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

Jairo Ferreira Gez

ANÁLISE DE ALGORITMOS DE BUSCA HIERÁRQUICA DE CAMINHOS COM A UTILIZAÇÃO DE REDES NEURAIS PROFUNDAS

Santa Maria, RS
2021

Jairo Ferreira Gez

ANÁLISE DE ALGORITMOS DE BUSCA HIERÁRQUICA DE CAMINHOS COM A UTILIZAÇÃO DE REDES NEURAIS PROFUNDAS

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: Prof. Dr. Luís Alvaro de Lima Silva

481

Santa Maria, RS
2021

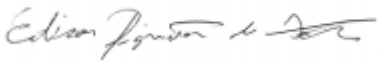
Jairo Ferreira Gez

ANÁLISE DE ALGORITMOS DE BUSCA HIERÁRQUICA COM A UTILIZAÇÃO DE REDES NEURAIS PROFUNDAS

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 09 de Fevereiro de 2021:

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



Edison Pignaton de Freitas, Dr. (UFRGS)

Felipe Martins Muller , Dr. (UFSM)

Santa Maria, RS
2021

RESUMO

ANÁLISE DE ALGORITMOS DE BUSCA HIERÁRQUICA COM A UTILIZAÇÃO DE REDES NEURAIS PROFUNDAS

AUTOR: Jairo Ferreira Gez
ORIENTADOR: Luís Alvaro de Lima Silva

Algoritmos de pathfinding e redes neurais profundas têm ganhado destaque na área de Inteligência Artificial (IA). Em geral, estes algoritmos contam com uma heurística que guia a busca a cada nodo expandido durante a busca do caminho. O problema é que funções heurísticas tradicionalmente usadas por vários algoritmos de pathfinding, a exemplo da distância Euclidiana e a distância de Manhattan, não consideram as características dos mapas virtuais em que o caminho é computado, tendendo a expandir cada vez mais nodos a medida em que o comprimento do caminho cresce. Tal expansão de nodos acaba gerando tempos de busca elevados quando grandes espaços de busca estão sendo tratados. Para abordar esse problema, este trabalho investiga como usar redes neurais profundas na construção de funções heurísticas para algoritmos de pathfinding hierárquicos, permitindo assim otimizar a busca de caminhos em mapas virtuais de grandes dimensões e diferentes naturezas. Especificamente, este trabalho explora a implementação de um algoritmo hierárquico de busca, o qual divide os mapas virtuais em sub-mapas, gerando um nível de abstração para acelerar o processo de busca. Uma vez que o caminho abstrato está pronto, o algoritmo faz um refinamento no nível mais baixo de abstração, empregando o algoritmo de busca A*. O principal objetivo deste trabalho é, portanto, verificar se o A* empregado em cada sub-mapa pode ser otimizado com a utilização de redes neurais profundas como funções heurísticas. Experimentos foram realizados com o emprego de três diferentes tipos de mapas virtuais, onde resultados obtidos foram analisados a partir de técnicas estatísticas de regressão. A partir dos resultados obtidos, o algoritmo hierárquico explorado neste trabalho foi otimizado em termos de tempo e nodos expandidos.

Palavras-chave: Pathfinding hierárquico. Redes neurais profundas. Pathfinding. Inteligência Artificial.

Santa Maria, RS
2021

ABSTRACT

ANALYSIS OF HIERARCHICAL PATHFINDING ALGORITHMS WITH THE USE OF DEEP NEURAL NETWORKS

AUTHOR: Jairo Ferreira Gez
ADVISOR: Luís Alvaro de Lima Silva

Pathfinding algorithms and deep neural networks have gained prominence in the area of Artificial Intelligence (AI). In general, these algorithms have a heuristic that guides the search for each expanded node during the path search. The problem is that heuristic functions traditionally used by various pathfinding algorithms, such as the Euclidean distance and the Manhattan distance, do not consider the characteristics of the virtual maps in which the path is computed, tending to expand more and more nodes as the path length grows. Such expansion of nodes ends up generating long search times when large search spaces are being treated. To solve this problem, this work investigates how to use deep neural networks in the construction of heuristic functions for hierarchical pathfinding algorithms, thus allowing to optimize the search for paths in large virtual maps and different natures. Specifically, this work explores the implementation of a hierarchical search algorithm, which divides virtual maps into sub-maps, generating a level of abstraction to speed up the search process. Once the abstract path is ready, the algorithm makes a refinement at the lowest level of abstraction, using the search algorithm A *. The main objective of this work is, therefore, to verify if the A * used in each sub-map can be optimized with the use of deep neural networks as heuristic functions. Experiments were carried out using three different types of virtual maps, where results obtained were analyzed using statistical regression techniques. From the results obtained, the hierarchical algorithm explored in this work was optimized in terms of time and expanded nodes.

Keywords: Hierarchical Pathfinding. Deep neural networks. Pathfinding. Artificial intelligence.

Santa Maria, RS
2021

LISTA DE FIGURAS E ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

1 INTRODUÇÃO

O estudo de Redes Neurais Profundas (Deep Neural Networks - DNN) (Goodfellow 2016) na área de Inteligência Artificial (IA) está em crescente expansão. Entre outras pesquisas em IA, DNN podem ser exploradas na otimização de algoritmos de pathfinding que utilizam funções heurísticas para guiar o processo de busca por nodos de menor custo. Tal como no algoritmo A* (Algfoor et al., 2015), a função de custo explorada é dada por: f(n) = g(n) + h(n), onde f(n) retorna o custo total do nodo (n), g(n) retorna o custo do caminho até o nodo (n) e h(n) retorna uma estimativa heurística de custo do nodo (n) até o nodo objetivo. Essas funções heurísticas oferecem uma estimativa de custo para alcançar o estado objetivo a cada estado do problema investigado no processo de busca no espaço de estados. Tais estimativas heurísticas são normalmente computadas por meio de funções conhecidas, como a distância Euclidiana e a distância de Manhattan, por exemplo.

DNN vêm sendo exploradas na otimização de algoritmos de busca em diferentes aplicações. Entre outras abordagens, essa otimização tem sido voltada para a construção de funções heurísticas ajustadas para problemas de aplicação, permitindo obter melhores estimativas de custo para alcançar um estado objetivo. Neste contexto, (Ariki e Narihira., 2019) utilizam redes neurais convolucionais para a geração de mapas heurísticos utilizados por planejadores de caminhos em grids regulares; (Agostinelli et al., 2019) exploram o problema de pathfinding em jogos tais como o cubo mágico com o emprego de diversas técnicas de redes neurais artificiais; (Jindal et al., 2017) propõem o uso de RNA para computar distâncias a partir de coodenadas de GPS sem utilizar caminhos; (Li et al., 2016) utilizam redes nerais de regressão para computar caminhos dentro de grids regulares. Apesar destes trabalhos, ainda permanece em aberto o emprego de DNN na otimização de computações de caminhos em grandes espaços de estados gerados pela utilização de mapas virtuais de grandes dimensões e diferentes naturezas.

No projeto de pesquisa onde este Trabalho de Conclusão de Curso – TCC está inserido, (Doebber, 2019) descreve como implementar, treinar e testar DNN no desenvolvimento de algoritmos hierárquicos de pathfinding. Embora não considerando o emprego de DNN, tal como investigado neste TCC, algoritmos hierárquicos descritos em (Algfoor et al., 2015), por exemplo, realizam um pré-processamento de mapas virtuais, dividindo-os em clusters. A partir do conjunto de nodos que conectam as bordas desses clusters, uma estrutura abstrata de grafo permite orientar e otimizar a busca de caminhos em mapas de grandes dimensões. Neste contexto, (Souza, 2021) apresenta a integração de DNN

com algoritmos hierárquicos de busca para tratamento de mapas virtuais de grandes dimensões. Embora os trabalhos descritos em (Doebber, 2019) e (Souza, 2021) tenham apresentado resultados promissores, um maior conjunto de testes das técnicas de pathfinding e DNN propostas ainda precisa ser desenvolvido. Além de obter resultados experimentais em diferentes tipos de mapas virtuais, expandindo os testes apresentados em (Doebber, 2019) e (Souza, 2021), existe a necessidade de propor e testar outras técnicas hierárquicas de pathfinding onde as funções heurísticas usadas pelos algoritmos de busca são computadas por DNN.

O objetivo deste trabalho é, portanto, investigar como explorar algoritmos de pathfinding hierárquicos a partir do emprego de DNN como suas funções heurísticas. A partir disso, a pesquisa deve realizar análises a partir de resultados experimentais de pathfinding obtidos em testes com um conjunto mais amplo de mapas virtuais de natureza e dimensões distintas.

Resultados experimentais são analisados via modelos estatísticos de regressão linear, permitindo comparar os diferentes algoritmos e técnicas implementadas. Por fim, é importante salientar que este TCC está inserido no contexto do projeto de pesquisa “Sistema Integrado de Simulação ASTROS - Grupo de Mísseis e Foguetes” (SIS-ASTROS GMF), o qual é desenvolvido em parceria com o Exército Brasileiro. Pelo emprego de diferentes algoritmos de pathfinding na busca de caminhos para agentes inseridos em ambientes virtuais simulados representando terrenos reais de grandes dimensões, o simulador SIS-ASTROS explora a utilização de mecanismos de IA no treinamento de comandantes de frações em exercícios táticos de Reconhecimento, Escolha e Ocupação de Posição (REOP) de baterias de mísseis de foguetes.

O TCC está organizado da seguinte maneira: o segundo capítulo apresenta uma revisão da literatura, dos principais algoritmos de busca empregados na solução proposta e dos mecanismos de redes neurais, além de fazer uma revisão do estado da arte para apresentar e contextualizar a pesquisa acerca de trabalhos relacionados. O terceiro capítulo aborda a metodologia explorada no desenvolvimento da pesquisa, descrevendo como os objetivos almejados foram abordados. O quarto capítulo trata dos experimentos e resultados obtidos. Finalmente, o quinto capítulo apresenta considerações finais e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Sistemas de simulação visam, por meio de ferramentas matemáticas, imitar um processo ou simulação do mundo real. Diversas técnicas de IA são constantemente exploradas na busca e aprimoramento dessas ferramentas. Entre outras técnicas, estão os algoritmos de busca de caminhos, os quais são investigados neste TCC. Entre outras aplicações, a busca de caminhos visa planejar rotas para o deslocamento de agentes terrestres sendo simulados, os quais interagem com o ambiente simulado, buscando expor um comportamento que mais se pareça com o que é naturalmente desempenhado pela entidade real simulada.

2.1 ALGORITMOS DE BUSCA DE CAMINHOS

Algoritmos de busca de caminhos (pathfinding) (Algfoor *et al.*, 2015) são ferramentas utilizadas para planejar a movimentação de agentes em jogos de computador e sistemas de simulação, entre outras aplicações da vida real. Para isso, tais algoritmos podem considerar as características e restrições inerentes aos mapas virtuais em que os mesmos estão sendo usados. Como elementos primordiais à busca de caminhos, (Botea et al.,2013) define a existência de uma estrutura de dados contendo a representação do mapa. Essa estrutura é comumente representada por um grafo.

Um algoritmo de busca de caminhos pode explorar uma função heurística para guiar a busca. (Anguelov, 2011) explica que esses algoritmos podem ser divididos em duas frentes:

- Soluções heurísticas, as quais são guiadas por uma função heurística que provê uma estimativa de qual melhor trajeto a seguir, guiando a busca;
- Soluções não-heurísticas, as quais não contam com funções de estimativa.

Uma vasta quantidade de algoritmos de busca é proposta na literatura. As implementações desenvolvidas neste trabalho utilizam os seguintes: Dijsktra (Dijkstra, 1959), Busca em Largura (Breadth-First Search - BFS) (Zuse, 1972), A* (Hart et al.,1968) e HPA* (Botea et al.,2004). Dentre as funções heurísticas utilizadas no trabalho, estão a distância Euclidiana e a distância de Manhattan.

2.1.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra (Dijkstra, 1959) gera uma árvore de extensão mínima

(Minimum Spaninng Tree – MST) por meio de uma solução não-heurística. A busca consiste em adicionar iterativamente nodos candidatos a uma fila com prioridades cujo primeiro elemento é o que contém o menor custo. A busca encerra quando o nodo objetivo é retirado da fila ou quando não houverem mais candidatos na fila – significando que não há caminho resultante. O pseudocódigo do algoritmo de Dijkstra é apresentado no Algoritmo 2.1.

