UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Daniel Matheus Doebber

USO DE REDES NEURAIS PROFUNDAS PARA O APRENDIZADO DE FUNÇÕES HEURÍSTICAS PARA ALGORITMOS DE BUSCA DE CAMINHOS

Santa Maria, RS
2019

Daniel Matheus Doebber

USO DE REDES NEURAIS PROFUNDAS PARA O APRENDIZADO DE FUNÇÕES HEURÍSTICAS PARA ALGORITMOS DE BUSCA DE CAMINHO

Trabalho de Conclusão de Curso apresentado ao Programa de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM), como requisito parcial para a obtenção do título de **Bacharel em Ciência da Computação**.

Orientador: Luís Alvaro de Lima Silva

460

Santa Maria, RS

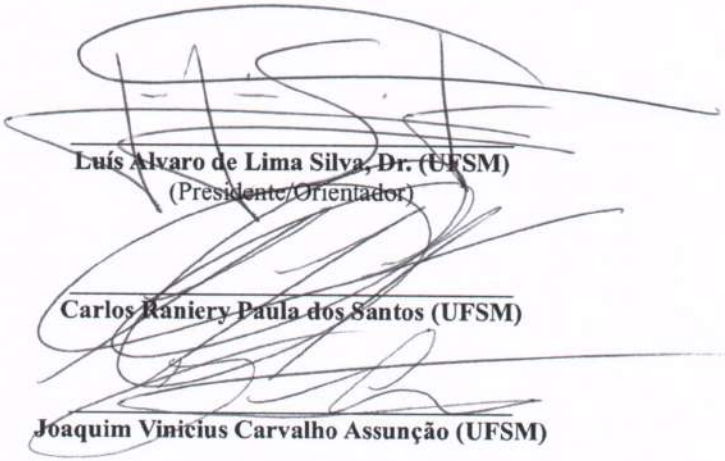
2019

Daniel Matheus Doebber

USO DE REDES NEURAIS PROFUNDAS PARA O APRENDIZADO DE FUNÇÕES HEURÍSTICAS PARA ALGORITMOS DE BUSCA DE CAMINHO

Trabalho de Conclusão de Curso apresentado ao Programa de Graduação em Ciência da Computação da Universidade Federal de Santa Maria (UFSM), como requisito parcial para a obtenção do título de **Bacharel em Ciência da Computação**.

Aprovado em 02 de dezembro de 2019:



Luís Alvaro de Lima Silva, Dr. (UFSM)
(Presidente/Orientador)

Carlos Raniery Paula dos Santos (UFSM)

Joaquim Vinicius Carvalho Assunção (UFSM)

AGRADECIMENTOS

Agradeço primeiramente a todas as pessoas que me apoiaram e incentivaram durante toda a graduação. Em especial, minha mãe Marli, namorada Giovana e irmã Aline, sem vocês provavelmente não estaria onde estou. Também agradeço a outras pessoas muito importantes, meus sogros, Cinara e Luiz Vanelli, por todos os conselhos e incentivos proporcionados. Agradeço a minha irmã Daiana, que mesmo distante, sempre esteve torcendo pelo meu sucesso. Em especial também, ao meu pai Irajá, que mesmo não estando mais aqui, tenho certeza que estaria imensamente feliz por tudo.

Agradeço aos professores da graduação, por todo conhecimento compartilhado e auxílio no aprendizado, além da disponibilidade em auxiliar no que fosse necessário. Em especial, agradeço ao meu orientador Luís Alvaro de Lima Silva, pela orientação neste trabalho, atenção e ensinamentos. Ainda, o agradeço por ter me selecionado para desenvolver pesquisa em sua equipe. Foi uma experiência que contribui muito para minha formação. Agradeço aos professores Carlos Raniery e Joaquim Assunção, pela disponibilidade de participarem da minha banca avaliadora, contribuindo para o aperfeiçoamento deste trabalho.

Agradeço aos meus colegas e amigos, Caroline, Francisco e Gabriel por todo companheirismo durante toda graduação.

Agradeço à equipe de Inteligência Artificial do projeto SIS-ASTROS, que me auxiliaram na adaptação ao projeto, no aprendizado e também na realização de minhas pesquisas e deste trabalho. Em especial, agradeço aos colegas Eliakim e Caroline, por todo conhecimento compartilhado, horas discutindo como melhorar um código e todas risadas proporcionadas por isso.

RESUMO

USO DE REDES NEURAIS PROFUNDAS PARA O APRENDIZADO DE FUNÇÕES HEURÍSTICAS PARA ALGORITMOS DE BUSCA DE CAMINHO

AUTOR: Daniel Matheus Doebber
ORIENTADOR: Luís Alvaro de Lima Silva

A busca de caminhos (pathfinding) é uma importante área da Inteligência Artificial que vem sendo pesquisada pelo emprego de redes neurais profundas. Mais especificamente, a pesquisa está voltada para o desenvolvimento de novos algoritmos de busca de caminhos guiados por funções heurísticas, onde essas funções são implementadas por redes neurais profundas. Em essência, funções heurísticas são usadas por variados algoritmos de busca para estimar a distância de uma posição atual até uma posição objetivo, onde a posição atual (representada como um nodo do espaço de buscas) que tem a melhor estimativa é priorizada no processo de investigação do menor caminho entre tais posições. Funções heurísticas tradicionais, como a distância Euclidiana e distância de Manhattan, por exemplo, guiam de forma ótima a busca de caminhos em mapas que não possuem muitos obstáculos. Porém, quando obstáculos são comuns no ambiente de busca, essas funções heurísticas começam a perder a eficiência visto que muitos nós do espaço de busca podem ter que ser analisados. Este trabalho propõe e testa o emprego de redes neurais profundas no aprendizado de funções heurísticas que permitam melhor guiar algoritmos de busca de caminhos. O objetivo é verificar se tais funções heurísticas fundamentadas em resultados de redes neurais profundas permitem ou não a obtenção de caminhos com boa qualidade em mapas de navegação representados como grades regulares. Experimentos desenvolvidos foram realizados pela execução de buscas de caminhos utilizando funções heurísticas tradicionais, as quais foram comparadas com funções aprendidas por uma rede neural profunda implementada. Os resultados mostram que o uso de redes neurais profundas permite reduzir a expansão dos nodos do espaço de busca mantendo a qualidade dos caminhos e, em algumas situações, reduzir o tempo de execução do algoritmo de busca.

Palavras-chave: Redes neurais. Busca de caminhos. Função heurística. Inteligência artificial.

ABSTRACT

USE OF NEURAL NETWORKS TO LEARNING HEURISTIC FUNCTIONS FOR PATHFINDING ALGORITHMS

AUTHOR: Daniel Matheus Doebber
ADVISOR: Luís Alvaro de Lima Silva

Pathfinding is an important area of Artificial Intelligence that has been researched through the use of deep neural networks. More specifically, the research focuses on the development of new path search algorithms guided by heuristic functions, where these functions are implemented by deep neural networks. In essence, heuristic functions are used by various search algorithms to estimate the distance from a current position to an objective position, where the current position (represented as a search space node) that has the best estimate is prioritized in the research process the shortest path between such positions. Traditional heuristic functions such as Euclidean distance and Manhattan distance, for example, optimally guide the search for paths on maps that do not have many obstacles. However, when obstacles are common in the search environment, these heuristic functions begin to lose efficiency as many search space nodes may need to be analyzed. This paper proposes and tests the use of deep neural networks in the learning of heuristic functions that better guide path search algorithms. The objective is to verify if such heuristic functions based on deep neural network results allow to obtain good quality paths in navigation maps represented as regular grids. Developed experiments were performed by performing path searches using traditional heuristic functions, which were compared with functions learned by a deep neural network implemented. The results show that the use of deep neural networks allows to reduce the expansion of the search space nodes maintaining the quality of the paths and, in some situations, reduce the search algorithm execution time.

Keywords: Neural networks. Pathfinding. Heuristic function. Artificial intelligence.

LISTA DE FIGURAS

LISTA DE ALGORITMOS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A Inteligência Artificial abrange diversos ramos de estudo, dentre eles, a área de redes neurais (Gama *et al.*, 2011). Redes neurais profundas vêm cada vez tendo uma maior relevância na literatura, sendo considerado o estado da arte para solução de diversos problemas como reconhecimento de objetos, reconhecimento da fala, diagnósticos médicos, entre outros (Alom *et al.*, 2018). Mais recentemente, uma área que vem aderindo o uso de redes neurais é a de busca de caminhos, mais especificamente sobre a pesquisa e desenvolvimento de algoritmos de busca de caminhos guiados por heurísticas (Ariki e Narihira, 2019; Takahashi *et al.*, 2019; Wang *et al.*, 2019). Contudo, tais funções heurísticas não são utilizadas somente para busca de caminhos, mas também para diversos problemas de busca baseados no emprego de heurísticas, como o cubo mágico, quebra-cabeças de peças deslizantes, entre outros (Agostinelli *et al.*, 2019).

Algoritmos heurísticos de busca de caminhos encontram o caminho que possuir o menor custo, onde o processo de busca é guiado por funções heurísticas. Em essência, uma função heurística estima a distância de uma posição atual até uma posição objetivo. A qualidade de tal estimativa está associada ao quão a função se aproxima da distância real até o objetivo e influencia diretamente no desempenho do algoritmo de busca. As funções heurísticas tradicionais, como a distância Euclidiana e distância de Manhattan, guiam bem a busca dos caminhos que não possuem muitos obstáculos. Porém, quando obstáculos são comuns no ambiente de busca, essas funções heurísticas começam a perder a eficiência, pois elas não consideram característica importantes do ambiente na realização das computações.

A partir desse cenário de pesquisa, esse trabalho propõe e testa o emprego de redes neurais profundas no aprendizado de funções heurísticas que possam melhor guiar os algoritmos de busca de caminhos. O objetivo é verificar se tais funções permitem ou não a obtenção de caminhos mais consistentes com o caminho executado por agentes reais imersos em ambientes do mundo real ao mesmo tempo que mantenham a qualidade dos caminhos (por exemplo, menor distância entre dois pontos) encontrados. A aplicação do trabalho está direcionada a sistemas de simulação e jogos digitais, embora tais técnicas possam ser empregadas na solução de outras aplicações. Em geral, o uso de tais técnicas visa reduzir o custo computacional e melhorar a qualidade dos caminhos computados nestes ambientes.

Em termos práticos, o objetivo do trabalho é propor uma arquitetura de rede neural profunda e treiná-la para que aprenda uma função heurística a ser usada em algoritmos de busca de caminhos. Em detalhes, o trabalho visa:

- Apresentar uma revisão da literatura a respeito de redes neurais profundas aplicadas a problemas de navegação.
- Descrever formas de realizar a utilização de redes neurais profundas em conjunto com algoritmos de busca de caminhos.

Este TCC está inserido no contexto do SIS-ASTROS (Sis-Astros, 2014). O SIS-ASTROS é um projeto de pesquisa e desenvolvimento para a implementação de um sistema de simulação tática virtual. O sistema de simulação fornece suporte para o treinamento tático militar de baterias de artilharia inseridos em terrenos de larga escala de batalha virtual. Este trabalho dá sequência a pesquisas realizadas no contexto de navegação do simulador (Brondani *et al.*, 2017; Brondani *et al.*, 2018; Brondani *et al.*, 2019).

O trabalho se organiza da seguinte forma: o capitulo 2 apresenta uma revisão bibliográfica sobre redes neurais, treinamento de redes neurais, algoritmos de busca de caminhos e funções heurísticas. O capítulo 3 apresenta a proposta deste trabalho, os mapas de navegação utilizados, a arquitetura de rede neural abordada e como a função heurística aprendida pela rede neural foi utilizada juntamente com o algoritmo de busca A*. O capítulo 4 apresenta os experimentos realizados e os resultados obtidos com o uso da técnica proposta. Por fim, o capitulo 5 apresenta as conclusões obtidas, e descreve trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

2.1 REDES NEURAIS

No cotidiano, um ser humano tem a capacidade de realizar diversas ações, como conversar, se locomover, realizar cálculos matemáticos, entre outras. No contexto de redes neurais artificiais (Gama *et al.*, 2011), a capacidade de realizar tais ações está vinculada a múltiplas camadas de neurônios que interagem entre si. Na literatura, essa arquitetura teve sucesso, por exemplo, em aplicações de reconhecimento de objetos (Krizhevsky *et al.*, 2012), reconhecimento de fala (Hinton *et al.*, 2012), tradução de textos (Sutskever *et al.*, 2014), jogos digitais (Mnih *et al.*, 2015) e diagnósticos médicos (Amato *et al.*, 2013).

2.1.1 Redes Neurais Artificiais

Redes neurais artificiais (RNA) são modeladas a partir de neurônios artificiais. Um neurônio artificial é descrito por um modelo matemático, no qual recebe a entrada ($x = x_1, x_2, \dots, x_n$) e realiza uma soma ponderada com pesos ($w = w_1, w_2, \dots, w_n$) dessas entradas. Ao final, o resultado da soma é adicionado um *bias b*, e então passa por uma função de ativação *σ*, resultando em uma saída final *z*. Um neurônio pode ser descrito pela seguinte equação:

$$z = \sigma(\sum_{i=0}^{n} x_i w_i + b)$$

A função de ativação realiza a conversão do sinal de entrada para um sinal de saída, definindo as conexões entre os neurônios artificiais. O objetivo das funções de ativação é adicionar uma não linearidade ao modelo matemático. Funções de ativação frequentemente usadas e exploradas neste trabalho são:

- **Sigmoide:** A característica da sigmoide é o formado de 'S'. Essa função é frequentemente utilizada em modelos de predição de probabilidades (Figura 2.1 A).

