UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

CAROLINE CHAGAS

Vizinhança 3^k para busca de caminhos em ambientes 3D representados por grids cúbicos

Dissertação apresentada como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Orientador: Prof. Dr. Edison Pignaton de Freitas

Porto Alegre
Julho 2022

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Chagas, Caroline

Vizinhança 3^k para busca de caminhos em ambientes 3D representados por grids cúbicos / Caroline Chagas. – Porto Alegre: PPGC da UFRGS, Julho 2022.

101 f.: il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, Julho 2022. Orientador: Edison Pignaton de Freitas.

1. Grande vizinhança. 2. Vizinhança 3^k. 3. Ambientes 3D. 4. Busca de caminhos. I. de Freitas, Edison Pignaton. II. Título.

AGRADECIMENTOS

Gostaria de agradecer ao meu orientador, Edison Pignaton de Freitas, pela oportunidade em trabalhar nessa pesquisa como sua aluna orientada. Agradeço pela atenção, orientação e auxílio nas pesquisas, compartilhando seu conhecimento e revisões para a concretização deste trabalho. Agradeço também ao professor Luis Álvaro de Lima Silva, que me acompanha e auxilia desde a graduação, fornecendo conhecimentos e correções que contribuem para a qualidade dos trabalhos desenvolvidos por mim.

Agradeço à minha família, que sempre apoiou meus objetivos e me incentivaram ao estudos para atingí-los, vibrando comigo a cada conquista.

Agradeço aos colegas e amigos que acompanharam essa jornada, muitas vezes auxiliando e incentivando meu trabalho. Em especial, aos colegas Gabriel, Daniel e Eliakim, que sempre estiveram dispostos a trocar ideias e a compartilhar seus conhecimentos.

À CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) por me conceder a bolsa que apoiou minha pesquisa. Agradeço também à banca avaliadora, pela disponibilidade e contribuições para aperfeiçoamento do trabalho.

Agradeço também à equipe do projeto SIS ASTROS GMF e ao Exército Brasileiro, por colaborarem com as pesquisas no projeto e fornecerem recursos para realização dos trabalhos e experimentos.

RESUMO

Planejamento de rotas é uma importante área de estudo da Inteligência Artificial (IA), visto seu emprego em diversos domínios de aplicação. A execução de jogos e simulações considerando espaço tridimensional (3D) se enquadram nesse contexto apresentando elevado grau de complexidade. Consequentemente, os algoritmos usados nesse caso também apresentam maior complexidade, visto que há mais aspectos a serem tratados quando comparados a ambientes 2D. Adaptar um algoritmo 2D para ambientes 3D não é uma tarefa trivial. Muitos dos algoritmos que funcionam de forma adequada em ambientes planos, no espaço não oferecem mesmo desempenho. Perante este cenário, foi pensada uma nova abordagem de expansão da vizinhança no processo de busca, visando contribuir para o planejamento de caminhos de forma que a técnica desenvolvida possa compor implementações básicas de tais algoritmos, visando resolver o problema em ambientes 3D. Poucos trabalhos na literatura abordam técnicas de ampliação da vizinhança que possibilitem ao algoritmo de busca de caminhos ampliar suas direções de movimento. Essa é uma forma dos algoritmos de busca melhorarem a qualidade das soluções retornadas, promovendo suavização dos caminhos encontrados. Partindo deste princípio, o trabalho apresenta uma nova e extensa expansão de vizinhança para ambientes 3D. A chamada *"Vizinhança* 3^k*"* foi desenvolvida com a finalidade de fornecer benefícios próximos aos dos algoritmos *any-angle*, porém com implementações mais simples. A *Vizinhança* 3^k pode ser aplicada a qualquer algoritmo que não tenha como característica fundamental uma expansão de vizinhança própria. Porém, por ser uma expansão mais ampla e complexa, a principal ideia é que seja aplicada a implementações simples, de forma a produzir caminhos de melhor qualidade, resultados próximos aos de implementações complexas. Os resultados dos experimentos, realizados com os algoritmos A*, JPS e Lazy Theta*, demonstraram que o uso da técnica de vizinhança proposta proporcionou suavização dos caminhos retornados pelos algoritmos testados, melhorando a qualidade final dos caminhos resultantes.

Palavras-chave: Grande vizinhança. Vizinhança 3^k. Ambientes 3D. Busca de caminhos.

The 3k neighborhood for grid path planning in 3D environments

ABSTRACT

Path planning is an important area of study in Artificial Intelligence (AI) employed in different application domains. Games and simulations running in three-dimensional space (3D) are in this context presenting an important complexity. Consequently, their algorithms are also more complex, since there are more aspects to be dealt with compared to 2D environments. Adapting 2D algorithms to 3D environments is not a trivial task. Many of the algorithms that work adequately in flat environments, do not offer the same performance in 3D spaces. Given this scenario, a new approach of neighborhood expansion in the search process was conceived, aiming to contribute to path planning by composing with basic implementations of such algorithms, aiming to solve the problem in 3D environments. Few works in the literature address neighborhood expansion techniques that allow pathfinding algorithms to expand the possible movement directions. This is a way the algorithms can improve the qualities of the provided solutions, smoothing the resulting paths. Based on this principal, this paper presents a new and extended neighborhood expansion approach for 3D environments. The so-called *"3k Neighborhood"* was developed with the purpose of providing benefits close to those of any-angle algorithms, but with simpler implementations. The *"*3^k *Neighborhood"* can be applied to any algorithm that does not have as a fundamental characteristic a particular neighborhood expansion. Nevertheless, because it is a broader and more complex expansion, the main idea is to apply it to easy implementations, in order to produce better quality paths, with results close to those of complex implementations. The results of the experiments, performed with the A*, JPS and Lazy Theta* algorithms, demonstrated that the use of the proposed neighborhood technique provided smoothing to the paths provided by the tested algorithms, improving the quality of the resulting paths.

Keywords: Large neighborhood. 3^k neighborhood. 3D environments. Pathfinding.

LISTA DE FIGURAS

LIST OF ALGORITHMS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IA	Inteligência Artificial
JPS	Jump Point Search
LT*	Lazy Theta*
LNS	Large Neighborhood Search
VLSN	Very Large Scale Neighborhood Search
ALNS	Adaptive Large Neighborhood Search
FPS	Frames Por Segundo
VANT	Veículo Aéreo Não Tripulado
GPU	Graphics Processing Unit
CPU	Central Processing Unit
VRP	Vehicle Routing Problem
COSPS	Critical Obstacles and Surrounding Point Set
USQ	Uneven Search-space Quantization
VCS	Visual Cone Search
G-SpAR	GPU-Based Spatial Audio Rendering
PBS	Priority-Based Search
PRP	Pollution Routing Problem
VRPTW	Vehicle Routing Problem with Time Windows

SUMÁRIO

1 INTRODUÇÃO

Planejamento de rotas é uma importante área de estudo da Inteligência Artificial (IA), visto que a aplicação de algoritmos voltados para este planejamento pode ser atribuída a diferentes contextos. Ao tratar-se de ambientes em três dimensões, algumas dificuldades podem ser encontradas no planejamento de caminhos eficientes voltados ao mundo real ou a simulações realizadas em ambientes virtuais do mundo real, quando comparados a ambientes 2D. A execução de algoritmos no espaço 3D apresentam maior complexidade, visto a maior complexidade destes ambientes. Muito é explorado em aplicações voltadas a contextos bidimensionais; contudo, pesquisas voltadas à aplicações tridimensionais ainda são necessárias. Nos últimos anos, problemas em cenários tridimensionais têm recebido atenção dos pesquisadores, visto o crescente uso de ambientes 3D em aplicações voltadas à jogos (MILLINGTON; FUNGE, 2009), sistemas de simulação (ABAR et al., 2017), robótica (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011), algoritmos de navegação para veículos aéreos não tripulados (VANTs) (MANDLOI; ARYA; VERMA, 2021)(FARIA et al., 2019), entre outras.

Quando trabalha-se com mapas e estruturas para representar tais ambientes 3D, aumentamos o espaço de busca e consequentemente o consumo de memória. Neste trabalho, compreende-se por ambientes 3D o uso estruturas de representação de mapas de navegação 3D. Aumentando a complexidade do ambiente, consequentemente, os algoritmos a serem executados neste cenário também apresentam maior complexidade, visto que há mais aspectos a serem tratados quando comparados a ambientes 2D. Por exemplo, representando os mapas com grids regulares, enquanto tem-se 8 vizinhos em ambientes de duas dimensões (considerando também movimentos diagonais), em ambientes 3D, representados por grids 3D, os nodos do grid passam a ter 26 vizinhos. Ampliando-se as direções de movimentação, mais condições e tratamentos devem ser aplicados. Além disso, tratamentos e adaptações visando melhorar desempenho e uso de memória devem ser considerados. Dessa forma, torna-se interessante explorar técnicas e algoritmos que sejam aplicáveis em ambientes 3D.