Algoritmo 2.1: Dijkstra	
1	**Dijkstra(Grafo, NodoInicial, NodoObjetivo)**
2	**Para cada** nodo **em** Grafo **faça**
3	nodo.g = **infinito**
4	nodo.visitado = **falso**
5	nodo.pai = **nulo**
6	**Fim para**
7	NodoInicial.g = 0
8	NodosAbertos = CriaFilaComPrioridades()
9	NodosAbertos.Inserir(NodoInicial)
10	**Enquanto** (!NodosAbertos.Vazio()) faça
11	nodoAtual = NodosAbertos.RemoveFrente()
12	nodoAtual.Visitado = verdadeiro
13	**Se** (nodoAtual == NodoObjetivo) **faça**
14	**Retorne** nodoAtual
15	**Fim se**
16	**Para cada** vizinho **em** nodoAtual.Vizinhos **faça**
17	G = nodoAtual.g + distancia(nodoAtual, vizinho)
18	**Se** (G <= vizinho.g) **faça**
19	vizinho.g = G
20	vizinho.pai = nodoAtual
22	**Se** (!vizinho.visitado) **faça**
22	NodosAbertos.Inserir(vizinho)
23	**Fim se**
24	**Fim se**
25	**Fim para**
26	**Fim enquanto**
27	**Retorne nulo**

Fonte: Autor.

O comportamento do algoritmo de Dijkstra pode ser verificado na Figura 2.1. Essa figura apresenta um grid regular, com nodo de partida colorido em verde escuro e nodo objetivo colorido em vermelho. Os nodos coloridos em cinza escuro estão bloqueados, ou seja, não são trafegáveis. Os nodos coloridos em branco estão livres para tráfego. Os nodos expandidos estão coloridos de azul e verde claro, sendo os verde-claros pertencentes ao conjunto de nodos abertos quando o melhor caminho foi encontrado. A linha amarela representa o caminho resultante.

Figura 2.1 - Visualização da execução do algoritmo de Dijkstra

Fonte: https://qiao.github.io/PathFinding.js/visual/.

Por gerar a árvore de extensão mínima, pode-se notar que o algoritmo expande os nodos de uma maneira equilibrada em termos de distância do nodo inicial.

2.1.2 Algoritmo de Busca em Largura

O algoritmo de busca em largura (Zuse, 1972), segue a mesma lógica do algoritmo de Dijkstra. Porém, ele utiliza uma fila simples em sua implementação, de tal maneira que todo nodo vizinho identificado é automaticamente adicionado ao final da fila.

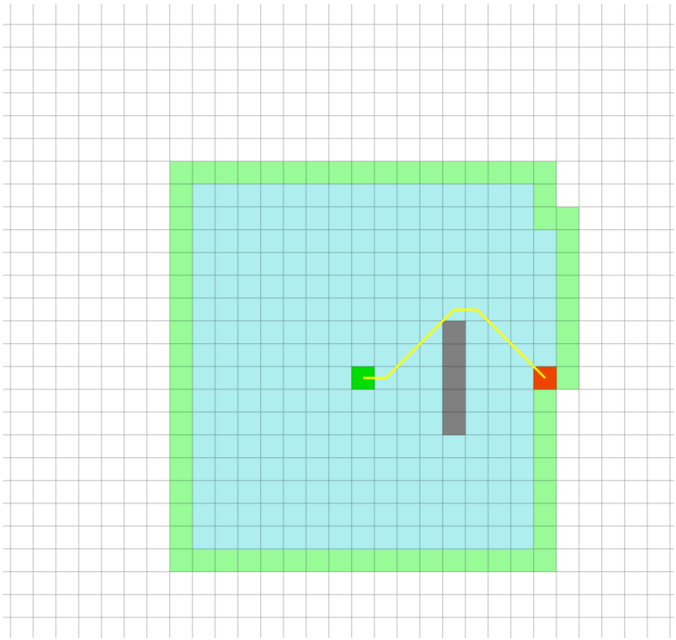
Algoritmo 2.2: BFS	
1	**BFS(Grafo, NodoInicial, NodoObjetivo)**
2	**Para cada** nodo **em** Grafo **faça**
3	nodo.g = **infinito**
4	nodo.visitado = **falso**
5	nodo.pai = **nulo**
6	**Fim para**
7	NodoInicial.g = 0
8	NodosAbertos = CriaFilaSimples()
9	NodosAbertos.Inserir(NodoInicial)
10	**Enquanto** (!NodosAbertos.Vazio()) **faça**
11	nodoAtual = NodosAbertos.RemoveFrente()
12	nodoAtual.Visitado = verdadeiro
13	**Se** (nodoAtual == NodoObjetivo) **faça**
14	**Retorne** nodoAtual
15	**Fim se**
16	**Para cada** vizinho **em** nodoAtual.Vizinhos **faça**
17	G = nodoAtual.g + distancia(nodoAtual, vizinho)
18	**Se** (G <= vizinho.g) **faça**
19	vizinho.g = G
20	vizinho.pai = nodoAtual
22	**Se** (!vizinho.visitado) **faça**
22	NodosAbertos.Inserir(vizinho)
23	**Fim se**
24	**Fim se**
25	**Fim para**
26	**Fim enquanto**
27	**Retorne** nulo

Fonte: Autor.

O comportamento do algoritmo de busca em largura pode ser verificado na Figura 2.2. Essa figura apresenta um grid regular, com nodo de partida colorido em vermelho e nodo objetivo colorido em azul. Os nodos coloridos em cinza escuro estão bloqueados, ou seja, são não-

trafegáveis. Já os nodos coloridos em branco estão livres para tráfego. Os nodos expandidos estão coloridos de azul e verde claro, sendo os verde-claros pertencentes ao conjunto de nodos abertos quando o melhor caminho foi encontrado. A linha amarela representa o caminho resultante.

Figura 2.2 - Visualização da execução algoritmo de Busca em Largura



Fonte: https://qiao.github.io/PathFinding.js/visual/.

Em comparação com o algoritmo de Dijkstra, o BFS também expande os nodos de maneira a manter sempre pouca variação na distância entre os nodos abertos e a origem. Contudo, o comportamento do algoritmo BFS não gera a árvore de extensão mínima por empregar uma fila simples.

2.1.3 Algoritmo A*

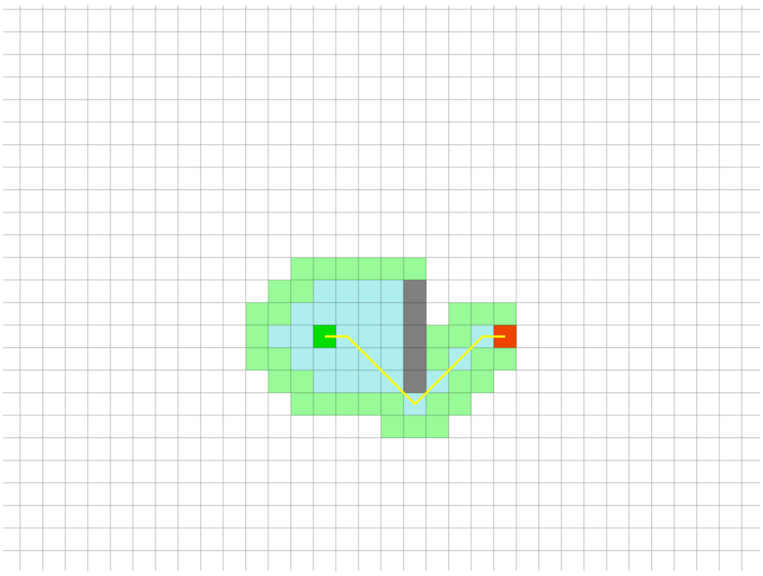
Baseado no algoritmo de Dijkstra, (Hart et al.,1968) propôs o algoritmo A*. Essa algoritmo apresenta uma abordagem heurística que adiciona o resultado de uma função heurística implementada ao custo de cada nodo, tendo o custo do nodo como: $f(n) = g(n) + h(n)$.

Algoritmo 2.3: A* tradicional	
1	**Astar(Grafo, NodoInicial, NodoObjetivo)**
2	**Para cada** nodo **em** Grafo **faça**
3	nodo.g = **infinito**
5	nodo.pai = **nulo**
6	**Fim para**
7	NodoInicial.g = 0
8	NodosAbertos = CriaFilaComPrioridades()
9	NodosAbertos.Inserir(NodoInicial)
10	NodosFechados = CriaFilaSimples()
10	**Enquanto** (!NodosAbertos.Vazio()) faça
11	nodoAtual = NodosAbertos.RemoveFrente()
12	NodosFechados.Inserir(nodoAtual)
13	**Se** (nodoAtual == NodoObjetivo) **faça**
14	**Retorne** nodoAtual
15	**Fim se**
16	**Para cada** vizinho **em** nodoAtual.Vizinhos **faça**
17	**Se (!NodosFechados.Contem(nodoAtual) faça**
18	G = nodoAtual.g + distancia(nodoAtual, vizinho)
19	**Se** (G <= vizinho.g ou !NodosAbertos.Contem(vizinho)) **faça**
20	vizinho.g = G
21	vizinho.h = CalculaHeuristica(vizinho, NodoObjetivo)
22	vizinho.pai = nodoAtual
23	**Se** (!NodosAbertos.Contem(vizinho)) **faça**
24	NodosAbertos.Inserir(vizinho)
25	**Fim se**
26	**Fim se**
27	**Fim se**
28	**Fim para**
29	**Fim enquanto**
30	**Retorne nulo**

Fonte: Autor.

O comportamento do algoritmo A* pode ser verificado na Figura 2.3, a qual apresenta um grid regular, com nodo de partida colorido em vermelho e nodo objetivo colorido em azul. Os nodos coloridos em cinza escuro estão bloqueados, ou seja, são não-trafegáveis. Os nodos coloridos em branco estão livres para tráfego. Os nodos expandidos estão coloridos de azul e verde claro, sendo os verde-claros pertencentes ao conjunto de nodos abertos quando o melhor caminho foi encontrado. A linha amarela representa o caminho resultante.

Figura 2.3- Visualização da execução do algoritmo A*



Fonte: https://qiao.github.io/PathFinding.js/visual/

Pode-se notar que, em comparação com os algoritmos de Dijkstra e BFS, o A* expande menos nodos na busca, resultado da utilização de funções heurísticas.

2.1.4 Algoritmo Hierarchical Pathfinding A* (HPA*)

Definido por (Botea et al., 2004), o Hierarchical Pathfinding A* surgiu com a proposta de otimizar a busca em mapas de grandes dimensões, adicionando níveis de hierarquia ao processo de busca em etapas:

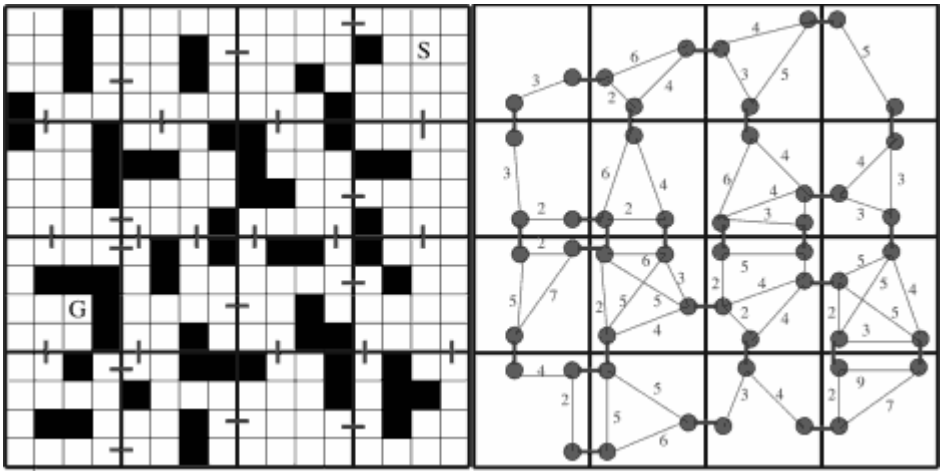
1. O mapa de estados, tomado como um grid regular, é dividido em sub-mapas, chamados de clusters. Essa subdivisão é realizada como parte de atividades de pré-processamentos dos mapas, antes do algoritmo de busca ser computado em uma determinada aplicação;

2. Os nodos trafegáveis pertencentes as fronteiras entre os clusters são detectados e interligados, onde o custo de movimentação deste nodos é atribuído com o custo mínimo para movimentação -normalmente uma unidade. Essas informações computadas são então representadas em um grafo abstrato que descreve o terreno, o qual é usado no processo hierárquico de busca.

Algoritmo 2.4: Pré-processamento feito pelo HPA*	
1	**PreProcessamento(GrafoConcreto, TamanhoDoCluster)**
2	LarguraCluster = **MAX**(GrafoConcreto.Largura / TamanhoDoCluster)
3	AlturaCluster = **MAX**(**GrafoConcreto**.Altura / TamanhoDoCluster)
5	GrafoAbstrato.Clusters = Cluster[LarguraCluster][AlturaCluster]
6	**Para cada** cluster1 e cluster2 **em** GrafoAbstrato.Clusters **faça**
7	**Se** Vizinhos(cluster1, cluster2) **faça**
8	**Para cada** nodo1 **em** cluster1 **faça**
9	**Para cada** nodo2 **em** cluster2 **faça**
10	**Se**(Vizinhos(nodo1,nodo2) **&** !Bloqueado(nodo1) **&** !Bloqueado(nodo2)) f**aça**
10	GrafoAbstrato**.Add(nodo1, nodo2, distancia = 1)**
11	**Fim se**
12	**Fim para**
13	**Fim para**
14	**Para cada** nodo1 **em** cluster1 **faça**
15	**Para cada** nodo2 **em** cluster1 **faça**
16	**Se** Borda(nodo1) **&** Borda(nodo2) **&** !Bloqueado(nodo1) **&** !Bloqueado(nodo2)
17	GrafoAbstrato.Add(nodo1, nodo2, distancia = AlgoritmodeBusca(nodo1,nodo2)
18	**Fim se**
19	**Fim para**
20	**Fim para**
21	**Fim para**
22	**Retorne** GrafoAbstrato

Fonte: Autor.