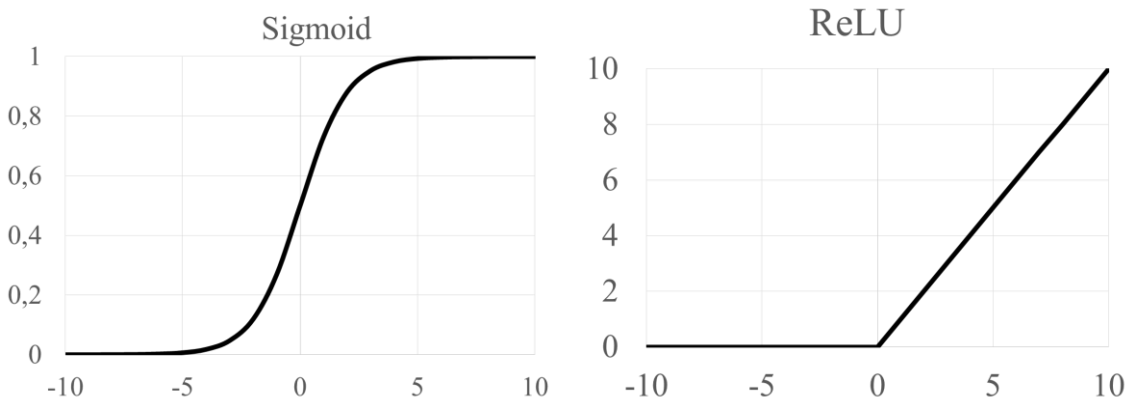
$$\sigma(\mathrm{z}) = \frac{1}{1 + e^{-z}}$$

- **ReLU (Unidade Linear Retificada).** Essa função limita inferiormente a saída para 0. A principal vantagem desta função é o baixo custo computacional em relação a função sigmoide (Maas *et al.*, 2013) (Figura 2.1 B).

$$R(z) = MAX(0, z)$$

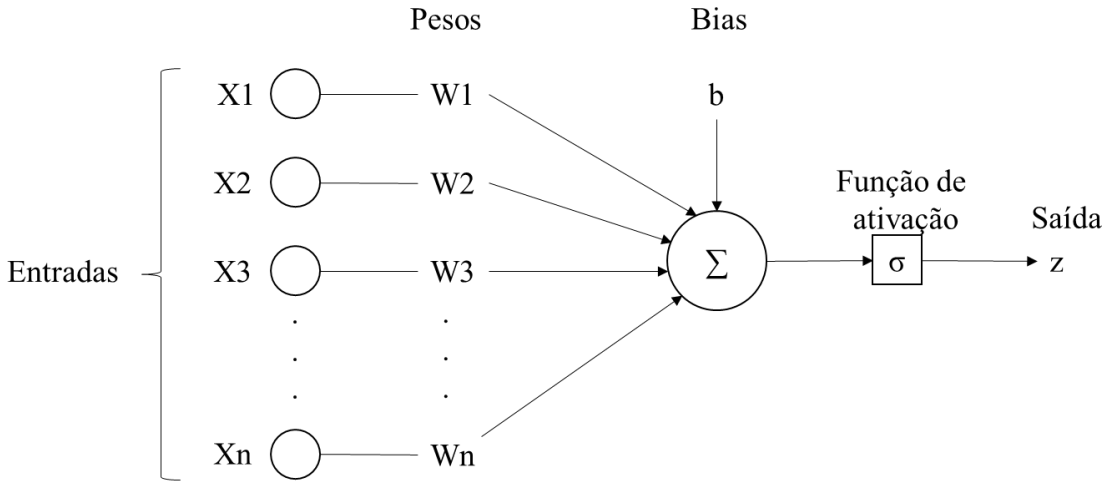
A Figura 2.2 representa a estrutura de um neurônio artificial. Ajustado com os diferentes pesos e bias, um neurônio é treinado para aproximar uma função de acordo com a entrada.

Figura 2.1 - Gráficos das funções de ativação Sigmoide (a) e ReLU (b).



Fonte: Autor

Figura 2.2 - Representação de um neurônio artificial.



Fonte: Autor

Tradicionalmente, redes neurais artificiais são formadas por somente três camadas (*layers*): camada de entrada, camada oculta e camada de saída. A camada de entrada é composta pelos neurônios artificiais que recebem os dados de entrada. O número de neurônios

nessa camada é igual ao número de dados de entrada. A camada oculta é composta pelos neurônios que realizam o processamento dos dados recebidos da camada de entrada. A camada de saída disponibiliza o resultado da rede.

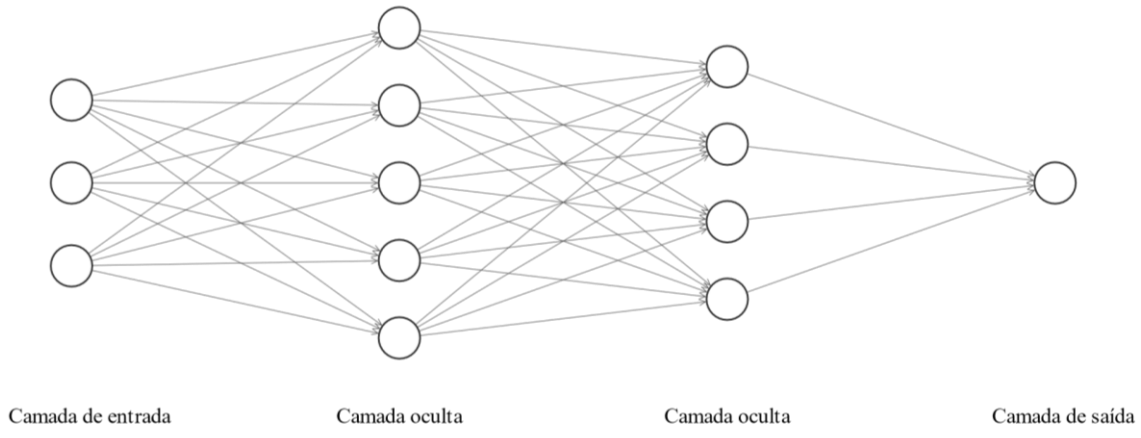
Em geral, as redes neurais são implementadas com dois objetivos, classificação e regressão. As redes neurais de classificação têm como objetivo rotular a entrada, isto é, dado uma entrada ela tenta predizer a qual classe a entrada pertence, por exemplo dado uma imagem de entrada, a rede diz se o que tem na imagem é da classe A ou B. As redes neurais para regressão tentam prever um valor real para a entrada, por exemplo dado as características de uma casa, a rede tenta prever o valor da mesma.

2.1.2 Redes Neurais Profundas

As redes neurais profundas (DNN) são redes neurais artificiais com mais de uma camada de neurônios oculta (Nielsen, 2015). Na camada oculta, podem existir quantidades arbitrárias de camadas, cada uma contendo diferentes números de neurônios. A quantidade de camadas e neurônios podem variar de acordo com a aplicação da rede. Geralmente, o número de neurônios e camadas são definidos de forma empírica (Gama *et al.*, 2011).

A Figura 2.3 representa um exemplo de rede neural profunda. Esta rede é composta por uma camada de entrada com três neurônios, duas camadas ocultas, uma com 5 neurônios e outra com 4 neurônios. Por último, a camada de saída possui apenas um neurônio.

Figura 2.3 - Representação de uma rede neural profunda.



Em geral, as arquiteturas de redes neurais são divididas em duas categorias: *Feedforward neural network* (FNN) *e Recurrent neural network* (RNN) (Alom *et al.*, 2018).

As FNNs possuem uma camada de entrada, uma camada de saída e diversas camadas ocultas. A principal característica destas redes é que os valores computados são sempre passados para frente nas diferentes camadas da rede. Existem diversas arquiteturas de FNNs. Na arquitetura de redes neurais totalmente conectas (*fully connected*), todos os neurônios de uma camada são conectados com todos neurônios da camada seguinte. Esse tipo de rede se torna custosa à medida que o tamanho da entrada aumenta. As redes neurais convolucionais (CNN) utilizam filtros de convolução para reduzir o tamanho da entrada (Goodfellow *et al.*, 2016). As CNNs também podem ser implementadas como RNNs.

As RNNs também possuem uma camada de entrada e saída. A diferença está nas camadas ocultas visto que elas possuem laços que permite que as informações persistam na rede. Tais redes são projetadas para reconhecimento de sequências de dados, como texto, séries numéricas e linguagem natural, por exemplo. Esse tipo arquitetura permite que a rede fique ciente do contexto memorizando as ativações anteriormente computadas na rede.

2.2 TREINAMENTO DE REDES NEURAIS

O treinamento de uma rede neural é o processo de aprendizado para que a rede consiga aproximar os resultados esperados como saída a partir dos padrões de dados recebidos como entrada. A maioria dos algoritmos de aprendizado de redes neurais pode ser dividido em três categorias: supervisionado, não supervisionado e por reforço.

O aprendizado supervisionado é um modelo de treinamento baseado na utilização de exemplos. Durante o treinamento é obtido um resultado da rede, o qual é comparado com o resultado esperado. A diferença entre os resultados é conhecida com erro. O erro é utilizado para atualizar os pesos dos neurônios da rede neural. O objetivo do treinamento é alcançar um ponto em que a rede possa generalizar além dos dados de treinamento, tendo um bom desempenho em entradas nunca antes vistas. Geralmente grandes conjuntos de dados podem vir a aumentar a capacidade de generalização da rede (Schmidhuber, 2015).

Diferentemente do aprendizado supervisionado, o aprendizado não supervisionado não tem um rótulo de comparação na saída. Neste caso, a rede neural visa aprender padrões a partir dos dados de entrada. Esse tipo de aprendizado pode ser usado para agrupar dados semelhantes (Alom *et al.*, 2018).

No aprendizado por reforço, um agente aprende um comportamento interagindo com o ambiente, que retorna uma recompensa ao agente. Em um jogo, essa recompensa por ser uma pontuação ou um resultado. No caso, se o agente ganhou ou perdeu a partida. O objetivo da rede é adaptar o comportamento para maximizar a recompensa. Uma característica dessa técnica é que o conjunto de dados de treinamentos não é fixo. Os maiores desafios no aprendizado por reforço é definir como serão atribuídas as recompensas para cada ação (Schmidhuber, 2015).

A otimização de uma rede neural que utiliza o aprendizado supervisionado é constituída pela busca automática dos pesos e bias para que a mesma convirja pra uma saída desejada. A otimização também se refere a minimização da função de erro. Tal função mede o quão errado está o modelo, isto é, o quão distante a saída esperada está do resultado da rede. A técnica para minimização do erro é conhecida como gradiente descendente.

O gradiente descendente busca encontrar um mínimo local para a função de erro. Inicialmente, os pesos e bias recebem valores aleatórios, e iterativamente a função segue em direção ao maior gradiente descendente. A direção do gradiente descendente é dada pelo cálculo do gradiente negativo da função de erro.

$$MAE = \frac{1}{n}\sum_{n}^{0} |y - y'|$$

O treinamento de FNN's é composto por duas etapas, *forward propagation* e *backpropagation*. Na primeira etapa, os valores *x* são inseridos na camada de entrada, que propagados pelas camadas ocultas, produzindo um resultado *y* na camada de saída. Essa saída é comparada a saída desejada produzindo um valor de erro. A segunda etapa consiste na propagação do erro de volta pela rede ajustando os pesos de acordo com o cálculo do gradiente descendente.

Para o treinamento de redes neurais a partir de grandes bases de dados, o ajuste dos pesos não pode ser realizado para cada item do conjunto de dados, devido ao fato de que tornaria inviável o tempo de treinamento de uma rede neural. Para isso, existe o treinamento em *batch*, que consiste em propagar um certo número de entradas. Esse número deve ser maior que um e menor que o número total de entradas. Após, é calculado o erro para esse número de entradas, e feito o *backpropagation* com esse valor de erro.

2.3 BUSCA DE CAMINHOS

Em sistemas de simulação, assim como em jogos digitais, um dos requisitos mais básicos dos agentes é permitir que eles possam navegar com êxito pelo terreno virtual. Para isso, algoritmos de busca de caminho são explorados. Geralmente, estes algoritmos buscam o menor caminho trafegável dado um ponto de origem e um ponto de destino dentro da área compreendida pelo terreno virtual. Os três elementos principais da busca de caminhos são um grafo de representação de um mapa, um algoritmo de busca e uma função heurística para orientar a pesquisa (Botea *et al.*, 2013).

