações até que tenhamos como resultado apenas um *cluster*.

Durante as décadas de 1950 e 1960, vários algoritmos para implementar K-means foram propostos por pesquisadores de diversas áreas (PÉREZ-ORTEGA et al., 2019) (BOCK, 2008). Temos primeiramente o algoritmo proposto pelo matemático Hugo Steinhaus em seu artigo "*Sur la division des corps materiels en parties*" no qual foi abordado o problema de particionamento em aplicações industriais e antropológicas (STEINHAUS, 1956). Posteriormente no ano seguinte, foi publicado um artigo por Stuart Lloyd que lidava com o problema de transmissão de um sinal aleatório em um espaço multidimensional, abordando portanto o problema de particionamento para uma aplicação de eletrônica e telecomunicações (LLOYD, 1982). MacQueen também tratou o problema em 1967, sendo ele o responsável pela atribuição do nome K-means (MACQUEEN et al., 1967). Temos também Jancey em 1966 com outro algoritmo variante de K-means (JANCEY, 1966).

O método K-means define um protótipo em termos do centroide (que é recorrentemente a média do grupo de pontos que vamos clusterizar). Existem diferentes funções de proximidade que podemos utilizar quando consideramos clusterização por K-means contemplando garantir a convergência. Por exemplo, para a distância Manhattan (L1), o centroide mais apropriado é a mediana dos pontos em um *cluster*, pois esta minimiza a soma da distância L1 de um objeto ao centroide do *cluster*. Para a distância euclidiana (L2), o centroide apropriado é a média, pois minimiza a soma do quadrado da distância L2. Já para uma distância cosseno, devemos maximizar a soma da similaridade de cossenos de um objeto até o centroide mais próximo, obtendo como métrica novamente a média. Por fim, para uma divergência de Bregman, minimizamos a soma da divergência de Bregman de um objeto até o centroide do *cluster*, também obtendo como centroide apropriado a média (TAN; STEINBACH; KUMAR, 2016). Já fica claro neste ponto que uma métrica amplamente utilizada como forma de encontrar o centro de um *cluster* em K-means é a média, pois esta métrica em muitos casos minimiza a distância entre o centroide e os elementos associados ao correspondente *cluster*. Neste trabalho consideramos uma métrica de distância mais sofisticada, que é obtida por meio do algoritmo A*. Minimizamos a distância utilizando uma busca exaustiva, ou seja, calculamos as distâncias de todos os pontos disponíveis no cenário considerado até todos os elementos do *cluster*.

Como mencionado, o K-means é um método simples e intuitivo com múltiplas aplicações. Porém ele apresenta três importantes limitações.

a) Alta sensibilidade à escolha dos centroides iniciais.

b) Necessidade do número de *clusters* ser especificado pelo usuário (ALPAYDIN, 2020).

c) Impossibilidade de levar em conta obstáculos no processo de clusterização.

A primeira limitação pode ser resolvida utilizando K-means++. O K-means++ é um algoritmo de inicialização utilizado para a escolha dos valores iniciais do método K-means. Este algoritmo foi proposto por David Arthur e Sergei Vassilvitskii em 2007 (ARTHUR; VASSILVITSKII, 2007) e consiste em inicializar o primeiro centroide como um ponto aleatório c_1 dentre os pontos do conjunto de dados que vamos clusterizar, escolhendo os próximos pontos por meio de uma equação probabilística (HUSSAIN; HARIS, 2019). Para o próximo centroide "c_i", a probabilidade de um ponto "x" ser escolhido como centroide é proporcional ao quadrado da distância de "x" até o centroide mais próximo. Portanto, K-means++ sempre busca selecionar centroides distantes uns dos outros produzindo uma melhoria na performance geral de K-means (naturalmente resultando, no entanto, em um tempo de execução maior). A segunda limitação dificilmente resulta em problemas, porque muitas vezes desejamos de fato escolher a quantidade de *clusters*, no entanto, podemos solucionar esta limitação utilizando validação cruzada. A validação cruzada é um método utilizado para avaliar e comparar algoritmos de aprendizado dividindo os dados em dois segmentos. Um dos segmentos é utilizado para treinar o modelo enquanto o outro é utilizado para avaliar o modelo (REFAEILZADEH; TANG; LIU, 2009). Por fim, a terceira limitação foi uma das motivações para o desenvolvimento deste trabalho. Uma vez que não têm sido estudado casos de clusterização utilizando K-means levando em conta obstáculos no cenário, buscamos explorar e solucionar esta limitação através do desenvolvimento de um novo algoritmo.

1.2 CONTRIBUIÇÃO DO TRABALHO

Embora a técnica de K-means tenha sido amplamente utilizada, conduzindo melhoria no desempenho das redes em termos de escalabilidade, desempenho e capacidade, sabemos que existem limitações neste algoritmo. Uma destas limitações é a impossibilidade do K-means considerar obstáculos no processo de clusterização. Então, o principal objetivo deste trabalho foi modificar a clusterização por K-means para melhorar o desempenho da rede considerando um cenário mais complexo, levando em conta obstáculos. Encontrar caminhos em cenários que possuem obstáculos não é uma tarefa simples. Em alguns casos a busca de caminhos em cenários complexos caracterizados por obstáculos pode se tornar até mesmo inviável. Chris Crawford desistiu de implementar cenários com obstáculos em formato de "U" em um jogo eletrônico que desenvolvia:

> O lago em formato de U criou uma armadilha para o meu algoritmo de reconhecimento artificial. Eu fiquei muito tempo trabalhando em uma rotina de reconhecimento artificial mais inteligente que não ficasse presa por estes lagos (...) Depois de muito esforço desperdiçado eu descobri uma solução melhor: deletar lagos em formato de U do mapa.(CRAWFORD, 1982, p.100, tradução nossa).[1]

Ainda assim, existem algoritmos eficientes capazes de encontrar caminhos em cenários que possuem vários obstáculos, geralmente utilizados em robótica. Um algoritmo que é amplamente utilizado com esta finalidade é o A*, desenvolvido inicialmente em um projeto de robótica (e posteriormente muito

[1] The U-shaped lake created a trap for my artificial reckoning algorithm. I expended a great deal of time working on a smarter artificial reckoning routine that would not be trapped by such lakes (...) After much wasted effort I discovered a better solution: delete U-shaped lakes from the map.

utilizado em jogos eletrônicos que utilizam inteligência artificial para buscas de caminhos). Então, com a motivação de superarmos a limitação imposta pelo K-means, buscamos alterar a forma de cálculo das distâncias utilizadas no K-means (geralmente distância euclidiana ou distância Manhattan) para considerar distâncias calculadas utilizando A*. Após uma busca bibliográfica, não deparamos com casos de integração destes dois algoritmos. Por meio da integração do algoritmo A* com o algoritmo de Lloyds aparentemente se tornaria possível a clusterização de usuários em redes cujo cenário é composto por obstáculos. No entanto, o volume de cálculos efetuados no decorrer do algoritmo poderia resultar em uma alta complexidade computacional. Então, como forma de superar longos períodos de simulação e evitar a repetição de cálculos desnecessários, foi utilizado o seguinte procedimento no algoritmo implementado: inicialmente são calculadas todas as distâncias entre cada coordenada do cenário utilizando o algoritmo A*. Em seguida, estas distâncias são armazenadas em um tensor. Agora, definidas as posições dos centroides iniciais e dos usuários iniciamos a simulação com todas as distâncias já calculadas. Portanto, supondo que estejamos motivados a simular diferentes posições de usuários e centros de *cluster* sem modificar o cenário em questão, teremos uma grande redução no tempo de simulação (e quantidade de cálculos a serem efetuados pelo computador). Mas, caso seja necessário modificar o cenário teremos de efetuar um novo cálculo de tensor de distâncias considerando as novas posições de obstáculos e valores de distâncias calculadas utilizando o algoritmo A*.

1.3 ESTRUTURA DO DOCUMENTO

Este documento encontra-se estruturado da seguinte forma: No Capítulo 2 descrevemos o algoritmo de Lloyds para a implementação de K-means. Posteriormente, introduzimos o algoritmo A* para obter rotas em cenários com obstáculos. Por fim, discutimos a integração destes dois algoritmos com o objetivo de desenvolver um algoritmo de clusterização capaz de levar em conta obstáculos. Então, exemplos da utilização do algoritmo desenvolvido são apresentados e discutidos no Capítulo 3. Finalmente, as conclusões e possibilidades de trabalhos futuros são apresentadas no Capítulo 4.

2 INTEGRAÇÃO DE K-MEANS E A*

Neste capítulo descrevemos os algoritmos K-means e A*, assim como propomos a integração de ambos. Inicialmente na Seção 2.1 introduzimos o método de K-means. Em seguida, na Seção 2.2. apresentamos o algoritmo A* para obter rotas em entornos com obstáculos. Por fim, na Seção 2.3 propomos uma forma de integração destes dois algoritmos que resulta em um algoritmo de clusterização capaz de considerar cenários com obstáculos.

2.1 ALGORITMO DE CLUSTERIZAÇÃO K-MEANS

O K-means é um método de clusterização. Trata-se de um algoritmo de classificação não supervisionado, ou seja, adequado para dados não rotulados (LOPEZ et al., 2018) e é utilizado como forma de segregar um conjunto de dados em "k" *clusters* de maneira que os pontos associados a um mesmo *cluster* possuam maior similaridade entre si do que aos pontos associados aos outros *clusters*. Neste sentido, a similaridade pode ser interpretada como uma distância mínima entre os pontos (AYTAÇ, 2020). Uma das implementações mais utilizadas, o denominado algoritmo padrão, foi desenvolvido por Stuart Lloyd em 1957 no artigo "*Least Squares Quantization in PCM*" (LLOYD, 1982). Entretanto, a idéia inicial do algoritmo foi proposta por Hugo Steinhaus em 1956 (mais cientistas contribuíram para o desenvolvimento do algoritmo, entre eles: James MacQueen, Edward W.Forgy e Jancey, R.C.). Sua fácil implementação, simplicidade e eficiência resultaram em ampla utilização desde sua publicação até os dias atuais. Como já mencionado, apesar de apresentar importantes vantagens, este algoritmo apresenta algumas limitações: por um lado, a alta sensibilidade à escolha das posições iniciais dos centroides e, por outro lado, a dificuldade de associar todos os pontos ao centro de um *cluster*. Estes problemas são solucionados em versões variantes do K-means. É importante ressaltar que a versão mais popular de K-means é o "Algoritmo de Lloyds".

De forma simplificada, o algoritmo de Lloyds é inicializado com a escolha dos "k" centroides iniciais. Então, cada ponto é associado ao centroide mais próximo e cada agrupamento de pontos associado a um mesmo centroide é designado como um *cluster*. Em sequência, cada centroide é atualizado com base nos pontos associados ao *cluster*. Este processo se repete até que não ocorram variações nos pontos associados à cada *cluster* (ou seja, os centroides não são mais atualizados) e por fim o algoritmo é encerrado. Podemos descrever o algoritmo formalmente nos quatro passos enumerados a seguir.

I) Selecionar "k" pontos como centroides iniciais.

II) Formar "k" *clusters* associando cada ponto ao centroide mais próximo.

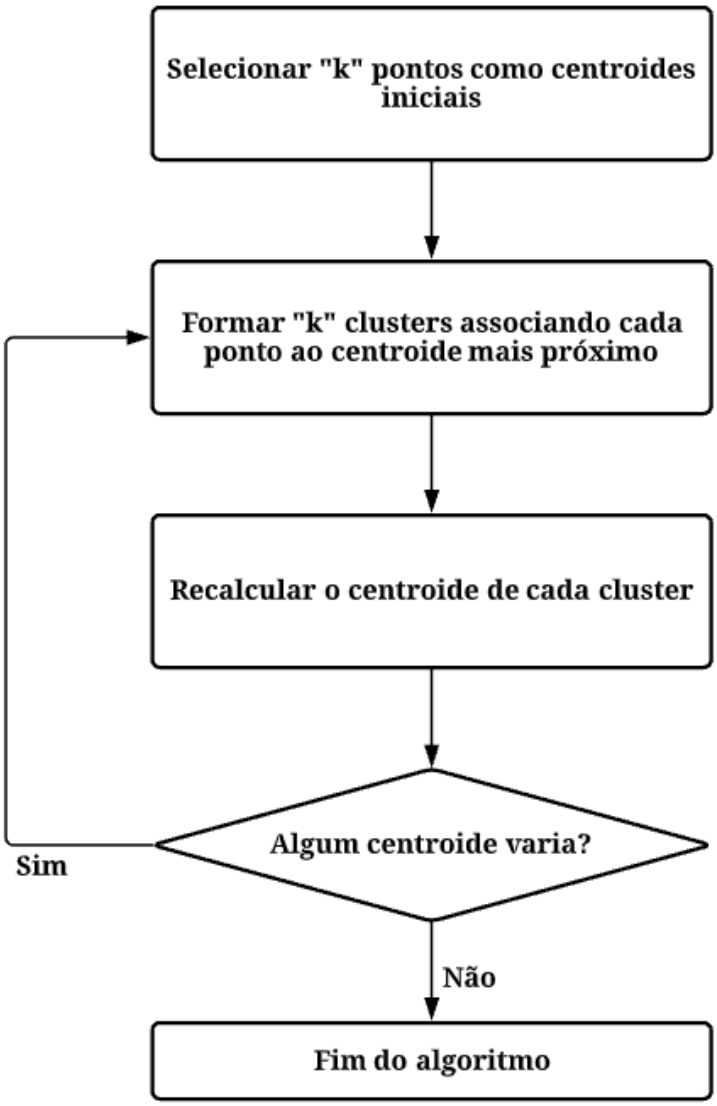
III) Recalcular o centroide de cada *cluster*.