Figura 2.4 - Visualização da execução do pré-processamento do algoritmo HPA*



Fonte: adaptado de (Botea, 2004).

O grid à esquerda da Figura 2.4 mostra os nodos selecionados, nas bordas dos clusters. O conjunto de nodos resultante desta primeira etapa de busca que envolve as bordas dos clusters, a qual ocorre uma vez só quando o mapa é carregado em uma determinada aplicação, representa o primeiro conjunto de nodos em que o algoritmo é aplicado. Esse processo de busca é realizado em um nível abstrato do mapa. Ao repetir esse processo, é possível adicionar mais níveis de hierarquia ao processo, sempre que for necessário em um determinado problema de aplicação. Concluída a etapa de construção do grafo abstrato (Figura 4), o algoritmo HPA* pode então realizar a busca de caminhos em quaisquer combinações de nodos do mapa. Isso acontece da seguinte maneira:

1. Os nodos de origem e destino são adicionados ao grafo abstrato, tendo seus custos até as fronteiras de seus respectivos clusters computados;
2. Um algoritmo de busca, Dijkstra no caso deste TCC, é utilizado para encontrar o melhor caminho entre esses nodos no grafo abstrato;
3. Uma vez que o caminho abstrato foi encontrado, um processo de refinamento é realizado dentro de cada cluster. Neste refinamento, todos os nodos pertencentes ao mapa de estados original são usados;
4. O conjunto de caminhos refinados que são encontrados em cada cluster é aglutinado, gerando um caminho concreto. Esse caminho concreto é então retornado pelo algoritmo.

Algoritmo 2.5: Busca online feita pelo HPA*	
1	**BuscaOnline(GrafoAbstrato, GrafoConcreto, Nodoinicial, NodoObjetivo)**
2	InsereNodo(GrafoAbstrato, NodoInicial)
3	InsereNodo(GrafoAbstrato, NodoObjetivo)
5	CaminhoAbstrato = GrafoAbstrato.Busca(NodoInicial, NodoObjetivo)
6	CaminhoConcreto = CriaFila()
7	**Enquanto** CaminhoAbstrato.Tamanho > 1 **faça**
8	i = IndiceCluster(CaminhoAbstrato.Top)
9	CaminhoLocal = Astar(Clusters[i], CaminhoAbstrato.Pop, CaminhoAbstrato.Pop)
10	**Para cada** nodo **em** CaminhoLocal **faça**
10	**CaminhoConcreto.Add(nodo)**
11	**Fim para**
12	**Fim enquanto**
13	**Se** CaminhoAbstrato.Tamanho == 1 **faça**
14	CaminhoConcreto.Add(CaminhoAbstrato.Pop)
15	**Fim se**
16	RemoveNodo(GrafoAbstrato, NodoInicial)
17	RemoveNodo(GrafoAbstrato, NodoObjetivo)
18	**Retorne** CaminhoConcreto

Fonte: Autor.

A fim de reduzir a complexidade do A* tradicional na busca em mapas de grandes dimensões, (Botea et al.,2004) descreve como explorar uma abordagem que utiliza níveis de hierarquia no processo de busca. O algoritmo faz um pré-processamento do mapa original, dividindo-o em sub-mapas ou clusters, onde as bordas de cada cluster são interligadas e as distâncias entre suas adjacências são armazenadas em um grafo. Isso permite gerar uma representação abstrata do mapa real: um grafo abstrato. Na execução do HPA*, em cada um dos clusters do mapa original é gerado um "caminho abstrato". Esse caminho abstrato é gerado a partir da aplicação de um algoritmo A* tradicional. Posteriormente, os sub-caminhos entre cada nodo desse grafo abstrato são computados no mapa original, gerando um "caminho concreto". Segundo (Botea et al.,2004), o processamento do algoritmo é pelo menos 10 vezes mais rápido do que o A* tradicional, encontrando caminhos que estão dentro do 1% do ideal (caminhos ótimos de menor custo/menor distância). Neste TCC, esta abordagem hierárquica de busca é investigada em conjunto com o emprego de DNN. Neste caso, diferentes DNN

devidamente treinadas são empregadas para oferecer estimativas heurísticas de custo, as quais guiam a análise do espaço de estados em cada um dos diferentes clusters determinados de acordo com o algoritmo HPA*.

2.2 REDES NEURAIS

No contexto de aprendizado e reconhecimento de padrões em grandes volumes de dados, na área de Inteligência Artificial (IA), as Redes Neurais Artificiais (RNA) visam modelar um comportamento complexo, não-linear e paralelo do conjunto de neurônios biológicos humanos.

2.2.1 Redes Neurais Artificiais

As RNA são modelos matemáticos que adquirem, armazenam e utilizam conhecimento através da experiência (Takahashi et al., 2019). O modelo matemático que descreve o neurônio biológico é chamado de “perceptron” (Ilustração 2.1), o qual possui um conjunto de entradas (x1, x2, x3, ..., xn). Atrelados a essas entradas estão os pesos sinápticos (w1, w2, w3, ..., wn). Ao somatório do produto dessas entradas com seus respectivos pesos sinápticos é adicionado um bias b, que serve para controlar o grau de liberdade dos ajustes dos pesos.

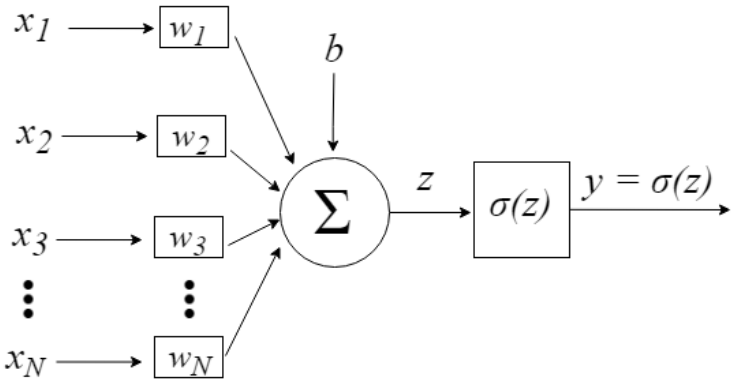
Ilustração 2.1 – Somatório que define parcialmente a saída produzida pelo neurônio

$$z = \sum_{i=1}^{N} x_i w_i + b$$

Fonte: Autor.

O resultado dessas operações passa por uma função de ativação, que visa dar não-linearidade à saída y do neurônio. Ilustração 2.2 mostra a estrutura da ferramenta matemática que modela o neurônio.

Ilustração 2.2 – Modelo matemático do neurônio artificial



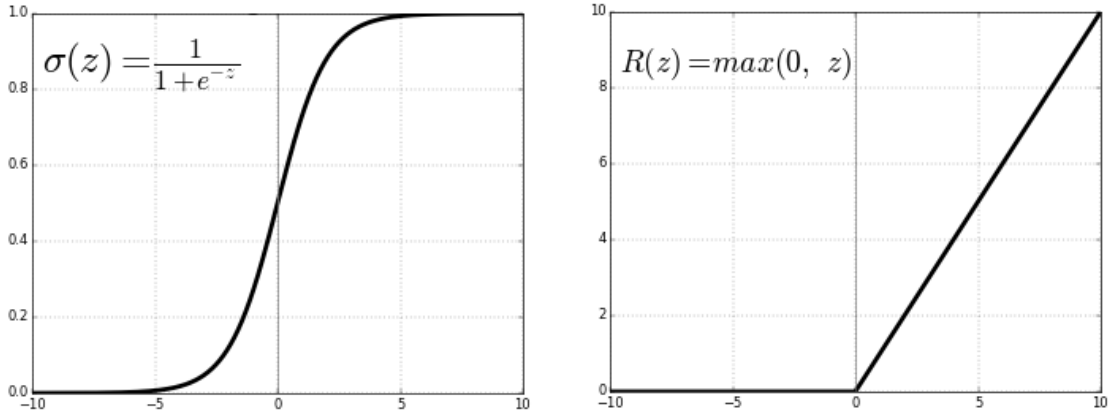
Fonte: Autor.

As funções de ativação mais conhecidas, e utilizadas neste trabalho, são:

- Sigmoide: com o contradomínio compreendido entre 0 e 1, a função sigmoide tende a produzir resultados próximos de seus extremos, o que é bastante útil para classificação de objetos. A fórmula dessa função é a seguinte: $f(x)= 1/(1+e^{-x})$;
- ReLU: com contradomínio compreendido entre 0 e mais infinito, essa função de ativação se caracteriza por ignorar os valores negativos, retornando 0 para eles. É a função mais comumente utilizada para projetar redes neurais atualmente.

Figura 2.6 mostra o comportamento gráfico dessas funções.

Figura 2.6 - Visualização gráfica das funções de ativação Sigmoide (esq.) e ReLU (dir.)

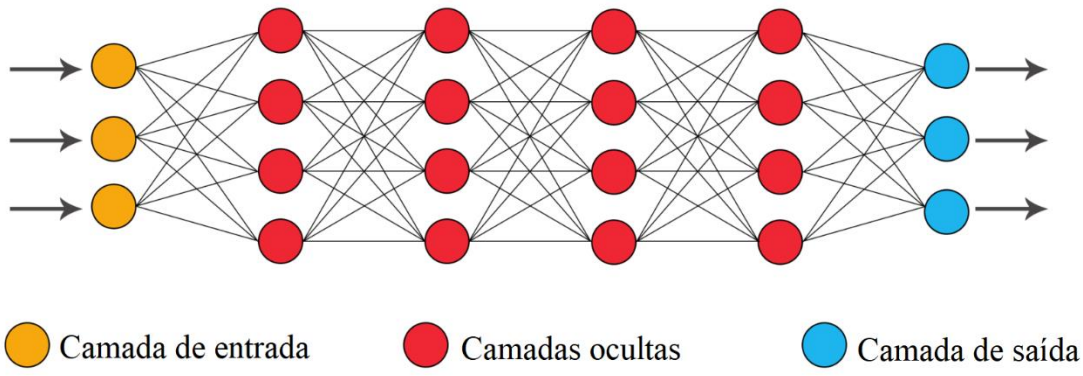


Fonte: Autor.

2.2.2 Redes Neurais Profundas

Normalmente, uma RNA possui uma camada de neurônios de entrada – a qual recebe as informações a serem processadas, uma camada intermediária – a qual processa essas informações - e uma camada de saída, que contém o resultado produzido. Quando uma rede neural possui mais de uma camada intermediária, ou oculta, diz-se que essa rede neural é do tipo profunda (Deep Neural Network – DNN). Figura 2.7 permite visualizar a organização dos neurônios de uma rede neural profunda, com três neurônios na camada de entrada, quatro camadas ocultas contendo quatro neurônios cada, e uma camada de saída contendo três neurônios.

Figura 2.7 – Estruturação de uma rede neural profunda



Fonte: Autor.

As redes neurais profundas compõem uma importante subárea de pesquisa da IA, chamada de aprendizado profundo (Deep Learning) (Goodfellow et al., 2016). As principais arquiteturas de redes neurais profundas são:

- Redes neurais feedforward: este tipo de rede neural opera uma série de cálculos que transforma e altera as semelhanças entre os padrões analisados. As computações aplicadas em cada camada são uma função não-linear das atividades na camada anterior, sempre passadas a frente;
- Redes neurais recorrentes: diferentemente das redes neurais feedforward, esse tipo de rede neural opera em ciclos e a informação passada de uma camada a outra pode voltar e ser reprocessada.

2.2.3 Treinamento das Redes Neurais Profundas

Os mecanismos que permitem que essas ferramentas matemáticas aprendam com as informações adquiridas podem ser implementados por meio de algoritmos de treinamento. Esse algoritmos se dividem, principalmente, em duas frentes:

- Treinamento supervisionado, o qual dispõe de um conjunto de saída para comparação e treinamento das saídas do modelo utilizado;
- Treinamento não-supervisionado, o qual não dispõe de exemplos de comparação, classificando as entradas em classes de dados semelhantes.

O algoritmo de treinamento utilizado neste TCC é conhecido como retro-propagação (backpropagation), o qual é do tipo supervisionado. Com base no cálculo do erro ocorrido na camada de saída da rede neural, são recalculados os valores dos pesos do vetor de entrada da última camada de neurônios. Efetuando o mesmo processo para as camadas anteriores, de trás para a frente, o algoritmo atinge a camada de entrada da rede, e visa diminuir esse erro a cada iteração. As etapas do backpropagation são as seguintes:

1. Inicializar todos os pesos atrelados aos neurônios da rede com valores aleatórios;
2. Alimentar a rede e calcular o erro obtido ao comparar a saída atual com o valor de saída pré-estabelecido (uma vez que o treinamento é supervisionado);
3. Calcular o vetor gradiente associado a cada neurônio da rede;

4. Atualizar o peso de cada neurônio de maneira iterativa a partir dos gradientes obtidos no passo três e voltar ao passo dois até que o erro obtido seja inferior ao requerido ou que a quantidade máxima de iterações, chamadas de épocas, seja atingida.