Para encontrar caminhos de melhor qualidade em tais ambientes, explorando melhor a ampliação de direções de movimento, e ainda utilizar algoritmos de busca em grids ou grafos, pesquisas na área de planejamento de caminhos ampliaram o tradicional algoritmo A*, desenvolvendo variações deste que tenham a capacidade de encontrar movimentos com maior diversidade de ângulos (por exemplo, (NASH et al., 2007), (HA-

RABOR; GRASTIEN, 2011) e (NASH; KOENIG; TOVEY, 2010). Ou ainda, propostas de algoritmos que podem considerar diretamente o uso de ângulos arbitrários na busca de caminhos - por exemplo, (HARABOR et al., 2016). Apesar de algoritmos que tenham o intuito de melhorar a qualidade dos caminhos, tornando-os mais diretos e realistas, pouco é aprofundado na literatura sobre ampliação da vizinhança para tais fins. Ainda assim, alguns trabalhos propõem técnicas para isso, majoritariamente voltadas a ambientes 2D, como a chamada vizinhança 2^k para ambientes bidimensionais proposta em (RIVERA; HERNÁNDEZ; BAIER, 2017)(RIVERA et al., 2020). Ou mesmo o uso de grandes vizinhanças para fins de otimização de soluções, como a introdução de *Large Neighborhood Search (LNS)* em (SHAW, 1998), uma heurística adaptativa de busca em grande vizinhança em (DEMIR; BEKTAŞ; LAPORTE, 2012), abordagem de LNS em sistema multiagente para otimizar a solução em (LI et al., 2021), entre outras propostas. Conforme citados, quando trata-se do uso de grandes vizinhanças em problemas de otimização de soluções, a variedade em trabalhos disponíveis na literatura é maior. Há inclusive trabalhos que revisam métodos de *Very Large Scale Neighborhood Search* (VLSN - que refere-se à ampla classe de algoritmos que pesquisam vizinhanças muito grandes, abrangendo LNS)(AHUJA; ORLIN; SHARMA, 1998) e trabalhos que discutem variações e extensões dos métodos (por exemplo, (PISINGER; ROPKE, 2010) e (PISINGER; ROPKE, 2019)). No âmbito de grandes vizinhanças em estruturas de representação para solucionar problemas de busca de caminhos, pesquisas mais contundentes sobre esse assunto ainda precisam ser apresentadas na literatura.

Em algoritmos de busca de caminhos, o uso de ampliação da vizinhança permite que algoritmos ampliem as possibilidades em movimentação. Isso permite uma maior variedade em direções de movimento, em adição às direções fornecidas por vizinhanças convencionais em grids. Ainda, algoritmos básicos e com implementações menos complexas também podem suavizar o caminho. Além de pouco aprofundada, a pesquisa tem focalizado a solução de problemas envolvendo duas dimensões. Quando trata-se de ambientes de três dimensões, tal como em grids 3D, por exemplo, técnicas de ampliação da vizinhança são ainda menos exploradas.

Adaptar um algoritmo 2D para ambientes 3D não é uma tarefa trivial, principalmente quando a implementação do algoritmo de busca é complexa. Ainda, é interessante desenvolver novas técnicas que possam ser acopladas a tais algoritmos voltados para espaços 3D. No contexto de desenvolvimento em ambientes 3D, sistemas voltados para jogos e simulações exigem bom desempenho de algoritmos de planejamento de caminhos

à medida que realizam uma exploração eficiente do espaço de busca. Ampliar a exploração da vizinhança em tais sistemas, aplicando novas técnicas de expansão aos algoritmos de busca, pode fornecer boas soluções para otimizar implementações padrão destes algoritmos. Ainda, é mais prático aliar outras técnicas aos algoritmos, como tratamentos de restrições, as quais podem ser necessárias para resolver problemas de aplicação particulares.

Desenvolvendo novas propostas para técnicas de busca a serem aplicadas neste contexto, pode-se reduzir a complexidade exigida na maioria das implementações de algoritmos voltados ao espaço 3D. Associado a isso, é possível melhorar a qualidade dos caminhos retornados com a suavização promovida, de forma que a tarefa de busca de caminhos seja otimizada - o que não ocorre com todos os algoritmos quando adaptados para versão 3D. Perante este cenário, este trabalho apresenta uma nova abordagem de expansão da vizinhança no processo de busca, visando contribuir para o planejamento de caminhos por meio da associação da nova vizinhança ao emprego de implementações básicas de algoritmos de planejamento de caminhos. Dessa forma, algoritmos de busca de caminhos de menor complexidade podem ser utilizados em diferentes aplicações fornecendo caminhos de melhor qualidade em ambientes 3D.

A motivação em pesquisar soluções em busca de caminhos para ambientes 3D surgiu da necessidade de tratar problemas reais em um sistema de simulação tático militar. Dessa forma, o trabalho está inserido no contexto do projeto do sistema de simulação SIS-ASTROS GMF (descrito em (SIS-ASTROS. . . , 2020), (POZZER C. T.; FREITAS, 2022) e (BRONDANI et al., 2018)), no qual outros trabalhos de navegação também foram desenvolvidos a fim de tratar problemas de navegação terrestre em mapas 2.5D (como, por exemplo, (CHAGAS et al., 2022), (NEISSE C.; FREITAS E. P., 2022) e (BRONDANI et al., 2019)). Ou seja, mapas 2D eram utilizados e representados por meio de estruturas de representação 2D, contudo continham informações de inclinações do relevo. Concretizando as implementações de algoritmos de navegação terrestre dos agentes envolvidos, novas necessidades surgiram no projeto. Dentre estas, a implementação de algoritmos que possam ser aplicados à problemas do espaço aéreo, como navegação aérea de VANTs. A fim de tratar problemas de navegação no espaço aéreo, um estudo sobre ambientes 3D foi iniciado, o que permitiu identificar a complexidade de algoritmos de busca de caminhos em tais ambientes. A partir do estudo, o assunto foi explorado através da investigação de algoritmos e técnicas que com potencial para endereçar os desafios do problema em questão.

1.1 Objetivos

O objetivo do trabalho é investigar e propôr uma vizinhança ampliada para algoritmos que tratam o problema de busca de caminhos envolvendo ambientes 3D. Por meio da chamada *Vizinhança* 3^k, o trabalho permite computar caminhos suavizados e de melhor qualidade, além de empregar implementações que não possuam grande complexidade. Ainda, a técnica é projetada e testada em cenários de ambientes 3D com variado número de obstáculos. Apesar de alguns autores na literatura realizarem trabalhos de análise teórica de algoritmos e técnicas propostas (por exemplo, (ALGFOOR; SUNAR; KOLIVAND, 2015) e (PISINGER; ROPKE, 2010)), o presente trabalho apresenta uma análise experimental da técnica de vizinhança proposta na computação de caminhos em ambientes 3D. Como objetivos específicos do trabalho, listam-se os seguintes:

- Revisar algoritmos de busca voltados ao tratamento de ambientes 3D;
- Revisar técnicas de ampliação de vizinhança disponíveis na literatura;
- Desenvolver a *Vizinhança* 3^k, uma vizinhança ampliada a ser aplicada em algoritmos de busca executados em ambientes 3D;
- Realizar uma análise experimental da *Vizinhança* 3^k;
- Implementar algoritmos utilizando a *Game Engine Unity 3D*;
- Avaliar estatisticamente os dados experimentais obtidos através da execução de um protocolo de testes visando a comparação do trabalho desenvolvido com o estado da arte.

1.2 Contribuições

O trabalho apresenta uma nova expansão de vizinhança para algoritmos de busca voltados para ambientes 3D. Partindo da técnica proposta em (RIVERA; HERNÁNDEZ; BAIER, 2017)(RIVERA et al., 2020), e ampliando a vizinhança para *"Vizinhança* 3^k*"*, amplia-se os saltos entre nodos durante a busca de caminho, promovendo a suavização ao passo que a busca do caminho é realizada. Como técnicas de expansão da vizinhança para tratar problemas do contexto 3D, no geral, ainda são pouco exploradas na literatura quando comparadas a técnicas voltadas para ambientes 2D, este trabalho apresenta e discute a exploração da técnica de vizinhança 3^k para ambientes 3D. Em geral, essa técnica exige a consideração de mais fatores (como movimentação em mais direções, incluindo

diagonais de dois e três eixos, campo de visão para estrutura 3D, entre outros), visto que os algoritmos tornam-se mais complexos no espaço. Com planejamento de caminhos em espaços 3D utilizando algoritmos suavizados, este trabalho apresenta as seguintes contribuições:

- Proposição da *vizinhança* 3^k, uma vizinhança expandida voltada para ambientes 3D. Tal vizinhança proporciona mais direções de movimentação e é aplicável idealmente em grids regulares;
- Emprego da técnica de *vizinhança* 3^k para melhorar a qualidade dos caminhos retornados, realizando suavização durante o processo de busca por meio de implementações com menor grau de complexidade amplamente exploradas pela indústria no desenvolvimento de jogos e sistemas de simulação;
- Adaptação de diferentes algoritmos de busca de caminhos (nominalmente, A*, JPS e Lazy Theta*) para uso da *vizinhança* 3^k em ambientes 3D;
- Por meio de resultados estatísticos de modelo de regressão generalizados, apresentação e discussão de diferentes conjuntos de testes experimentais da vizinhança 3^k em três diferentes ambientes/mapas 3D, com diferentes complexidades em termos de número de obstáculos.