IV) Repetir os passos II e III até que não ocorram mudanças nos centroides.

Na Figura 1 é apresentado o fluxograma do algoritmo de Lloyds.

A partir deste ponto, podemos também descrever matematicamente o K-means: dado um conjunto de pontos $X = \{x_1, ..., x_n\}$ tal que $x_i \in \mathbb{R}^d$ para $i = 1, ..., n$ temos que $d \geq 1$ é a dimensão dos dados pertencentes ao conjunto X. Então, seja o conjunto de partições $C = \{C_1, ..., C_k\}$ tal que $k \geq 2$, para

Figura 1 – Fluxograma do algoritmo de Lloyds.



Fonte: (SCABURI et al., 2021)

k partições e sendo c_i o centroide do *cluster* C_i (ou seja, o ponto que minimiza a distância até todos os pontos associados ao *cluster* C_i) o erro entre c_i e os pontos pertencentes ao *cluster* C_i é definido como:

$$J(C_k) = \sum_{x_i \in C_k} d(c_i, x), \tag{2.1}$$

em que $d(c_i, x)$ é a distância entre x_i e c_k. Então, o objetivo do algoritmo é minimizar a soma da distância para todos os *clusters*:

$$J(C) = \sum_{i=1}^{K} \sum_{x_i \in C_k} d(c_i, x). \tag{2.2}$$

Agora, o problema de clusterização é reformulado como um problema de otimização. Desde os estudos pioneiros de Steinhaus e Lloyd, diferentes abordagens ao problema foram propostas. Uma

observação importante: minimizar esta função de custo é um problema computacionalmente difícil (NP-difícil) (JAIN, 2010) mesmo para $k = 2$ (DRINEAS et al., 2004). Portanto, obter uma solução ótima para muitos casos pode resultar em um problema intratável [2] (PÉREZ-ORTEGA et al., 2019). Posteriormente ao surgimento do algoritmo, uma variedade de algoritmos heurísticos foram propostos com o objetivo de obter uma solução para este tipo de problema de otimização.

2.2 ALGORITMO DE BUSCA DE PERCURSO A*

O algoritmo A* foi desenvolvido pelos cientistas da computação Bertram Raphael, Nils Nilson e Peter Hart no projeto do robô "Shakey" (HART; NILSSON; RAPHAEL, 1968). Enquanto outros projetos de robótica da época recebiam instruções individuais separadas em passos para posteriormente executar longas tarefas, "Shakey" tinha a capacidade de analisar comandos e planejar suas ações, sendo então o primeiro robô projetado com inteligência artificial capaz de mover-se e executar tarefas sem a necessidade de instruções segmentadas (BAJCSY; ALOIMONOS; TSOTSOS, 2018)(NILSSON, 1969). O desenvolvimento de "Shakey" causou grande impacto no campo da robótica e da inteligência artificial, em partes devido ao desenvolvimento do algoritmo A* que passou a ser utilizado popularmente como método de busca de percurso em situações em que as informações do ambiente já são conhecidas.

Formalmente, o algoritmo é definido por uma função de custo $f(n)$ que estima o custo total do percurso formado pelo caminho do nó de origem até o nó de destino. Considerando n como o nó atual do percurso, temos:

$$f(n) = g(n) + h(n). \tag{2.3}$$

Então, $g(n)$ corresponde ao custo do percurso partindo do nó inicial até n. Além disso, $h(n)$ é denominada de "função heurística", responsável por estimar o custo de n até o nó de destino. A* é capaz de encontrar caminhos ótimos mesmo em cenários caracterizados por vários obstáculos, evitando nós com valores de $f(n)$ maiores do que outros nós candidatos a integrarem o percurso que leva até o destino final. Existem ainda dois conceitos importantes no algoritmo: a denominada "lista aberta" e a "lista fechada". A lista fechada contém todos os nós que já foram processados pelo algoritmo, enquanto a lista aberta contém os nós vizinhos aos nós processados (que ainda devem ser processados pelo algoritmo). Os nós presentes na lista aberta serão processados pelo algoritmo. Os nós da lista fechada serão ignorados (SHARMA et al., 2012).

O algoritmo é executado seguindo os seguintes passos: inicialmente criamos a lista aberta e a lista fechada. Então adicionamos o nó inicial na lista aberta, adicionamos todos os obstáculos na lista fechada e iniciamos um *loop*. Em cada iteração calculamos os valores de $f(n)$, $g(n)$ e $h(n)$ dos nós vizinhos ao nó atual e transferimos o nó com menor valor de custo $f(n)$ da lista aberta para a lista fechada. Já sabemos neste ponto que os nós da lista fechada são ignorados, portanto, ao transferir o nó com menor valor de custo para a lista fechada após a iteração estamos garantindo que não vamos visitar novamente este nó durante a execução do algoritmo (os nós já visitados possuem valor de $f(n)$

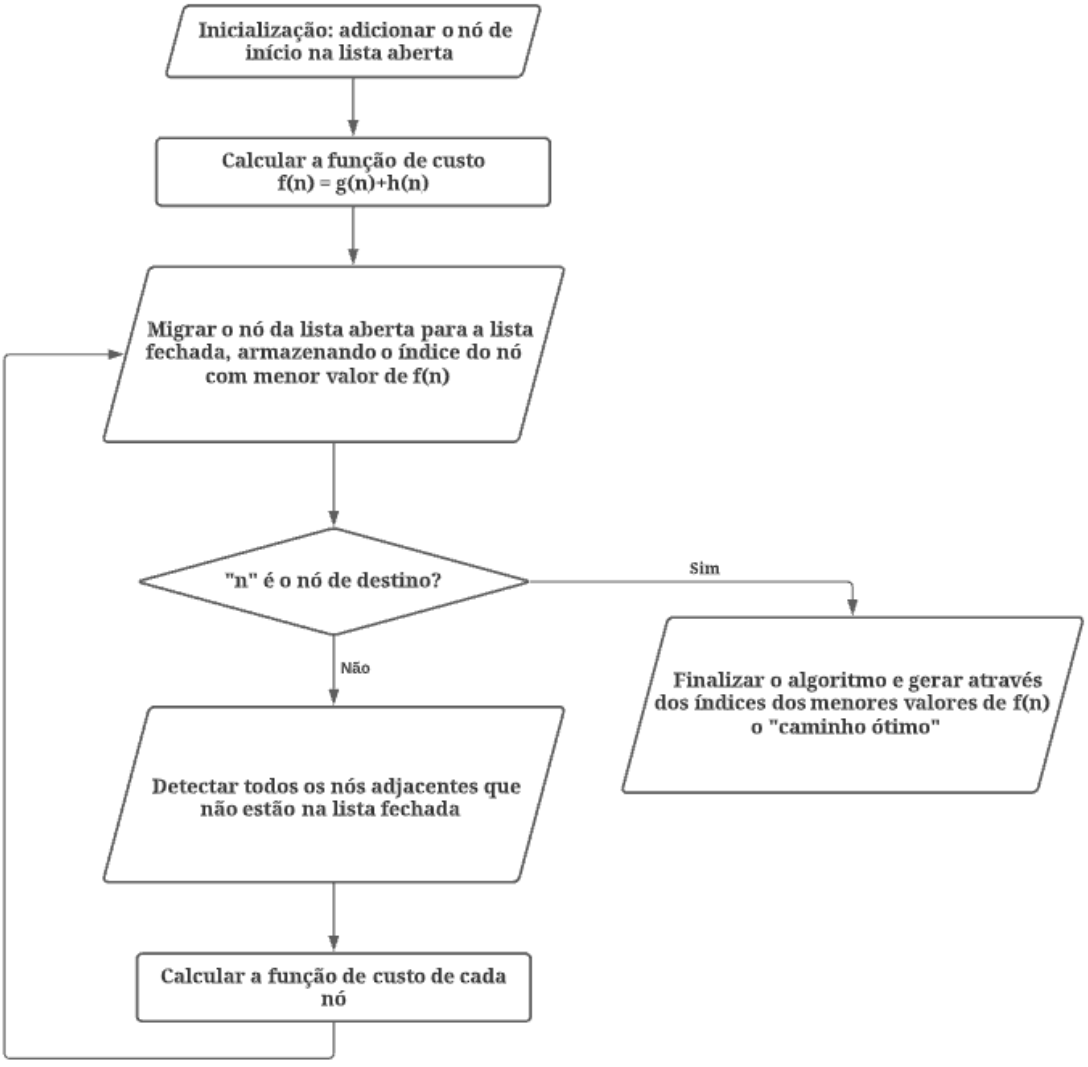
[2] Do ponto de vista da teoria da computação um problema intratável é um problema para o qual não existem algoritmos eficientes que o resolvam de forma ótima.

menor, no entanto não devemos escolhe-los, pois desejamos avançar em direção ao nó de destino, nunca retornando em direção ao nó de início). Agora, consideramos este nó como nosso nó atual, adicionamos todos os seus nós vizinhos na lista aberta e classificamos o nó pelo qual alcançamos este nó atual como "nó parente". Então, repetimos o processo até que o nó atual coincida com o nó de destino e neste ponto encerramos o *loop*. Agora, basta retornarmos do nó de destino até o nó de origem através dos nós definidos como "nós parentes" e este conjunto de nós corresponde ao caminho ótimo encontrado pelo algoritmo.

Tenhamos um olhar agora para a seguinte situação: durante a execução do algoritmo, após efetuarmos algumas iterações transferimos todos os nós da lista aberta para a lista fechada e apesar disso não fomos conduzidos à situação em que o nó atual coincide com o nó de destino. Isto ocorre quando o nó de destino está cercado por obstáculos e não é possível encontrarmos um caminho ótimo até o destino, sendo necessário então finalizar o algoritmo uma vez que o nó de destino encontra-se totalmente bloqueado.

Na Figura 2 é apresentado o fluxograma do algoritmo A*.

Figura 2 – Fluxograma do algoritmo A*.



Fonte: (ZIDANE; IBRAHIM, 2017)

2.3 CLUSTERIZAÇÃO CONSIDERANDO OBSTÁCULOS

Um fato que motivou o desenvolvimento deste trabalho foi que não fomos capazes de encontrar anteriormente na literatura casos de integração entre estes dois algoritmos (Algoritmo de Lloyds e A*). Sabemos que o método K-means utiliza uma distância calculada entre os pontos e os centroides $d(c_i, x)$. Na maioria dos casos esta distância é euclidiana, ou seja:

$$d(c_i, x) = ||c_i - x||^2. \tag{2.4}$$

Portanto, contemplamos minimizar a soma do quadrado da distância do ponto até o centro do *cluster*. A métrica que minimiza a distância dos pontos até o *cluster* para o caso da distância euclidiana é a média. Podemos mostrar isto da seguinte forma: considerando o caso unidimensional $d = 1$, desejamos minimizar a Equação 2.2. Então, diferenciando com relação a c_k e igualando a zero temos:

$$\frac{\partial J(C)}{\partial c_k} = \frac{\partial \sum_{i=1}^{K} \sum_{x_i \in C_k} (c_i - x)^2}{\partial c_k} \tag{2.5}$$

$$= \sum_{i=1}^{K} \sum_{x_i \in C_k} \frac{\partial (c_i - x)^2}{\partial c_k} \tag{2.6}$$

$$= \sum_{x_i \in C_k} 2(c_k - x_k) = 0 \tag{2.7}$$

$$\sum_{x_i \in C_k} 2(c_k - x_k) = 0 \implies m_k c_k = \sum_{x_i \in C_k} x_k \implies c_k = \frac{1}{m_k} \sum_{x_i \in C_k} x_k. \tag{2.8}$$