Outra característica importante a ser levada em consideração no treinamento de redes neurais é o número de dados (batch size), que cada iteração, ou época, possui. Esse parâmetro é útil, porque é ele quem dita a frequência com a qual a rede neural atualiza os pesos sinápticos e o grau de liberdade. Deve-se levar em consideração que, quanto maior for o conjunto de dados de treinamento, mais memória é utilizada na computação, e mais tempo é demandado na execução de cada interação. Por outro lado, menores tamanhos de batch levam a menores custos em termos de memória e tempo para cada iteração, pois não carregam todos os dados para a memória de uma só vez.

2.3 TRABALHOS RELACIONADOS

Algoritmos de pathfinding buscam, geralmente, o menor caminho trafegável (ou com menor custo de acordo com diferentes critérios) deste uma posição inicial (nodo inicial) até uma posição final (nodo objetivo) usando a representação do mapa virtual. Os elementos principais da busca de caminhos são um grafo de representação do espaço de estados, um algoritmo de busca e uma função heurística para orientar a pesquisa. Funções heurísticas amplamente usadas em algoritmos de pathfinding (Algfoor et al., 2015), tais como distância Euclidiana e distância de Manhattan, por exemplo, mostram-se cada vez menos eficientes à medida que o tamanho do espaço de busca e o número de obstáculos nos terrenos virtuais usados aumentam.

Neste contexto de pesquisa, (Takahashi et al., 2019) abordam uma alternativa ao uso de funções heurísticas tradicionais ao propor a utilização da rede neural U-Net. Essa DNN do tipo convolucional extrai informações de imagens de mapas coletadas para o aprendizado de heurísticas utilizadas no planejamento de rotas de robôs móveis. Tendo o mapa de estados e uma matriz tridimensional, chamada de tensor meta, a qual contém informações do nodo objetivo - coordenadas x, y e o ângulo de orientação, a rede produz um valor heurístico que é o resultado de diversos produtos vetoriais aplicados entre as características do mapa e do nodo objetivo, como as dimensões do mapa e a quantidade de orientações do nodo objetivo. Assim como neste TCC, os autores utilizaram a saída do algoritmo de Dijkstra (Dijkstra, 1959) para

produzir o dataset de treinamento da rede, o qual foi realizado em domínios discretos e contínuos, apresentando resultados satisfatórios na redução do número de nodos abertos na busca de caminhos. Diferente deste trabalho, este TCC não explora redes convolucionais, uma vez que os dados provenientes dos mapas são transformados em coordenadas, não sendo analisadas as imagens cruas.

(Ariki e Narihira., 2019) também empregam redes neurais convolucionais para o aprendizado de heurísticas usadas por planejadores de caminhos. A rede neural é alimentada com matriz contendo as posições dos obstáculos e a posição objetivo existentes no mapa de estados, tomado como um grid regular, assim como apresentado neste TCC. Produzindo como resultado uma imagem que representa o mapa heurístico, a saída da rede contém todos os valores heurísticos a partir do ponto objetivo. Esse mapa heurístico é então utilizado pelo planejador. Os experimentos compararam a utilização do mapa heurístico com as tradicionais funções heurísticas existentes (distância Euclidiana e distância de Manhattan), reduzindo significativamente o custo de busca, em termos de tempo, em detrimento da qualidade do caminho. O trabalho apresentado neste TCC também explora a ideia de construção e uso de um mapa heurístico. Nos testes desenvolvidos, os autores utilizaram mapas de 224x224, enquanto o presente trabalho explora o processo de busca hierárquico que permite lidar com DNN na busca de caminhos em mapas de maiores dimensões (404x404).

Dado o aumento massivo na urbanização ocorrido nas últimas décadas, (Khekare et al., 2020) propõem um algoritmo de busca que utiliza uma abordagem de aprendizagem por reforço para tomada de decisão. A proposta leva em consideração múltiplas características do cenário como a densidade de tráfego, a distância a ser percorrida e dados históricos das rotas. Na abordagem descrita, os autores tomam um sistema de tráfego inteligente como um grafo direcionado, o qual atribui custos a diferentes partes do caminho quando atingidos sob diferentes direções. Os nodos desse grafo alimentam a rede neural utilizada a fim de prover caminhos sem congestionamentos e com comprimentos próximos ao ótimo. Posteriormente, uma análise do algoritmo de rota ideal com aprendizagem por reforço (optimal route algorithm with reinforcement learning - ORAWRL) é feita, comparando-o com abordagens que utilizam algoritmos genéticos, algoritmos de redes neurais e algoritmos de otimização por enxame de partículas. Segundo os autores, o algoritmo é útil em cenários onde há congestionamentos, acidentes e bloqueios, tendo melhores resultados quando comparado com os outros algoritmos estudados. Em contraste com a pesquisa apresentada neste TCC, o trabalho utiliza a técnica de aprendizado por reforço ao analisar diferentes características dos

mapas virtuais e do processo de busca. Em linhas gerais, o emprego de técnicas de aprendizado por reforço profundo parece ser atrativo neste cenário de busca de caminhos onde aspectos de diferentes naturezas devem ser considerados nas escolhas dos caminhos resultantes. Embora esse TCC explore técnicas de aprendizado supervisionadas mais tradicionais, aprendizado por reforço também poderia ser explorado de acordo com a abordagem hierárquica de busca investigada neste trabalho. Trabalhos futuros, portanto, poderiam indicar as vantagens e desvantagens dessas tecnicas a partir de análise experimentais comparativas.

(Arfaee et al., 2011) utilizam técnicas de machine learning para aprimoramento de heurísticas no planejamento de rotas em grandes espaços de estados. A partir de dada heurística h0 e uma série de problemas que podem ser resolvidos por ela, um algoritmo de bootstrapping gera, a partir do treinamento da rede neural, uma sequência de heurísticas h1 que otimizam h0. Se a heurística h0 não for suficientemente forte para resolver nenhum dos problemas propostos, o algoritmo gera caminhos aleatórios cada vez menores, mas não tão simples que possam ser resolvidos por algoritmos como o de busca em largura, a fim de encontrar dados de treinamento que são garantidamente resolvidos por h0. O algoritmo foi testado para quebra-cabeças deslizantes e cubos mágicos de 25x25. Segundo os autores, a técnica produziu heurísticas que permitiram resolver os problemas rapidamente com soluções próximas do ótimo. Embora este trabalho e o presente TCC utilizem dados do mapa para treinarem redes neurais cuja saída é o valor heurístico a ser utilizado, este TCC considera apenas os problemas que podem ser resolvidos pelas heurísticas comparadas, não aplicando nenhuma modificação às tradicionais heurísticas.

(Reijnen et al.,2020) propõem a combinação de DNN com aprendizagem por reforço com heurísticas de busca para resolver um problema de transporte de bagagens feito por veículos autônomos em mapas virtuais baseados em segmentos. Nesses mapas, as rotas dos agentes são definidas por caminhos pré-computados e os mesmos podem ocupar segmentos acessíveis ao seu redor ao invés das posições tradicionalmente existentes em mapas baseados em grids regulares. Para atacar o problema, os autores adaptaram o algoritmo de busca para mapas baseados em segmentos, WHCA* (Silver 2005), a fim de gerar uma solução cooperativa entre os agentes presentes no terreno. Enquanto o WHCA* tradicional leva em consideração apenas o terreno em que o agente está inserido, o algoritmo proposto pelos autores utiliza uma rede de aprendizagem por reforço para gerenciar as rotas traçadas pelos veículos autônomos, evitando colisões e deadlocks. Em contraste com a pesquisa apresentada

neste TCC, o trabalho lida com mapas virtuais baseados em segmentos, enquanto este TCC explora grids regulares de grandes dimensões.

Utilizando redes neurais com aprendizado por reforço, (Agostinelli et al., 2019) combinam abordagens por reforço profundo, por reforço clássico e outros métodos, como programação dinâmica, para treinar uma rede neural profunda cujo resultado é usado para aproximar funções heurísticas para estimar o custo para atingir metas. A solução desenvolvida, DeepCubeA, é capaz de resolver diversos jogos como cubo mágico, quebra cabeça de peças deslizantes e Sokoban. Após o treinamento, a rede neural implementada é capaz de gerar heurísticas para problemas com espaços de busca extremamente elevados, tal como na solução do problema do cubo mágico, por exemplo. Os resultados mostraram que em quase 100% dos casos analisados, as soluções chegaram perto do ótimo. Embora este trabalho não trate da otimização de algoritmos de pathfinding por si, tal como analisado neste TCC, o trabalho ilustra o emprego de DNN e outras técnicas na solução de um problema de busca complexo. Em especial, o trabalho demonstra a aplicabilidade de diferentes técnicas de aprendizado por reforço que são usadas de forma combinada para a construção de heurísticas otimizadas para apoiar a resolução do problema. Embora o foco deste trabalho seja demonstrar como explorar diferentes técnicas de aprendizado de máquina de forma combinada para resolver problemas de busca complexos, o presente TCC apenas explora uma técnica de aprendizado de máquina na otimização de uma algoritmo hierárquico de busca de caminhos.

Combinando previsões do tempo de viagem com a hora do dia da viagem, ST-NN (Rede Neural Espacial-Temporal) proposto por (Jindal et al., 2017)) estima distâncias entre coordenadas GPS de origem e destino. Nesse modelo, coordenadas brutas do GPS são combinadas com as informações do horário atual, possibilitando prever uma distância de viagem sem construir uma rota entre dois pontos. Bem como neste TCC, as redes treinadas levam em consideração as posições atual e objetivo da busca em seu treinamento. Diferentemente deste TCC (Jindal et al., 2017), apenas estimam distâncias, sem gerar, contudo, o caminho resultante.

(Li et al.,2016) desenvolvem um algoritmo que integra o A* tradicional a uma rede neural de regressão, por meio da função heurística utilizada, na solução de problemas de busca de caminhos em mapas tomados como grids regulares. A rede neural utilizada é alimentada através de um algoritmo de geração de mapas aleatórios parametrizáveis, contendo informações acerca da largura, tamanho e outras característica do mapa, como o total de

nodos bloqueados. O processamento da rede gera pontos aleatórios dentro do mapa que são posteriormente unidos pelo algoritmo de Dijkstra, uma vez que tal algoritmo gera caminhos mínimos. Em comparação com outras funções heurísticas tradicionais, como a distância de Manhattan e a distância Euclidiana, a heurística proveniente da saída da rede neural proposta obteve sucesso na redução de nodos expandidos quando aplicada em mapas contendo um total de nodos entre 1000 e 25000. Em contraste com a pesquisa apresentada neste TCC, (Li et al., 2016) fazem um pré-processamento de dados para geração das informações que alimentarão a rede, enquanto o presente trabalho utiliza os valores computados por Dijkstra para os pontos amostrados, não fazendo qualquer processamento de outros dados tais como largura e altura dos grids.

No grupo de pesquisa onde este trabalho está inserido, (Doebber, 2019) detalha a implementação e teste de uma DNN do tipo “feedforward” no desenvolvimento de novos algoritmos de pathfinding. Embora não explorado em (Doebber, 2019), (Botea et al., 2004) propõem o emprego de algoritmos hierárquicos de pathfinding (HPA*). Visando o uso de mapas virtuais de grandes dimensões, o trabalho descreve um algoritmo que divide o mapa virtual em clusters (sub-regiões), os quais são previamente analisados e guardados na memória. Posteriormente, caminhos descobertos nessas sub-regiões são unidos, permitindo encontrar caminhos mais detalhados de forma mais rápida que o algoritmo A* tradicional. Expandindo os trabalho de (Doebber, 2019), (Souza, 2021), mostra uma integração de DNN com algoritmos hierárquicos de busca. O trabalho explora duas arquiteturas de redes neurais profundas para prover funções heurísticas em mapas (labirintos) tomados como grids regulares. Nos experimentos realizados, as redes neurais obtiveram melhores resultados em termos de nodos expandidos e tempo de execução da busca de caminhos em um único tipo de mapa de até (404x404). Este TCC visa dar continuidade a esse trabalho, ajustando e explorando os algoritmos desenvolvidos para apoiar a busca de caminhos em outros tipos de mapas.

Embora as arquiteturas e métodos de treinamento das DNN utilizadas nos trabalhos relacionados variem, eles propõem e analisam o emprego de DNN treinadas (por aprendizado supervisionado, aprendizado por reforço ou combinações dessas e outras técnicas de aprendizado de máquina) a partir de dados de várias características dos problemas alvos sendo abordados. Uma vez treinadas, essas DNN são usadas para guiar e possivelmente otimizar o processo de busca desenvolvido. Em muitos sentidos, o presente trabalho integra, expande e aprofunda as propostas apresentadas em (Doebber, 2019), (Souza, 2021) e (Boeta et al.,

2004), detalhando como empregar uma abordagem hierárquica de pathfinding pelo emprego de DNN em mapas de naturezas distintas.