1.3 Estrutura do trabalho

O trabalho é organizado da seguinte forma. A seção 2 apresenta a Definição do problema, juntamente com as definições dos termos, estruturas e técnicas utilizadas neste trabalho. Na seção 3 é apresentado o Referencial teórico da implementação, abordando conceitos, técnicas e algoritmos utilizados no trabalho, além de apresentar trabalhos relacionados na subseção 3.2. Na seção 4 é introduzida a proposta do trabalho, apresentando em 4.1 a ideia na qual a proposta é baseada e descrevendo em 4.2 a proposta de *Vizinhança* 3^k deste trabalho. Na seção 5 os experimentos, métodos analíticos e análises dos resultados obtidos são apresentados e discutidos. Finalizando, a seção 6 apresenta considerações finais.

2 DEFINIÇÃO DO PROBLEMA

Planejamento de caminhos é um problema de busca de caminhos dados um ponto inicial e um ponto de destino em um determinado grid G, representando um domínio D. Este é um problema com muitas aplicações, como robótica (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011), vídeo games (MILLINGTON; FUNGE, 2009), rede de navegação em estradas (BAST et al., 2016), sistemas de simulação (ABAR et al., 2017), entre outras.

Um dos aspectos mais importantes para o planejamento de caminhos é o *ambiente* onde o caminho resultante é usado por um agente. Geralmente, a estrutura de representação utilizada armazena informações que modelam as principais características do ambiente. Entre outras, o modelo indica quais são os locais transitáveis ou não do ambiente. Os obstáculos são áreas instransponíveis representadas por nodos bloqueados na estrutura. A quantidade de informações representada neste modelo aumenta na proporção em que aumenta-se o tamanho físico do ambiente e a granularidade desejada.

Na literatura de busca de caminhos ((ALGFOOR; SUNAR; KOLIVAND, 2015), (CHOSET et al., 2005), (NASH; KOENIG, 2013), (NILSSON N. J., 1998), (SOUISSI et al., 2013), (URAS; KOENIG, 2015), (YANG et al., 2016) e (YAP et al., 2011)), a execução de algoritmos de busca de caminhos é primariamente explorada em grids regulares e outras estruturas de dados utilizadas na representação de ambientes 2D. O acréscimo de uma dimensão, necessário para o desenvolvimento de diferentes sistemas que necessitam a utilização de ambientes 3D, amplia o espaço de busca a ser explorado pelos algoritmos de busca de caminhos, aumentando também a complexidade da tarefa de planejamento de caminhos. Tarefas que podem parecer triviais em ambientes 2D podem não ser bem sucedidas quando reproduzidas no espaço tridimensional. Entre outros exemplos que podem ser citados, a qualidade dos caminhos pode não ser adequada para as necessidades do problema a ser resolvido. Em termos de desempenho, os tempos de computação dos caminhos podem ser impraticáveis para muitas aplicações. Outro problema que pode ocorrer é o consumo elevado de memória do algoritmo de busca devido ao aumento do espaço de busca. De forma geral, as tarefas de computação de caminhos devem ser adaptadas ou até mesmo replanejadas visto as dificuldades adicionais apresentadas na execução de algoritmos de busca de caminhos em espaços de estados resultantes de ambientes 3D.

Um mapa $M \times N \times L$ é representado por (G, O), onde G é o grid e $O \subseteq G$ representa o conjunto de obstáculos. A definição de um grid no espaço 3D é dada pelas

dimensões de cada eixo. Assim, um grid de dimensões $M \times N \times L$ é um conjunto de tuplas ordenadas tal que G = { (x, y, z) | 0 ≤ x ≤ M, 0 ≤ y ≤ N, 0 ≤ z ≤ L }.

Formalizando, o domínio de um planejamento de caminhos é uma tupla D = $< N, O, E, c >$, onde:

- N é o conjunto de nodos;
- O é o conjunto de obstáculos;
- $E \subseteq M \times N \times L$ é o conjunto de arestas (conexões) entre os nodos;
- c: $E \mapsto R_0^+$ retorna o custo de travessia entre nodos.

Um problema de planejamento de caminho é denotado por uma tupla $<D$, s, $g>$, onde:

- $D = < N, O, E, c >$ é o domínio do planejamento de caminhos;
- $s \in N$ é o ponto inicial;
- $g \in N$ é o ponto destino.

A solução de um problema dessa natureza é um caminho π no domínio D correspondente, tal que π^0 = s and $\pi^{|\pi-1|}$ = g. O custo entre dois nodos é o custo de um caminho entre eles. Assim, o custo ótimo entre dois nodos n_i e n_j é dado por $c(n_i, n_j)$. A fim de encontrar um caminho com custo ótimo, muitas abordagens em IA utilizam variações do tradicional A* (HART; NILSSON; RAPHAEL, 1968). Todavia, quando problemas complexos são abordados, onde o desempenho dos algoritmos em grandes espaços de busca é uma questão importante a ser considerada, caminhos subótimos são geralmente aceitos. Neste trabalho, tais aspectos são considerados, buscando comparar e avaliar as relações custo-benefício dentre as implementações.

Este trabalho utiliza um grid regular como estrutura de representação do ambiente, e o tratamento de restrições típicas de aplicações e outros fatores de custo que poderiam ser associados aos caminhos computados não são abordados. Logo, o custo de travessia dos nodos do grid é uniforme. O conjunto de todos os caminhos no domínio denota-se por Π, e o conjunto de todos os caminhos π iniciados em $\pi^0 = n_1$ e tendo o destino em $\pi^{|\pi-1|} = n_2$ denota-se por $\Pi(n_1, n_2)$.

Para encontrar caminhos, cada nodo analisado pelo algoritmo de busca possui uma sequência de movimentos possíveis. Os nodos que podem ser atingidos por esses movimentos compõem a *vizinhança* no nodo. As conexões de vizinhança em grids 2D geralmente ocorrem na forma de 4 direções (nas orientações horizontal e vertical) ou 8 direções (orientações horizontal, vertical e diagonais).

De acordo com Rivera, Hernández and Baier (2017), a movimentação de agentes entre nodos vizinhos pode ser definida da seguinte forma: dado um grid G_{2D}, considerando um ambiente 2D, denota-se um vetor movimento por (u, v) tal que u e v são dois inteiros diferentes de 0. Quando o movimento (u, v) é executado a partir de um nodo de coordenadas (x, y), o local resultante, ou seja, o ponto que finaliza o movimento, é dado por $(x + u, y + v)$. O movimento (m, n) é viável no nodo (x, y) do grid desde que o nodo objetivo $(x+u, y+v) \in G_{2D}$ não seja um obstáculo. Além disso, outros nodos que sejam tocados pelo segmento entre esses dois nodos, de início e fim do movimento, não podem ser obstáculos. Isso indica que ambos os nodos devem ter *visibilidade* de um para o outro. O custo de um movimento (u, v) é definido por $\sqrt{u^2 + v^2}$. Logo, o custo do caminho final π é dado pela soma dos custos de todos os k movimentos que o compõem: $\sum_{i=0}^{k}(u_i, v_i)$. Um grid G_{2D} satisfaz uma vizinhança de 4 conexões quando os movimentos permitidos para (x, y) são aqueles movimentos executáveis em:

$$M^4 = \{(u, v) \mid u, v \in \{-1, 0, 1\}, |u| + |v| = 1\}, \tag{2.1}$$

que interpreta-se como movimentos verticais e horizontais de custo = 1 (Figura 2.1(a)). Para grids de 8 conexões, os movimentos executáveis são dados por:

$$M^8 = \{(u, v) \mid u, v \in \{-1, 0, 1\}, |u| + |v| > 0\}, \tag{2.2}$$

onde o conjunto de movimentos em M^4 é ampliado, incluindo movimentos diagonais (Figura 2.1(b)). De acordo com técnicas que permitam ampliar a vizinhança de um nodo, por exemplo (RIVERA; HERNÁNDEZ; BAIER, 2017)(RIVERA et al., 2020), é possível ampliar a possibilidades de direções de movimento de um agente que trafega de um nodo a outro. Assim, o conjunto de nodos visitados pela ampliação de um movimento (u, v) em um nodo (x, y) é dado por:

$$(x', y') = \{(x, y) + \lambda(u, v) \mid \lambda \in]0, 1[\}, \tag{2.3}$$

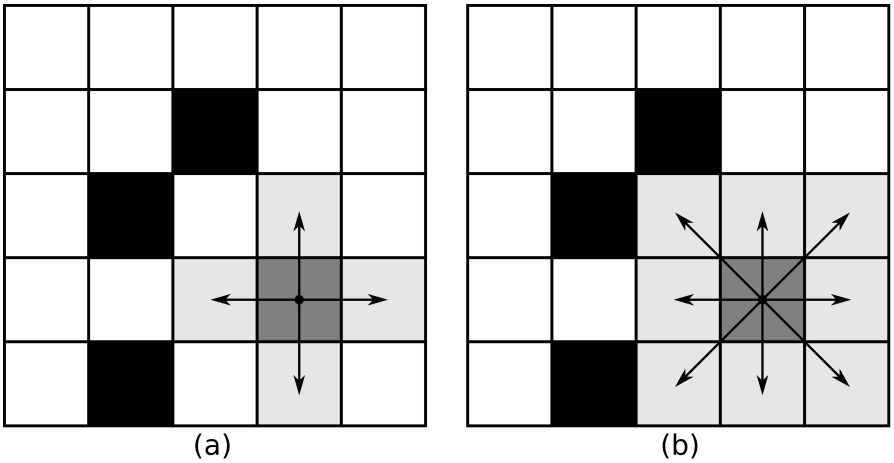
Dessa forma, o conjunto de sucessores de um nodo n, cujas coordenadas são (x, y), é definido como:

$$Succ(n) = \{(u, v) + n \mid (u, v) \in N\}, \tag{2.4}$$

e (u, v) é aplicável em n.