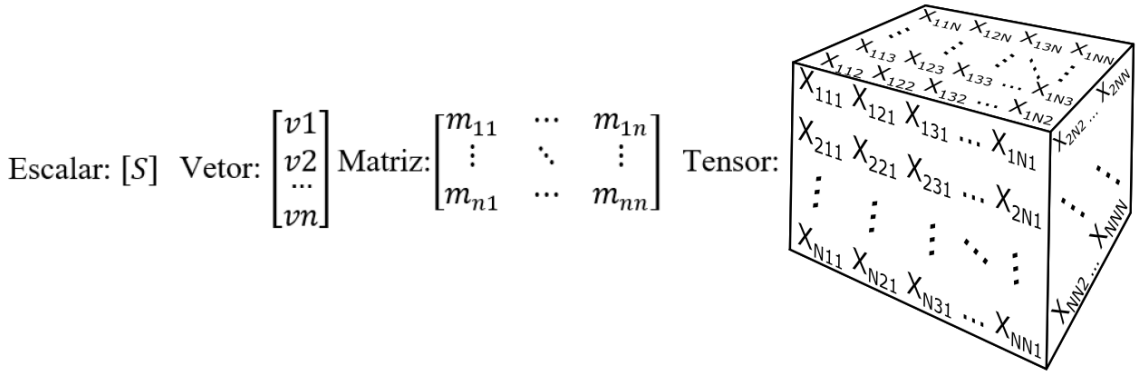
Em que c_k é o centroide do *cluster* C_k e m_k é o número de dados no *cluster* k. Assim, comprova-se que quando consideramos a distância euclidiana a posição obtida calculando a média dos elementos em cada dimensão corresponde ao ponto que minimiza a distância a todos os elementos do *cluster*. No entanto, quando uma métrica de distância diferente é considerada, como no caso de considerarmos a distância utilizando o algoritmo A*, a posição do ponto que minimiza a distância a todos os elementos do *cluster* não pode ser obtida diretamente calculando a média. Neste caso, devemos considerar uma métrica que leve em conta a presença dos obstáculos, por exemplo, uma busca exaustiva. Por meio da busca exaustiva calculamos as distâncias de todos os pontos disponíveis no cenário até todos os elementos do *cluster*, obtendo assim uma métrica capaz de considerar os obstáculos.

Para integrar a distância calculada por meio do algoritmo A* ao algoritmo de Lloyds, devemos efetuar uma grande quantidade de cálculos computacionais. Devemos efetuar todos os cálculos de distâncias entre os pontos e os centroides por meio do algoritmo A* repetidas vezes, necessitando provavelmente várias iterações até que não ocorram mais variações nas coordenadas dos centroides.

Isto implica em um alto custo computacional. Contemplando a possibilidade de reduzir o tempo de simulação e uma vez que muitas das distâncias são repetidas durante a execução do algoritmo, decidimos armazenar todas as distâncias entre cada coordenada para o cenário utilizado. Agora, com todas as distâncias possíveis calculadas, podemos utilizar os valores de distâncias do cenário previamente caracterizado para diferentes pontos e centroides iniciais, apenas sendo necessário efetuar mais cálculos quando estivermos motivados a modificar o cenário de simulação.

Para armazenar esta grande quantidade de distâncias utilizamos um "tensor". Podemos definir tensores como uma generalização de escalares, vetores e matrizes para dimensões arbitrárias [3] (SHULGA et al., 2019). Na notação tensorial, um escalar é definido como um tensor de ordem zero, um vetor é um tensor de ordem um e uma matriz é um tensor de ordem dois (PANIGRAHY; MISHRA, 2020). Uma forma de representação comparativa de tensores com escalares, vetores e matrizes é mostrada na Figura 3.

Figura 3 – Comparação entre escalar, vetor, matriz e tensor.



Fonte: (EIGENCHRIS, 2017)

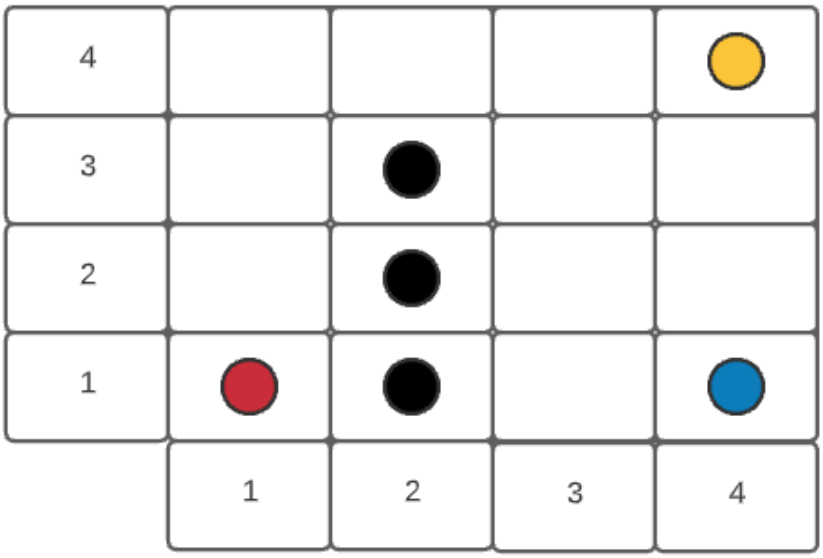
Neste trabalho efetuamos a implementação do algoritmo integrado para cenários bidimensionais. Isto significa que cada valor de distância é definido pelas coordenadas cartesianas "x" e "y" do ponto de origem e pelas coordenadas cartesianas "x" e "y" do ponto de destino. Portanto buscando uma forma de armazenar e posteriormente acessar as distâncias calculadas, optamos pela utilização de um tensor de ordem quatro.

Para ilustrar melhor a diferença entre a utilização da distância euclidiana e da distância obtida com A*, podemos considerar o seguinte exemplo: suponhamos que vamos utilizar para a clusterização a distância euclidiana entre um ponto x_1 localizado na coordenada (1,1) e os centroides c_1 localizado na coordenada (4,1) e c_2 na coordenada (4,4) respectivamente. Consideramos também obstáculos nos pontos (2,1), (2,2) e (2,3). Este cenário é ilustrado na Figura 4.

Então, temos que $d(x_1, c_1) = \sqrt{(1-4)^2 + (1-1)^2} = 3$ e $d(x_1, c_2) = \sqrt{(1-4)^2 + (1-4)^2} = 4,242$. Portanto, como $d(x_1, c_1) < d(x_1, c_2)$ o ponto x_1 pertenceria ao *cluster* de c_1. Agora, vamos considerar o cálculo da distância utilizando o algoritmo A*. Com este propósito calculamos todos

[3] Como utilizaremos tensores apenas para armazenar valores de distâncias, podemos representa-los desta forma, no entanto, caso utilizássemos as implicações geométricas mais complexas dos tensores precisaríamos de uma definição mais elaborada.

Figura 4 – Cenário bidimensional 4×4 para um ponto (indicado em vermelho), dois centroides (amarelo e azul) e três obstáculos (representados em preto).



Fonte: (SHRIKANT; SELVAKUMAR, 2018)

Coordenada	$f(n)$	$g(n)$	$h(n)$
(1,2)	4,414	1	3,414
(1,3)	5,828	2	3,828
(1,4)	7,242	3	4,242
(2,4)	7,242	3,414	3,828
(3,1)	7,828	6,828	1
(3,2)	7,242	5,828	1,414
(3,3)	7,242	4,828	2,414
(3,4)	7,828	4,414	3,414
(4,2)	7,242	6,242	1
(4,3)	7,828	5,828	2

Tabela 1 – Valores de $f(n)$,$g(n)$ e $h(n)$ para c_1

os valores de $f(n)$, $g(n)$ e $h(n)$ para o centroide c_1 na Tabela 1. Cabe mencionar que neste cálculo, utilizamos como $h(n)$ a distância em linhas retas e diagonais até o destino ignorando os obstáculos no percurso. No entanto, seria possível utilizar outras distâncias para o cálculo da heurística, por exemplo, poderíamos efetuar um simples cálculo de distância Manhattan entre o ponto de origem e o ponto de destino. Para o centroide c_2 temos também os resultados na Tabela 2. Partindo da coordenada inicial (1,1) até (2,4) temos apenas um caminho possível que evita os obstáculos e tem por direção as coordenadas dos centroides. Agora, na coordenada (2,4) temos duas possibilidades: a coordenada (3,4) ou a coordenada (3,3). Utilizando os menores valores de $f(n)$ obtemos dois traçados diferentes de percurso: Seguimos por (3,4) até (4,4) ou alternativamente por (3,3) passando por (4,2) até (4,1). Por fim, temos dois valores de distâncias diferentes dos valores obtidos para a distância euclidiana. Denotando a distância calculada utilizando o algoritmo A* por $dA^*(x, c_k)$ obtemos: $dA^*(x_1, c_1) = 7,242$ e $dA^*(x_1, c_2) = 5,414$. Neste caso, como $dA^*(x_1, c_1) > dA^*(x_1, c_2)$,

Coordenada	$f(n)$	$g(n)$	$h(n)$
(1,2)	4,828	1	3,828
(1,3)	5,414	2	3,414
(1,4)	6	3	3
(2,4)	5,414	3,414	2
(3,1)	10,242	6,828	3,414
(3,2)	8,242	5,828	2,414
(3,3)	6,242	4,828	1,414
(3,4)	5,414	4,414	1
(4,2)	8,242	6,242	2
(4,3)	6,828	5,828	1

Tabela 2 – Valores de $f(n)$,$g(n)$ e $h(n)$ para c_2

x_1 pertence ao *cluster* c_2.

Portanto, podemos concluir que a utilização de A* para calcular as distâncias resulta em uma modificação da associação de centroides e consequentemente em uma solução diferente para o problema de clusterização K-means. Então, notamos que é possível a integração dos dois algoritmos (Algoritmo de Lloyds e A*) como forma de obter um método de clusterização capaz de levar em conta obstáculos presentes no cenário. Tal algoritmo naturalmente produziria como resultado uma forma de clusterização mais realista, utilizando distâncias obtidas por meio de A* ao invés um simples cálculo de distância Manhattan ou de distância euclidiana.

3 RESULTADOS NUMÉRICOS

Neste capítulo implementamos a integração dos algoritmos de Lloyds e de A* e aplicamos o algoritmo que obtivemos como produto final em um cenário arbitrário. Na Seção 3.1 implementamos o algoritmo A* de forma isolada para verificar sua eficiência. Em seguida na Seção 3.2 descrevemos como armazenamos os valores das distâncias no tensor. Na seção 3.3 explicamos como encontramos as coordenadas dos pontos que minimizam as distâncias aos outros pontos dos *clusters* (i.e., as posições dos centroides). Finalmente, na Seção 3.4, apresentamos o algoritmo de A* integrado ao algoritmo de Lloyds e avaliamos seu desempenho em um cenário que possui obstáculos.

3.1 AVALIAÇÃO DO ALGORITMO A*

É muito importante que o algoritmo A* esteja de fato evitando os obstáculos presentes no cenário e encontrando a menor distância entre o ponto de origem e o ponto de destino, porque ele será utilizado durante toda a execução do algoritmo de Lloyds como forma de efetuar os cálculos das distâncias. Portanto, como uma maneira de comprovar o correto desempenho do algoritmo de Lloyds integrado com A*, decidimos implementar primeiramente o algoritmo A* e verificar o seu desempenho. Utilizamos o algoritmo de A* disponibilizado no *site MathWorks* com algumas modificações (ver Apêndice A) e iniciamos a seguinte simulação utilizando o *software* Matlab:

I) criamos o cenário com obstáculos: criamos um grid retangular bidimensional (optamos por utilizar um cenário 20x20) onde os obstáculos estão identificados pelo valor “1” e a ausência de obstáculos pelo valor “0”.

II) Selecionamos os pontos de origem: (18,14), (14,13), (10,1) e (15,7) (selecionados arbitrariamente).