2.3.1 Estruturas de representação de mapas de navegação

A representação do terreno é uma característica importante a ser considerada para realizar a busca de caminho em um ambiente virtual. A estrutura deve armazenar os dados de forma a facilitar e simplificar o acesso aos mesmos durante o processo de busca (Anguelov, 2011). Esses dados representam características do mapa em cada item da estrutura, que indicam se o mesmo é trafegável ou não. Também devem ser definidas as adjacências de cada item, que representam a conexão entre um item e outro.

As estruturas podem ser dividias em duas classes: regulares e irregulares. Estruturas regulares são de fácil implementação visto que a quantidade de nodos é sempre a mesma (Souissi *et al.*, 2013). Essa estrutura divide a área do terreno em forma de polígonos regulares, ou seja, os polígonos devem possuir o mesmo tamanho.

2.3.2 Dijkstra e suas variações

Uma variedade de algoritmos de busca é apresentada na literatura. Tais algoritmos funcionam de diferentes formas, com ou sem o uso de heurísticas. Estes algoritmos podem utilizar resultados pré-processados (uso de memória) ou ser computados em tempo de execução, a partir do uso de mapas de navegação com estruturas dinâmicas ou estáticas. A qualidade de um algoritmo é validada pela sua completude e otimalidade, isto é, se existir um caminho, ele vai encontrar esse caminho caso o algoritmo for completo, e o caminho encontrado tem o menor custo possível caso o algoritmo seja ótimo.

O algoritmo de Dijkstra (Dijkstra, 1959) é um algoritmo não heurístico que realiza a busca do caminho gerando uma árvore de comprimento total mínimo. Uma busca mantém um conjunto, “NodosAbertos”, do qual iterativamente remove e expande o nodo com o menor

custo. O custo é determinado pela distância real entre o nodo inicial e nodo atual. Após o nodo ser expandido, ele é marcado como visitado. A busca é finalizada quando o nodo objetivo é encontrado, ou quando todos os nodos são expandidos, resultando em um caminho não encontrado. Uma representação do pseudocódigo do algoritmo de Dijkstra é apresentada no Algoritmo 2.1.

Algoritmo 2.1-Pseudocódigo do algoritmo de Dijkstra.

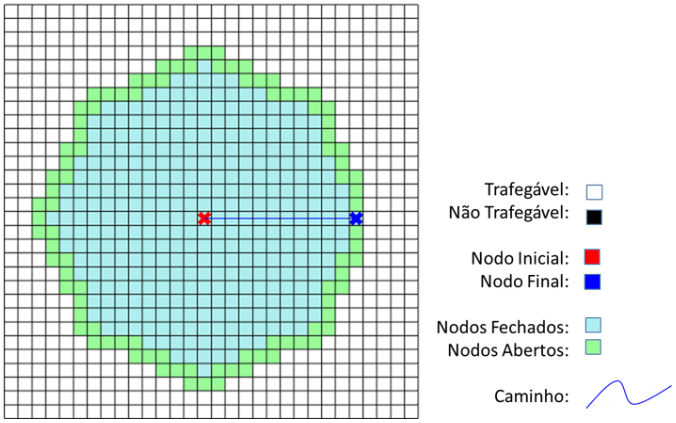
Algoritmo 1: Dijkstra

```
   Entrada: Grafo, NodoInicial, NodoObjetivo
1  Para cada nodo em Grafo faça
2     nodo.g ← infinito
3     nodo.visitado ← falso
4     nodo.pai ← indefinido
5  Fim para
6  NodoInicial.g = 0
7  NodosAbertos.Inserir(NodoInicial)
8  Enquanto (NodosAbertos.Vazio() = falso) faça
9     nodoAtual = NodosAbertos.RemoverMelhor() // Nodo com a menor distância entre o inicial e o atual
10    nodoAtual.visitado ← verdadeiro
11    Se (nodoAtual == NodoObjetivo) faça
12       Retornar ConstruirCaminho (NodoInicial, NodoObjetivo)
13    Fim se
14    Para cada vizinho em nodoAtual.vizinhos faça
15       G ← nodoAtual.g + distancia (nodoAtual, vizinho)
16       Se (G ≤ vizinho.g) faça
17          vizinho.g ← G
18          vizinho.pai ← nodoAtual
19          Se (vizinho.visitado = falso) faça
20             NodosAbertos.Inserir(vizinho)
21          Fim se
22       Fim se
23    Fim para
24 Fim enquanto
25 Retornar Caminho não encontrado
```

Fonte: Adaptado de (Ammar *et al.*, 2016).

Uma ilustração da execução do algoritmo de Dijkstra pode ser visualizada na Figura 2.4. Nessa figura, pode ser notado que mesmo com nenhum nodo bloqueado, o algoritmo expandiu uma grande quantidade de nodos. Isso acontece devido ao fato que a expansão dos nodos realizada pelo algoritmo leva em conta apenas a distância para a posição de início da busca.

Figura 2.4 - Ilustração da execução do algoritmo de Dijkstra.



Fonte: Autor

Uma versão do algoritmo de Dijkstra, o Relaxed Dijkstra (Ammar *et al.*, 2016), faz uma alteração na forma como o nodo a ser expandido é selecionado. Nessa versão, uma fila é a responsável pelo controle dos nodos. O nodo a ser expandido é removido do início da fila, enquanto os nodos que serão expandidos posteriormente a partir desse são adicionados ao final da fila. Dessa forma, o tempo computacional usado pelo algoritmo é reduzido, mantendo a otimalidade em estruturas de grade 4-conexas e a sub-otimalidade em estruturas de grade 8-conxeas.

O algoritmo A* é baseado no algoritmo de Dijkstra. O diferencial do algoritmo A* é que ele possui uma heurística que guia a busca do caminho. Se a heurística for consistente, o caminho encontrado entre um nodo inicial *ns* e um nodo objetivo *ng* será ótimo (Russell e Norvig, 2016). Uma busca mantém um conjunto, “NodosAbertos”, do qual iterativamente remove e expande o nodo com o menor custo. O custo de cada nodo *n* é determinado pela função $f(n) = g(n) + h(n)$. Nesta função, $g(n)$ é o custo do caminho, considerando a distância entre *ns* e *n*. $h(n)$ é a função heurística, a qual estima a distância entre *n* e *ng*. Depois que um nodo é expandido, esse nodo é adicionado a outro conjunto, “NodosFechados”, e seus filhos que ainda não estão em “NodosFechados” são adicionados ao “NodosAbertos”. O algoritmo começa com apenas o nó inicial no “NodosAbertos” e termina quando o nodo objetivo é removido do “NodosAbertos” (Hart *et al.*, 1968). Uma representação do pseudocódigo do algoritmo de A* é apresentada no Algoritmo 2.2.

Algoritmo 2.2 - Pseudocódigo do algoritmo de A*.

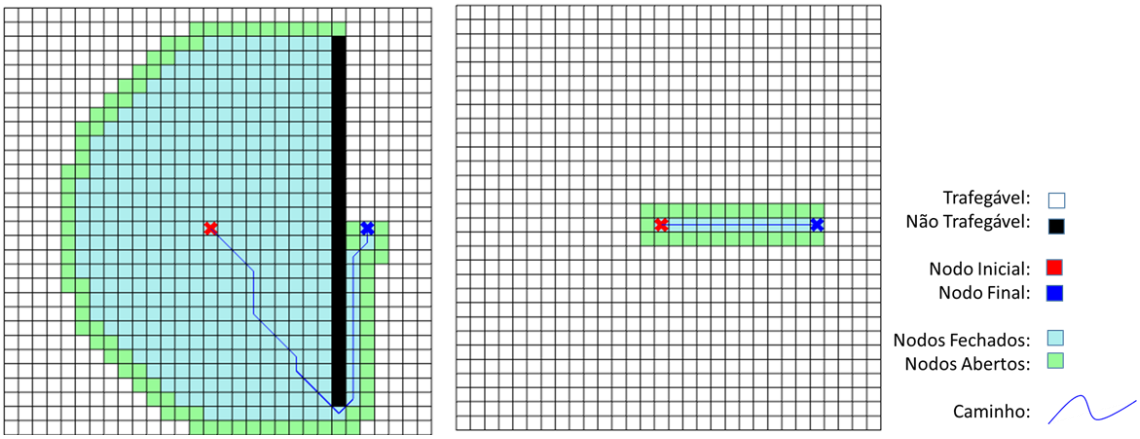
Algoritmo 2: A*

```
   Entrada: Grafo, NodoInicial, NodoObjetivo
1  NodoInicial.g = 0
2  NodoInicial.pai = Indefinido
3  NodosAbertos.Inserir(NodoInicial)
4  Enquanto (NodosAbertos.Vazio() = falso) faça
5     nodoAtual = NodosAbertos.RemoverMelhor() // Nodo com o menor custo f (g + h)
6     NodosFechados.Adicionar(nodoAtual)
7     Se (nodoAtual == NodoObjetivo) faça
8        Retornar ConstruirCaminho (NodoInicial, NodoObjetivo)
9     Fim se
10    Para cada vizinho em nodoAtual.vizinhos faça
11       Se vizinho ∉ NodosFechados faça
12          G ← nodoAtual.g + distancia (nodoAtual, vizinho)
13          Se G ≤ vizinho.g ou vizinho ∉ NodosAbertos faça
14             vizinho.g ← G
15             vizinho.h ← distância (vizinho, NodoObjetivo)
16             vizinho.pai ← nodoAtual
17             Se vizinho ∉ NodosAbertos faça
18                NodosAbertos.Adicionar(vizinho)
19             Fim se
20          Fim se
21       Fim se
22    Fim para
23 Fim enquanto
24 Retornar Caminho não encontrado
```

Fonte: Adaptado de (Ammar *et al.*, 2016)

Uma ilustração da execução do algoritmo A* pode ser visualizada na Figura 2.5. O principal diferencial em relação ao Dijkstra (Figura 2.4), é que esse algoritmo conseguiu reduzir significativamente a quantidade de nodos expandidos. Porém, quando o mapa possui obstáculos entre o ponto inicial e o ponto final, o algoritmo pode expandir uma grande quantidade de nodos. Isso acontece devido ao fato que o cálculo da função heurística não leva em consideração que possa existir uma área bloqueada para movimentação entre o nodo inicial e o nodo final.

Figura 2.5 - Exemplos de execução do algoritmo A* em um mapa com obstáculos e um sem obstáculos.



Fonte: Autor

Hierarquical PathFinding A* (HPA*) (Botea *et al.*, 2004) usa uma abordagem hierárquica para reduzir a complexidade do problema da busca de caminho. A hierarquia pode ser de dois ou mais níveis. Essa técnica divide o mapa de navegação do terreno virtual em clusters. Para cada cluster são pré-calculadas as distâncias ideais para atravessá-lo. Essas distâncias são armazenadas em um cache (uma memória, a qual é reusada em computações de caminhos). A busca de caminhos começa a nível de clusters, resultando em um caminho abstrato. Em seguida, o algoritmo encontra o caminho refinado no nível mais abaixo da hierarquia. Este caminho refinado passa por um processo de suavização resultando em um caminho final. De acordo com (Botea *et al.*, 2004), o HPA* algoritmo produz caminhos dez vezes mais rápidos que o A* tradicional, e com soluções dentro do 1% do ideal. Assim como apresentado no algoritmo HPA*, o uso de informações de caminhos que são pré-processadas e armazenadas em uma memória para auxiliar o algoritmo de busca indica que tal memória pode ser usada para melhorar a execução da busca de caminhos. Esta ideia é utilizada neste trabalho a partir da utilização de uma memória de caminhos pré-computados que é mantida por uma rede neural profunda. Para isso, esta rede utiliza para treinamento as informações de caminhos ótimos entre diferentes pontos de início e fim em um terreno virtual. Uma vez treinada, os resultados da rede neural são usados em tempo de execução do algoritmo de busca. A ideia é reduzir o número de nodos do espaço de estados que são analisados na busca de caminhos, bem como o tempo de computação do algoritmo de busca.

2.4 FUNÇÕES HEURÍSTICAS

Heurística é uma função arbitraria não negativa, de problemas específicos, com uma restrição, se qualquer nodo *n* for um nodo objetivo então *h(n) = 0*. Essa função tem como objetivo guiar a solução de um problema. Para a solução ótima do problema, a primeira condição é que *h(n)* seja uma heurística admissível. Uma heurística admissível é a que nunca superestima o custo para atingir o objetivo, ou seja, maior que custo real. A segunda condição é a consistência da heurística, necessária apenas para algoritmos A*. Uma heurística *h(n)* será consistente se, para cada nodo *n* e para todo sucessor *n'*, o custo de *n* até *n'* mais a *h(n')* for maior ou igual que *h(n)* (Russell e Norvig, 2016).