3 METODOLOGIA

(Doebber, 2019) e (Souza, 2021) apresentaram e compararam resultados de pathfinding baseados no uso de DNN. Apesar disso, os resultados experimentais apresentados estavam limitados a análise de apenas um tipo de mapa envolvendo labirintos. Em (Doebber, 2019), DNN em conjunto com o A* tradicional foram usados na busca de caminhos em mapas de no máximo 101x101 nodos. Expandindo esse trabalho, (Souza, 2021) explorou DNN em conjunto com uma abordagem hierárquica de busca onde os mapas usados foram de no máximo 404x404 nodos. Usando os mapas de labirintos, eles foram mesclados lado a lado para criar mapas cada vez maiores. Isso permitiu analisar o impacto das computações nestes mapas de diferentes dimensões. Para desenvolver um maior conjunto de testes dos algoritmos de pathfinding e DNN propostos em (Doebber, 2019) e (Souza, 2021), este trabalho de TCC explora mapas virtuais com naturezas distintas de labirintos.

3.1 MAPAS UTILIZADOS

Figura 3.1 (A, B, C) ilustra os mapas utilizados neste TCC, todos com tamanhos máximos de 404x404 nodos. Esses mapas estão em conformidade com o mapa de maior dimensão utilizado por (Souza, 2021), com tamanhos totais de 163.216 nodos. Seguindo o que é descrito em (Sturtevant, 2012), os mapas usados neste TCC foram obtidos a partir de um repositório de mapas comum para teste de algoritmos de pathfinding (http://movingai.com/benchmarks/). Para poder comparar com os resultados obtidos por (Souza, 2021), o algoritmo implementado também foi testado para o maior mapa de labirinto utilizado naquele trabalho Figura 3.1 (D). Estes mapas possuem tamanhos maiores do que mapas utilizados na maioria dos jogos virtuais e sistemas simulados, tendo também naturezas parecidas com as dos mapas encontrados em aplicações de simulação.

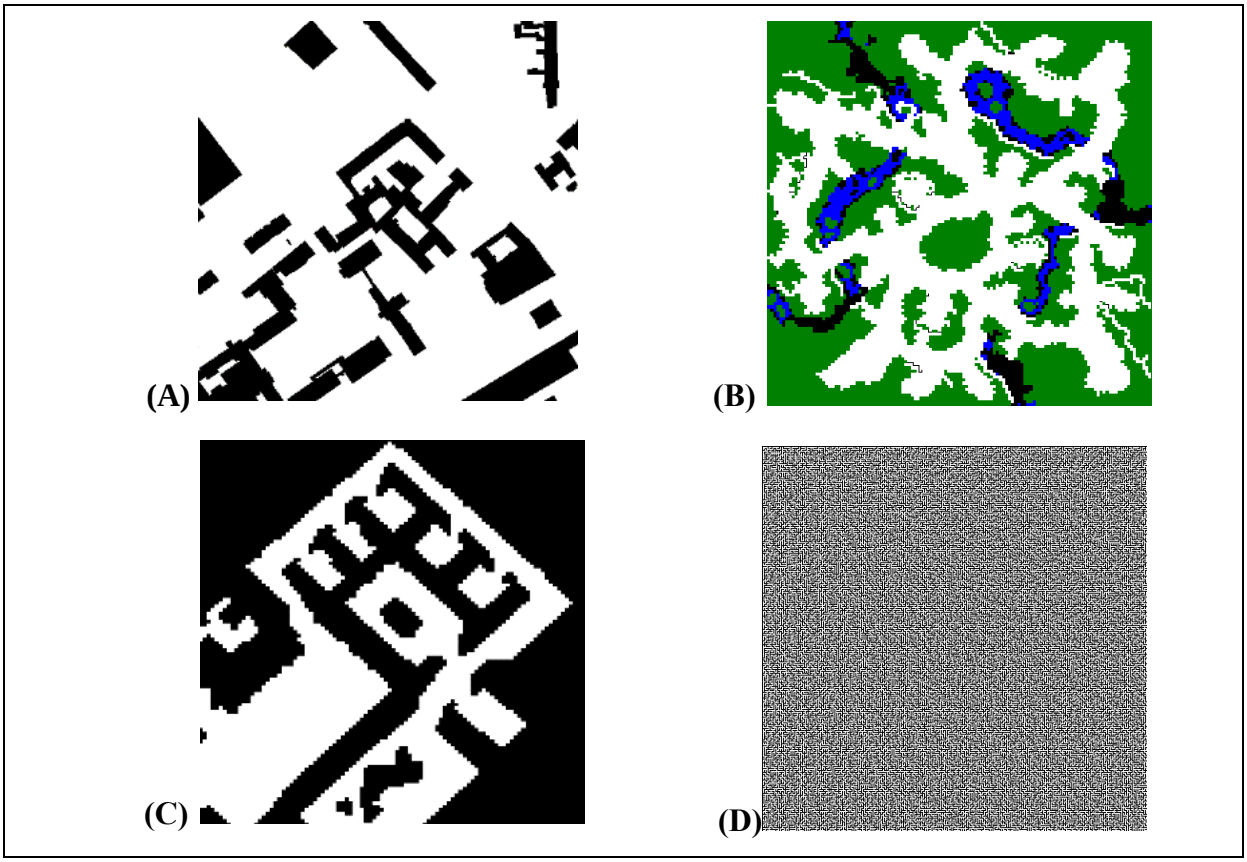
Em particular, os seguintes mapas foram explorados:

i) mapa virtual gerado a partir da discretização de uma imagem aérea da cidade de Berlim (Figura 10, A), que tem aproximadamente 120.000 nodos livres para computação de caminhos;

ii) mapa virtual de uma floresta usada em um jogo virtual (Figura 10, B), com aproximadamente 65.000 nodos livres;

iii) mapa virtual descrevendo o interior de um prédio (Figura 10, C), também usado em um jogo digital, com aproximadamente 70.000 nodos livres.

Figura 3.1 – Mapas virtuais explorados neste TCC



3.2 TREINAMENTO DAS REDES NEURAIS

Um dos principais desafios encontrados neste TCC foi a escolha de suas arquiteturas e o treinamento das redes. Para atacar estes problemas, uma amostragem dos pontos dos mapas foi feita, selecionando 50% do total de nodos disponíveis para movimentação em cada mapa.

Após combinar os 50% nodos selecionados dois a dois (para refletir pontos de início e fim usados nas computações de busca de caminhos), cluster a cluster (neste caso, pontos internos a cada clusters, visto que diferentes redes são treinadas em cada cluster), tem-se 25% das combinações dos pontos originais em cada cluster, totalizando aproximadamente 10% das combinações totais de nodos do mapa original. Isso indica que, para mapas com complexidades similares aos mapas utilizados neste TCC, a mesma amostragem de combinações pode ser feita. Tabela 3.1 mostra o total de nodos disponíveis e a quantidade de combinações utilizadas para o treinamento em cada um dos três mapas.

Tabela 3.1 – Total de nodos amostrados em cada mapa

Mapa	Total de nodos livres	Total de nodos amostrados
Berlin	126.144	63.072
Floresta	72.122	36.061
Prédio	62.207	31.103

Fonte: Autor.

Para gerar os datasets de treinamento das DNN, as quais são usadas como funções heurísticas para o algoritmo hierárquico de pathfinding, o algoritmo de Dijkstra foi utilizado. Para isso, a posição objetivo para este algoritmo não foi determinada, permitindo gerar um grafo contendo caminhos com custos mínimos entre todos os nodos do mapa virtual (Minimum Spanning Tree). Os caminhos computados então foram usados no treinamento das DNN, com 70% do total dos dados para treinamento e os 30% restantes para validação dos resultados. Tabela 3.2 mostra o total de nodos livres em cada um dos 48 sub-mapas gerados pela divisão dos mapas originais. A mesma arquitetura de rede foi utilizada no treinamento realizado com os caminhos computados em cada clusters. Contudo, devido as diferenças no número de nodos disponíveis para navegação em cada cluster, arquiteturas de rede diferentes poderiam ter sido exploradas, assim ajustando as redes usadas as necessidades de treinamento de cada cluster.

Tabela 3.2 – Total de nodos livres para navegação em cada sub-mapa.

	Total de nodos livres para navegação		
Cluster	Berlin	Floresta	Prédio
0x0	10.177	2.159	68
0x1	7.463	5.132	4.194
0x2	7.713	4.494	1.966
0x3	5.865	1.617	561
1x0	8.373	3.238	3.228
1x1	8.602	5.737	4.675
1x2	7.002	6.787	5.905
1x3	7.698	6.093	5.593
2x0	9.197	4.252	4.146
2x1	7.084	6.023	4.420
2x2	7.606	4.935	7.235
2x3	9.027	4.477	5.071
3x0	8.250	3.975	126
3x1	9.068	3.511	2.417
3x2	6.544	5.301	3.740
3x3	6.475	4.391	8.862

Fonte: Autor.

O treinamento das redes foi feito em Python, utilizando a plataforma TensorFlow (Abadi *et. al*, 2019). Os dados de saída foram normalizados entre 0 e 1, utilizando como fator divisor o comprimento máximo de caminho gerado pelo algoritmo de Dijkstra em cada cluster. Por exemplo: se o comprimento máximo encontrado pelo algoritmo de Dijkstra em determinado cluster foi 140, todos os valores de caminhos encontrados naquele cluster foram divididos por 140. Os resultados destas divisões são as saídas esperadas pelas redes neurais durante o processo de treinamento supervisionado. Tal normalização não somente contribuiu

para um treinamento mais rápido das redes (Goodfellow et al., 2016), como também auxilia no processo de busca do algoritmo implementado neste trabalho. Tabela 3.3 mostra os comprimentos máximos em todos os 48 clusters dos mapas utilizados nos experimentos.

Tabela 3.3 – Maiores distâncias encontradas por sub-mapa

	Tamanhos máximos dos caminhos		
Cluster	Berlin	Floresta	Prédio
0x0	140	128.8	14.8
0x1	160.6	143.4	135.6
0x2	153.6	124.6	65.6
0x3	114.2	63	42.2
1x0	160.4	107	138.4
1x1	460	125.8	99.2
1x2	193.6	139.4	161
1x3	136.4	143.8	128.6
2x0	173.8	182	137.6
2x1	191.8	160.4	79.4
2x2	158.4	145	121.8
2x3	144.2	193.8	133.6
3x0	237.8	103.8	18.8
3x1	144.4	136.2	113.4
3x2	190.4	164.6	147.2
3x3	136.6	196.8	134

Fonte: Autor.

Após experimentação de diversas arquiteturas para treinamento dos dados oriundos de cada um dos clusters dos diversos mapas, a arquitetura que atingiu pelo menos 3,5% de acurácia após treinamento foi a arquitetura proposta em (Doebber, 2019), baseada no trabalho de (Jindal *et al.*, 2017). Tabela 3.2 mostra essa arquitetura de DNN em termos de neurônios e

funções de ativação.

Tabela 3.4 – Arquitetura da rede neural profunda utilizada neste TCC

Quantidade de neurônios	Função de ativação
4	Linear
20	ReLU
100	ReLU
200	ReLU
300	ReLU
400	ReLU
500	ReLU
400	ReLU
300	ReLU
200	ReLU
100	ReLU
20	ReLU
1	Sigmóide

Fonte: Autor.

Após o treinamento, as redes neurais foram salvas em Protocol Buffers - mecanismos extensíveis e neutros em termos de linguagem e plataforma do Google para serializar dados estruturados. Tais estruturas de dados foram posteriormente carregadas pelo algoritmo hierárquico de busca para fazer as predições necessárias.

3.3 ALGORITMO IMPLEMENTADO

Para tratar a busca de caminhos em grandes espaços de busca (por exemplo, ~120k nodos no mapa da Figura 3.1, B), este TCC explora uma abordagem hierárquica de busca de caminhos. A implementação desta abordagem foi escrita em C# com o framework .NET Core, diferente dos trabalhos de (Souza, 2021), o qual utilizou C++ e (Doebber, 2019), o qual integrou C# e Unity3D (Unity Technologies, 2019) com Python.

De acordo com a implementação desenvolvida neste TCC, o mapa virtual original foi dividido em 16 clusters de tamanhos iguais (101x101). Esses clusters foram previamente analisados e armazenados em memória. Para construção do grafo abstrato uma abordagem diferente da utilizada por (Doebber, 2019) e (Souza, 2021) foi utilizada: as bordas destes clusters foram unidas fazendo uma amostragem dos pontos disponíveis para ligação entre os clusters. O objetivo deste tipo de amostragem de bordas foi construir uma representação que distribuisse igualmente as conexões entre as fronteiras dos clusters. Cada um dos nodos selecionados dentro de seus respectivos clusters passou por um processo de cálculo de distância, o qual foi realizado com o uso do algoritmo BFS. Tais distâncias foram representadas em um grafo abstrato, o qual contém o espaço da busca abstrata. As arestas deste grafo contêm as distâncias calculadas anteriormente entre os nodos, enquanto os nodos representam os estados do nível abstrato do mapa, tal como proposto por (Botea et al., 2004). O nível abstrato é, portanto, um subconjunto do espaço de busca concreto.