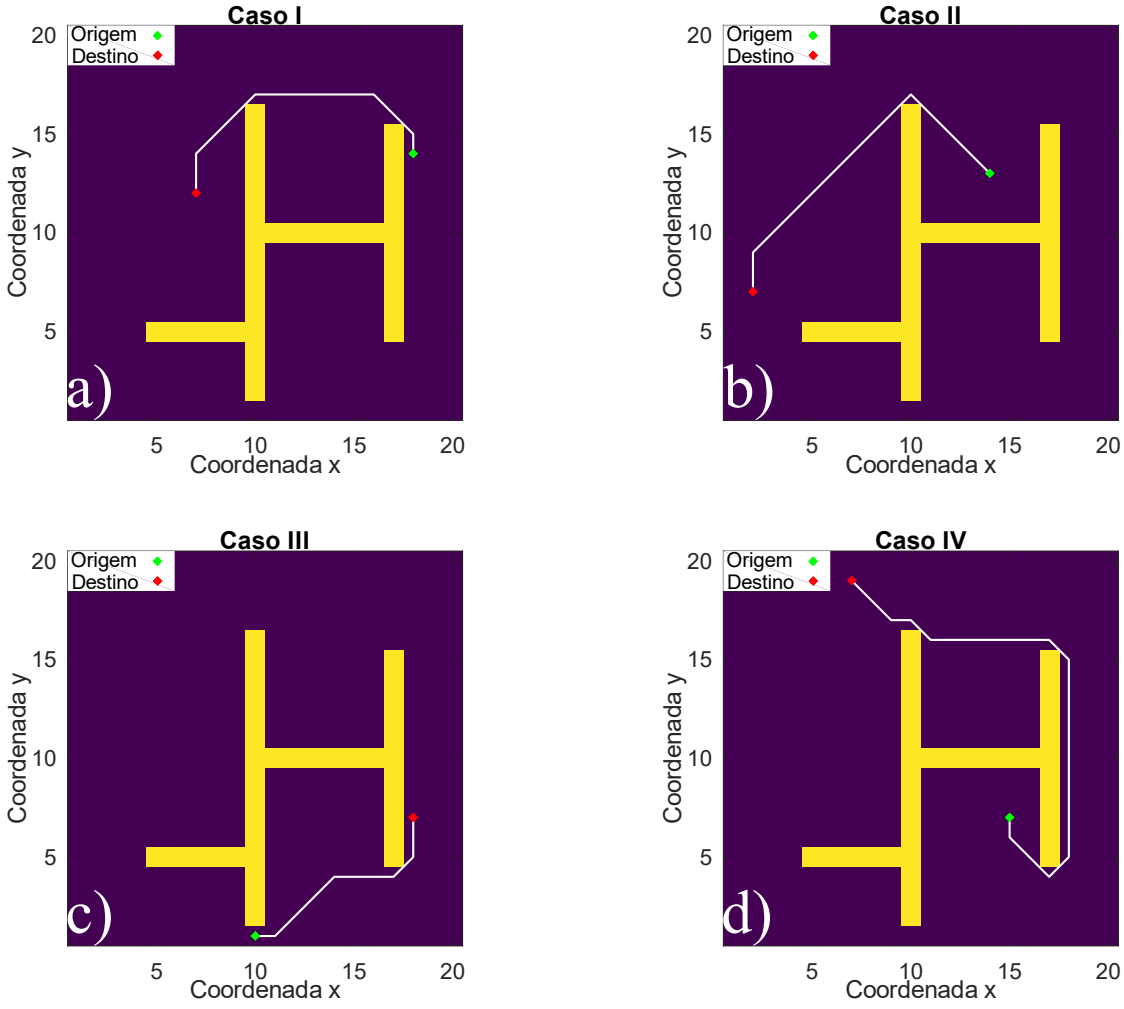
III) Escolhemos os pontos de destino: Decidimos utilizar (7,12), (2,7), (18,7) e (7,19).

Agora, o cenário é caracterizado por uma matriz esparsa. Então, na implementação do código podemos colocar todos os pontos associados ao valor “1” da matriz que representa o cenário diretamente dentro da lista fechada. A propósito, definimos o ponto de início como valor “-1” e o ponto de destino como valor “-2” na matriz que representa o cenário. Executamos o algoritmo para cada um destes quatro casos, obtendo os resultados apresentados na Figura 5. Neste ponto, fica claro que o algoritmo é de fato capaz de encontrar percursos corretos, evitando os obstáculos e sempre percorrendo as coordenadas na direção do ponto de destino de maneira a encontrar o menor valor de distância possível.

3.2 GERAÇÃO DO TENSOR DE DISTÂNCIAS

O cálculo das distâncias utilizando A* é eficiente, mas ainda tem um alto custo computacional. Buscando evitar o cálculo de distâncias repetidas de forma desnecessária, optamos por armazenar primeiramente as distâncias entre todos os elementos em um tensor. Uma vez que o cenário é bidimensional, cada distância é caraterizada pelos quatro índices correspondentes às coordenadas “x” e “y” da posição origem e destino. Como forma de armazenar e posteriormente acessar estes

Figura 5 – Algoritmo A* implementado para quatro combinações de origem e destino.

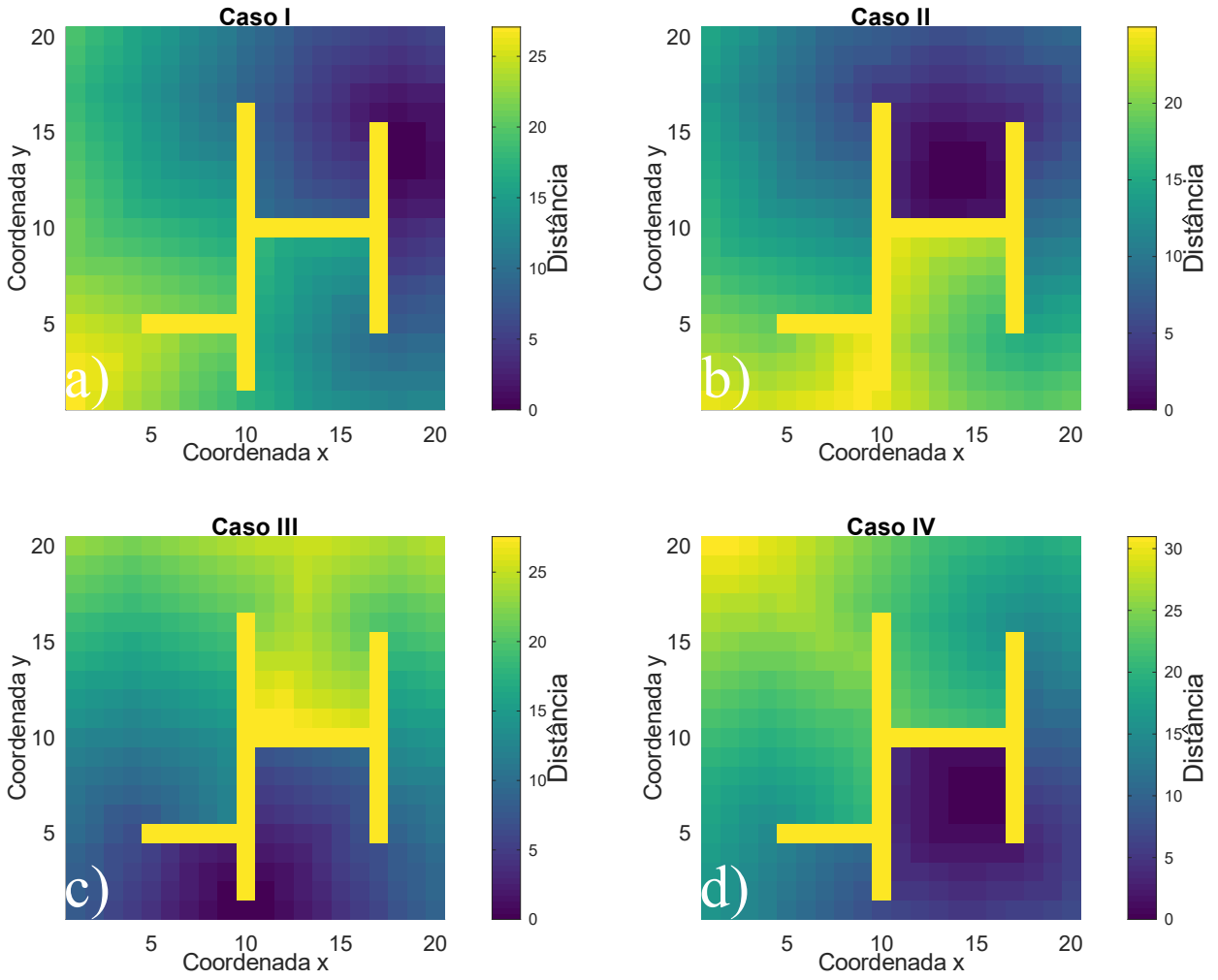


Fonte: Elaborado pelo autor

valores de distâncias podemos utilizar um tensor de ordem quatro. Desta forma, antes de efetuarmos a execução do algoritmo integrado vamos calcular todos os valores de distâncias possíveis e armazena-las neste tensor, para posteriormente acessarmos as distâncias que utilizaremos de acordo com a situação específica que vamos simular, i.e., dependendo da quantidade e da posição dos centroides iniciais e dos usuários presentes no cenário. Novamente, considerando os pontos de origem (18,14), (14,13), (10,1) e (15,7) calculamos com o algoritmo A* as distâncias entre todas as coordenadas do cenário. A Figura 6 apresenta os diagramas de contorno correspondentes às distâncias desde os quatro pontos considerados. Não é de surpreender que os valores máximos de distâncias são os valores referentes às coordenadas dos obstáculos e os valores mínimos são referentes às coordenadas adjacentes. Inclusive, regiões distantes do ponto de origem e obstruídas por obstáculos apresentam um valor muito alto de distância, enquanto que regiões próximas às coordenadas iniciais e sem obstáculos nas coordenadas adjacentes apresentam valores de distâncias menores.

Então, temos agora todos os valores das distâncias armazenados no tensor e temos também representações gráficas que mostram como a presença de obstáculos afeta fortemente os valores de distâncias calculados com A* para cada coordenada do cenário em cada um dos casos considerados.

Figura 6 – Diagramas de contorno da distância até cada um dos pontos considerados para os quatro casos.



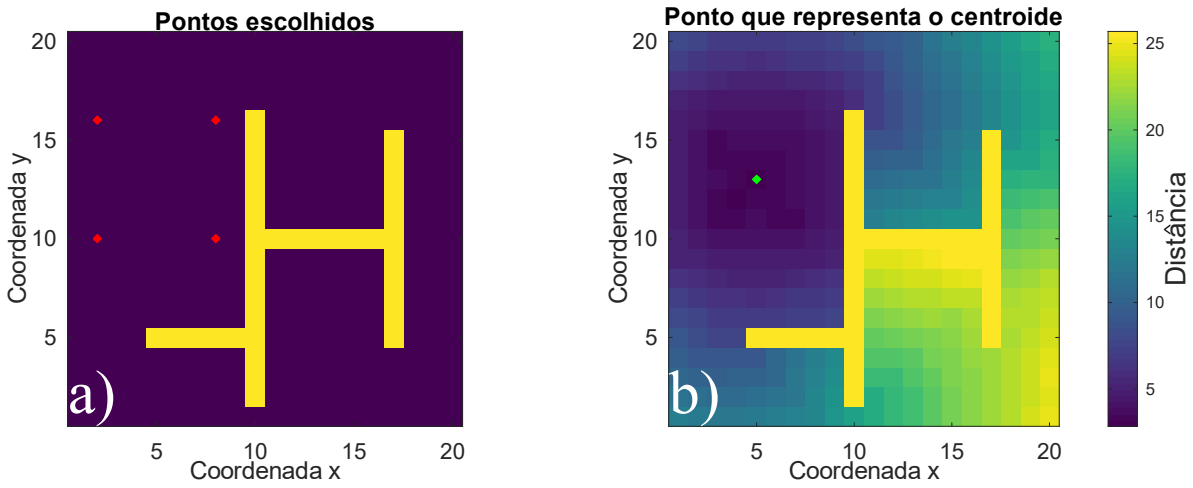
Fonte: Elaborado pelo autor

3.3 CÁLCULO DAS COORDENADAS DO PONTO QUE MINIMIZA A DISTÂNCIA AOS PONTOS DO CLUSTER

Agora, depois de confirmar que o algoritmo A* está precisamente obtendo percursos ótimos no cenário para diferentes coordenadas de origem e destino e possuirmos todos os valores de distâncias calculados e armazenados no tensor, podemos iniciar a integração com o algoritmo de Lloyds. Sabemos que o algoritmo de Lloyds após associar os pontos aos centroides mais próximos busca na terceira etapa de execução a posição do ponto que minimiza a distância para todos os pontos do *cluster*. Então, devemos integrar também ao algoritmo de A* uma maneira de minimizar a distância de todos os pontos para um único ponto que é, efetivamente, o cálculo da posição do centroide.

Visando encontrar este ponto de minimização efetuamos o seguinte procedimento: primeiramente, acessamos as distâncias previamente calculadas no tensor entre cada ponto e todos os pontos possíveis no cenário. Uma observação: apesar do cálculo do tensor que vimos no passo anterior ser significativamente longo, uma vez que possuímos o tensor já calculado, acessar as distâncias armazenadas é

Figura 7 – Representação gráfica do cálculo do centroide sem efeito dos obstáculos. (a) Cenário com os pontos pertencentes ao *cluster*. (b) Ponto que minimiza a distância para os quatro pontos escolhidos e o diagrama de contorno obtido quando não consideramos obstáculos.