Cada problema requer uma forma de calcular uma função heurística. No algoritmo de busca de caminhos A*, algumas métricas que podem ser usadas para o cálculo da função heurística são a distância Euclidiana (Distância em linha reta) e distância de Manhattan (Soma das diferenças absolutas em cada espaço de busca). Porém, tais heurísticas são usadas para guiar a solução de muitos outros problemas. Um exemplo é o quebra-cabeça de peças deslizantes, no qual a heurística pode ser calculada como a quantidade de peças fora do lugar, ou a soma das distâncias das peças até a suas posições corretas (Russell e Norvig, 2016).

$$distânciaEuclidiana\ (x1, y1, x2, y2) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2}$$

$$distânciaManhattan\ (x1, y1, x2, y2) = |x1 - x2| + |y1 - y2|$$

A função heurística controla o fluxo de execução do algoritmo A*. Características relevantes dessas funções são:

- Se h(n) = 0, apenas o g(n) desempenha o papel de seleção do nodo a ser expandido.
- Se h(n) for menor ou igual a distância de deslocamento até o objetivo, é garantido que o caminho encontrado é ótimo. Quanto menor h(n), mais o A * se expande, tornando-o mais lento.
- Se h(n) for exatamente igual ao custo de passar n para a posição objetivo, o A * seguirá apenas o melhor caminho e nunca expandirá mais nada, tornando-o muito rápido.
- Se h(n) for maior que a distância de deslocamento até o objetivo, não é garantido que o caminho encontrado é ótimo.

Em geral, as funções heurísticas são utilizadas por algoritmos de busca. Elas são calculadas de diferentes formas para cada problema. Quanto mais próximas tais heurísticas forem da solução real, melhor elas guiam a solução dos problemas as quais são direcionas.

2.5 TRABALHOS RELACIONADOS

Uma abordagem para o aprendizado de funções heurísticas usando rede neural profunda é proposta por (Takahashi *et al.*, 2019). O objetivo é melhorar a eficiência computacional do planejamento de caminhos de robôs móveis. Neste estudo, foi adotado a rede neural U-Net (inicialmente proposta para segmentação semântica), para gerar valores heurísticos. A rede neural U-Net resulta na geração de imagens precisas usando as informações das primeiras camadas convolucionais diretamente nas últimas camadas (Ronneberger *et al.*, 2015). A entrada da rede é um mapa de ocupação e um tensor meta. Esse tensor meta consiste em duas matrizes que representam sgoal = (xgoal, ygoal, θgoal) onde (xgoal, ygoal) é a posição de meta e θgoal é a orientação do objetivo. A saída da rede é um valor heurístico que é um tensor W × H × O, onde W e H é o número de colunas e linhas no mapa de ocupação, respectivamente, e O é o número de orientações. Para o treinamento da rede foram geradas aleatoriamente posições de início e meta. Em seguida, a heurística ideal foi criada utilizando o algoritmo de Dijkstra a partir da posição de meta. Para a avaliação o resultado da rede, um algoritmo de A* foi utilizado. Os testes foram realizados em um domínio contínuo (domínio onde os estados não podem ser mapeados) e um domínio discreto (domínio onde os estados podem ser mapeados), como diferentes funções de erro para avaliação de perda da heurística aprendida e da heurística real, apresentando resultados bem-sucedidos com a redução do número de nodos explorados. A objetivo dessa proposta é similar ao objetivo de pesquisa deste TCC: a redução do número de nodos expandidos durante a execução da busca de um caminho. A principal diferença está na arquitetura de rede neural. Enquanto este TCC utiliza redes neurais totalmente conectadas, essa proposta utiliza uma U-Net, que é uma rede neural convolucional. Embora as redes sejam diferentes, existem técnicas semelhantes usadas em ambos os trabalhos. Neste TCC, por exemplo, o algoritmo de Dijkstra também foi explorado para gerar as heurísticas. Além disso, este TCC também usou a rede neural juntamente com o algoritmo A*. Os experimentos realizados naquele trabalho foram realizados em mapas de 32x32. Neste TCC, os mapas foram de 101x101.

DeepCubeA (Agostinelli *et al.*, 2019) é uma abordagem de aprendizado por reforço profundo, combinado com aprendizado por reforço clássico e métodos de localização de caminhos. O DeepCubeA é capaz de resolver quebra-cabeças combinados como o cubo mágico, quebra cabeça de peças deslizantes de 15, 24, 35 e 48, Lights Out e Sokoban. O DeepCubeA trabalha usando iteração de valor aproximado (técnica de programação dinâmica que melhora iterativamente uma função de custo) para treinar uma DNN para aproximar uma função que gera o custo para atingir a meta. Após o treinamento, a função de custo aprendida é usada como uma heurística para resolver os quebra-cabeças usando Weighted A* (Ebendt e Drechsler, 2009). Comparações entre o DeepCubeA e outros solucionadores de caminho mais curto são demonstradas em diferentes quebra-cabeças mencionados naquele trabalho. Os resultados obtidos mostram que todos os casos de testes para o cubo magico foram resolvidos. Nos outros quebra-cabeças, todos os casos de testes também foram resolvidos, e em quase 100% dos casos, a solução era a melhor possível. Em geral, a pesquisa desenvolvida neste TCC foi motivada nessa proposta. Embora os propósitos sejam diferentes, os dois trabalhos utilizam redes neurais para o aprendizado de funções heurísticas. Tais funções heurísticas são utilizadas na busca em espaços finitos, porém em diferentes escalas visto que quebra-cabeças possuem um espaço de busca muito maior que mapas de navegação em geral (por exemplo, os mapas possuem no máximo 5000 diferentes estados, enquanto o cubo magico possui aproximadamente 4,3x10^19 diferentes estados).

O modelo chamado ST-NN (Rede Neural Espacial-Temporal) (Jindal *et al.*, 2017) prevê a distância de viagem entre coordenadas GPS de origem e destino. Em seguida, esse algoritmo combina essa previsão com a hora do dia para prever o tempo de viagem. A abordagem proposta usa apenas coordenadas de GPS brutas de origem e destino e informações do horário do dia para realizar a predição de distância e tempo. De acordo com os autores, foram os primeiros a estimar a distância da viagem a partir de coordenadas de GPS, sem construir nenhuma rota entre os dois locais. A partir desse modelo, foi possível propor uma arquitetura de rede neural inicial para a predição de distância entre dois pontos. Essa foi a principal contribuição desse modelo para o desenvolvimento deste TCC.

O modelo NASR (Neuralized A-Star) (Wang *et al.*, 2019) propõe o uso de redes neurais para aprender as funções de custo de um algoritmo heurístico usado na solução de uma tarefa de recomendação de rota personalizada. A recomendação de rota personalizada

envolve a geração de sugestões de rotas específicas para usuário, as quais são emitidas a partir de consultas voltadas para o planejamento de caminho entre uma origem e um destino. Isso é desenvolvido por que uma rota padrão baseada na menor distância entre dois pontos pode não atender às necessidades personalizadas de um usuário específico. O modelo consiste em dois componentes. O primeiro é baseado no emprego de Redes Neurais Recorrentes (RNN) para modelar o custo de posição de origem até a posição corrente. O segundo propõe o uso de uma rede de valor (funções que atribuem pesos a nodos) para estimar o custo da posição candidata até a posição destino. Os resultados indicam que essa combinação obteve sucesso na tarefa de recomendação de rotas personalizadas. Enquanto o NASR utiliza redes neurais para calcular o custo de movimentação da posição inicial até a posição corrente, este TCC utiliza redes neurais para a predição da distância da posição corrente até a posição final. A principal semelhança entre os dois trabalhos é a utilização de redes neurais para aprendizado de uma função de custo.

Uma técnica de utilização de redes neurais convolucionais (CNN) para o aprendizado de funções heurísticas para planejadores de caminhos é apresentado por (Ariki e Narihira, 2019). Neste trabalho, assim como explorado neste TCC, grades regulares 2D são utilizadas com estruturas de navegação. Os mapas utilizados possuem diferentes características, sendo mapas de labirinto, mapas com *bugtraps* (estruturas locais com apenas uma entrada), mapas de florestas e outros mapas com as características anteriores misturadas. A entrada da rede é um mapa de obstáculos e uma posição objetivo. A saída da rede é um mapa heurístico, que é uma imagem que contém todos os valores heurísticos a partir do ponto objetivo. O mapa heurístico previsto pela rede neural é usado como uma tabela de consulta para consultar um valor heurístico durante o planejamento do caminho. Esta ideia de construção e uso de um mapa heurístico também é explorada neste TCC, assim permitindo reduzir o tempo de computação de caminhos requerido pelo algoritmo de busca explorado. Para validação da técnica, foi avaliado o custo de busca e a qualidade do caminho, com diferentes funções heurísticas: distância ótima, distância Euclidiana e a distância adquirida através do mapa heurístico. Os resultados mostram que foi possível reduzir significativamente o custo de busca com perda de qualidade do caminho. A principal diferença entre os trabalhos está na arquitetura de rede neural utilizada. Esse trabalho utiliza redes neurais convolucionais enquanto este que o TCC utiliza redes neurais totalmente conectadas. A técnica utilizada nos dois trabalhos envolve a geração de um mapa heurístico a partir do resultado da rede neural.

Os mapas testados são de 224×224. Porém, nos resultados apresentados, nenhum dos casos o custo da busca de caminho foi próximo ao ótimo.

O ANN* (Li *et al.*, 2016) é um algoritmo de busca heurística em grades baseado em uma rede neural artificial. Esse algoritmo é semelhante ao A*, embora ele empregue uma função heurística modificada. Essa função heurística é gerada a partir de uma rede neural de regressão. A rede neural é utilizada para aprender os recursos dos mapas e prever a dificuldade da busca de caminho. Esse algoritmo utiliza a mesma função de custo que o algoritmo A*, usando uma função heurística tradicional e um multiplicador. Esse multiplicador é o resultado da rede neural, f(n)=g(n)+hT(n)*hNN(n). A base de dados para o treinamento da rede neural foi construída através de um algoritmo de geração de mapas aleatórios parametrizáveis. Os parâmetros são largura, comprimento e complexidade do mapa (porcentagem de nodos bloqueados). Com tais parâmetros, são gerados pontos aleatórios de acordo com o valor de complexidade. Usando os pontos, é gerado uma árvore mínima aleatória, a qual é utilizada pelo algoritmo para conectar os pontos na grade. Redes neurais possuem um tamanho de entrada fixo. Então, para o treinamento da rede, os mapas gerados com tamanho aleatório são redimensionados. Diferentes arquiteturas de rede foram testadas e avaliadas. A rede neural apresentada ao final generaliza a função heurística para diferentes mapas. Para avaliação do modelo proposto, foram utilizados cinco diferentes tamanhos de mapas (10x10, 20x20, 30x30, 40x40 e 50x50) e duas funções heurísticas (distância Euclidiana e distância de Manhattan). Os resultados apresentados envolviam a quantidade de nodos expandidos em relação ao tamanho e complexidade do mapa. Segundo o autor, o modelo obteve sucesso na redução de nodos expandidos pelo algoritmo. Este trabalho generaliza a rede neural para diversos mapas. Porém, os mapas são de 50x50, enquanto que neste TCC os mapas são de 101x101.

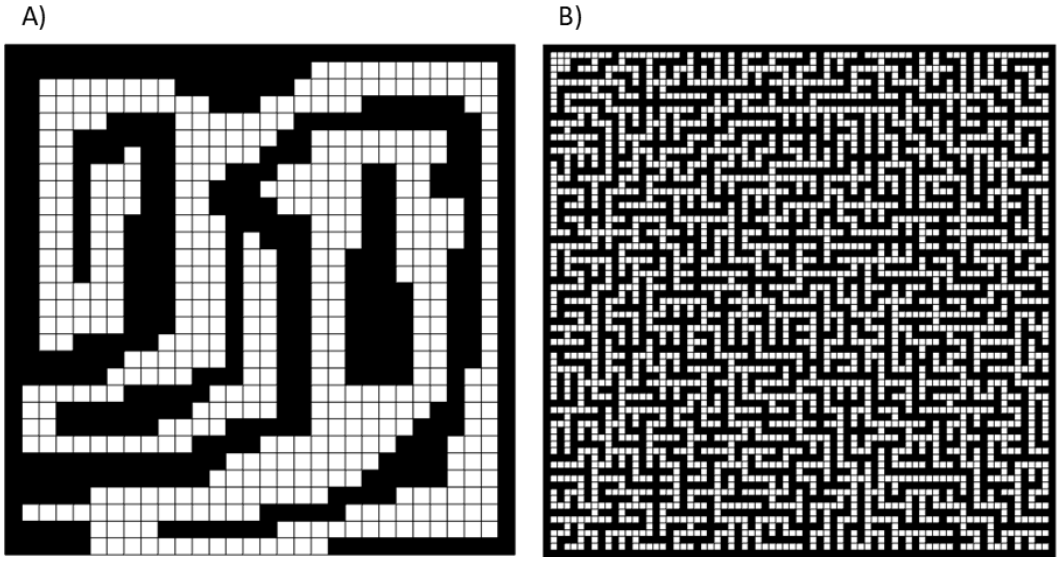
Em resumo, os trabalhos encontrados descrevem formas de utilização de redes neurais para o aprendizado de funções heurísticas a fim de serem usadas em algoritmos de busca (Agostinelli *et al.*, 2019; Wang *et al.*, 2019). Eles também descrevem técnicas de auxiliam a utilização dessas redes. No TCC, as principais ideias utilizadas foram a geração do mapa heurístico apresentado por (Ariki e Narihira, 2019), a utilização do algoritmo de Dijkstra para gerar a heurística ideal apresentada por (Takahashi *et al.*, 2019) e o modelo de arquitetura de rede neural proposto por (Jindal *et al.*, 2017).