Em resumo, um caminho de s para g por meio da vizinhança N é uma sequência de nodos n_1, n_2,...,n_n, tal que n_1 = s e n_n = g e para todo $i \in \{0, ..., n-1\}$. Logo, $n_{i+1} \in Succ(n_i)$.

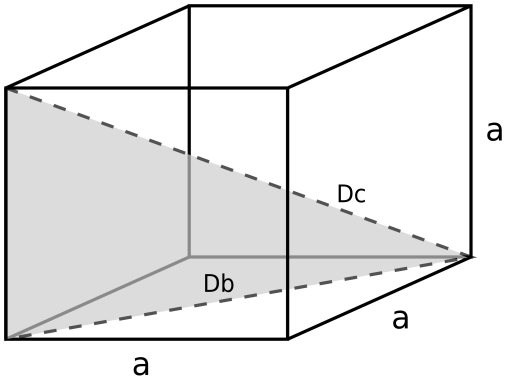
Figura 2.1: Vizinhança 2D representada pelos nodos em cinza claro. a) Movimentos de vizinhança em direções cardeais M^4; b) Movimentos de vizinhança incluindo direções diagonais M^8.



Fonte: Autora

Para um grid uniforme cúbico (3D), os nodos vizinhos são semelhantes e, considerando vizinhanças convencionais, terão sempre o mesmo número de conexões. Com a terceira dimensão, além dos movimentos padrão em eixos horizontal e vertical, de custo a (onde a é a medida da lateral do nodo), e diagonal de dois eixos, de custo Db = $a\sqrt{2}$, tem-se o custo de movimentação em diagonal nos três eixos, dado por Dc = $a\sqrt{3}$. Esse tipo de movimento é ilustrado na Figura 2.2.

Figura 2.2: Custos de movimentação com nodos cúbicos.



Fonte: Autora

No cenário 3D, as conexões de vizinhança em grids cúbicos ocorrem na forma de 6 direções (nas orientações horizontal, vertical e profundidade) ou 26 direções (ori-

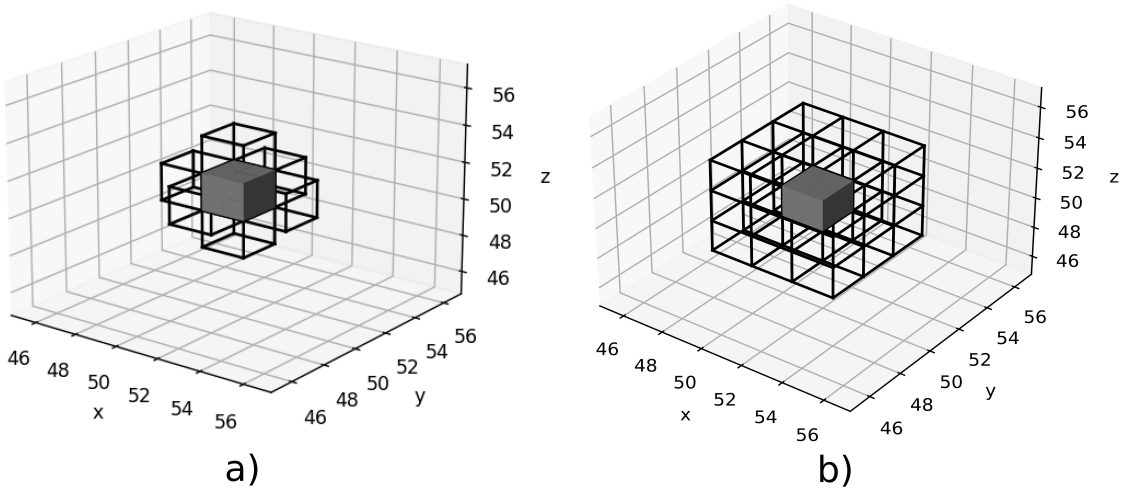
entações horizontal, vertical, profundidade e diagonais). Quando o movimento (u, v, w) é executado a partir de um nodo de coordenadas (x, y, z), o ponto destino do caminho é dado por $(x + u, y + v, z + w)$. Um grid G_{3D} satisfaz uma vizinhança de 6 conexões quando os movimentos permitidos para (x, y, z) são aqueles movimentos executáveis em:

$$M^6 = \{(u, v, w) \mid u, v, w \in \{-1, 0, 1\}, |u| + |v| + |w| = 1\}, \tag{2.5}$$

onde existem movimentos verticais (para cima a para baixo), horizontais (para direita e esquerda) e de profundidade (para frente e para trás) de custo = 1 (Figura 2.3(a)). Para grids cúbicos de 26 conexões (Figura 2.3(b)), quando movimentos diagonais no espaço são incluídos, os movimentos executáveis são dados por:

$$M^{26} = \{(u, v, w) \mid u, v, w \in \{-1, 0, 1\}, |u| + |v| + |w| > 0\}, \tag{2.6}$$

Figura 2.3: Vizinhança 3D representada pelos nodos contornados em preto. a) Movimentos de vizinhança em direções cardeais M^6; b) Movimentos de vizinhança incluindo direções diagonais M^{26}.



Fonte: Autora

Este trabalho aborda o problema de computação de caminhos suavizados em mapas 3D, que representam ambientes de três dimensões. Para isso, técnicas de busca que consideram uma *vizinhança ampliada* são investigadas. Tal aspecto é analisado pois algoritmos de computação de caminhos que consideram a vizinhança ampliada podem retornar caminhos mais diretos, ou seja, algoritmos que promovem a determinação de caminhos suavizados. Ao utilizar uma vizinhança ampliada, isso pode ocorrer mesmo em casos em que o algoritmo de busca não realiza suavização como parte da implementação utilizada. Geralmente esses algoritmos são de natureza *any-angle*, ou seja, algoritmos que

buscam caminhos de qualquer ângulo.

Um caminho de qualquer ângulo π é uma sequência de pontos $<p_1, ..., p_k>$ onde cada p_i é visível de p_{i-1} e tem visibilidade para p_{i+1}. O comprimento de π é a distância acumulada entre cada par de pontos sucessivos $d(p1, p2)$ + ... + $d(p_{k-1}, p_k)$, onde $d((x, y, z), (x', y', z'))$ = $\sqrt{(x' - x)^2 + (y' - y)^2 + (z' - z)^2}$ é um métrica de distância Euclidiana uniforme. Denominamos $p_i \in \pi$ como um *ponto de inflexão* se os segmentos (p_{i-1}, p_i) e (p_i, p_{i+1}) formam um ângulo diferente de 180º. É importante ressaltar que tais pontos de inflexão em caminhos de qualquer ângulo ótimos são pontos que contornam obstáculos.

Visto que em ambientes 3D os algoritmos apresentam maior complexidade, alguns algoritmos aplicáveis em problemas 2D por vezes tornam-se inviáveis de adaptação para problemas 3D. Visando simplificar as implementações de algoritmos 3D e ainda fornecer caminhos de melhor qualidade, o trabalho investiga uma implementação de uma expansão de vizinhança ampliada, chamada *Vizinhança* 3^k. Por meio do desenvolvimento desta técnica, pretende-se reduzir os pontos de inflexão ao longo do caminho, a exemplo do comportamento resultante do emprego de algoritmos *any-angle*, mantendo os comprimentos dos caminhos retornados pelas versões padrão dos algoritmos de busca, ou até mesmo reduzindo tais comprimentos.

3 REFERENCIAL TEÓRICO

Esta seção apresenta conceitos relevantes para a pesquisa desenvolvida. O trabalho investiga a "expansão de vizinhança". Entre outras características, essa expansão amplia as direções de movimentação para nodos vizinhos explorados pelos algoritmos de planejamento de caminhos, tornando os algoritmos testados implementações com resultados próximos aos de implementações *any-angle*. Mesmo em implementações padrão de algoritmos de planejamento de caminhos, como o algoritmo A*, o uso de maiores vizinhanças permite ampliar as possibilidades de movimentação, suavizando os caminhos retornados. Em algoritmos de natureza *any-angle*, maiores vizinhanças também podem contribuir para alcançar melhores aspectos de suavização. Dessa forma, as implementações utilizando a vizinhança 3^k são analisadas de acordo com algoritmos que possuam essa característica. Ainda, estudos de algoritmos de planejamento de caminhos realizados em outros cenários tridimensionais são interessantes ao trabalho. Assim, uma revisão sobre abordagens 3D e sobre expansão de vizinhança em grids regulares é apresentada de acordo com algoritmos que realizam suavização dos caminhos, além de outros algoritmos padrão que foram utilizados nos experimentos desenvolvidos nesta pesquisa.