Com o mapa abstrato pronto, o algoritmo de busca insere os nodos de início e objetivo no grafo a cada busca, calculando as distâncias entre esses nodos e os nodos de borda de seus clusters. Tal mecanismo de inserção teve um custo relativamente baixo (~2ms) para as dimensões dos mapas utilizados e também pela amostragem dos nodos das bordas. Contudo, esse mecanismo tende a perder eficiência (Botea, 2004) conforme o tamanho dos mapas e a quantidade de nodos nas bordas dos clusters aumentam, podendo demandar outras soluções.

Uma vez que os pontos de início e fim são inseridos na representação abstrata do mapa, o algoritmo de Dijkstra pode calcular a menor distância para gerar o caminho abstrato. Posteriormente, caminhos concretos computados dentro de cada cluster pertencente ao caminho abstrato (somente os clusters que o caminho abstrato computado utilizou) são computados com o A*, onde buscas em cada um dos clusters envolvidos no caminho abstrato encontrado são computadas de forma paralela nas implementações realizadas neste TCC. Ao final, os caminhos retornados pelas buscas de caminhos em cada uma dos clusters selecionados são unidos como parte de um processo de refinamento dos caminhos encontrados no primeiro passo de busca. Nesta última etapa de refinamento, as funções heurísticas tradicionais são computadas normalmente pelo algoritmo de busca. Quando o algoritmo utiliza as saídas das redes neurais como função heurística, os resultados são obtidos sob demanda pelas redes neurais, trazidas para memória com suas arquiteturas, pesos e outras informações alocados em grafos, previamente salvos em Protocol Buffers. Tais representações, alocadas com a API

provida pela Google (Abadi et al., 2016), são ferramentas poderosas de computação, as quais utilizam a capacidade de lidar com grandes volumes de cálculos matemáticos da unidade de processamento gráfico (Graphics Processing Unity – GPU) para calcular as saídas produzidas pelas redes neurais. Apesar de retornar caminhos sub-ótimos (1% maiores que os menores caminhos, por exemplo), esse processo hierárquico de pathfinding permite otimizar a computação de rotas em mapas de maiores dimensões (em comparação a mapas normalmente usados no desenvolvimento de jogos digitais, por exemplo), tais como os mapas usados nos experimentos deste TCC.

4 EXPERIMENTOS E RESULTADOS

O objetivo dos experimentos desenvolvidos neste TCC é comparar a computação de caminhos com uso de funções heurísticas tradicionais, nominalmente: distância de Manhattan e distância Euclidiana, com a utilização das saídas das redes neurais profundas, desenvolvendo experimentos adicionais aos apresentados em (Doebber, 2019) e (Souza, 2021). Os resultados destes experimentos são analisados de acordo com as seguintes métricas: número de nodos expandidos, tempo de execução e distância entre os pontos de início e destino dos caminhos computados. Em muitos sentidos, este TCC expande as pesquisas de (Doebber, 2019) e (Souza, 2021), permitindo analisar o uso DNN na computação hierárquica de caminhos em outros tipos de mapas além de labirintos.

O treinamento das redes neurais profundas foi feito em Python, com a API Keras, a qual utiliza TensorFlow em seu back-end. A implementação do algoritmo de busca hierárquica foi feita em C#, no ambiente de desenvolvimento integrado Visual Studio Community.

Tabelas 4.1-3 mostram dados pertinentes ao treinamento das redes neurais para cada um dos mapas. O conjunto de dados é uma amostragem (25%) das combinações de início e destino dos pontos disponíveis para navegação. O tamanho do conjunto de validação é um percentual relativo a 30% do conjunto de dados. Os tempos de treinamento das redes neurais variaram conforme a complexidade de cada cluster, sendo possível notar que clusters com menores complexidades em termos de distribuição irregular de nodos bloqueados (a exemplo do cluster 0x0 do mapa de Berlin) treinaram mais rápido do que clusters com distribuições irregulares, mesmo que estes tenham tamanhos de conjuntos de dados menores em comparação com aqueles. A acurácia mínima foi obtida via resultados experimentais com a própria API Keras, sendo definida como 2% para o mapa de Berlin e 3,5% para os demais mapas, uma vez que os treinamentos das DNN relativas aos mapas da floresta e do prédio não atingiram o limiar de 2% em tempos aceitáveis, passando várias horas sem convergir. Tal fator é um indicativo de que a amostragem de 10% das combinações dos pontos disponíveis para movimentação nestes mapas pode ser um limiar inferior para treinamento da arquitetura de rede neural proposta.

Tabela 4.1 – Informações acerca do treinamento das redes neurais para o mapa de Berlin

Cluster	Tamanho do conjunto de dados	Tamanho do conjunto de validação	Tempo de treinamento da rede neural	Acurácia mínima da rede neural treinada
0x0	25.887.744	7.766.323	12min.	98%
0x1	13.920.361	4.176.108	14min.	98%
0x2	14.868.736	4.460.620	1h 37min.	98%
0x3	8.596.624	2.578.987	23min.	98%
1x0	17.522.596	5.256.778	7min.	98%
1x1	18.498.601	5.549.580	18min.	98%
1x2	12.257.001	3.677.100	2h 24min.	98%
1x3	5.331.481	1.599.444	1h 52min.	98%
2x0	21.141.604	6.342.481	44min.	98%
2x1	12.545.764	3.763.729	13min.	98%
2x2	14.462.809	4.338.842	58min.	98%
2x3	20.367.169	6.110.150	17min.	98%
3x0	17.015.625	5.104.687	1h 23min.	98%
3x1	20.557.156	6.167.146	15min.	98%
3x2	10.705.984	3.211.795	13min.	98%
3x3	10.478.169	3.143.450	8min.	98%

Fonte: Autor.

Tabela 4.2 – Informações acerca do treinamento das redes neurais para o mapa da floresta

Cluster	Tamanho do conjunto de dados	Tamanho do conjunto de validação	Tempo de treinamento da rede neural	Acurácia mínima da rede neural treinada
0x0	1.165.320	349.596	6min.	97,5%
0x1	6.584.356	1.975.307	53min.	97,5%
0x2	5.049.009	1.514.703	24min.	97,5%
0x3	653.672	196.101	23min.	97,5%
1x0	2.621.161	786.348	3h 15min.	97,5%
1x1	8.228.292	2.468.488	18min.	97,5%
1x2	11.515.842	3.454.753	12min.	97,5%
1x3	9.281.162	2.784.349	1h 31min.	97,5%
2x0	4.519.876	1.355.963	2h 02min.	97,5%
2x1	9.069.132	2.720.740	1h 25min.	97,5%
2x2	6.088.556	1.826.567	51min.	97,5%
2x3	5.010.882	1.503.265	1h 12min.	97,5%
3x0	3.950.156	1.185.047	2h 14min.	97,5%
3x1	3.081.780	924.534	1h 04min.	97,5%
3x2	7.025.150	2.107.545	1h 45min.	97,5%
3x3	4.820.220	1.446.066	16min.	97,5%

Fonte: Autor.

Tabela 4.3 – Informações acerca do treinamento das redes neurais para o mapa do interior de um prédio

Cluster	Tamanho do conjunto de dados	Tamanho do conjunto de validação	Tempo de treinamento da rede neural	Acurácia da rede neural treinada
0x0	1.156	346	3min.	97,5%
0x1	4.397.409	1.319.223	1h 13min.	97,5%
0x2	966.289	289.886	47min.	97,5%
0x3	78.680	23.604	8min.	97,5%
1x0	2.604.996	781.498	21min.	97,5%
1x1	5.463.906	1.639.172	39min.	97,5%
1x2	8.717.256	2.615.177	41min.	97,5%
1x3	7.820.412	2.346.124	17min.	97,5%
2x0	4.297.329	1.289.199	15min.	97,5%
2x1	4.884.100	1.465.230	28min.	97,5%
2x2	13.086.306	3.925.892	51min.	97,5%
2x3	6.428.760	1.928.628	36min.	97,5%
3x0	3.969	1.190	3min.	97,5%
3x1	1.460.472	438.141	11min.	97,5%
3x2	3.496.900	1.049.070	17min.	97,5%
3x3	19.633.761	5.890.128	28min.	97,5%

Fonte: Autor.

Para os experimentos, combinações de pares de pontos (ínicio e destino) foram selecionadas aleatoriamente nos mapas, sempre em clusters diferentes (para permitir analisar o impacto das computações de caminhos que envolvem no mínimo dois clusters diferentes), tendo suas execuções computadas pelo algoritmo hierárquico. Foram medidos o tempo de execução, a quantidade de nodos expandidos e a distância entre os pontos, métricas sugeridas em (Algfoor et al., 2015). No algoritmo de busca testado, três diferentes opções de computações heurísticas foram usadas: distância de Manhattan, distância Euclidiana e DNN.

Os resultados experimentais foram avaliados via modelos estatísticos de regressão linear (Mccullagh e Nelder, 1989), utilizando o pacote de software estatístico R. Em modelos de regressão linear generalizada, as variáveis representam os conceitos ou operações que se quer analisar. Normalmente, usam-se variáveis quantitativas, cujos valores são expressos numericamente. No entanto, existe a possibilidade de incluir informações qualitativas em tais modelos. As variáveis qualitativas expressam qualidades ou atributos dos agentes ou indivíduos analisados. Como as variáveis qualitativas normalmente coletam aspectos da presença ou não de um determinado atributo, são utilizadas variáveis construídas artificialmente, também chamadas de variáveis dummy. Essas variáveis geralmente assumem dois valores, 1 ou 0, indicando se uma determinada qualidade ou atributo está presente. Normalmente, a variável dummy recebe o valor 1 na presença da qualidade e 0 caso contrário. Variáveis que assumem os valores 1 e 0 também são chamadas de variáveis dicotômicas ou binárias. Variáveis dicotômicas podem ser combinadas para caracterizar variáveis definidas por pertencerem ou não a determinados grupos. Ao incluir uma variável qualitativa que define a ordem de determinado algoritmo, como é feito neste trabalho, introduzem-se, então, variáveis qualitativas ao modelo, associadadas ao fato de pertencerem ou não a cada grupo de algoritmos - os que utilizam DNN como heurísticas, os que utilizam a distância Euclidiana e os que utilizam a distância de Manhattan. Especificamente, duas variáveis dummy (D1 e D2) foram adicionadas aos modelos para comparar as diferentes abordagens (Tabela 4.4).

Tabela 4.4 – Variáveis adicionadas ao modelo de regressão linear

Abordagem	D1	D2
DNN	0	0
Manhattan	0	1
Euclidiana	1	0

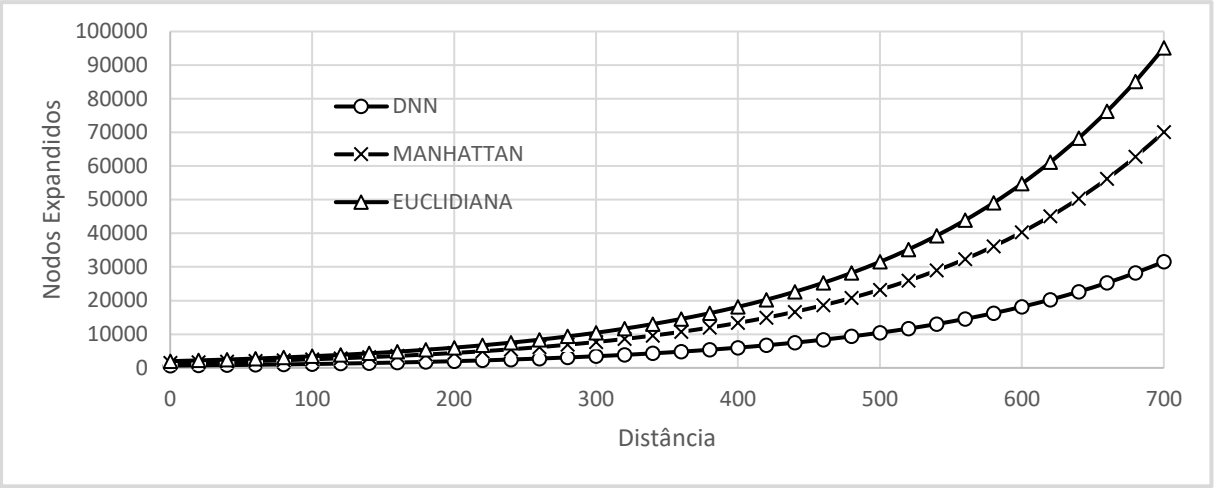
Fonte: Autor.