3 METODOLOGIA

3.1 MAPAS E ESTUTURAS UTILIZADOS

Diferentes mapas de navegação são abordados neste trabalho, sendo um mapa genérico desenhado a mão com tamanho de 30x30 (Figura 3.1 A) e três mapas representando labirintos com tamanho de 51x51, 75x75 (Figura 3.1 B) e 101x101. Para representar as informações dos mapas, a estrutura de grade regular foi usada na representação. Tais mapas foram criados para avaliar como o algoritmo proposto de sai em mapas de complexidade variada.

Figura 3.1 - Exemplos de mapas/estruturas utilizados neste trabalho.



Fonte: Autor

3.2 REDE NEURAL

O uso de uma rede neural profunda para o aprendizado de funções heurísticas é o objetivo principal deste trabalho. No algoritmo de A*, a função heurística tenta estimar a distância para alcançar o objetivo. Para que o caminho encontrado seja o menor possível, a heurística deve ser admissível. Porém, como é a função heurística que guia o processo de análise dos nodos do mapa, quanto mais essa função apresentar uma aproximação da distância real de um caminho até um objetivo, menor número de nodos do mapa serão analisados pelo algoritmo. Para o treinamento da rede neural, distâncias reais entre pares de pontos dos mapas descritos na Seção 3.1 foram computadas usando o algoritmo Dijkstra.

3.2.1 Conjunto de dados

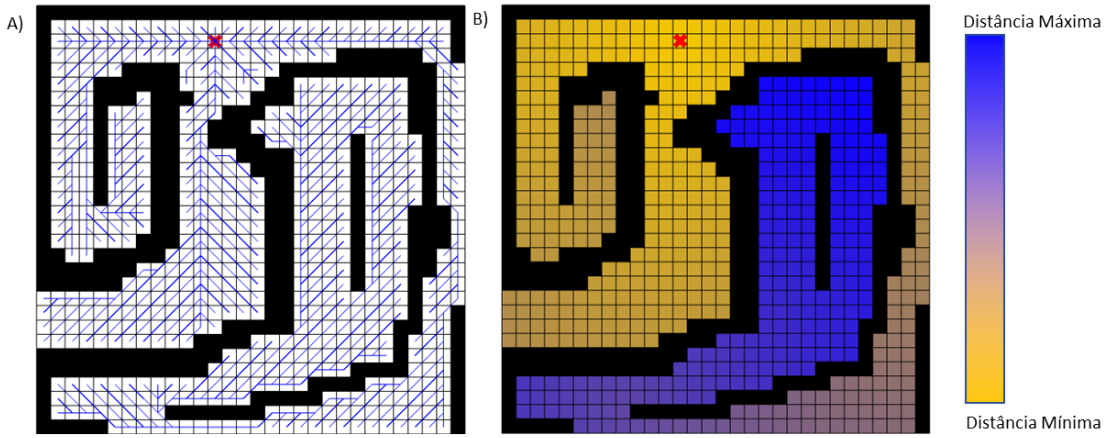
Para construir uma base de dados a ser usada no treinamento da rede neural, as informações essenciais para utilização de funções heurísticas no algoritmo A* foram identificadas. Tais atributos são:

- Ponto pC (xC, yC) que representa a posição do nodo corrente;
- Ponto pT (xT, yT) que representa a posição do nodo objetivo;
- Valor de distância entre eles, que representa o custo de deslocamento do nodo pC até pT.

$$h(n) = f(xC,\ yC,\ xT,\ yT) = Distância$$

Para gerar o conjunto de dados usados para treinamento e teste da rede neural, duas formas de computação foram testadas. A primeira é baseada na execução do A* para cada par de pontos inicial e final do caminho. A segunda forma é baseada na execução do algoritmo de Dijkstra sem que este use uma posição final/objetivo, devido ao fato de que o Dijkstra produz a árvore geradora mínima. Com isso, é possível gerar a distância de um nodo n para todos os outros nodos acessíveis a partir de n no mapa. Uma representação de como funciona a computação da distância entre os pontos inicial e final na estrutura de grade regular pode ser visualizada na Figura 3.2 A. Tal figura ilustra a árvore geradora mínima iniciando a partir do X (marcado em vermelho) localizado na parte superior central da Figura 3.2. A Figura 3.2 B representa a mesma árvore, porém em forma de gradiente de cores, onde a distância mínima calculada é representada pela cor amarela e a distância máxima é representada pela cor azul.

Figura 3.2 - Representação do processamento de distância calculada pelo algoritmo Dijkstra em um mapa.



Fonte: Autor

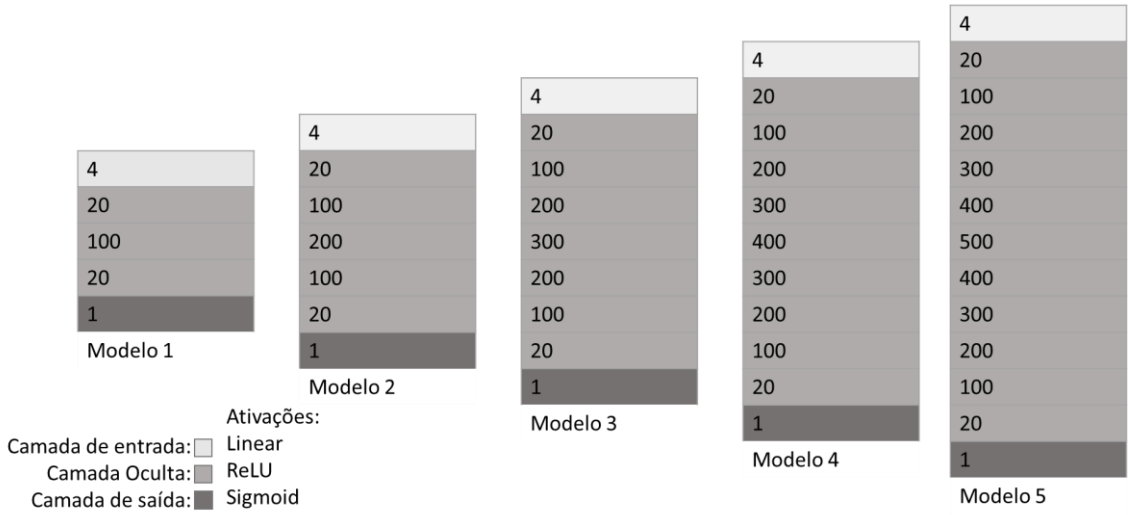
As distâncias entre diferentes pontos iniciais e finais processadas foram armazenadas em formato CSV. O cabeçalho do CSV foi xInicial, yInicial, xFinal, yFinal e distância.

3.2.2 Arquitetura da rede neural

Para definição da arquitetura da rede neural profunda, diferentes números de neurônios e camadas foram testados. Neste trabalho, um modelo de rede neural *feedforward* totalmente conectado foi explorado, com as camadas ocultas possuindo uma certa simetria em relação ao número de neurônios por camada. A partir de testes realizados, o modelo que obteve o melhor resultado foi semelhante ao proposto por (Jindal *et al.*, 2017), o qual utilizava a computação de heurísticas para apoiar a predição de distâncias em uma aplicação voltada para apoiar a computação de viagens de taxi. Contudo, como as funções de ativação usadas na rede neural não foram mencionadas em (Jindal *et al.*, 2017), as funções utilizadas neste trabalho podem não ser a mesmas. Em resumo, o modelo de rede usado pode ser visualizado na Figura 3.3 (Modelo 1).

As funções de ativação escolhidas foram: a) para a camada de entrada, função linear; b) para as camadas ocultas, a escolhida foi função ReLU pois, segundo (Maas *et al.*, 2013), a função ReLU permite melhorar o desempenho das redes neurais em relação as outras funções de ativação e c) para a camada de saída, a função sigmoide foi a escolhida, por ajustar valores entre zero e um.

Figura 3.3 - Arquitetura da rede neural implementada neste TCC.



Fonte: Autor

3.2.3 Treinamento da rede neural

Para o treinamento, os valores dos atributos da base de dados foram normalizados entre zero e um visto que tal normalização geralmente tem um impacto positivo no processo de treinamento de redes neurais. A normalização foi feita de acordo com o conjunto de dados disponível. Os pontos utilizados para gerar o conjunto de dados pertencem a parte positiva do plano cartesiano, e os valores de distância são sempre valores reais e positivos. Com isso sabe-se que os valores estão sempre entre 0 e N, sendo N um valor real e positivo. Os valores de pontos e distâncias não possuem grande diferença de escala em relação aos pares de pontos do conjunto de dados. Desta forma, eles foram normalizados pelo mesmo valor. Tal valor é o maior valor contido no conjunto de dados encontrado na última coluna do arquivo CSV usados.

Para o treinamento da rede neural, os dois principais parâmetros são a função de perda e a otimização. A função de perda escolhida foi a média do erro absoluto (apresentada na Seção 2.2). A função de otimização foi a otimização de Adam (Kingma e Ba, 2014), que é baseado na decida do gradiente. As bases de dados foram duplicadas com os pontos invertidos para auxiliar o treinamento. Esta base foi dividida em duas partes: treinamento e validação. A proporção foi 70% para treinamento e 30% para validação. O *batch size* e número de *epochs* (número de vezes que os dados são propagados pela rede neural para o treinamento) foram definidos de acordo com o tamanho de cada base de dados.

Tabela 3.1 - Parâmetros e dados utilizados para o treinamento das redes neurais.

	Mapa Genérico	**Labirinto 1**	**Labirinto 2**	**Labirinto 3**
Tamanho do conjunto de dados	375.769	1.562.500	7.502.121	25.000.000
Tamanho do conjunto de treinamento	263.038	1.068.550	5.251.484	17.500.000
Número de épocas	100	300	500	720
Tamanho de batch	32	128	512	2048
Tempo de treinamento	50 min	1,5h	5,25h	12h

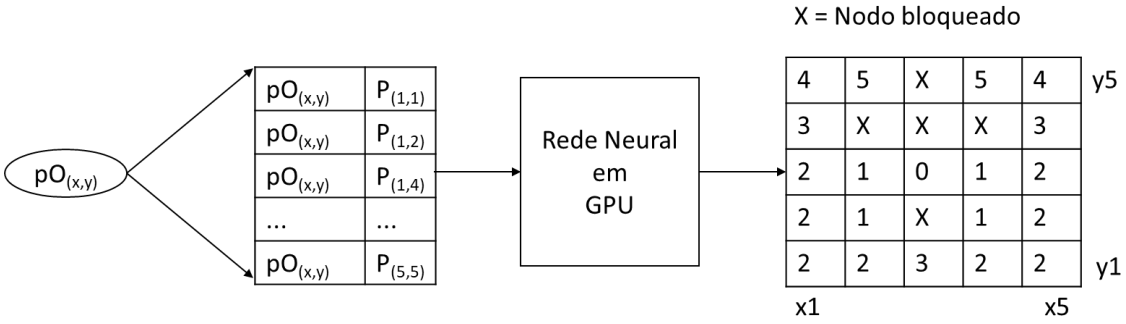
3.3 ALGORITMO DE A* COM REDES NEURAIS

A utilização da rede neural consiste na substituição do método que calcula a função heurística no algoritmo de busca de caminhos. Essa substituição acontece na linha 15 do pseudocódigo do Algoritmo 2.2. Com a substituição de tal linha, o restante o algoritmo executa da mesma forma.

Para que seja possível inserir a função heurística da rede neural no algoritmo de busca A*, a cada nodo descoberto, o valor heurístico da função de custo deve ser computado pela rede neural. Dessa forma, a execução do algoritmo de busca de caminhos se torna lenta à medida que o número de neurônios contidos na rede neural cresce. Tal como explorado neste trabalho, outra forma de gerar os valores para as funções heurísticas é utilizar o paralelismo em GPU. Como resultado, é gerado um mapa heurístico.

Neste trabalho, o mapa heurístico é representado por uma matriz bidimensional, onde os valores da matriz armazenam a distância entre todos os pontos em relação a um ponto objetivo (pO). Essa matriz é alimentada com os resultados da rede neural. Nesse caso, quando o algoritmo de busca de caminho requerer o valor heurístico para um nodo, ele não executa a rede neural, mas executa uma pesquisa no mapa heurístico previamente computado.

Figura 3.4 - Exemplo de execução da rede neural para geração do mapa heurístico.



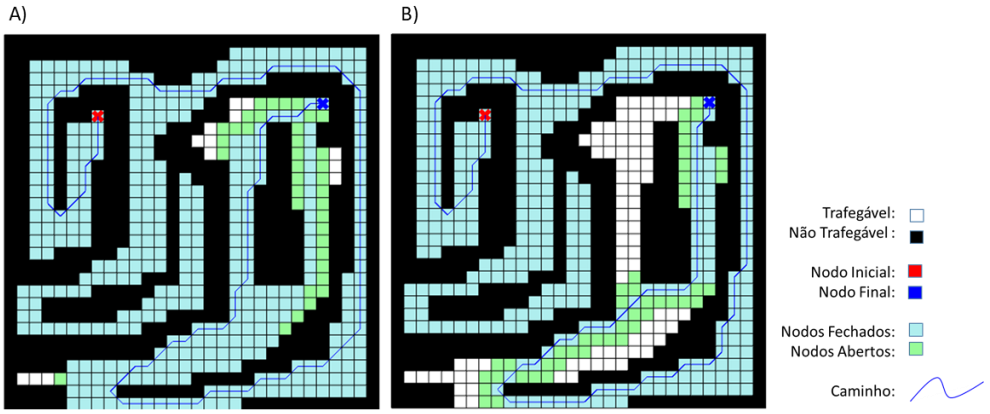
Fonte: Autor

De acordo com experimentos realizados neste trabalho, os tempos requeridos para a geração do mapa heurístico ganharam relevância à medida que o tamanho do mapa usado cresce.