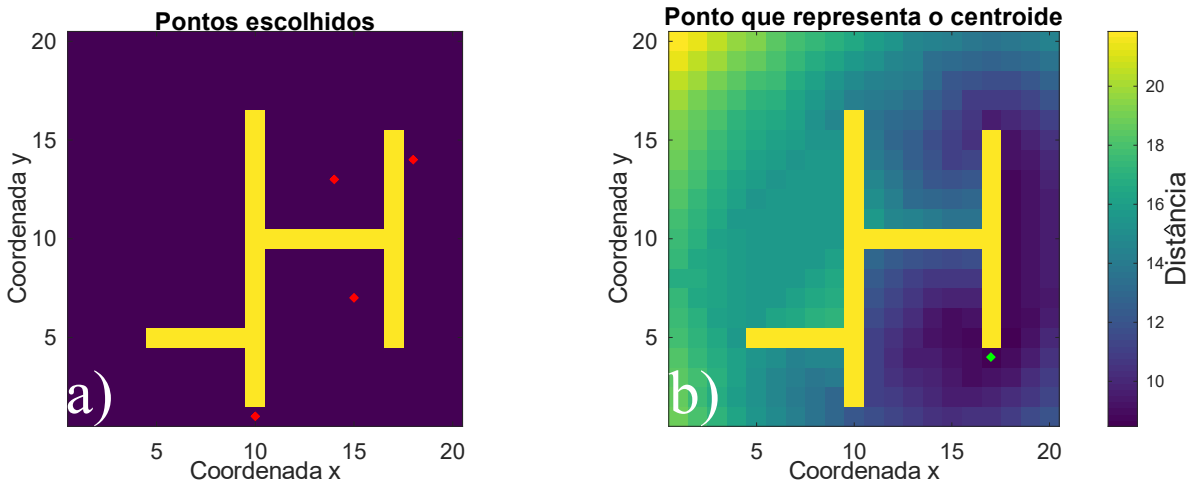


Fonte: Elaborado pelo autor

um processo muito rápido. Então, calculamos a média das distâncias destes pontos e os armazenamos em uma nova matriz. Agora, acessamos esta nova matriz e identificamos o menor valor dentre todas as médias calculadas. As coordenadas correspondentes a este valor devem identificar o ponto que minimiza a distância para todos os outros pontos do *cluster*. Cabe mencionar que mesmo pontos separados por uma distância euclidiana pequena (sem considerar obstáculos) poderiam pertencer a diferentes *clusters* caso existam obstáculos entre eles. Devido à presença dos obstáculos esperamos que no algoritmo integrado ocorra uma tendência de afastamento do centroide com relação aos obstáculos (uma vez que esta posição é obtida mediante uma busca exaustiva de distâncias obtidas com A*).

Por ora, como forma de comprovar a eficiência do nosso método utilizado para encontrar o ponto que minimiza a distância aos pontos que pertencem aos *clusters* utilizando a busca exaustiva, decidimos escolher inicialmente quatro pontos que não são influenciados por obstáculos. Escolhendo os pontos (2,16), (8,16), (2,10) e (8,10) obtemos como resultado que o ponto que minimiza a distância é o ponto (5,13). Na Figura 7 são representados de forma gráfica os pontos no cenário considerado e o diagrama de contorno correspondente à distância mínima a estes pontos. O ponto (5,13) está totalmente centralizado com relação aos outros pontos. Ou seja, o nosso método está correto quando não consideramos obstáculos. Vamos agora encontrar o ponto que minimiza a distância para os pontos que consideramos inicialmente para avaliar o desempenho do algoritmo A*, ou seja, os pontos (18,14), (14,13), (10,1) e (15,7). O ponto que minimiza a distância neste caso é o ponto (17,4). O diagrama de contorno e a representação dos pontos no cenário são apresentados na Figura 8. Agora, faz sentido que não obtivemos como resultado um ponto centralizado, porque temos neste caso a influência dos obstáculos.

Figura 8 – Representação gráfica do cálculo do centroide com efeito dos obstáculos. (a) Cenário com os pontos pertencentes ao *cluster*. (b) Ponto que minimiza a distância para os quatro pontos escolhidos e o diagrama de contorno obtido quando consideramos obstáculos.



Fonte: Elaborado pelo autor

3.4 AVALIAÇÃO DO ALGORITMO DE CLUSTERIZAÇÃO INTEGRADO AO A*

Uma vez que estamos agora familiarizados com os procedimentos necessários para efetuarmos a integração do algoritmo de Lloyds ao algoritmo de A* podemos considerar as etapas enumeradas a seguir.

I) Caracterizar o cenário (comprimento e largura do cenário e coordenadas dos obstáculos).

II) Armazenar o cenário no tensor (calcular e armazenar todas as distâncias entre todas as coordenadas do cenário).

III) Carregar o tensor associado ao cenário previamente caracterizado.

IV) Escolher o valor de “k”.

V) Escolher as coordenadas iniciais dos centroides.

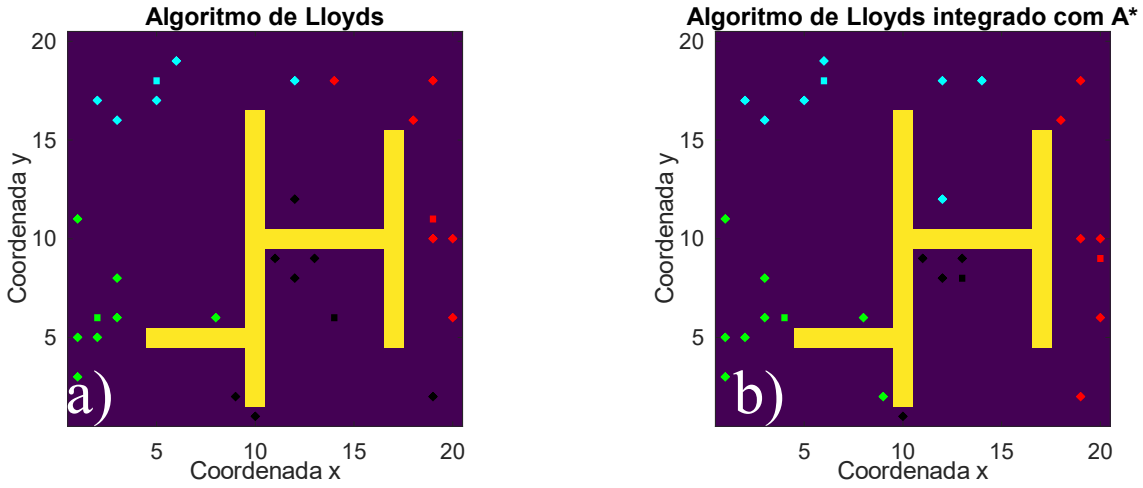
VI) Escolher as coordenadas dos usuários.

VII) Iniciar o algoritmo de Lloyds integrado com A*

Buscando apresentar o funcionamento do algoritmo de maneira comparativa decidimos simular o mesmo cenário tanto para o algoritmo de Lloyds clássico quanto para o algoritmo integrado com A*, mostrando os diferentes resultados obtidos para os dois métodos. Novamente, utilizando os pontos (18,14), (14,13), (10,1) e (15,7) agora como posições iniciais dos centroides e considerando 25 pontos posicionados de forma aleatória para representar os usuários realizamos a clusterização utilizando os dois métodos previamente mencionados. A Figura 9 representa o resultado obtido.

As posições finais dos centroides variam muito de um método para o outro. Isto significa que a presença de obstáculos influencia de forma significativa no cálculo do ponto que minimiza a distância, de maneira que quando temos obstáculos próximos este ponto não ocupa uma posição centralizada com relação aos pontos utilizados no cálculo do ponto que minimiza a distância como mencionamos anteriormente. No algoritmo de Lloyds tradicional o ponto que minimiza a distância é meramente o cálculo da média, portanto, o ponto que minimiza a distância é um ponto bem centralizado com relação

Figura 9 – Resultado da clusterização considerando centroides iniciais em (18,14), (14,13), (10,1) e (15,7) utilizando (a) o algoritmo de Lloyds clássico e (b) o algoritmo A* integrado ao algoritmo de Lloyds para os quatro pontos escolhidos definidos como centroides iniciais.



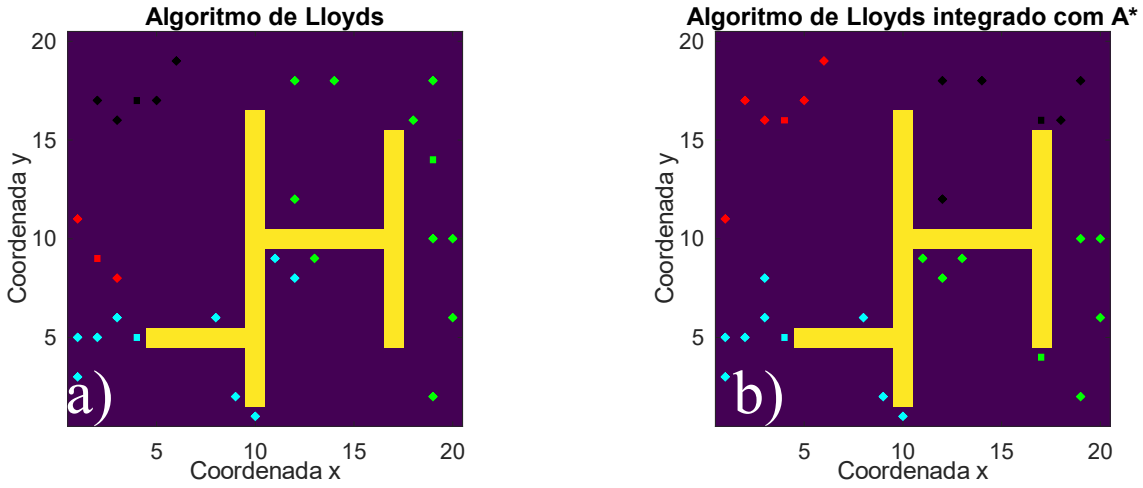
Fonte: Elaborado pelo autor

aos outros pontos que contemplamos clusterizar, no entanto, isto não acontece no algoritmo integrado, exceto quando consideramos pontos que não sofrem influência dos obstáculos (ver Figura 7). Tenhamos um olhar agora para a relação entre os pontos clusterizados e os centroides, mais precisamente para os pontos próximos de obstáculos. Enquanto os obstáculos são totalmente ignorados no algoritmo de Lloyds clássico, no caso do algoritmo integrado um ponto dificilmente é associado a um centroide que está cercado por obstáculos. Isto evidencia que encontramos a solução para uma das limitações do algoritmo de Lloyds que foi identificada anteriormente no início do trabalho: o algoritmo integrado que obtemos como resultado final considera de fato os obstáculos presentes no cenário simulado.

Decidimos efetuar uma última simulação, desta vez para posições aleatórias de centroides iniciais. Apresentamos na Figura 10 o resultado final. Novamente, poucos pontos submetidos ao processo de clusterização são associados a centroides obstruídos por obstáculos. Tenhamos um olhar atento para o *cluster* representado pela cor ciano cuja posição final de centroide foi a mesma nos dois métodos. Os dois pontos localizados no centro do cenário naturalmente pertencem a este *cluster* quando consideramos o algoritmo de Lloyds, no entanto, quando consideramos o algoritmo integrado com a presença dos obstáculos eles passam a pertencer a outro *cluster*. Este resultado é esperado, pois as distâncias destes pontos até o centroide representado pela cor ciano deve aumentar significativamente quando consideramos obstáculos no cálculo da distância. Agora, quando consideramos os cinco pontos localizados na região noroeste não podemos notar nenhuma diferença. Este resultado também é esperado, uma vez que estes pontos estão muito próximos do centroide localizado nesta região e não existe a presença de obstáculos nas adjacências.

Finalmente, quantificamos a diferença obtida com o cálculo das distâncias por meio dos dois algoritmos. Com esta finalidade, acessamos os valores das distâncias entre os usuários e os centroides calculadas computacionalmente com o algoritmo A* integrado ao algoritmo de Lloyds. No cálculo da distância total do algoritmo A* consideramos os nós que pertencem ao caminho ótimo. Estes nós

Figura 10 – Resultado da clusterização considerando centroides iniciais em posições aleatórias utilizando (a) o algoritmo de Lloyds clássico e (b) o algoritmo A* integrado ao algoritmo de Lloyds para os quatro pontos escolhidos definidos como centroides iniciais.