3.1 Algoritmos que utilizam técnicas de suavização

Dentre os algoritmos de busca, o algoritmo A* (NILSSON N. J., 1998) e variações deste estão entre os mais explorados. A* é um algoritmo heurístico que garante a menor quantidade de nodos expandidos em relação a outros algoritmos. Apesar de ser um algoritmo ótimo e completo, o custo computacional deste algoritmo aumenta à medida que a complexidade do problema aumenta. Isso ocorre quando, por exemplo, o espaço de busca e a quantidade de agentes são aumentados. O algoritmo A* utiliza uma heurística para auxiliar a ordem em que os nodos que representam a estrutura do mapa são processados, com o intuito de diminuir o tempo de processamento. Sua execução ocorre baseada na seguinte lógica: o nodo inicial do caminho é adicionado a uma lista, a qual mantém os nodos abertos pelo algoritmo ao longo de sua execução. Enquanto houver nodos nesta lista, o algoritmo escolhe o melhor nodo e verifica se este é um nodo objetivo. Se este for o destino da busca, o caminho foi encontrado. Caso não seja, o algoritmo busca por vizinhos do nodo atual ainda não visitados. Para cada um destes, é verificado o custo de movimento do ponto inicial até a posição atual e o custo de movimento para o vizinho.

Os caminhos retornados pelos algoritmos de planejamento de caminhos, em geral, fornecem uma trajetória que conecta nodos consecutivos de um grid G, gerando segmentos de caminho relativamente curtos. Os caminhos encontrados pelo algoritmo A*, por exemplo, são compostos por segmentos que conectam vizinho a vizinho, desde o ponto inicial até o ponto objetivo, por vezes produzindo caminhos sinuosos, com pequenas curvas que poderiam ser evitadas. Dessa forma, o caminho resultante pode parecer irreal quando agentes trafegando pelos caminhos retornados são observados. Contudo, existe a possibilidade de realizar um pós-processamento desses caminhos, aplicando um algoritmo de suavização sobre o caminho retornado pelo algoritmo de busca. Este segundo algoritmo geralmente substitui pontos desnecessários do caminho por linhas retas (abordado em (THORPE; MATTHIES, 1984), (BOTEA; MÜLLER; SCHAEFFER, 2004), (MILLINGTON; FUNGE, 2009)), desde que não haja bloqueios ao longo da linha de suavização.

Desenvolvidos a partir do A*, com variações para atender diferentes objetivos, é possível notar a classe dos algoritmos *any-angle* (alguns deles abordados e discutidos em (NASH; KOENIG, 2013), (YAP et al., 2011) e (URAS; KOENIG, 2015)). Algoritmos *any-angle* exploram mais direções de movimentação e, consequentemente, maior suavização e realismo ao movimento dos agentes. Tais algoritmos buscam combinar os benefícios do A*, geralmente executado em grids, com a execução deste algoritmo em *grafos de visibilidade* ((LEE, 1978),(LOZANO-PÉREZ; WESLEY, 1979)). Um destes benefícios é o desempenho computacional. O outro é o retorno de caminhos mais curtos, fornecidos pelo A* com o uso desses grafos de visibilidade. Para isso, os algoritmos *any-angle* propagam informações ao longo dos nodos/arestas do grafo (como o A* em grids, para ganhar desempenho), sem restringir os caminhos resultantes aos nodos ou arestas do grid (como o A* em grafos de visibilidade, para encontrar caminhos curtos). Apesar da tendência de encontrar caminhos de menor custo, os algoritmos *any-angle* não garantem isso. Embora não haja garantia de otimização, eles tentam concluir a busca com eficiência, simplicidade e generalidade.

Quando utilizamos algoritmos de planejamento do tipo *any-angle*, a suavização do caminho ocorre simultaneamente ao processo de busca. Esses algoritmos não estão limitados a explorar vizinhos adjacentes a 45º ou 90º. Eles exploram uma maior possibilidade de ângulos uma vez que exista a visibilidade para os nodos a serem atingidos. Apresentando poucas curvas ao longo do caminho encontrado, algoritmos *any-angle* são diferentes do A*, o qual gera caminhos restritos à grade da estrutura na qual executam,

produzindo caminhos indiretos e irregulares.

Neste trabalho, os algoritmos de suavização denominados JPS e Lazy Theta* são abordados, sendo Lazy Theta* do tipo *any-angle*. A finalidade é comparar distâncias e suavização (além de questões referentes a tempo de computação e uso de memória) entre algoritmos de suavização, algoritmos de natureza *any-angle* e algoritmos de implementações mais básicas, onde a exploração de uma expansão da vizinhança venha a proporcionar a suavização do caminho retornado.

3.1.1 Jump Point Search (JPS)

Jump Point Search (HARABOR; GRASTIEN, 2011)(HARABOR; GRASTIEN, 2014)(NOBES et al., 2022) é um algoritmo de busca de caminhos baseado no algoritmo A*, mantendo sua otimalidade e apresentando melhor desempenho em termos de tempo de execução. Diferentemente do A*, ao invés de utilizar uma busca gulosa e expandir um grande número de nodos, o JPS pode selecionar e excluir nodos que seriam irrelevantes para o caminho final. Essa característica, referida como quebra de simetria, é o principal motivo que leva o JPS a superar um algoritmo A*.

O JPS realiza a busca de caminho utilizando a ideia de encontrar um conjunto de "*jump points*". Diferente do A* que avalia dois vizinhos adjacentes por vez, o JPS avalia *jump points* ao longo do caminho, onde estes *jump points* possuem distâncias maiores que os vizinhos mais próximos do nodo atual. Esse processo, juntamente com a seleção planejada dos vizinhos diretos, além de otimizar a busca, contribui para a suavização do caminho. Entre *jump points* consecutivos, pequenos caminhos ótimos são formados, os quais irão compôr o caminho ótimo final.

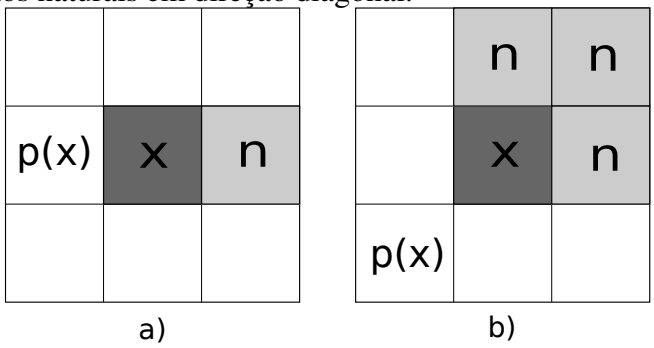
Para realizar uma melhor seleção de sucessores, o algoritmo JPS verifica vizinhos *naturais* e *forçados* do nodo expandido. No entanto, não inclui esses vizinhos na lista de nodos abertos. Essa etapa é importante para o bom desempenho do JPS, visto que somente os vizinhos com potencial de pertencerem ao caminho ótimo são selecionados. Para a seleção de *vizinhos naturais*, o algoritmo seleciona somente os vizinhos que pertencem à direção do movimento executado. Para um nodo x, cujo movimento vem de um pai $p(x)$, a seleção dos vizinhos deve satisfazer as seguintes condições: Equação 3.1, para movimentos nas direções cardeais; Equação 3.2, para movimentos diagonais. Em suma, um nodo n é um vizinho natural quando o caminho de $p(x)$ passando por x até n é o

menor caminho.

$$len(< p(x), ...n > \backslash x) \leq len(< p(x), x, n >) \tag{3.1}$$

$$len(< p(x), ...n > \backslash x) < len(< p(x), x, n >) \tag{3.2}$$

Figura 3.1: Em ambiente 2D: a) Exemplo de vizinhos naturais em direção cardeal; b) Exemplo de vizinhos naturais em direção diagonal.

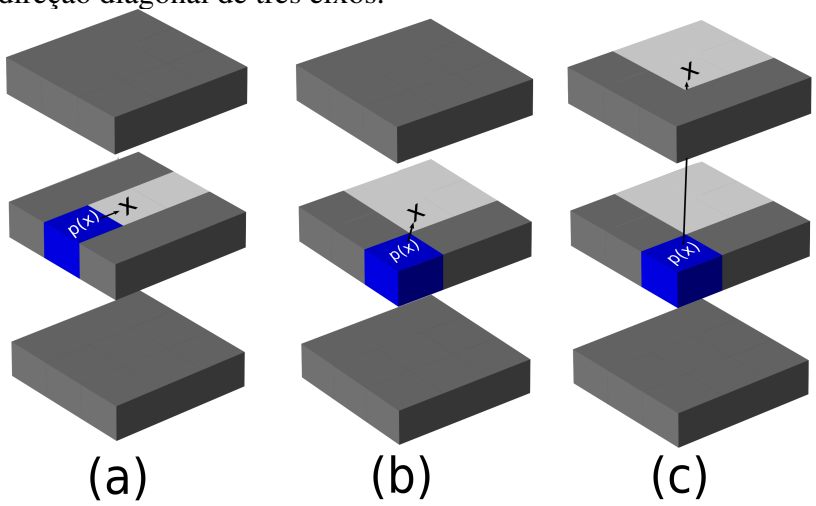


Fonte: Adaptado de (RANTTILA, 2019).