Tabela 3.2 - Tempos de geração dos mapas heurísticos.

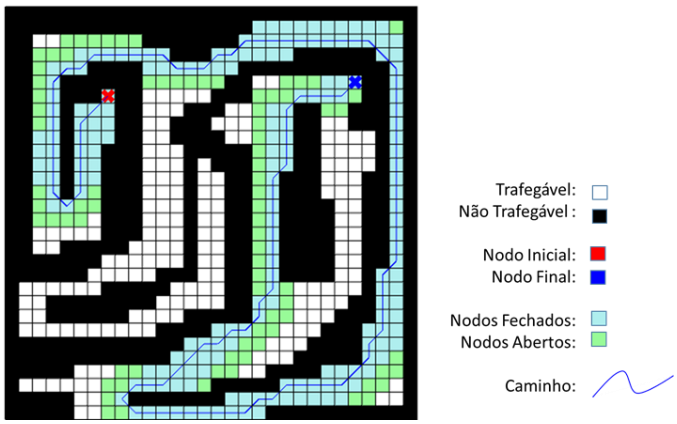
	Mapa Genérico	**Labirinto 1**	**Labirinto 2**	**Labirinto 3**
Tempo	1ms	1.5ms	3ms	5ms

Figura 3.5 - Caminhos computados entre dois pontos via A) A* com função heurística baseada na distância Euclidiana B) A* com função heurística baseada na distância de Manhattan.



Fonte: Autor

Figura 3.6 - Exemplo de execução do algoritmo A* usando o mapa heurístico resultante da rede neural.



Fonte: Autor

Comparado as Figura 3.5 e Figura 3.6, foi possível notar que os valores heurísticos retornados pela rede neural permitiram direcionar corretamente busca, pois somente nodos próximos ao caminho final foram expandidos. Apesar dos caminhos encontrados serem diferentes, o comprimento final de todos foi o mesmo.

4 EXPERIMENTOS E RESULTADOS

O objetivo dos experimentos desenvolvidos neste trabalho é avaliar a viabilidade do uso de redes neurais para o aprendizado de funções heurísticas usadas por algoritmos de busca de caminhos (em especial, o algoritmo A*). A hipótese é que a função heurística aprendida pela rede neural permite reduzir o tempo de execução do processo de busca e direcionar/reduzir a expansão dos nodos processados pelos algoritmos de busca de caminhos.

Quatro mapas foram utilizados nos experimentos: um mapa genérico desenhado a mão e três outros em formato de labirinto. Todos os mapas possuem como característica grande porcentagem de nodos bloqueados. Os dados referentes a cada mapa são apresentados na Tabela 4.1. Um processador Intel i7 - 4790 (3,6 GHz) com 12 GB de RAM foi utilizado para execução dos algoritmos. Para a execução da rede neural, a GPU GTX 1660 Ti 6 GB foi utilizada.

Para cada mapa, foram gerados pares de pares de pontos, com quantidade equivalente ao número de nodos existentes na estrutura. Neste caso, uma cobertura completa de diferentes posições dos mapas foi considerada, solução que pode não ser viável para mapas de maiores dimensões tais como os usados no sistema de simulação SIS-ASTROS. Para cada par de pontos, foi executado o algoritmo de busca A* com três diferentes funções heurísticas. Duas dessas funções heurísticas não levam em consideração nenhuma característica do mapa (Distância Euclidiana e Distância de Manhattan), e a função heurística obtida através da execução da rede neural. As funções heurísticas de Distância Euclidiana e Distância de Manhattan foram calculadas normalmente durante a execução do algoritmo de busca de caminhos. A função heurística calculada pela rede neural foi viabilizada pela construção do mapa heurístico. Por fim, para cada execução foram medidos e armazenados os valores referentes ao comprimento do caminho resultante, número de nodos expandidos e tempo de execução.

Tabela 4.1 - Características dos mapas de navegação e protocolo de testes.

	Mapa genérico	**Labirinto 1**	**Labirinto 2**	**Labirinto 3**
Dimensão	30x30	51x51	75x75	101x101
Número de nodos	900	2.601	5.625	10.201
Percentual de nodos bloqueados	49%	48%	49%	51%
Amostragem	70%	70%	70%	70%
Métricas	Tempo Nodos expandidos	Tempo Nodos expandidos	Tempo Nodos expandidos	Tempo Nodos expandidos
Funções heurísticas	Rede neural Distância de Manhattan Distância Euclidiana	Rede neural Distância de Manhattan Distância Euclidiana	Rede neural Distância de Manhattan Distância Euclidiana	Rede neural Distância de Manhattan Distância Euclidiana
Conjunto de dados de teste	900	2.601	5.625	10.201

A implementação dos algoritmos de busca de caminhos e mapas de navegação foram feitas utilizando a Unity3D (Unity Technologies, 2019). A rede neural foi implementada em Python utilizando a biblioteca de redes neurais TensorFlow (Abadi *et al.*, 2016). Para a execução da busca de caminhos, a comunicação entre a Unity3D e o Python foi feita via sockets. Porém, quando uma busca de caminho é executada, a comunicação entre os dois é realizada, resultando em um atraso na execução do algoritmo de busca. Para a execução dos testes, esse atraso foi computado e desconsiderado nos valores de tempo resultantes. Apenas o tempo de geração do mapa heurístico e do algoritmo A* foi considerado.

Para analisar estatisticamente os resultados obtidos nos experimentos desenvolvidos, foram utilizados modelos de regressão linear generalizada (Mccullagh e Nelder, 1989). Tais modelos foram implementados utilizando o pacote de software estatístico R. Mesmo com tamanha complexidade desses modelos generalizados, eles apresentam, de maneira confiável,

medidas de tendência central e disseminação, além de serem robustos modelos não paramétricos de regressão. Essas técnicas de regressão generalizada permitem explorar diferentes tipos de distribuição na construção de modelos para descrever resultados experimentais. Em particular, considerando os resultados obtidos nos experimentos realizados, a distribuição Gama foi explorada na construção dos modelos de regressão. O modelo Gama representa melhor os valores reais positivos, que são o comprimento do caminho em relação ao número de nodos expandidos e o tempo de execução do algoritmo. Para comparação estatística, resultados obtidos pela computação das três funções heurísticas foram inclusos nos modelos de regressão.

Os modelos de regressão para a comparação dos resultados foram descritos como g(mu) = Beta_0 + Beta_1 * X + Beta_2 * D1 + Beta_3 * D2, onde g é a função de log. Na prática, este modelo representa os valores de nodos expandidos em função da distância resultante do caminho entre a posição de início e a posição objetivo, onde tais distâncias são representadas pela variável X do modelo. Da mesma forma, esse tipo de representação foi usado para descrever os valores de tempo em relação ao comprimento do caminho resultante. As variáveis *dummy* D1 e D2 foram utilizadas para representar a funções heurísticas comparadas, onde as comparações foram feitas em relação a um método base, a qual é a função heurística aprendida pela rede neural. Desta forma, os modelos de regressão descrevem a função heurística obtida pela rede neural quando D1 = 0 e D2 = 0. Respectivamente, eles descrevem a função heurística cujo valor é obtido pela distância Euclidiana quando D1 = 1 e a função heurística cujo valor é obtido pela distância de Manhattan quando D2 = 1. O valor da i-ésima função é obtido quando o Di = 1 e os outro são iguais a zero.

Na comparação das funções heurísticas, as estimativas resultantes representam os valores de nodos expandidos em relação a distância do caminho percorrido e os valores de tempo de execução em relação a distância do caminho percorrido. As amostras experimentais foram ordenadas de acordo com o comprimento do caminho. Devido ao fato de que os resultados das funções construídas podem assumir qualquer valor real, a função de vinculação usada nos modelos de regressão é a função log. Para desenvolver a análise estatística, foi definido um nível de significância de 1%. Então, as hipóteses H0 e H1 foram definidas com Beta_i = 0 ou Beta_i $\neq$ 0, respectivamente para i = 1, 2 e 3. Com isso o alfa foi comparado

com o valor-p. Se alfa > P-valor, então H0 não é rejeitado, do contrário é rejeitado. Beta_1 > 0, significa que distâncias maiores entre as posições de origem e destino implicam em uma maior quantidade de nodos expandidos e maior tempo de execução. Para Beta_i = 0, com i = 2 e 3, significa que o método D1 é equivalente ao método base. Beta_i > 0 significa que o Di é menos eficiente que o método base. Beta_i < 0 significa que Di é mais eficiente que o método base.

Nos resultados, os valores de P obtidos com os modelos de regressão são todos próximos de zero e inferiores ao nível de alpha considerado de 1%, portanto, é possível afirmar que todos os resultados obtidos são estatisticamente significativos.

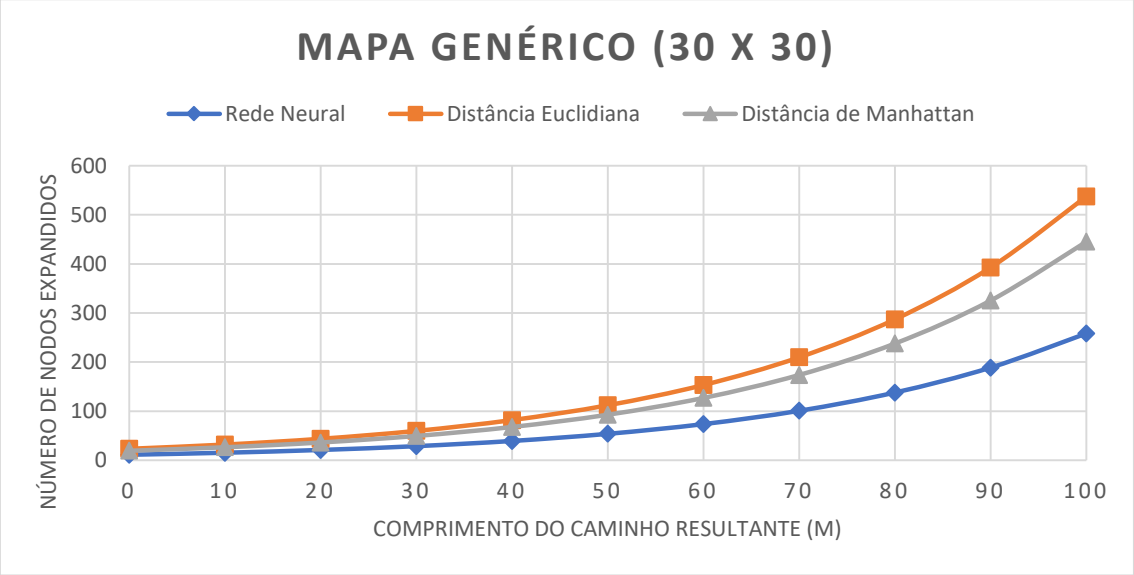
Analisando os gráficos da Figura 4.1, Figura 4.3, Figura 4.5 e Figura 4.7, que comparam a expansão de nodos, para todos os mapas, todos os valores são positivos (maiores que o método base, a rede neural), os modelos de regressão mostram que as funções heurísticas de distância de Manhattan e distância Euclidiana resultam em uma maior quantidade de nodos expandidos em comparação a quando foi utilizado a função heurística como sendo o resultado da rede neural. Este é um resultado já esperado, visto que apenas a rede neural leva em consideração as características do mapa (por exemplo, posição de nodos bloqueados). Outro ponto importante a ser notado é que em todos os casos, a função heurística sendo a distância de Manhattan foi melhor que a função heurística de distância Euclidiana, devido ao fato de que o mapa de navegação utilizado foi o de grade regular.

Entretanto, quando se trata do tempo de execução dos algoritmos de busca, os resultados obtidos não foram os mesmos. Analisando a Figura 4.2, o uso da distância Euclidiana e da distância de Manhattan descritas no modelo de regressão em relação ao método base (rede neural) foram negativos. Isso claramente indica que a rede neural obteve um desempenho pior que a computação de funções de distância tradicionais. Analisando os resultados apresentados na Figura 4.4, o modelo estatístico indicou que a rede neural obteve um desempenho pior em relação ao uso da distância de Manhattan e melhor em relação ao uso da distância Euclidiana. Analisando os resultados da Figura 4.6 e Figura 4.8, o modelo de regressão mostrou que os resultados foram equivalentes aos obtidos quando a expansão de nodos foi comparada. Sabendo que o tempo de execução é proporcional ao número de nodos expandidos pelos algoritmos de busca, a redução da quantidade de nodos compensou o tempo de execução da rede neural (mais alto que simples computações de distâncias via equações

matemáticas). Esses resultados de tempo de processamento também eram esperados (no caso, tendo a rede neural como a melhor função heurística na Figura 4.6 e Figura 4.8, e não sendo a melhor na Figura 4.2 e Figura 4.4), devido ao fato de que a rede neural possui um tempo adicional de computação, que está relacionado ao tempo de geração do mapa heurístico. Nos mapas de menor escala, o tempo de computação do mapa heurístico teve maior relevância em comparação aos mapas maiores.