Fonte: Elaborado pelo autor

correspondem a todos os nós percorridos pelo algoritmo, desde o nó de origem até o nó de destino, porém não consideramos em nossas simulações o nó de origem. Portanto, o algoritmo produz alguns valores de distâncias iguais a “0” quando calculamos a distância entre nós adjacentes. Por esta razão também, o resultado final apresenta um pequeno erro de “1” quando partimos do nó de origem em um ângulo de 90 graus ou “1,4142” quando partimos do nó de origem com ângulos de 45 graus. Na Tabela 3 apresentamos os valores de distâncias obtidos considerando os centroides calculados com o algoritmo A* integrado ao algoritmo de Lloyds para a situação (posições de usuários e centroides) ilustrada na Figura 10 b). Na Tabela 4 apresentamos os valores obtidos com o algoritmo de Lloyds clássico, i.e., a situação ilustrada na Figura 10 a). Uma vez encontrados os centroides (independentemente do método) as distâncias desde os elementos até os centroides são calculadas utilizando o algoritmo A*. Então, observamos o seguinte: alguns valores de distâncias quando consideramos o algoritmo de Lloyds clássico são grandes, especificamente os valores referentes aos usuários “8”, “9” e “18” na tabela 4, em que obtemos “14,657”, “14,071” e “16,485” respectivamente. Estes valores estão muito elevados quando comparados com os valores obtidos na tabela 3 referentes ao algoritmo A* integrado ao algoritmo de Lloyds. No algoritmo integrado obtemos como valor máximo de distância 7,0711 para o usuário “19”. Observamos mais um fato. A distância média total é menor quando utilizamos o algoritmo integrado que desenvolvemos. Obtemos 3,0833 enquanto que para o algoritmo de Lloyds (clássico) a distância média total resulta em 4,7. Isto comprova quantitativamente que o algoritmo integrado produz redução nos valores de distâncias entre os usuários e os centroides quando comparado ao algoritmo de Lloyds, cumprindo o objetivo inicial de considerar a presença dos obstáculos presentes no cenário de simulação no cálculo das distâncias.

Usuários	Coordenada x	Coordenada y	*Cluster*	Distância	Dist. média	Dist. média total
1	1	3	1	2,4142	2,6856	3,0833
2	1	5	1	2		
3	2	5	1	1		
4	3	6	1	0		
5	3	8	1	2		
6	8	6	1	3		
7	9	2	1	4,8284		
8	10	1	1	6,2426		
9	1	11	2	5,2426	1,8142	
10	2	17	2	1,4142		
11	3	16	2	0		
12	5	17	2	0		
13	6	19	2	2,4142		
14	12	12	3	5,2426	2,7799	
15	12	18	3	4,8284		
16	14	18	3	2,4142		
17	18	16	3	0		
18	19	18	3	1,4142		
19	11	9	4	7,0711	4,6610	
20	12	8	4	5,2426		
21	13	9	4	5,2426		
22	19	2	4	1,4142		
23	20	6	4	2,4142		
24	19	10	4	5,4142		
25	20	10	4	5,8284		

Tabela 3 – Distâncias entre os usuários e os centroides considerando o algoritmo de Lloyds integrado com A*.

Usuários	Coordenada x	Coordenada y	*Cluster*	Distância	Dist. média	Dist. média total
1	1	3	1	2,4142	5,3570	4,7
2	1	5	1	2		
3	2	5	1	1		
4	3	6	1	0		
5	8	6	1	3		
6	9	2	1	4,8284		
7	10	1	1	6,2426		
8	11	9	1	14,657		
9	12	8	1	14,071		
10	3	8	2	0	0,5	
11	1	11	2	1		
12	3	16	3	0	0,60355	
13	2	17	3	1		
14	5	17	3	0		
15	6	19	3	1,4142		
16	19	2	4	11	6,5869	
17	20	6	4	7,4142		
18	13	9	4	16,485		
19	20	10	4	3,4142		
20	19	10	4	3		
21	12	12	4	8,0711		
22	18	16	4	1		
23	19	18	4	3		
24	14	18	4	5,2426		
25	12	18	4	7,2426		

Tabela 4 – Distâncias entre os usuários e os centroides considerando o algoritmo de Lloyds clássico.

4 CONCLUSÕES E TRABALHO FUTURO

No presente trabalho foi desenvolvido um novo algoritmo que soluciona uma das limitações do algoritmo de Lloyds, ou seja, é capaz de executar a clusterização em um cenário que possui obstáculos. Este novo algoritmo é de interesse para o projeto de redes de telecomunicações uma vez que é capaz de prover os benefícios da clusterização (aumento de escalabilidade e capacidade) em cenários realistas. Efetuamos a simulação com diferentes posições de centroides iniciais e conseguimos através de uma análise comparativa verificar que de fato o algoritmo é capaz de levar em conta obstáculos diferentemente do algoritmo clássico. Através de uma análise quantitativa, conseguimos também verificar que ocorre uma redução nos valores das distâncias obtidas mediante o algoritmo de Lloyds. Para a situação específica que analisamos quantitativamente, a distância média total foi reduzida de 4,7 para 3,0833 quando consideramos o algoritmo desenvolvido ao invés do algoritmo de Lloyds. O algoritmo que obtemos como resultado final é essencialmente a integração do algoritmo A* ao K-means (a propósito: não encontramos esta integração anteriormente na literatura). Dentre as limitações do algoritmo integrado obtido como produto final temos o pequeno erro decorrente do fato de não considerarmos a coordenada de origem no cálculo do percurso ótimo e isto resulta em valores de distâncias iguais a “0” ou com um pequeno erro de “1” e “1,414”. É importante mencionar que nossos estudos ficaram restritos ao algoritmo de Lloyds.

Para trabalhos futuros consideramos a utilização de A* integrado com outros algoritmos de clusterização. Podemos alternativamente modificar o algoritmo de A* (ou utilizar algum outro algoritmo de busca de caminho). Consideramos também a utilização de traçado de raios no processo de busca de caminhos ótimos. Com este método aparentemente vamos obter maior redução da distância média total e manteremos, é claro, a capacidade de levar em conta os obstáculos presentes no cenário durante o processo de clusterização.

REFERÊNCIAS

AL-HADDAD, U. A.; ALDABBAGH, G. A classification and comparison between clustering algorithms for wireless networks. In: THE STEERING COMMITTEE OF THE WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER .... **Proceedings of the International Conference on Wireless Networks (ICWN)**. [S.l.], 2015. p. 46.

ALPAYDIN, E. **Introduction to machine learning**. [S.l.]: MIT press, 2020.

ARTHUR, D.; VASSILVITSKII, S. k-means++: the advantages of careful seeding, p 1027–1035. **Society for Industrial and Applied Mathematics**, 2007.

AYTAÇ, E. Unsupervised learning approach in defining the similarity of catchments: Hydrological response unit based k-means clustering, a demonstration on western black sea region of turkey. **International Soil and Water Conservation Research**, Elsevier, v. 8, n. 3, p. 321–331, 2020.

BAJCSY, R.; ALOIMONOS, Y.; TSOTSOS, J. K. Revisiting active perception. **Autonomous Robots**, Springer, v. 42, n. 2, p. 177–196, 2018.

BOCK, H.-H. Origins and extensions of the k-means algorithm in cluster analysis. **Electronic Journal for History of Probability and Statistics**, v. 4, n. 2, p. 1–18, 2008.

CISCO, U. Cisco annual internet report (2018–2023) white paper. **Cisco: San Jose, CA, USA**, 2020.

CRAWFORD, C. **Byte magazine volume 07 number 12 - game plan 1982 : Free Download, borrow, and streaming**. 1982. Disponível em: <https://archive.org/details/byte-magazine-1982-12/page/n101/mode/2up?view=theater>.

DRINEAS, P. et al. Clustering large graphs via the singular value decomposition. **Machine learning**, Springer, v. 56, n. 1, p. 9–33, 2004.

DRIVER, H. E.; KROEBER, A. L. **Quantitative expression of cultural relationships**. [S.l.]: Berkeley: University of California Press, 1932. v. 31.

EIGENCHRIS. **Tensors for beginners 0: Tensor definition**. YouTube, 2017. Disponível em: <https://www.youtube.com/watch?v=TvxmkZmBa-k>.

GUPTA, P.; KUMAR, P. The capacity of wireless networks. **IEEE Transactions on Information Theory**, v. 46, n. 2, p. 388–404, 2000.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE Transactions on Systems Science and Cybernetics**, v. 4, n. 2, p. 100–107, 1968.

HUSSAIN, S. F.; HARIS, M. A k-means based co-clustering (kcc) algorithm for sparse, high dimensional data. **Expert Systems with Applications**, Elsevier, v. 118, p. 20–34, 2019.

JAIN, A. K. Data clustering: 50 years beyond k-means. **Pattern recognition letters**, Elsevier, v. 31, n. 8, p. 651–666, 2010.

JANCEY, R. Multidimensional group analysis. **Australian Journal of Botany**, CSIRO Publishing, v. 14, n. 1, p. 127–130, 1966.

LLOYD, S. Least squares quantization in PCM. **Transactions on Information Theory**, IEEE, v. 28, n. 2, p. 129–137, 1982.

LOPEZ, C. et al. An unsupervised machine learning method for discovering patient clusters based on genetic signatures. **Journal of Biomedical Informatics**, v. 85, p. 30–39, 2018. ISSN 1532-0464. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1532046418301308>.

MACQUEEN, J. et al. Some methods for classification and analysis of multivariate observations. In: OAKLAND, CA, USA. **Proceedings of the fifth Berkeley symposium on mathematical statistics and probability**. [S.l.], 1967. v. 1, n. 14, p. 281–297.

MITRA, R.; NANDY, D. et al. A survey on clustering techniques for wireless sensor network. **International Journal of Research in Computer Science**, Citeseer, v. 2, n. 4, p. 51–57, 2012.

NILSSON, N. J. **A mobile automaton: An application of artificial intelligence techniques**. [S.l.], 1969.

PANIGRAHY, K.; MISHRA, D. On reverse-order law of tensors and its application to additive results on moore–penrose inverse. **Revista de la Real Academia de Ciencias Exactas, Físicas y Naturales. Serie A. Matemáticas**, Springer, v. 114, n. 4, p. 1–21, 2020.

PÉREZ-ORTEGA, J. et al. The k-means algorithm evolution. In: **Introduction to Data Science and Machine Learning**. [S.l.]: IntechOpen, 2019.

REFAEILZADEH, P.; TANG, L.; LIU, H. Cross-validation. **Encyclopedia of database systems**, Springer, v. 5, p. 532–538, 2009.

RODRIGUEZ, M. Z. et al. Clustering algorithms: A comparative approach. **PloS one**, Public Library of Science San Francisco, CA USA, v. 14, n. 1, p. e0210236, 2019.

SCABURI, A. et al. Aplicação da metodologia k-means e do algoritmo de little para roteirização de veículos no projeto timm. In: . [S.l.: s.n.], 2021.

SHARMA, H. et al. Determining similarity in histological images using graph-theoretic description and matching methods for content-based image retrieval in medical diagnostics. **Diagnostic pathology**, v. 7, p. 134, 10 2012.

SHRIKANT, N.; SELVAKUMAR, A. Implementation of a* algorithm to autonomous robots-a simulation study. **Eng Technol Open Acc**, v. 1, n. 3, p. 88–91, 2018.

SHULGA, D. et al. **Tensor B-Spline Numerical Methods for PDEs: a High-Performance Alternative to FEM**. 2019.

STEINHAUS, H. **Sur la division des corps materiels en parties. Bull. Acad. Polon. Sci., C1. III vol IV: 801-804**. [S.l.]: cf, 1956.

TAN, P.-N.; STEINBACH, M.; KUMAR, V. **Introduction to data mining**. [S.l.]: Pearson Education India, 2016.

XU, K.; HONG, X.; GERLA, M. An ad hoc network with mobile backbones. In: IEEE. **2002 IEEE international conference on communications. Conference Proceedings. ICC 2002 (Cat. No. 02CH37333)**. [S.l.], 2002. v. 5, p. 3138–3143.

ZIDANE, I. M.; IBRAHIM, K. Wavefront and a-star algorithms for mobile robot path planning. In: SPRINGER. **International Conference on Advanced Intelligent Systems and Informatics**. [S.l.], 2017. p. 69–80.