Para a versão 3D, a ideia de seleção dos vizinhos naturais utilizada no algoritmo 2D é generalizada e o grande espaço de estado tridimensional é explorado. Dessa forma, movimentos diagonais que ocorrem mais cedo no caminho são priorizados, sendo movimentos em diagonais de três eixos os de maior prioridade, seguidos por movimentos de dois eixos e, por fim, movimentos cardeais. Neste caso, são podados os nodos n cuja Equação 3.2 seja satisfeita, ou os nodos n nos quais satisfaçam a condição $\pi' = len(< p(x), ...n > \backslash x)\pi = len(< p(x), x, n >)$, de forma que π' tenha um movimento diagonal que ocorra mais cedo do que em π. Por meio da Figura 3.2, identificam-se (a) vizinhos naturais, ilustrados na cor branca, ao movimentar-se na direção de um eixo, do nodo pai $p(x)$, na cor azul, ao nodo atual x. Nessa mesma Figura 3.2(b), identificam-se os vizinhos naturais ao realizar um movimento em diagonal de dois eixos, no qual o exemplo ilustrado varia os eixos x e y. Na Figura 3.2(c), a variação nos três eixos é exemplificada.

Para verificar os vizinhos forçados, nodos bloqueados em torno do nodo x são identificados. Alguns nodos em torno do nodo atual x e do nodo vizinho bloqueado devem ser considerados como possíveis componentes do caminho resultante. Como tais nodos são foçadamente avaliados pelo algoritmo JPS, eles são chamados de *vizinhos forçados*. O conceito de vizinho forçado é uma importante abordagem do algoritmo JPS, visto que

Figura 3.2: Em ambiente 3D: a) Exemplo de vizinhos naturais em direção cardeal; b) Exemplo de vizinhos naturais em direção diagonal de dois eixos; c) Exemplo de vizinhos naturais em direção diagonal de três eixos.

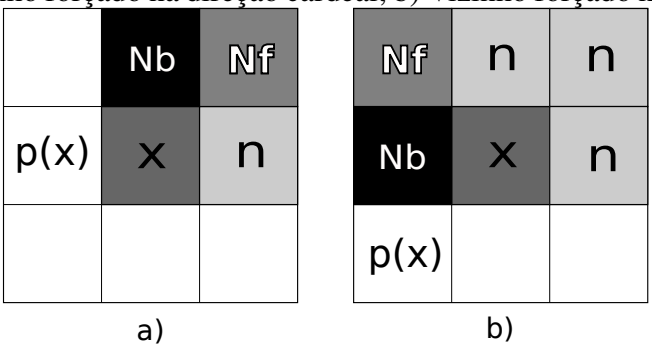


Fonte: Adaptado de (NOBES et al., 2022).

garante a otimalidade do caminho quando há bloqueios que são considerados no processo de busca. Em um caminho cujo movimento inicia em $p(x)$ e passar por x para atingir um vizinho n, porém onde exista um vizinho de x bloqueado n_b, o vizinho n_f torna-se um vizinho forçado a ser avaliado. Isso ocorre apesar das coordenadas do nodo forçado não seguirem a direção do movimento. Um nodo forçado n_f deve satisfazer dois requisitos:

- n_f não é um vizinho natural de x;
- len (<p(x), x, n>) $< len$ (<p(x), ...n>\x).

Figura 3.3: a) Vizinho forçado na direção cardeal; b) Vizinho forçado na direção diagonal.

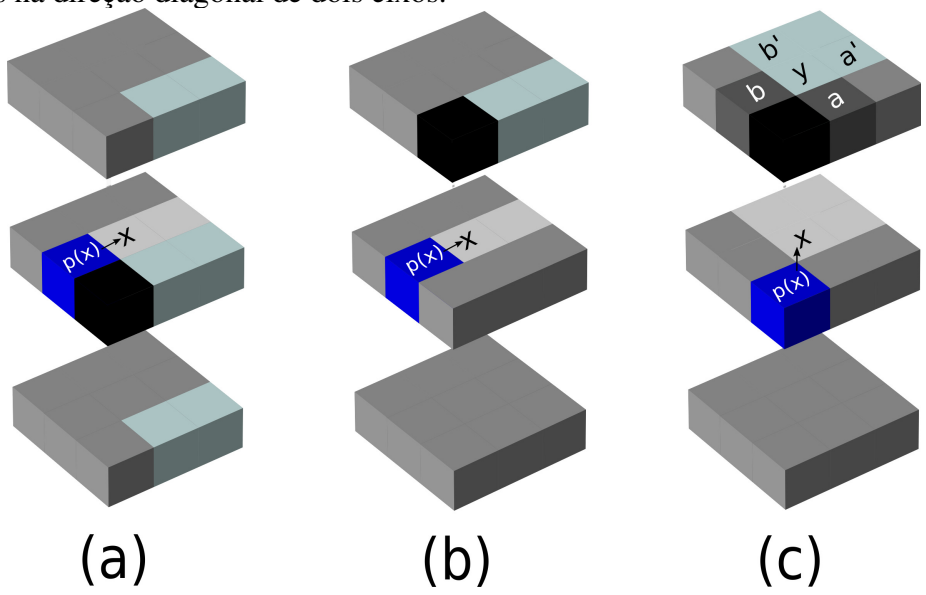


Fonte: Adaptado de (RANTTILA, 2019).

Para identificar vizinhos forçados no algoritmo JPS 3D, as condições definidas em (HARABOR; GRASTIEN, 2011) são mantidas, com uma ressalva em relação ao segundo requisito listado: n é um vizinho forçado caso exista um caminho cujo custo de travessia que seja menor **ou igual (diferença em 3D)** $\pi' = < p, y, n >$ para o caminho π =

$< p, x, n >$ incluindo x, que não é válido devido a uma obstrução causada por um obstáculo entre a passagem de p a y (NOBES et al., 2022). Através da Figura 3.4, os exemplos ilustram os vizinhos forçados na cor azul claro, gerados a partir do nodo bloqueado representado na cor preta, ao realizar um movimento do nodo pai $p(x)$ (representado em azul) para o nodo atual x. Os vizinhos naturais, na cor branca, também são apresentados nas imagens. Na movimentação em diagonal de dois eixos, ilustrada na Figura 3.4(c), havendo possíveis bloqueios dos nodos a ou b, determina que os nodos a' ou b' serão vizinhos forçados, respectivamente.

Figura 3.4: Em ambiente 3D: a) Vizinhos forçados na direção cardeal; b) Vizinhos forçados em diagonal ao *parent(x)* e x quando a direção do movimento é cardeal; c) Vizinhos forçados na direção diagonal de dois eixos.



Fonte: Adaptado de (NOBES et al., 2022).

A partir da seleção de vizinhos, os "*jump points*"são calculados, realizando uma verificação linear nas orientações cardeais e diagonais partindo de cada vizinho. Quando trata-se do algoritmo voltado ao ambiente 3D, a verificação ocorre em mais direções, incluindo diagonais de três eixos e diagonais de dois eixos, nos sentidos em que o movimento é realizado. Somente os *jump points* são adicionados à lista de nodos abertos, por meio da função que identifica tais sucessores. Esse processo de identificação dos sucessores, ou seja, os *jump points*, é central para esse algoritmo.

A definição de um *jump point* é dada por $y = x + k * \vec{d}$, na qual x é o nodo atual e $\vec{d}$ é a direção do movimento. Essa Equação define que um nodo y é um *jump point* somente se o fator k é minimizado e se y satisfaz as condições necessárias para um nodo ser um *jump point*, simultaneamente. Os *jump points* são os pontos de curvas, ou pontos

de inflexão, ao longo do caminho. A definição do ponto de inflexão é qualquer nodo n_i ao longo do caminho cuja direção de travessia em relação ao nodo antecessor n_{i-1} seja diferente em relação ao nodo sucessor n_{i+1}. Ou seja, o caminho de n_{i-1} a n_i possui uma direção diferente do caminho de n_i a n_{i+1}. Assim, esses pontos são selecionados de acordo com um conjunto de regras, listadas abaixo, e de acordo com a direção do movimento. A função de $jump$, que realiza os saltos na direção determinada, é a função que mais exige computação. Esta função é recursiva e constantemente realiza verificações para determinar se pode retornar um *jump point*. O retorno dessa função é um valor de nodo, um *jump point*, ou um valor nulo. Há três casos em que a função retorna um nodo:

1. 1º caso: quando o nodo verificado é o nodo objetivo, demonstrando que o caminho foi encontrado. Naturalmente, essa condição é satisfeita quando a pesquisa é bem sucedida;
2. 2º caso: quando o nodo verificado possui vizinhos forçados. Este caso ocorre quando o algoritmo de busca está contornando obstáculos;
3. 3º caso: quando a direção $\vec{d}$ do movimento é diagonal e existe um nodo $z = y + k_i \vec{d_i}$, no qual $k_i \in N$ passos na direção $\vec{d_i} \in \{\vec{d_1}, \vec{d_2}\}$ tal que z é um *jump point* de y por meio dos casos 1 ou 2;

A terceira formulação pode ser compreendida da seguinte forma: quando o movimento é diagonal, as chamadas de *jump* são feitas sequencialmente para parâmetros de direção vertical e horizontal. Caso qualquer uma dessas chamadas recursivas ao método não retornarem valor nulo, o nodo é um *jump point*. A função de *jump* para direções de diagonais sempre retornam para os casos base de direção direta (vertical e horizontal).