Na comparação de distâncias computadas, em todas execuções com as diferentes funções heurísticas, os resultados foram os mesmos (podendo ser diferentes caminhos, porém com o mesmo custo). Porém, é importante notar que a heurística aprendida pela rede neural não satisfaz os critérios e restrições de heurística admissível. Então, existem casos em que o valor retornado pela heurística aprendida é superior ao valor que representa a distância real entre pares de pontos. Também existem casos em que o valor resultante da rede neural é diferente de zero quando a entrada da rede é o nodo objetivo x nodo objetivo.

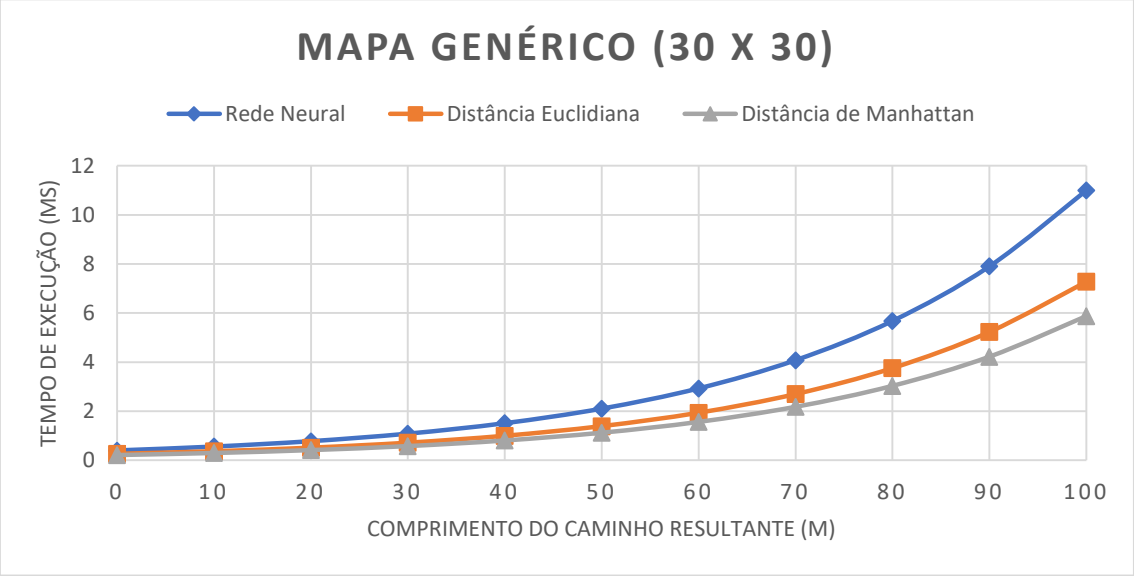
Figura 4.1 - Gráfico de comparação de expansão de nodos no mapa genérico.



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	2.4176910	0.0260131	92.94	< 2e-16
Rede Neural	0.0313640	0.0003867	81.11	< 2e-16
Euclidiana	0.7327879	0.0267524	27.39	< 2e-16
Manhattan	0.5444151	0.0268337	20.29	< 2e-16

Fonte: Autor

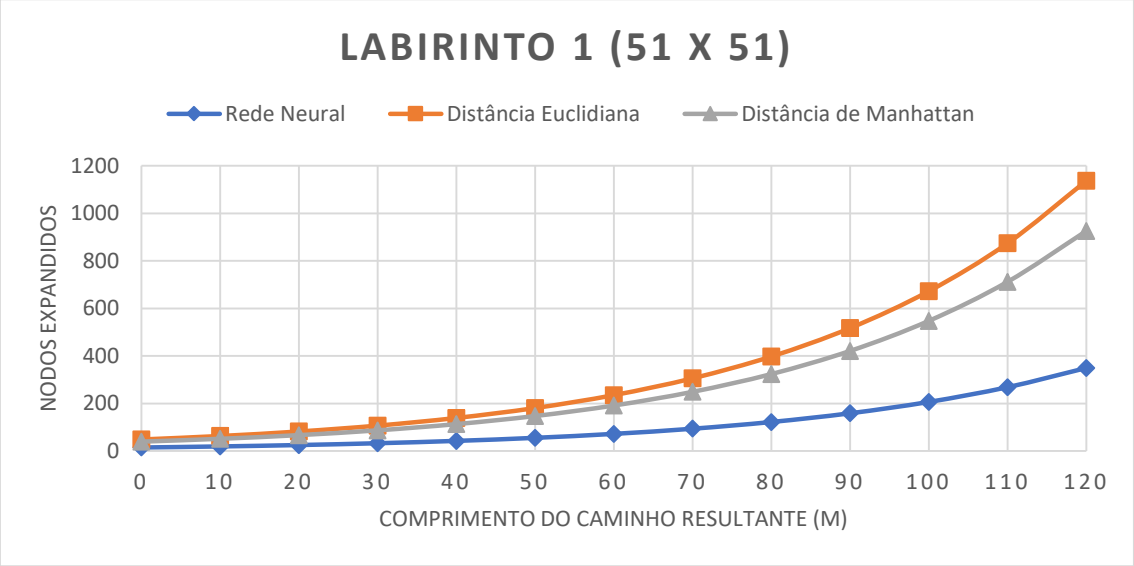
Figura 4.2 - Gráfico de comparação de tempo de execução no mapa genérico.



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	-0.9071863	0.0327056	-27.738	< 2e-16
Rede Neural	0.0330501	0.0005991	55.169	< 2e-16
Euclidiana	-0.4129531	0.0434716	-9.499	< 2e-16
Manhattan	-0.6266154	0.0439321	-14.263	< 2e-16

Fonte: Autor

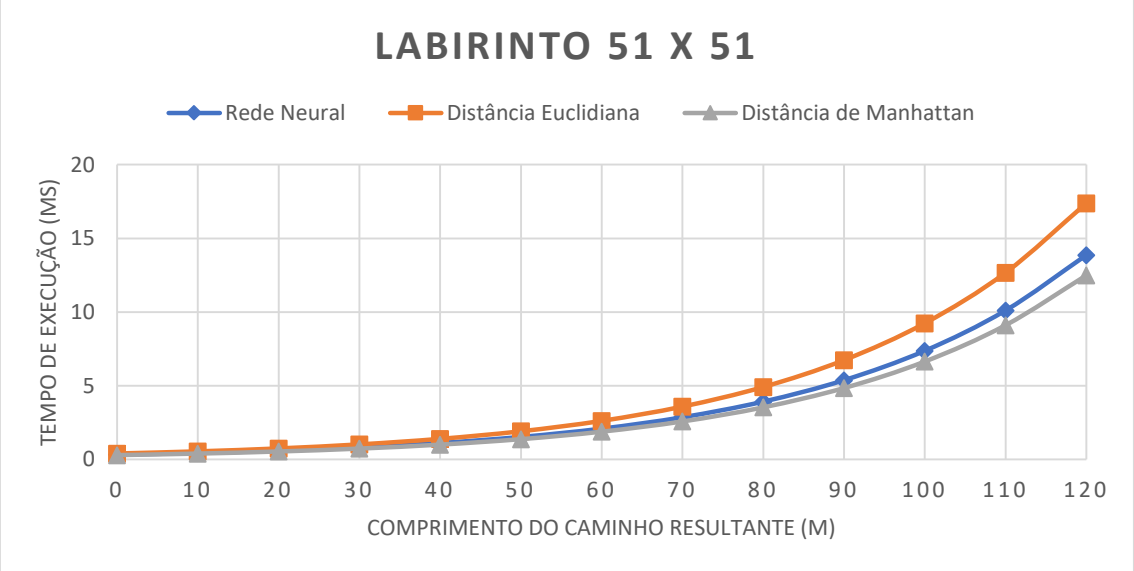
Figura 4.3- Gráfico de comparação de expansão de nodos no Labirinto 1.



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	2.7056329	0.0133905	202.06	< 2e-16
Rede Neural	0.0262517	0.0001727	152.00	< 2e-16
Euclidiana	1.1799651	0.0127127	92.82	< 2e-16
Manhattan	0.9745703	0.0127288	76.56	< 2e-16

Fonte: Autor

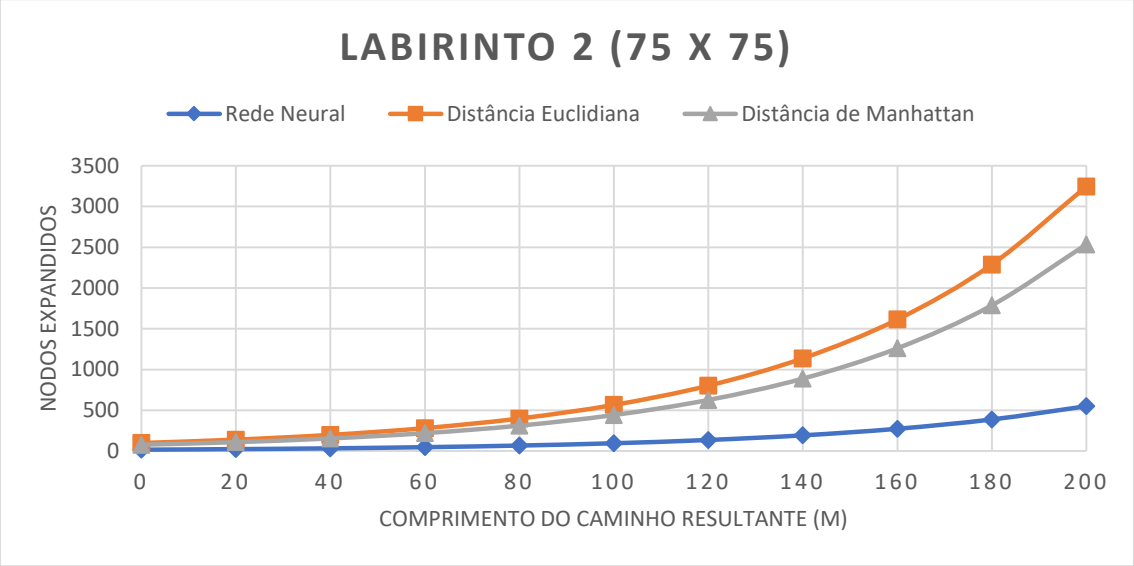
Figura 4.4 - Gráfico de comparação de tempo de execução no mapa Labirinto 1.



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	-1.1652741	0.0199882	-58.298	< 2e-16
Rede Neural	0.0316216	0.0003194	99.001	< 2e-16
Euclidiana	0.2263683	0.0247414	9.149	< 2e-16
Manhattan	-0.1044822	0.0248033	-4.212	2.55e-05

Fonte: Autor

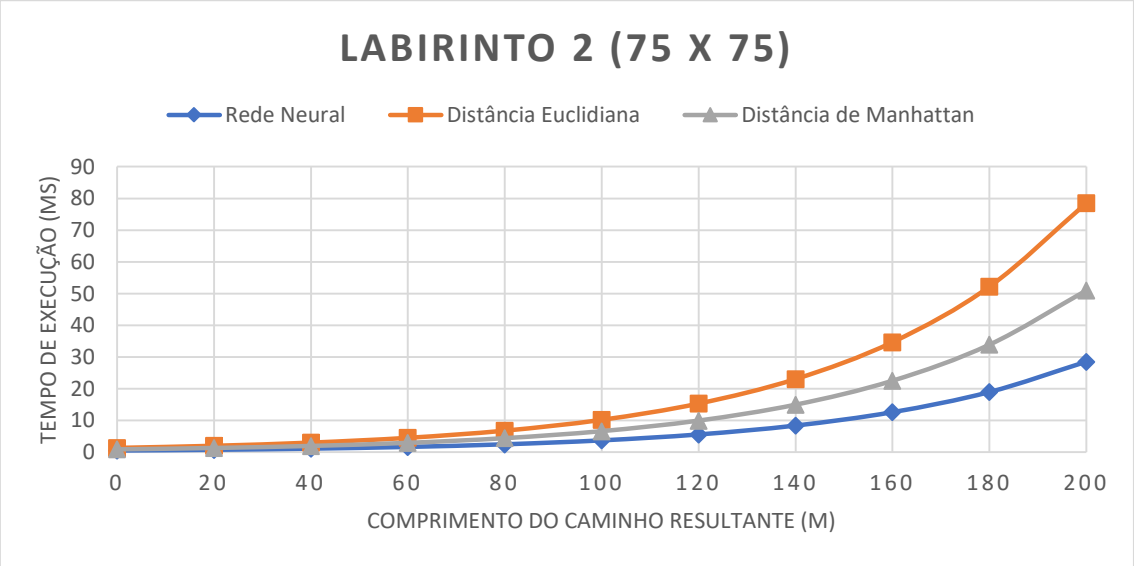
Figura 4.5 - Gráfico de comparação de expansão de nodos no Labirinto 2.