APÊNDICE A – CÓDIGO DA FUNÇÃO QUE CALCULA AS DISTÂNCIAS UTILIZANDO O ALGORITMO A*

```
% Modificado de: Paul Premakumar (2022). A* (A Star) search for path planning tutorial.(https://www.mathworks.com/matlabcentral/fileexchange/26248-a-a-star-search-for-path-planning-tutorial), MATLAB Central File Exchange. Retrieved February 13, 2022.
% Sintaxis: functionAstrela(x1,y1,x2,y2,ScenarioMatrix)
% function    [path,distance] = functionAestrela(x1,y1,x2,y2,ScenarioMatrix)
function    [Optimal_path,distanciaAstar] = functionAestrela(MAX_X,MAX_Y,MAX_VAL,xTarget,yTarget,xStart,yStart,xval,yval,MAP,plotflag)
  path = 0;
  distance = 0;
  distanciaAstar = 0;
  Optimal_path = 0;
  if (MAP(xTarget,yTarget)==1)      % Caso obstaculo no destino
    distanciaAstar = Inf
  else
    axis([1 MAX_X 1 MAX_Y])
    grid on
    hold on
    OPEN=[];
    CLOSED=[];
    k=1;
    for i=1:MAX_X
      for j=1:MAX_Y
        if(MAP(i,j) == 1)
        CLOSED(k,1)=i;
        CLOSED(k,2)=j;
        k=k+1;
      end
    end
  end
  CLOSED_COUNT=size(CLOSED,1);
  xNode=xStart;
  yNode=yStart;
  OPEN_COUNT=1;
```

```
path_cost=0;
dist=sqrt((xTarget-xNode)^2 + (yTarget-yNode)^2);
goal_distance=dist;
OPEN(OPEN_COUNT,:)=insert_open(xNode,yNode,xNode,yNode,path_cost,
    goal_distance,goal_distance);
OPEN(OPEN_COUNT,1)=0;
CLOSED_COUNT=CLOSED_COUNT+1;
CLOSED(CLOSED_COUNT,1)=xNode;
CLOSED(CLOSED_COUNT,2)=yNode;
NoPath=1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INICIO DO ALGORITMO:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while((xNode ~= xTarget || yNode ~= yTarget) && NoPath == 1)
exp_array=expand_array(xNode,yNode,path_cost,xTarget,yTarget,
    CLOSED,MAX_X,MAX_Y);
exp_count=size(exp_array,1);
for i=1:exp_count
  flag=0;
  for j=1:OPEN_COUNT
    if(exp_array(i,1) == OPEN(j,2) && exp_array(i,2) == OPEN(j,3)
        )
    OPEN(j,8)=min(OPEN(j,8),exp_array(i,5));
    if OPEN(j,8)== exp_array(i,5)
      OPEN(j,4)=xNode;
      OPEN(j,5)=yNode;
      OPEN(j,6)=exp_array(i,3);
      OPEN(j,7)=exp_array(i,4);
      end;
      flag=1;
      end;
      end;
      if flag == 0
        OPEN_COUNT = OPEN_COUNT+1;
        OPEN(OPEN_COUNT,:)=insert_open(exp_array(i,1),exp_array(i
            ,2),xNode,yNode,exp_array(i,3),exp_array(i,4),
            exp_array(i,5));
        end;
        end;
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        %Fim do loop while
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        index_min_node = min_fn(OPEN,OPEN_COUNT,xTarget,yTarget);
        if (index_min_node ~= 1)
          xNode=OPEN(index_min_node,2);
          yNode=OPEN(index_min_node,3);
          path_cost=OPEN(index_min_node,6);
          CLOSED_COUNT=CLOSED_COUNT+1;
          CLOSED(CLOSED_COUNT,1)=xNode;
          CLOSED(CLOSED_COUNT,2)=yNode;
          OPEN(index_min_node,1)=0;
        else
          % Sair do loop se nao existirem mais nos possiveis de serem explorados.
          NoPath=0;
          end;
          end;
          i=size(CLOSED,1);
          Optimal_path=[];
          xval=CLOSED(i,1);
          yval=CLOSED(i,2);
          i=1;
          Optimal_path(i,1)=xval;
          Optimal_path(i,2)=yval;
          i=i+1;
          if ( (xval == xTarget) && (yval == yTarget))
            inode=0
            parent_x=OPEN(node_index(OPEN,xval,yval),4);
            parent_y=OPEN(node_index(OPEN,xval,yval),5);
            while( parent_x ~= xStart || parent_y ~= yStart)
            Optimal_path(i,1) = parent_x;
            Optimal_path(i,2) = parent_y;
            % Retorna ate o no de inicio.
            inode=node_index(OPEN,parent_x,parent_y);
            parent_x=OPEN(inode,4);
            parent_y=OPEN(inode,5);
            i=i+1;
            end;
            j=size(Optimal_path,1);
            if plotflag == 1
```

```
105                 p=plot(Optimal_path(j,1),Optimal_path(j,2),'bd','
                        HandleVisibility','off');
106                 j=j-1;
107                 for i=j:-1:1
108                   pause(.25);
109                   set(p,'XData',Optimal_path(i,1),'YData',
                          Optimal_path(i,2));
110                   drawnow ;
111                   end;
112                   plot(Optimal_path(:,1),Optimal_path(:,2),'w','
                          MarkerSize',20,'HandleVisibility','off');
113                 end
114                 path = Optimal_path
115                 comprimento_path = rows(Optimal_path)
116                 if (MAP(xStart,yStart)==1)      % Caso obstaculo no
                        inicio
117                   distanciaAstar = Inf
118                 else
119                   if ( (Optimal_path(1,1) == Optimal_path(1,2)) &&
                          (xStart == yStart) && rows(Optimal_path) == 1)
                           % Caso nos adjacentes e 90 graus
120                     distanciaAstar = 0
121                   else
122                     if rows(Optimal_path) == 1 % Caso nos
                            adjacentes apenas
123                       distanciaAstar = 0
124                     else
125                       for i=1:length(Optimal_path)-1
126                         distanciai = sqrt(((Optimal_path(i+1,1)-
                              Optimal_path(i,1))^2+(Optimal_path(i
                              +1,2)-Optimal_path(i,2))^2)); % Caso
                              geral
127                         distanciaAstar = distanciaAstar +
                              distanciai;
128                       end
129                     endif
130                   endif
131                 endif
132               else
133                 pause(1);
```

```
134             h=msgbox('Sorry, No path exists to the Target!','
                    warn');
135             uiwait(h,5);
136           end
137         endif
138         xStart
139         yStart
140         xTarget
141         yTarget
142         distanciaAstar
```

APÊNDICE B – CÓDIGO UTILIZADO PARA GERAR O TENSOR DE DISTÂNCIAS

```
### A* com mapa 2d
close all
clear all
clc
### Definir o comprimento e a largura do cenario
# Cenario 20x20:
MAX_X=20;
MAX_Y=20;
MAX_VAL=20;
MAP=0*(ones(MAX_X,MAX_Y));

# Parede de obstaculos 1
ind1_x = round(MAX_X*0.5);
ind2_x = round(MAX_X*0.85);
ind1_y = round(MAX_Y*0.5);
ind2_y = round(MAX_Y*0.5);
MAP(ind1_x:ind2_x,ind1_y) = ones(ind2_x-ind1_x+1,1);

# Parede de obstaculos 2
ind1_x = round(MAX_X*0.25);
ind2_x = round(MAX_X*0.5);
ind1_y = round(MAX_Y*0.25);
ind2_y = round(MAX_Y*0.25);
MAP(ind1_x:ind2_x,ind1_y) = ones(ind2_x-ind1_x+1,1);

# Parede de obstaculos 3
ind1_x = round(MAX_X*0.5);
ind2_x = round(MAX_X*0.5);
ind1_y = round(MAX_Y*0.1);
ind2_y = round(MAX_Y*0.8);
MAP(ind1_x,ind1_y:ind2_y) = ones(1,ind2_y-ind1_y+1);

# Parede de obstaculos 4
ind1_x = round(MAX_X*0.85);
ind2_x = round(MAX_X*0.85);
ind1_y = round(MAX_Y*0.25);
ind2_y = round(MAX_Y*0.75);
```

```
38 MAP(ind1_x,ind1_y:ind2_y) = ones(1,ind2_y-ind1_y+1);
39
40 # Representacao do cenario
41 xval = 0;
42 yval = 0;
43 axis([0 MAX_X 0 MAX_Y]) ;
44 imagesc(MAP');
45 colorbar;
46 set(gca, 'YDir','normal');
47
48 # Tensor de distancias
49 disttens = zeros(MAX_X,MAX_X,MAX_X,MAX_X);
50 for ind1=1:MAX_X
51   for ind2=1:MAX_X
52     for ind3=1:MAX_X
53       for ind4=1:MAX_X
54         [Optimal_path,distanciaAstar] = functionAestrela(MAX_X,
              MAX_Y,MAX_VAL,ind1,ind2,ind3,ind4,xval,yval,MAP,0);
55         disttens(ind1,ind2,ind3,ind4) = distanciaAstar;
56       endfor
57     endfor
58   endfor
59 endfor
60 # Salvar o cenario no tensor
61 save('tensor20x20.mat','disttens');
```