A concepção por trás do processo de exploração da vizinhança do JPS em ambientes 2D é mantida pelo algoritmo adaptado para ser executado no espaço 3D (RANTTILA, 2019). Contudo, ao adicionarmos uma terceira dimensão, além de acrescentarmos uma direção cardeal (nos sentidos de profundidade), acrescentamos não somente diagonais de canto (união das três dimensões que compõem o espaço), mas diagonais entre duas dimensões também são avaliadas durante a busca de um caminho. Dessa forma, cada posição permite explorar 26 direções, diferentemente do algoritmo em 2D que explora somente 8 direções. Apesar de muitas características deste algoritmo serem mantidas quando ele é executado no espaço, algumas adaptações precisam ser realizadas.

Assim como na versão 2D do algoritmo JPS, antecedendo o processo de seleção dos sucessores por meio dos *jump points*, uma poda da vizinhança do nodo atual é reali-

zada. Para isso, somente os "vizinhos naturais"do nodo são selecionados. Estes vizinhos são aqueles que se encontram na direção do movimento que está sendo realizado do nodo pai $parent(n)$ para o nodo atual n. Ainda nesse processo de poda, os casos de "vizinhos forçados"são tratados. Os vizinhos forçados são aqueles que se encontram em torno do nodo atual. Estes também são adjacentes a um nodo bloqueado que faça parte da vizinhança. Este vizinho não se encontra na direção do movimento que está sendo realizado. Porém, ele deve ser avaliado em função do bloqueio adjacente. No algoritmo JPS, nodos bloqueados podem significar um ponto de inflexão no caminho, indicando que os vizinhos forçados não podem ser desconsiderados na busca de caminhos.

Visto que a função de *jump* do algoritmo JPS realiza uma exploração recursiva em diferentes direções na busca por "*jump points*", em grandes ambientes no espaço 3D, os quais possuem muitos nodos a serem explorados, é possível que haja uma sobrecarga na pilha de recursão. Essa ocorrência prejudica a execução do algoritmo, podendo gerar falhas quando o algoritmo é executado. Para evitar tal sobrecarga, um valor limite é atribuído para a profundidade da recursão, o qual é verificado a cada novo passo na função *Jump*, ou seja, a cada nova chamada recursiva. Atingindo o valor definido, o último nodo é retornado como um *jump point*. Além de evitar a sobrecarga na pilha, essa técnica também melhora o desempenho do algoritmo.

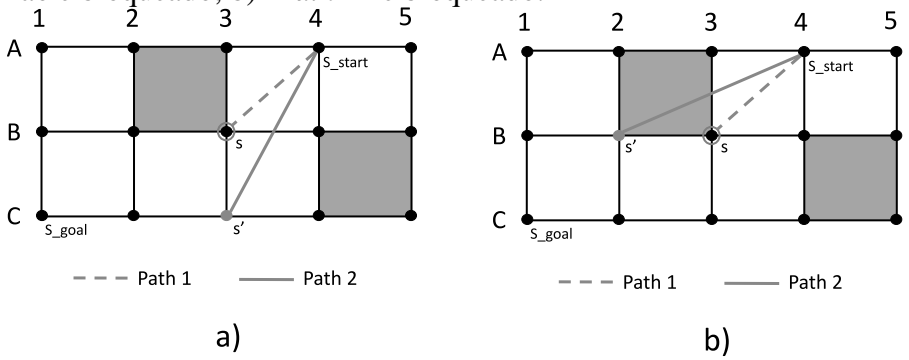
Neste trabalho, para definir o número limite para chamadas recursivas do algoritmo JPS, testes experimentais foram realizados. Inicialmente, foram calculadas quantas chamadas foram realizadas até ocorrer uma sobrecarga na pilha de recursão. O valor computado foi definido inicialmente como limite. Após, os valores definidos para limitar a recursão foram diminuídos, a fim de minimizar esses valores de forma que os resultados do caminho retornado pelo algoritmo JPS fossem mantidos (comprimento do caminho, nodos abertos, quantidade de nodos que compõem o caminho, entre outros) ao passo que o tempo de execução do algoritmo pudesse ser melhorado. Por fim, o valor definido com base nessa avaliação permitiu reduzir o tempo de execução do algoritmo em cerca de 60% em mapas de navegação 3D com maior número de obstáculos distribuídos nas diferentes regiões do mapa - neste trabalho, o mapa C (65% dos nodos bloqueados). Ainda, em mapas com menor número de obstáculos, ou seja, mais espaços abertos, a otimização do limite às chamadas recursivas reduziu o tempo de execução em quase 90%, com testes realizados nos mapas A (30% de nodos bloqueados) e B (45% de nodos bloqueados).

3.1.2 Theta* e Lazy Theta*

O algoritmo Theta* (NASH et al., 2007)(DANIEL et al., 2010) é uma variação do A* para realizar suavização do caminho enquanto a busca é executada. O algoritmo possibilita essa suavização ao expandir o caminho para o próximo nodo vizinho, quando verifica se é possível traçar uma rota do pai do nodo atual até o vizinho para onde algoritmo pretende expandir o caminho. Basicamente, o Theta* combina a ideia de busca de caminhos do A* em grafos de visibilidade com a ideia do A* em estruturas de grid.

Em contraste com o A*, ao atualizar o custo g e o pai de um vizinho visível não expandido, o Theta* considera duas possibilidades para a busca de caminho. A primeira opção de caminho é a computada pelo A*, que considera o caminho do nodo inicial até o nodo atual (n) verificado e a distância em linha reta para o vizinho (n') [= dist(n, n')] do nodo atual. A segunda opção de caminho considera o caminho do nodo inicial até o pai do nodo atual, além de verificar, usando a distância em linha reta, se é possível formar um caminho do pai para o vizinho visível [=dist($parent(n)$, n')]. Esta segunda opção é possível quando existe o campo de visão do pai do nodo atual para o nodo vizinho visível. Ter o campo de visão significa não existir algum nodo obstruído entre nodo observador (pai do nodo atual) e observado (vizinho visível), onde esse obstáculo impede a passagem do nodo pai ao vizinho. Na Figura 3.5, é possível visualizar esta ideia de seleção do caminho de acordo com o campo de visão (apresentada também no Algoritmo 1). Se não há obstrução, o caminho 2 é escolhido. Caso contrário, o caminho 1 é escolhido.

Figura 3.5: Comparação entre as duas opções de caminho que o Theta* pode executar. a) *"Path 2"* não é bloqueado; b) *"Path 2"* é bloqueado.



Fonte: Adaptado de (NASH et al., 2007).

O algoritmo Theta* é eficiente ao aplicar a ideia de suavização uma vez que já planeja o caminho de forma suavizada. Isso permite computar caminhos mais diretos e menores, sem que haja necessidade de pós-processamento. No entanto, uma desvantagem

deste algoritmo é que ele requer tempos de execução mais longos. Em ambientes 3D, os tempos de execução de algoritmos de busca de caminhos tornam-se ainda maiores, devido à maior complexidade do espaço de busca (mais nodos e necessidade de um maior tratamento de obstáculos).

Uma etapa que demanda tempo no algoritmo Theta* é a verificação de campo de visão. Esse é uma tarefa que ocorre para cada vizinho visível não expandido de cada nodo expandido. Em ambientes 2D, essa verificação ocorre para 8 vizinhos, enquanto a verificação passa a ser feita para 26 vizinhos em ambientes 3D, o que pode aumentar muito o tempo de computação do algoritmo. Além de haver muito mais linhas de visão por vértice expandido em grids cúbicos, o tempo de execução por verificação de campo de visão pode ser linear no número de nodos.

A fim de reduzir a grande frequência de verificações de campo de visão e otimizar o desempenho do algoritmo Theta*, (NASH; KOENIG; TOVEY, 2010) propõem Lazy Theta*. Esse algoritmo implementa uma variação do Theta* que utiliza uma "avaliação preguiçosa". A avaliação consiste em executar apenas uma verificação de campo de visão por vértice expandido. Enquanto o Theta* verifica se há linha de visão para cada vizinho n' de um nodo n, atualizando os custos de cada vizinho se necessário, o algoritmo Lazy Theta* não realiza a verificação na etapa de abertura dos vizinhos. Theta* atualiza o custo g e o nodo pai ($parent(n)$) de um vizinho visível não expandido n' de um nodo n no procedimento *ComputeCost*, onde considera o *Caminho 1* e o *Caminho 2*. Ele considera o Caminho 2 se n' e o $parent$ têm campo de visão. Caso contrário, o Caminho 1 é considerado.