Por melhor lidar com variáveis de saída contínuas -tempo, distância e nodos expandidos, a distribuição gama foi escolhida como ferramenta de análise, tendo sua fórmula descrita como $g(mu) = \beta0 + \beta1 * X + \beta2 * D1 + \beta2 * D2$ onde g é a função logaritmo. Este modelo foi utilizado, tanto para normalização dos resultados de nodos expandidos, quanto para os resultados de tempo, ambos em relação à distância. Isso quer dizer que quando D1 e D2 forem

iguais a 0, o resultado dessa função será a quantidade de nodos expandidos ou o tempo de execução do método base, aqui colocado como a DNN. Quando D1 for igual a 0 e D2 for igual a 1, a saída deste método resulta em valores relativos a execução do algoritmo hierárquico com a distância de Manhattan como heurística. Finalmente, quando D1 for 1 e D2 for 0, tem-se os valores de saída em termos de nodos expandidos e tempo de execução quando a função Euclidiana é a heurística usada. Tendo o nível de significância (α) definido em 1%, e as hipóteses H0 e H1 definidas como $\beta i = 0$ ou $\beta i \neq 0$ -para i = { 1, 2, 3}, o nível de significância foi comparado com o valor de Pr, sendo aceito quando $\alpha > Pr$ ou rejeitado caso contrário. Valores βi positivos indicam que o método é menos eficiente em relação ao método base (DNN), e valores βi negativos indicam o contrário.

Os experimentos (Figuras 4.1-6) mostraram que os valores de Pr são inferiores ao limiar α, indicando resultados estatisticamente significativos. Tendo sempre βi positivos, os resultados ainda demonstraram que as redes neurais profundas obtiveram melhores desempenhos em termos de nodos expandidos e tempos em relação a utilização das heurísticas tradicionais para os diversos tipos de mapas verificados. Tais resultados são consistentes com os resultados apresentados em (Doebber, 2019) e (Souza, 2021).

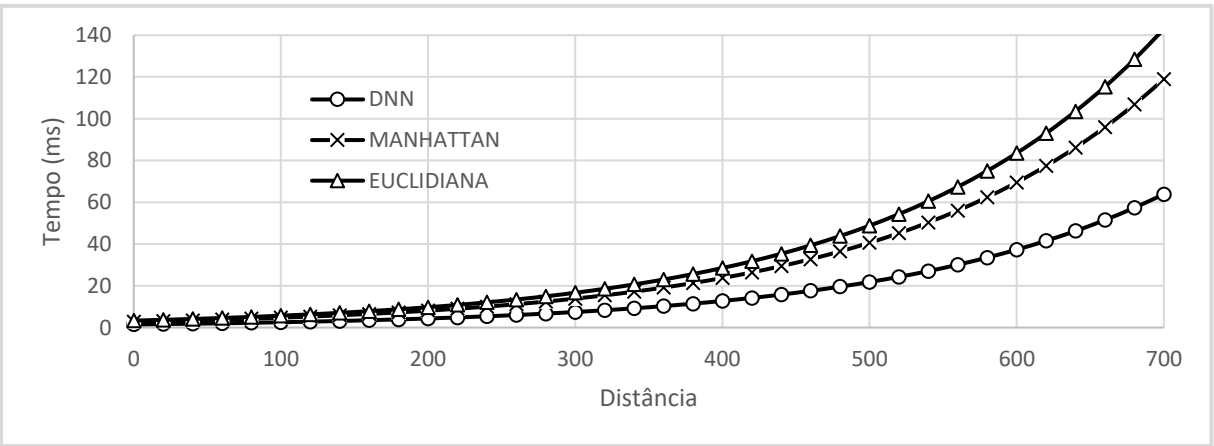
Figura 4.1 – Resultados obtidos para o mapa de Berlin (nodos expandidos X distância)



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	6.493e+00	2.259e-03	2874.4	< 2e-16
Rede Neural	5.523e-03	5.290e-06	1043.9	< 2e-16
Euclidiana	1.104e-00	1.653e-03	668.3	< 2e-16
Manhattan	7.984e-01	1.653e-03	483.1	< 2e-16

Fonte: Autor.

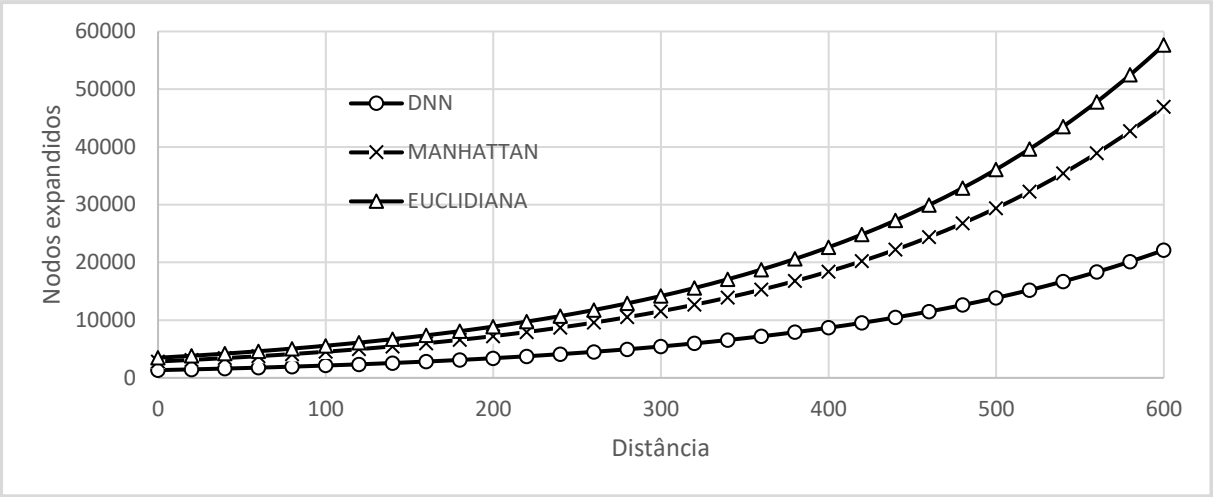
Figura 4.2 – Resultados obtidos para o mapa de Berlin (tempo X distância)



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	3.942e-01	2.300e-03	171.4	< 2e-16
Rede Neural	5.375e-03	5.299e-06	1014.3	< 2e-16
Euclidiana	8.060e+00	2.252e-03	579.7	< 2e-16
Manhattan	6.220e+00	2.254e-03	453.7	< 2e-16

Fonte: Autor.

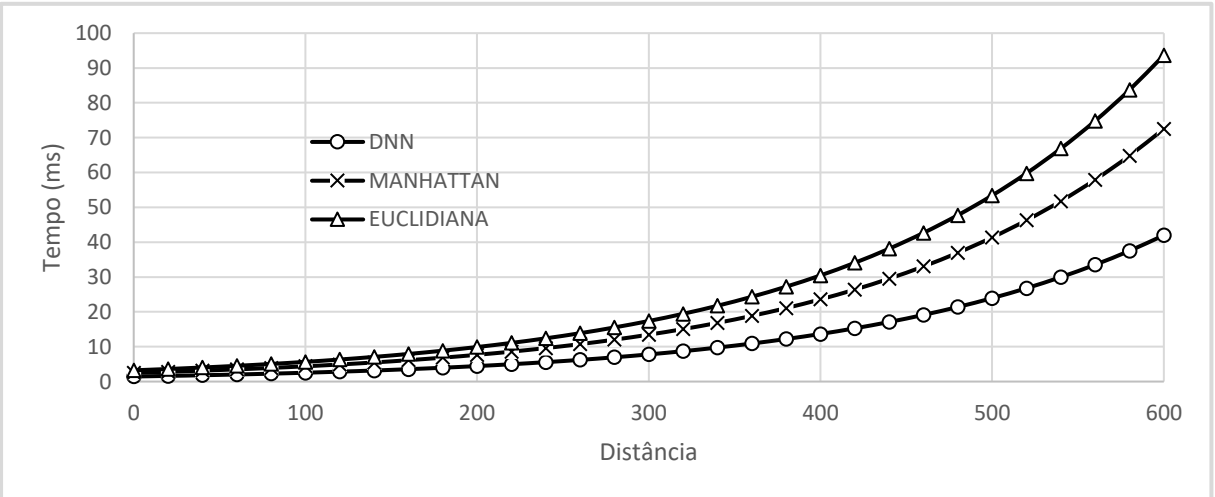
Figura 4.3 – Resultados obtidos para o mapa da floresta (nodos expandidos X distância)



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	7.198e-00	2.604e-03	3713.6	< 2e-16
Rede Neural	4.677e-03	5.413e-06	939.4	< 2e-16
Euclidiana	9.580e-01	2.345e-03	572.4	< 2e-16
Manhattan	7.524e-01	2.345e-03	449.7	< 2e-16

Fonte: Autor.

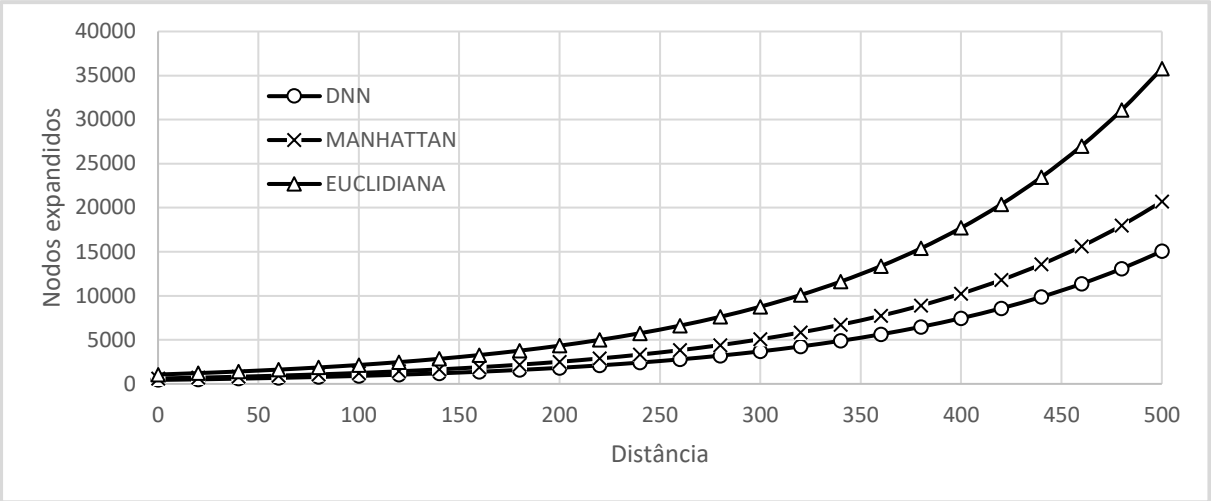
Figura 4.4 – Resultados obtidos para o mapa da floresta (tempo X distância)



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	3.683e-01	1.938e-03	141.4	< 2e-16
Rede Neural	5.615e-03	4.979e-06	926.5	< 2e-16
Euclidiana	8.024e-01	1.674e-03	342.2	< 2e-16
Manhattan	5.467e-01	1.673e-03	233.2	< 2e-16

Fonte: Autor.

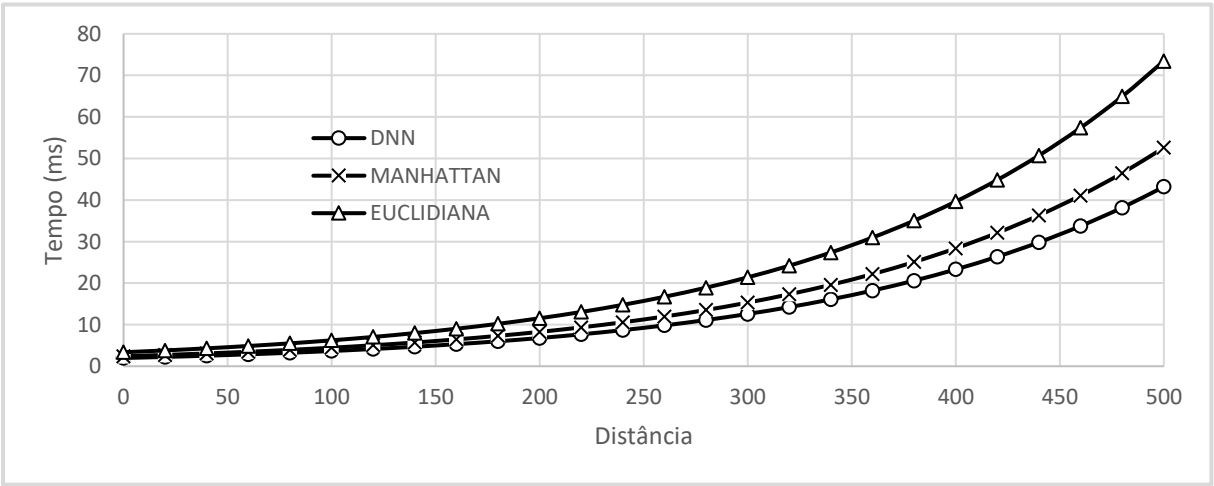
Figura 4.5 – Resultados obtidos para o mapa do interior de um prédio (nodos expandidos X distância)



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	6.102e+00	1.852e-03	1748.4	< 2e-16
Rede Neural	7.035e-03	5.474e-05	425.2	< 2e-16
Euclidiana	8.657e-01	1.742e-03	214.3	< 2e-16
Manhattan	3.181e-01	1.742e-03	203.6	< 2e-16

Fonte: Autor.