APÊNDICE C – CÓDIGO DO ALGORITMO DE LLOYDS INTEGRADO AO ALGORITMO A*

```
%%%
# Algortimo A* integrado com algoritmo de Lloyds.
%%%

clc
clear all
close all
load('disttens20x20.mat','disttens')
### Mapeamento para visualizacao
MAX_X = 20
MAX_Y = 20
MAX_VAL = 20
MAP=0*(ones(MAX_X,MAX_Y))
distanciamed = 0;
# Parede de obstaculos 1
ind1_x = round(MAX_X*0.5)
ind2_x = round(MAX_X*0.85)
ind1_y = round(MAX_Y*0.5)
ind2_y = round(MAX_Y*0.5)
MAP(ind1_x:ind2_x,ind1_y) = ones(ind2_x-ind1_x+1,1)
# Parede de obstaculos 2
ind1_x = round(MAX_X*0.25)
ind2_x = round(MAX_X*0.5)
ind1_y = round(MAX_Y*0.25)
ind2_y = round(MAX_Y*0.25)
MAP(ind1_x:ind2_x,ind1_y) = ones(ind2_x-ind1_x+1,1)
# Parede de obstaculos 3
ind1_x = round(MAX_X*0.5)
ind2_x = round(MAX_X*0.5)
ind1_y = round(MAX_Y*0.1)
ind2_y = round(MAX_Y*0.8)
MAP(ind1_x,ind1_y:ind2_y) = ones(1,ind2_y-ind1_y+1)
# Parede de obstaculos 4
ind1_x = round(MAX_X*0.85)
ind2_x = round(MAX_X*0.85)
ind1_y = round(MAX_Y*0.25)
```

```
37 ind2_y = round(MAX_Y*0.75)
38 MAP(ind1_x,ind1_y:ind2_y) = ones(1,ind2_y-ind1_y+1)
39 imagesc(MAP')
40 colorbar
41 set(gca, 'YDir','normal')
42 hold on
43 # Escolher o valor de "k"
44 nclusters = 6
45 k = nclusters
46 # Inicializacao dos centroides
47 for i=1:k
48   cx(i) = randi (MAX_VAL, 1);
49   cy(i) = randi (MAX_VAL, 1);
50 endfor
51 checkobst = zeros(1,k);
52 for i=1:k
53   disttens(cx(i),cy(i));
54   checkobst(i) = disttens(cx(i),cy(i));
55 endfor
56 checkobst;
57 nobst = ismember(Inf,checkobst);
58 while nobst>0
59   for i=1:k
60     cx(i) = randi (MAX_VAL, 1);
61     cy(i) = randi (MAX_VAL, 1);
62   endfor
63   checkobst = zeros(1,k);
64   for i=1:k
65     disttens(cx(i),cy(i));
66     checkobst(i) = disttens(cx(i),cy(i));
67   endfor
68   checkobst;
69   nobst = ismember(Inf,checkobst);
70 endwhile
71 p=0;
72 # Inicializacao dos usuarios
73 # Escolher o numero de usuarios
74 nusers=MAX_VAL
75 xUsuarios = randperm(MAX_VAL, nusers)
76 yUsuarios = randperm(MAX_VAL, nusers)
```

```
77
78 for i=1:nusers
79   disttens(xUsuarios(i),yUsuarios(i))
80   checkobst2(i) = disttens(xUsuarios(i),yUsuarios(i))
81 endfor
82 checkobst2;
83 nobst = ismember(Inf,checkobst2)
84 while nobst>0
85   for i=1:k
86     xUsuarios = randperm(MAX_VAL, nusers)
87     yUsuarios = randperm(MAX_VAL, nusers)
88   endfor
89   checkobst2 = zeros(1,15);
90   for i=1:nusers
91     disttens(xUsuarios(i),yUsuarios(i));
92     checkobst2(i) = disttens(xUsuarios(i),yUsuarios(i));
93   endfor
94   checkobst2
95   nobst = ismember(Inf,checkobst2);
96 endwhile
97 matrizcluster = zeros(k,length(xUsuarios));
98 for km=1:k
99   for i=1:length(xUsuarios)
100    matrizcluster(km,i)=disttens(xUsuarios(i),yUsuarios(i),cx(km),
         cy(km));
101  endfor
102 endfor
103 matrizcluster;
104 [valorminimo,vetorclusterold] = min(matrizcluster);
105 vetorclusterinicial = vetorclusterold;
106 vetorclusterinicial
107 vetorclusternew=0
108 for i=1:k
109   idx(i,:)= (vetorclusterold==i);
110 endfor
111 idx;
112 nusuarios={};
113 for i=1:k
114   nusuarios(i,:)=find(idx(i,:));
115 endfor
```

```
116 for idk=1:k
117   for ids=1:length(nusuarios{idk})
118     if ismember(idk,vetorclusterold)
119       cordx=xUsuarios(nusuarios{idk});
120       cordy=yUsuarios(nusuarios{idk});
121     endif
122   end
123   % Utilizar a distancia media para encontrar os novos centroides
        de cada cluster.
124   [m,n,k,l] = size(disttens);
125   for iy=1:m
126     for ix=1:n
127       for is=1:length(nusuarios{idk})
128         distanciaAstars=disttens(cordx(is),cordy(is),ix,iy);
129         sumv(is)=distanciaAstars;
130         distanciamed = sum(sumv)/length(nusuarios{idk});
131       endfor
132       matrizmeddist(ix,iy) = distanciamed;
133     endfor
134   endfor
135   matrizmeddist;
136   [minval,ind] = min(matrizmeddist(:));
137   [I,J] = ind2sub([size(matrizmeddist,1) size(matrizmeddist,2)],ind
        );
138   xCentroidnew(idk) = I;
139   yCentroidnew(idk) = J;
140   xCentroidnew(idk)
141   yCentroidnew(idk)
142   min(matrizmeddist(:));
143 endfor
144 matrizmeddist;
145 vetorclusterold
146 k=nclusters;
147 for idk=1:k
148   if ismember(idk,vetorclusterold)
149     cx = xCentroidnew;
150     cy = yCentroidnew;
151     for km=1:k
152       for i=1:length(xUsuarios)
153         matrizcluster(km,i)=disttens(xUsuarios(i),yUsuarios(i),cx(
```

```
            km),cy(km));
154       endfor
155     endfor
156   endif
157 endfor
158 cx;
159 cy;
160 matrizcluster;
161 for u=1:k
162   if not(ismember(u,vetorclusterold))
163     for i=1:length(xUsuarios)
164       matrizcluster(u,i)=Inf;
165     endfor
166   endif
167 endfor
168 matrizcluster;
169 [valorminimo,vetorclusterold] = min(matrizcluster);
170 vetorclusterold
171 for i=1:k
172   idx(i,:)= (vetorclusterold==i);
173 endfor
174 idx;
175 nusuarios={};
176 for i=1:k
177   nusuarios(i,:)=find(idx(i,:));
178 endfor
179 for idk=1:k
180   for ids=1:length(nusuarios{idk})
181     cordx=xUsuarios(nusuarios{idk});
182     cordy=yUsuarios(nusuarios{idk});
183   end
184   % Novamente, utilizar a distancia media para encontrar os novos
       centroides de cada cluster.
185   k=nclusters;
186   [m,n,k,l] = size(disttens);
187   for iy=1:m
188     for ix=1:n
189       for is=1:length(nusuarios{idk})
190         distanciaAstars=disttens(cordx(is),cordy(is),ix,iy);
191         sumv(is)=distanciaAstars;
```

```
192             distanciamed = sum(sumv)/length(nusuarios{idk});
193           endfor
194           matrizmeddist(ix,iy) = distanciamed;
195         endfor
196       endfor
197       matrizmeddist;
198       [minval,ind] = min(matrizmeddist(:));
199       [I,J] = ind2sub([size(matrizmeddist,1) size(matrizmeddist,2)],ind
            );
200       xCentroidnew(idk) = I;
201       yCentroidnew(idk) = J;
202       xCentroidnew(idk)
203       yCentroidnew(idk)
204       min(matrizmeddist(:));
205   endfor
206   matrizmeddist;
207   cx=xCentroidnew;
208   cy=yCentroidnew;
209   k=nclusters;
210   for km=1:k
211     for i=1:length(xUsuarios)
212       matrizcluster(km,i)=disttens(xUsuarios(i),yUsuarios(i),cx(km),
            cy(km));
213     endfor
214   endfor
215   matrizcluster;
216   [valorminimo,vetorclusternew] = min(matrizcluster);
217   vetorclusternew
218   matrizcluster
219   % Iterar novamente
220   while not(isequal(vetorclusterold,vetorclusternew))
221     matrizcluster = zeros(k,length(xUsuarios));
222     for km=1:k
223       for i=1:length(xUsuarios)
224         matrizcluster(km,i)=disttens(xUsuarios(i),yUsuarios(i),cx(km)
              ,cy(km));
225       endfor
226     endfor
227     matrizcluster;
228     [valorminimo,vetorclusterold] = min(matrizcluster);
```

```
229   vetorclusterold
230   for i=1:k
231     idx(i,:)= (vetorclusterold==i);
232   endfor
233   idx;
234   nusuarios={};
235   for i=1:k
236     nusuarios(i,:)=find(idx(i,:));
237   endfor
238   for idk=1:k
239     for ids=1:length(nusuarios{idk})
240       if ismember(idk,vetorclusterold)
241         cordx=xUsuarios(nusuarios{idk});
242         cordy=yUsuarios(nusuarios{idk});
243       endif
244     end
245     % Mais uma vez, utilizar a distancia media para encontrar os
          novos centroides de cada cluster.
246     [m,n,k,l] = size(disttens);
247     for iy=1:m
248       for ix=1:n
249         for is=1:length(nusuarios{idk})
250           distanciaAstars=disttens(cordx(is),cordy(is),ix,iy);
251           sumv(is)=distanciaAstars;
252           distanciamed = sum(sumv)/length(nusuarios{idk});
253         endfor
254         matrizmeddist(ix,iy) = distanciamed;
255       endfor
256     endfor
257     matrizmeddist;
258     [minval,ind] = min(matrizmeddist(:));
259     [I,J] = ind2sub([size(matrizmeddist,1) size(matrizmeddist,2)],
          ind);
260     xCentroidnew(idk) = I;
261     yCentroidnew(idk) = J;
262     xCentroidnew(idk)
263     yCentroidnew(idk)
264     min(matrizmeddist(:));
265   endfor
266   matrizmeddist;
```

```
267   vetorclusterold
268   k=nclusters;
269   for idk=1:k
270     if ismember(idk,vetorclusterold)
271       cx = xCentroidnew;
272       cy = yCentroidnew;
273       for km=1:k
274         for i=1:length(xUsuarios)
275           matrizcluster(km,i)=disttens(xUsuarios(i),yUsuarios(i),cx
                (km),cy(km));
276         endfor
277       endfor
278     endif
279   endfor
280   cx;
281   cy;
282   matrizcluster;
283   matrizcluster;
284   [valorminimo,vetorclusterold] = min(matrizcluster);
285   vetorclusterold
286   for i=1:k
287     idx(i,:)= (vetorclusterold==i);
288   endfor
289   idx;
290   nusuarios={};
291   for i=1:k
292     nusuarios(i,:)=find(idx(i,:));
293   endfor
294   for idk=1:k
295     for ids=1:length(nusuarios{idk})
296       cordx=xUsuarios(nusuarios{idk});
297       cordy=yUsuarios(nusuarios{idk});
298     end
299     k=nclusters;
300     [m,n,k,l] = size(disttens);
301     for iy=1:m
302       for ix=1:n
303         for is=1:length(nusuarios{idk})
304           distanciaAstars=disttens(cordx(is),cordy(is),ix,iy);
305           sumv(is)=distanciaAstars;
```

```
306             distanciamed = sum(sumv)/length(nusuarios{idk});
307           endfor
308           matrizmeddist(ix,iy) = distanciamed;
309         endfor
310       endfor
311       matrizmeddist;
312       [minval,ind] = min(matrizmeddist(:));
313       [I,J] = ind2sub([size(matrizmeddist,1) size(matrizmeddist,2)],
              ind);
314       xCentroidnew(idk) = I;
315       yCentroidnew(idk) = J;
316       xCentroidnew(idk)
317       yCentroidnew(idk)
318       min(matrizmeddist(:));
319     endfor
320     matrizmeddist;
321     cx=xCentroidnew;
322     cy=yCentroidnew;
323     k=nclusters;
324     for km=1:k
325       for i=1:length(xUsuarios)
326         matrizcluster(km,i)=disttens(xUsuarios(i),yUsuarios(i),cx(km)
              ,cy(km));
327       endfor
328     endfor
329     matrizcluster;
330     [valorminimo,vetorclusternew] = min(matrizcluster);
331     vetorclusternew
332     matrizcluster
333   endwhile
334   % Plotar usuarios e centroides
335   result1 = find(vetorclusternew==1)
336   plotar1x = zeros(1,length(result1));
337   plotar1y = zeros(1,length(result1));
338   for idd=1:length(result1)
339     plotar1x(idd) = xUsuarios(result1(idd));
340     plotar1y(idd) = yUsuarios(result1(idd));
341   endfor
342   plotar1x;
343   plotar1y;
```

```
344 result2 = find(vetorclusternew==2)
345 plotar2x = zeros(1,length(result2));
346 plotar2y = zeros(1,length(result2));
347 for idd=1:length(result2)
348   plotar2x(idd) = xUsuarios(result2(idd));
349   plotar2y(idd) = yUsuarios(result2(idd));
350 endfor
351 plotar2x;
352 plotar2y;
353 result3 = find(vetorclusternew==3)
354 plotar3x = zeros(1,length(result3));
355 plotar3y = zeros(1,length(result3));
356 for idd=1:length(result3)
357   plotar3x(idd) = xUsuarios(result3(idd));
358   plotar3y(idd) = yUsuarios(result3(idd));
359 endfor
360 plotar3x;
361 plotar3y;
362 result4 = find(vetorclusternew==4)
363 plotar4x = zeros(1,length(result4));
364 plotar4y = zeros(1,length(result4));
365 for idd=1:length(result4)
366   plotar4x(idd) = xUsuarios(result4(idd));
367   plotar4y(idd) = yUsuarios(result4(idd));
368 endfor
369 plotar4x;
370 plotar4y;
371 result5 = find(vetorclusternew==5)
372 plotar5x = zeros(1,length(result5));
373 plotar5y = zeros(1,length(result5));
374 for idd=1:length(result5)
375   plotar5x(idd) = xUsuarios(result5(idd));
376   plotar5y(idd) = yUsuarios(result5(idd));
377 endfor
378 plotar5x;
379 plotar5y;
380 result6 = find(vetorclusternew==6)
381 plotar6x = zeros(1,length(result6));
382 plotar6y = zeros(1,length(result6));
383 for idd=1:length(result6)
```

```
384   plotar6x(idd) = xUsuarios(result6(idd));
385   plotar6y(idd) = yUsuarios(result6(idd));
386 endfor
387 plotar6x;
388 plotar6y;
389 for idd=1:length(result1)
390   plot(plotar1x(idd),plotar1y(idd),'rd');
391   hold on;
392 endfor
393 for idd=1:length(result2);
394   plot(plotar2x(idd),plotar2y(idd),'bd');
395   hold on;
396 endfor
397 for idd=1:length(result3)
398   plot(plotar3x(idd),plotar3y(idd),'gd');
399   hold on;
400 endfor
401 for idd=1:length(result4)
402   plot(plotar4x(idd),plotar4y(idd),'md');
403   hold on;
404 endfor
405 for idd=1:length(result5)
406   plot(plotar5x(idd),plotar5y(idd),'cd');
407   hold on;
408 endfor
409 for idd=1:length(result6)
410   plot(plotar6x(idd),plotar6y(idd),'kd');
411   hold on;
412 endfor
413 % Coincidir o centroide com o usuario caso tenhamos apenas um
      usuario.
414 if length(result1)==1
415   cx(1)=plotar1x(1)
416   cy(1)=plotar1y(1)
417 endif
418 if length(result2)==1
419   cx(2)=plotar2x(1)
420   cy(2)=plotar2y(1)
421 endif
422 if length(result3)==1
```

```
423     cx(3)=plotar3x(1)
424     cy(3)=plotar3y(1)
425   endif
426   if length(result4)==1
427     cx(4)=plotar4x(1)
428     cy(4)=plotar4y(1)
429   endif
430   if length(result5)==1
431     cx(5)=plotar5x(1)
432     cy(5)=plotar5y(1)
433   endif
434   if length(result6)==1
435     cx(6)=plotar6x(1)
436     cy(6)=plotar6y(1)
437   endif
438   if length(result1)>0
439     plot(cx(1),cy(1),'r+');
440   endif
441   if length(result2)>0
442     plot(cx(2),cy(2),'b+');
443   endif
444   if length(result3)>0
445     plot(cx(3),cy(3),'g+');
446   endif
447   if length(result4)>0
448     plot(cx(4),cy(4),'m+');
449   endif
450   if length(result5)>0
451     plot(cx(5),cy(5),'c+');
452   endif
453   if length(result6)>0
454     plot(cx(6),cy(6),'k+');
455   endif
456   %%%%%%
```