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	2.815e+00	9.908e-03	284.1	< 2e-16
Rede Neural	1.747e-02	9.204e-05	189.8	< 2e-16
Euclidiana	1.776e+00	8.292e-03	214.2	< 2e-16
Manhattan	1.529e+00	8.289e-03	184.4	< 2e-16

Fonte: Autor

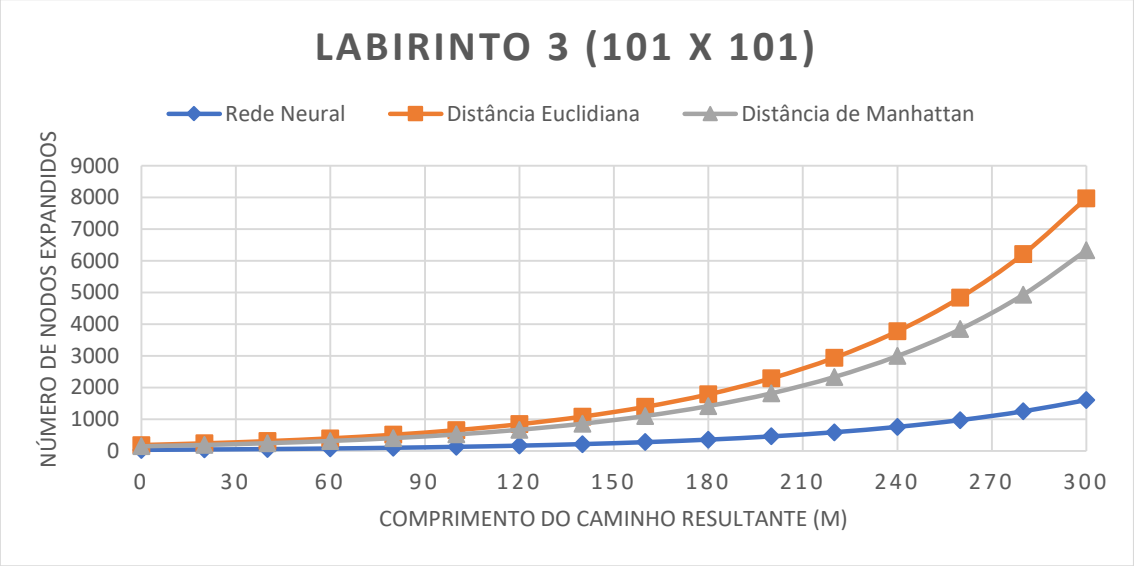
Figura 4.6 - Gráfico de comparação de tempo de execução no mapa Labirinto 2.



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	-0.7396022	0.0151888	-48.69	< 2e-16
Rede Neural	0.0204473	0.0001696	120.59	< 2e-16
Euclidiana	1.0135353	0.0165112	61.38	< 2e-16
Manhattan	0.5814473	0.0164290	35.39	< 2e-16

Fonte: Autor

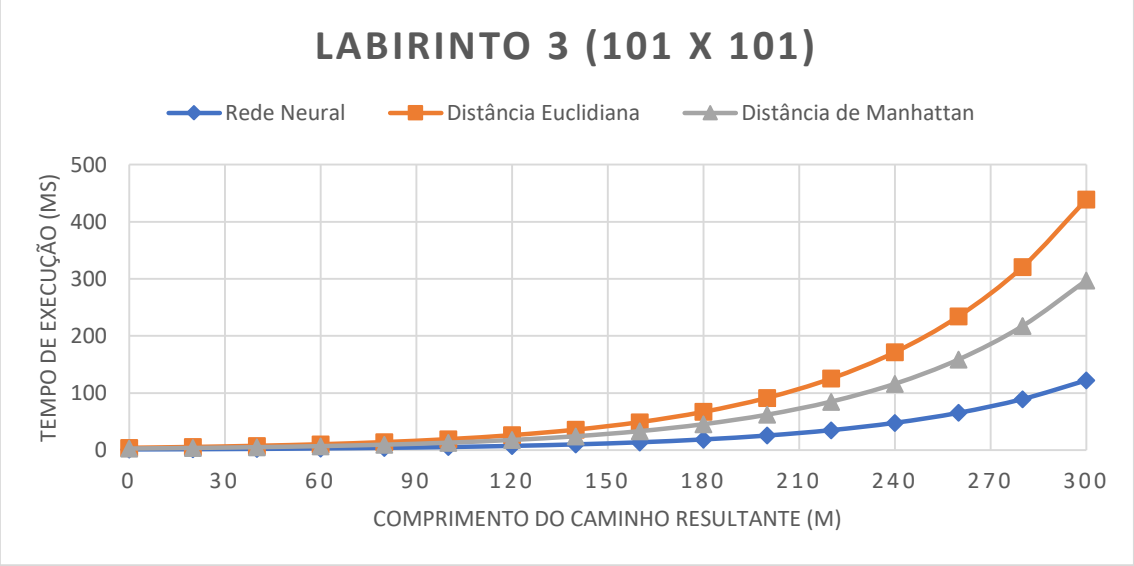
Figura 4.7 - Gráfico de comparação de expansão de nodos no Labirinto 3.



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	3.645e+00	8.068e-03	451.8	< 2e-16
Rede Neural	1.246e-02	4.931e-05	252.8	< 2e-16
Euclidiana	1.601e+00	6.894e-03	232.2	< 2e-16
Manhattan	1.370e+00	6.898e-03	198.6	< 2e-16

Fonte: Autor

Figura 4.8 - Gráfico de comparação de tempo de execução no mapa Labirinto 3.



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	1.046e-01	1.202e-02	8.701	< 2e-16
Rede Neural	1.567e-02	9.087e-05	172.391	< 2e-16
Euclidiana	1.279e+00	1.370e-02	93.364	< 2e-16
Manhattan	8.894e-01	1.376e-02	64.638	< 2e-16

Fonte: Autor

5 CONCLUSÃO

Algoritmos heurísticos de busca de caminhos são usados para encontrar caminhos entre dois pontos em um mapa de navegação. Embora tais heurísticas sejam utilizadas para guiar a busca, a computação de funções heurísticas tradicionais pode não considerar características impeditivas de trafegabilidade que possam estar representadas no mapa de navegação. Para investigar esse problema, a proposta analisada neste TCC envolve a utilização de funções heurísticas que permitam melhor direcionar a busca de caminhos. Mais especificamente, o objetivo do trabalho foi avaliar a viabilidade do uso de redes neurais para o aprendizado de funções heurísticas construídas a partir de dados de caminhos, os quais são computados por algoritmos de busca a partir da estrutura de mapas de navegação.

O desenvolvimento desse trabalho permite concluir que é possível realizar o treinamento de uma rede neural para o aprendizado de uma função heurística para apoiar a execução de algoritmos de busca de caminhos para um determinado mapa cuja estrutura de navegação seja uma grade regular de duas dimensões. Em geral, a técnica apresentada neste trabalho permite concluir que a) a função heurística aprendida pela rede neural reduz a dimensão de busca do algoritmo A*, b) o tempo de execução do algoritmo de busca com o uso de redes neurais também pode ser reduzido e c) o uso de redes neurais permite implementar o processo de busca de caminhos a partir da combinação entre pré-processamento de informações de caminhos (distâncias heurísticas) em mapas e processamento de caminhos em tempo de execução. Entretanto, a técnica investigada neste trabalho possui certas limitações. A principal é que deve ser treinado uma rede neural para cada mapa de navegação em especifico.

Como trabalhos futuros, novas arquiteturas de redes neurais para o aprendizado de funções heurísticas a serem usadas em diferentes algoritmos heurísticos de busca de caminhos podem ser investigadas. Neste TCC, por exemplo, apenas o algoritmo A* foi utilizado para os testes. Além disso, novos experimentos para diferentes cenários e estruturas de mapas de navegação podem ser realizados, com especial atenção para mapas de grandes dimensões e estruturas de navegação representadas de forma hierárquica e irregular, tais como mapas de navegação usados no projeto SIS-ASTROS. Por fim, trabalhos futuros podem propor comparações com algoritmos de busca de caminhos mais recentes, os quais melhor reflitam o estado da arte nessa área de pesquisa de IA.

REFERENCIAS BIBLIOGRÁFICAS

ABADI, M. et al. Tensorflow: A system for large-scale machine learning. 12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16), 2016. p.265-283.

AGOSTINELLI, F. et al. Solving the Rubik's cube with deep reinforcement learning and search. **Nature Machine Intelligence,** v. 1, n. 8, p. 356-363, 2019. ISSN 2522-5839.

ALOM, M. Z. et al. The history began from alexnet: A comprehensive survey on deep learning approaches. **arXiv preprint arXiv:1803.01164**, 2018.

AMATO, F. et al. **Artificial neural networks in medical diagnosis**: Elsevier 2013.

AMMAR, A. et al. Relaxed Dijkstra and A* with linear complexity for robot path planning problems in large-scale grid environments. **Soft Computing,** v. 20, n. 10, p. 4149-4171, 2016. ISSN 1432-7643.

ANGUELOV, B. **Video game pathfinding and improvements to discrete search on grid-based maps**. 2011. University of Pretoria

ARIKI, Y.; NARIHIRA, T. Fully Convolutional Search Heuristic Learning for Rapid Path Planners. **arXiv preprint arXiv:1908.03343**, 2019.

BOTEA, A. et al. **Pathfinding in games**. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

BOTEA, A.; MÜLLER, M.; SCHAEFFER, J. Near optimal hierarchical path-finding. **Journal of game development,** v. 1, n. 1, p. 7-28, 2004.

BRONDANI, J. R.; DE FREITAS, E. P.; SILVA, L. A. A task-oriented and parameterized (semi) autonomous navigation framework for the development of simulation systems. **Procedia Computer Science,** v. 112, p. 534-543, 2017. ISSN 1877-0509.

BRONDANI, J. R. et al. Pathfinding in hierarchical representation of large realistic virtual terrains for simulation systems. **Expert Systems with Applications,** v. 138, p. 112812, 2019. ISSN 0957-4174.

BRONDANI, J. R. et al. Semi-Autonomous Navigation for Virtual Tactical Simulations in the Military Domain. SIMULTECH, 2018. p.443-450.

DIJKSTRA, E. W. A note on two problems in connexion with graphs. **Numerische mathematik,** v. 1, n. 1, p. 269-271, 1959. ISSN 0029-599X.

EBENDT, R.; DRECHSLER, R. Weighted A∗ search–unifying view and application. **Artificial Intelligence,** v. 173, n. 14, p. 1310-1342, 2009. ISSN 0004-3702.

GAMA, J. et al. **Inteligência artificial: uma abordagem de aprendizado de máquina**. Grupo Gen - LTC, 2011. ISBN 9788521618805. Disponível em: < https://books.google.com.br/books?id=4DwelAEACAAJ >.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep learning**. MIT press, 2016. ISBN 0262337371.

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. ISSN 0536-1567.

HINTON, G. et al. Deep neural networks for acoustic modeling in speech recognition. **IEEE Signal processing magazine,** v. 29, 2012.

JINDAL, I. et al. A unified neural network approach for estimating travel time and distance for a taxi trip. **arXiv preprint arXiv:1710.04350**, 2017.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **arXiv preprint arXiv:1412.6980**, 2014.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012. p.1097-1105.

LI, G. et al. ANN: a heuristic search algorithm based on artificial neural networks. Proceedings of the 2016 International Conference on Intelligent Information Processing, 2016, ACM. p.51.

MAAS, A. L.; HANNUN, A. Y.; NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. Proc. icml, 2013. p.3.

MCCULLAGH, P.; NELDER, J. A. **Generalized Linear Models**. CRC Press, 1989. ISBN 0412317605.

MNIH, V. et al. Human-level control through deep reinforcement learning. **Nature,** v. 518, n. 7540, p. 529, 2015. ISSN 1476-4687.

NIELSEN, M. A. **Neural networks and deep learning**. Determination press San Francisco, CA, USA:, 2015.

RONNEBERGER, O.; FISCHER, P.; BROX, T. U-net: Convolutional networks for biomedical image segmentation. International Conference on Medical image computing and computer-assisted intervention, 2015, Springer. p.234-241.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. Malaysia; Pearson Education Limited, 2016.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. **Neural networks,** v. 61, p. 85-117, 2015. ISSN 0893-6080.

SIS-ASTROS. **SIS-ASTROS PROJECT - Project 3.07.0065/Agreement 813782/2014 .** Universidade Federal de Santa Maria 2014.

SOUISSI, O. et al. Path planning: A 2013 survey. Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM), 2013, IEEE. p.1-8.

SUTSKEVER, I.; VINYALS, O.; LE, Q. V. Sequence to sequence learning with neural networks. Advances in neural information processing systems, 2014. p.3104-3112.

TAKAHASHI, T. et al. Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks. Proceedings of the International Conference on Automated Planning and Scheduling, 2019. p.764-772.

UNITY TECHNOLOGIES, I. Software Unity®. unity3d.com, 2019.

WANG, J. et al. Empowering A* Search Algorithms with Neural Networks for Personalized Route Recommendation. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, ACM. p.539-547.