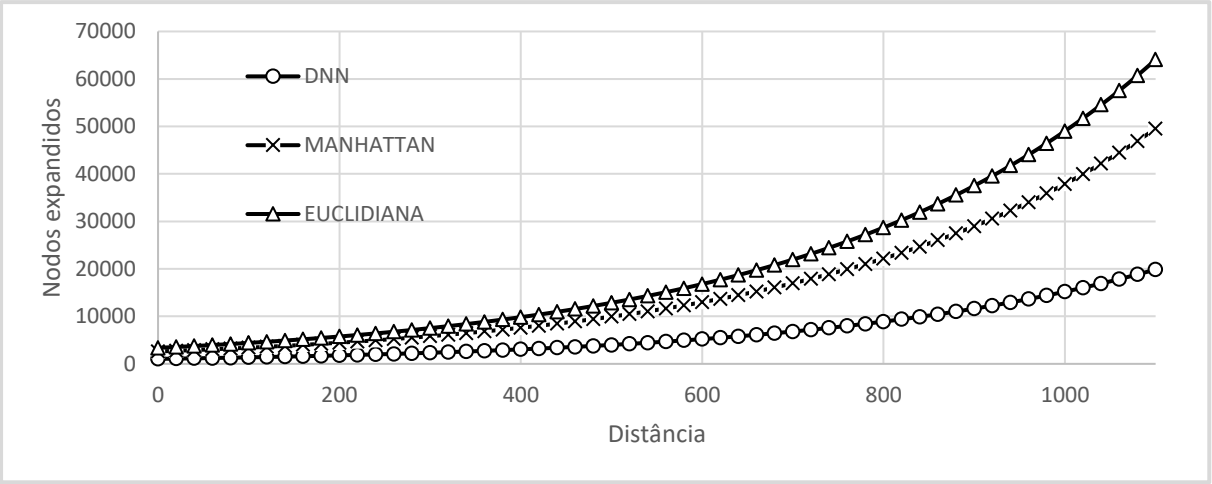
Figura 4.6 – Resultados obtidos para o mapa do interior de um prédio (tempo X distância)



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	6.820e-01	1.997e-03	131.7	< 2e-16
Rede Neural	6.167e-03	4.132e-06	514.3	< 2e-16
Euclidiana	5.311e-01	1.335e-03	248.2	< 2e-16
Manhattan	1.970e-01	1.334e-03	191.4	< 2e-16

Fonte: Autor.

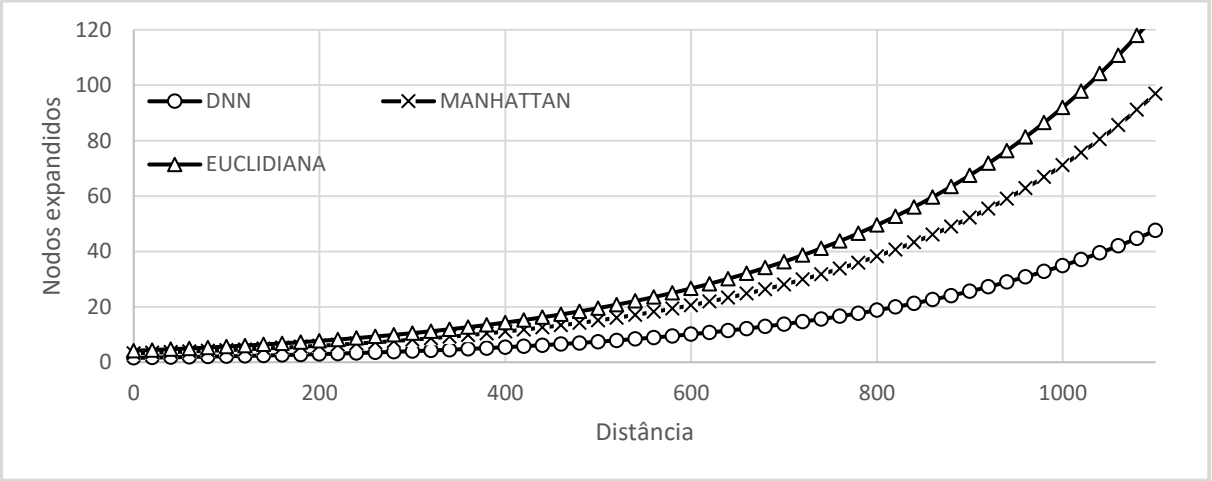
Figura 4.7– Resultados obtidos para o mapa do labirinto (nodos expandidos X distância)



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	6.950e+00	2.102e-03	1940.1	< 2e-16
Rede Neural	2.677e-03	3.785e-06	812.5	< 2e-16
Euclidiana	1.171e-00	1.224e-03	557.2	< 2e-16
Manhattan	9.134e -01	1.224e-03	497.5	< 2e-16

Fonte: Autor.

Figura 4.8– Resultados obtidos para o mapa do labirinto (tempo X distância)



	Estimativa	Erro padrão	Valor t	Pr (>|t|)
Intercepto	4.521e-01	1.978e-03	172.4	< 2e-16
Rede Neural	3.120e-03	2.661e-06	912.1	< 2e-16
Euclidiana	9.785e-00	1.521e-03	422.7	< 2e-16
Manhattan	7.213e -01	1.522e-03	375.5	< 2e-16

Fonte: Autor.

Ao comparar o crescimento das curvas que modelam o comportamento das heurísticas nos três mapas testados, é possível visualizar que as redes neurais profundas melhor otimizaram o algoritmo hierárquico nos mapas de Berlin e da floresta. No mapa do prédio, embora se observe melhor desempenho das DNN nas duas métricas analisadas, a estimativa da função de Manhattan teve rendimentos próximos. Pode-se inferir que tal comportamento deva-se ao fato de que o mapa do interior do prédio contém uma distribuição de obstáculos e áreas livres para busca de caminhos mais irregular, e portanto, mais complexa, comparado aos outros dois mapas. Além disso, o mapa do prédio tem uma menor quantidade de nodos disponíveis para navegação, evidenciando melhor eficiência no uso de redes neurais treinadas a partir de uma amostragem de 10% de caminhos computados em mapas com maiores quantidades de nodos livres. Por fim, quando o algoritmo implementado neste trabalho foi utilizado para comparar os diversos comportamentos das funções heurísticas para o maior mapa utilizado em (Souza, 2021), os resultados também foram positivos para as duas métricas analisadas, obtendo, porém, maiores tempos de execução para maiores caminhos, o que pode ser efeito da busca hierárquica. Para mapas mais complexos e irregulares além dos utilizados neste trabalho, um maior conjunto de dados de caminhos pode ser necessário no treinamento das redes neurais.

5 CONCLUSÕES

Por não considerarem as características de diferentes mapas virtuais, algoritmos de busca baseados em funções heurísticas tradicionais tendem a perder eficiência conforme o tamanho dos mapas crescem. Para atacar esse problema, o presente trabalho abordou a implementação e experimentação de DNN como funções heurísticas de algoritmos hierárquicos de pathfinding, utilizando sua capacidade de reconhecimento de padrões para aprimorar as estimativas heurísticas computadas durante o processo de busca. Tal integração entre heurísticas provenientes de DNN devidamente treinadas e algoritmos hierárquicos de busca é justificável visto que explora uma área de pesquisa recente e ainda em aberto na literatura.

Em muitos sentidos, a principal contribuição deste trabalho é apresentar uma expansão experimental e prática dos trabalhos apresentados em (Doebber, 2019) e (Souza, 2021), os quais foram desenvolvidos no grupo de pesquisa onde este TCC está inserido. Portanto, o trabalho apresenta evidência adicional que permitem analisar se tais funções heurísticas geradas a partir dos resultados de DNN devidamente treinadas podem ou não permitir obter melhores estimativas heurísticas para a computação de rotas de menor custo em diferentes tipos de mapas contendo grandes espaços de busca.

Este TCC permitiu verificar que as redes neurais profundas obtiveram melhores resultados em termos de nodos expandidos e tempos de execução em comparação com as funções heurísticas tradicionais – distância de Manhattan e distância Euclidiana. Para isso, foi utilizada uma arquitetura de DNN única que atendeu a demanda dos diversos tipos de mapas utilizados nos experimentos. Mesmo usando um conjunto de dados de treinamento bastante inferior (10%) ao conjunto de todos os caminhos que poderiam ser computados nos mapas usados (total de combinações possíveis para os nodos livres para movimentação em cada mapa), o emprego de DNN no papel das funções heurísticas permitiu obter melhores tempos de execução a partir da diminuição da quantidade de nodos expandidos, tendo sua eficiência melhor comprovada conforme o total de nodos disponíveis para navegação aumenta.

Trabalhos futuros podem investigar a exploração da abordagem de pathfinding utilizada neste TCC em outros tipos de mapas, o que poderia permitir apresentar uma validação mais compreensiva das técnicas propostas. Também é sugerida a utilização de DNN como função heurística em experimentos com mapas virtuais representados por outras estruturas de dados, como quadtrees ou waypoints (Algfoor et al., 2015), onde os mapas são

representados por estruturas de representação hierárquicas e irregulares que podem, em certas situações, favorecer ainda a representação de mapas virtuais de grandes dimensões e a otimização de computações realizadas por algoritmos de pathfinding. Por fim, análises adicionais das propostas apresentadas neste trabalho podem ser realizadas em comparação a outros algoritmos de pathfinding que exploram o emprego de dados pré-computados (algoritmos mais avançados apresentados na literatura (Algfoor et al., 2015)), não somente utilizando técnicas padrão de busca de caminhos que somente permitem demonstrar o quanto os resultados obtidos neste trabalho permitem melhoram resultados tomados como baseline.

REFERÊNCIAS BIBLIOGRÁFICAS

ABADI, M.; BARHAM, P.; CHEN, J.; CHEN, Z.; DAVIS, A., DEAN, J.; ZHENG, X. **Tensorflow: A system for large-scale machine learning**. 12th USENIX symposium on operating systems design and implementation (pp. 265-283), 2016.

AGOSTINELLI, F.; MCALEER, S.; SHMAKOV, A. **Solving the Rubik's cube with deep reinforcement learning and search.** Nature Machine Intelligence, 2019, v. 1, n. 8, p. 356-363.

ALGFOOR, A.; SUNAR, M. S.; KOLIVAND, H. **A comprehensive study on pathfinding techniques for robotics and video games**. Int. Journal of Computer Games Technology, 2015.

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

ARFAEE, Shahab Jabbari; ZILLES, Sandra; HOLTE, Robert C**. Learning heuristic functions for large state spaces.** Artificial Intelligence, v. 175, n. 16-17, p. 2075-2098, 2011.

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

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

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.

DOEBBER, D. M; **Uso de redes neurais profundas para o aprendizado de funções heurísticas para algoritmos de busca de caminhos.** Trabalho de Conclusão do Curso em Ciência da Computação - Universidade Federal de Santa Maria, Rio Grande do Sul, 2019.

GHORPADE, J.; PARANDE, J.; KULKARNE, M.; BAWASKAR, A. **GPGPU processing in CUDA architecture**. arXiv preprint arXiv:1202.4347. 2012.

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

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

JINDAL, I.; QIN, T.; XUEUE, Q. **A unified neural network approach for estimating travel time and distance for a taxi trip**. arXiv preprint arXiv:1710.04350, 2017.

KHEKARE, G. **The Optimal Path Finding Algorithm Based on Reinforcement Learning**. International Journal of Software Science and Computational Intelligence (IJSSCI), v. 12, n. 4, p. 1-18, 2020.

LI, G.; WANG, H.; WANG, N; YEUNG, Y. (2016) **ANN: a heuristic search algorithm based on artificial neural networks**. Proceedings of the International Conference on Intelligent Information Processing. ACM. p.51, 2016.

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

REIJNEN, R.; ZHANG, Y.; NUJITEN, W. P. M.; SENARAS, C.; GOLDAK-ALTGASSEN, M. **Combining Deep Reinforcement Learning with Search Heuristics for Solving Multi-Agent Path Finding in Segment-based Layouts**. IEEE Symposium Series on Computational Intelligence. IEEE Press, 2020.

SILVER, D.; **Cooperative pathfinding**. AIIDE, 1:117 –122, 2005.

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

SOUZA, L.; **Busca hierárquica de caminhos com redes neurais profundas.** Trabalho de Conclusão do Curso em Engenharia de Computação - Universidade Federal de Santa Maria, Rio Grande do Sul, 2021.

STURTEVANT, N. R. **Benchmarks for Grid-Based Pathfinding. IEEE Transactions.** Computational Intelligence and AI in Games, 4(2), 144–148. 2012.

TAKAHASHI, T.; SUN, H.; TIAN, T.; e WANG Y. **Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks**. Proceedings of the International Conference on Automated Planning and Scheduling. p.764-772, 2019.

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

ZUSE, K. (1972), **Der Plankalkül**. Konrad Zuse Internet Archive. p. 96–105, 1972.