Algorithm 1: Função *ComputeCost* do algoritmo Theta*

```
1  ComputeCost (n, n')
2      if LineOfSight(parent(n), n') then
3          /* Path 2 */
4          if g(parent(n)) + c(parent(n), n') < g(n') then
5              parent(n') = parent(n);
6              g(n') = g(parent(n)) + c(parent(n), n');
7          end
8      else
9          /* Path 1 */
10         if g(n) + c(n, n') < g(n') then
11             parent(n') = n;
12             g(n') = g(n) + c(n, n');
13         end
14     end
15 end
```

Função do algoritmo Theta* que seleciona o caminho a ser escolhido de acordo com obstrução ou não do campo de visão.

Data:

n: Current node.

n': Neighbour node.

De forma otimista e sem realizar uma verificação prévia, o algoritmo Lazy Theta* assume que não há obstrução do caminho entre $parent(n)$ e n', o sucessor de n. Assim, a verificação do campo de visão é postergada e somente o Caminho 2 é considerado (a função *ComputeCost* apresentada no Algoritmo 1, passa a ser definida pelo Algoritmo 2). Como tal suposição pode ter sido incorretamente estabelecida, o procedimento *SetVertex*, apresentado no Algoritmo 3, realiza uma verificação para validar a suposição inicial, imediatamente antes de expandir o nodo n'. Caso a suposição seja válida, o custo g e nodo pai não são alterados. Caso inválida, então o custo g e $parent(n')$ são atualizados de acordo com os princípios do Caminho 1. Quando essa situação ocorre, similar ao que ocorre quando uma vizinhança padrão de 26 vizinhos é usada na vizinhança 3^k, para diferentes valores de k, o nodo atual também possui em sua vizinhança o mesmo nodo vizinho que o expandiu. Ou seja, é considerado o caminho do ponto inicial s para cada vizinho vi-

sível expandido n'' de n' e de n'' a n' em linha reta, escolhendo o caminho mais curto. Sabemos que n tem pelo menos um vizinho visível expandido, pois n' foi adicionado à lista de nodos abertos quando Lazy Theta* expandiu tal vizinho. Portanto, para recalcular os custos e reeleger o nodo pai de acordo com a ideia do Caminho 1, esse nodo vizinho deve estar tanto na lista de vizinhança do nodo quanto na lista de nodos fechados, além de apresentar o menor custo g dentre os vizinhos que satisfizerem tais condições.

Algorithm 2: Função *ComputeCost* do algoritmo Lazy Theta*

```
1 ComputeCost (n, n')
2     /* Path 2 */
3     if g(parent(n)) + c(parent(n), n') < g(n') then
4         parent(n') = parent(n);
5         g(n') = g(parent(n)) + c(parent(n), n');
6     end
7 end
```

Função do algoritmo Lazy Theta* que computa custos considerando somente o caminho 2, assumindo de forma otimista que o campo de visão não seja obstruído.

Data:

n: Current node.

n': Neighbour node.

Algorithm 3: Função *SetVertex* do algoritmo Lazy Theta*

```
1 SetVertex (n)
2     if NOT LineOfSight(parent(n), n) then
3         /* Path 1 */
4         parent(n) = argmin_{n' ∈ neighbors_vis(n) ∩ closed} (g(n') + c(n', n));
5         g(n) = min_{n' ∈ neighbors_vis(n) ∩ closed} (g(n') + c(n', n));
6     end
7 end
```

Função do algoritmo Lazy Theta* que computa custos considerando somente o caminho 2, assumindo que o campo de visão não seja obstruído.

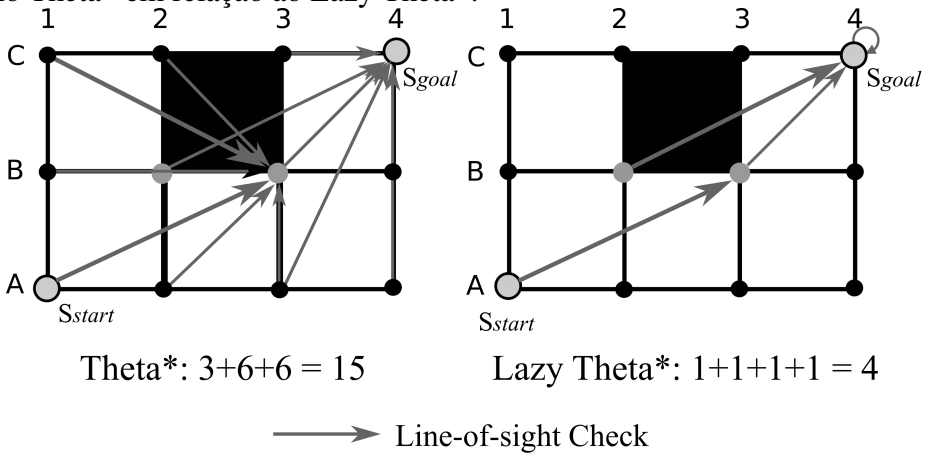
Data:

n: Current node.

O algoritmo Lazy Theta* expande mais nodos que o Theta*, visto que as suposi-

ções prévias de que o campo de visão não é obstruído podem fazer com que os valores g sejam mal informados, diferentemente do Theta*. O cálculo de um custo g para um caminho obstruído naturalmente vai ser diferente de um custo g para um caminho livre. Essa informação equivocada em relação a g (em função da suposição sobre a desobstrução do caminho) implica na ordenação da lista de nodos abertos e seleção do nodo de menor custo a ser expandido. Contudo, o algoritmo Lazy Theta* realiza um número muito menor de testes associados ao campo de visão, o que faz com que o tempo de execução deste algoritmo seja reduzido em relação ao tempo requerido pelo algorimto Theta* padrão. Tal como descrito em (NASH; KOENIG; TOVEY, 2010), o Lazy Theta* encontra caminhos mais rápido que Theta* em grids cúbicos, com menos verificações associadas ao campo de visão em uma ordem de magnitude e sem resultar em um aumento no comprimento do caminho.

Figura 3.6: Comparação da quantidade de verificações do campo de visão realizadas pelo algoritmo Theta* em relação ao Lazy Theta*.



Fonte: Adaptado de (NASH; KOENIG; TOVEY, 2010).

3.2 Trabalhos relacionados

Em sistemas de simulação e jogos digitais, agentes devem navegar com êxito em modelos virtuais de terrenos/mapas. Para isso, algoritmos de planejamento de caminhos de diferentes naturezas têm sido apresentados na literatura (ALGFOOR; SUNAR; KOLIVAND, 2015)(URAS; KOENIG, 2015)(HUNG; HE, 2016)(YANG et al., 2016). Estes algoritmos buscam rotas trafegáveis para agentes dados os pontos de origem e destino determinados no mapa. A busca de um caminho pode ser baseada em algum critério de prioridade tal como: menor custo, menor comprimento do caminho, melhor suavização,

entre outros.

Nesta seção são apresentados trabalhos que foram revisados e utilizados como embasamento para realização da pesquisa do presente trabalho. Como diferentes abordagens foram investigadas, os trabalhos relacionados estão divididos de acordo com os aspectos que são tratados em cada um deles (algoritmo de suavização de caminhos, soluções para problemas em ambientes 3D e uso de grandes vizinhanças).

3.2.1 Trabalhos relacionados com abordagem de suavização de caminhos

Além dos algoritmos que realizam suavização e de *any-angle* utilizados neste trabalho para fins de experimentos e avaliação do uso da vizinhança 3^k, que foram apresentados no início do capítulo de Referencial Teórico, outros trabalhos foram revisados durante a presente pesquisa. Nesta seção são apresentados alguns trabalhos que foram investigados e apresentaram propostas interessantes para algoritmos de busca de caminhos *any-angle*. Todavia as propostas apresentadas são voltadas a ambientes 2D. Visto que transformar tais implementações em versões de algoritmos que tratem problemas em ambientes 3D não é uma tarefa trivial e geram uma nova proposta de pesquisa por si só, os algoritmos *any-angle* apresentados nos próximos parágrafos não foram implementados neste trabalho.

Anya (HARABOR; GRASTIEN, 2013)(HARABOR et al., 2016) é um algoritmo que apresenta resultados ótimos. A principal ideia do Anya é realizar a busca em conjuntos contíguos de estados que formam intervalos. Dessa forma, os sucessores de um nodo n são identificados por intervalos de computação, que são construídos em tempo real. O algoritmo Dijkstra Contínuo (MITCHELL; MOUNT; PAPADIMITRIOU, 1987) emprega uma ideia semelhante. Porém, esse algoritmo exige uma etapa de pré-processamento anterior à resposta de qualquer consulta aos intervalos durante a busca e apenas a partir de um único ponto de partida fixo. Cada ponto p pertencente ao intervalo I deve ser visível pelo root r. Root r é o nodo selecionado no passo anterior. Assim, um nodo de busca (I, r) é composto pelo intervalo de sucessores I e o root r, tal que $r \notin I$. O nodo root r é escolhido em cada intervalo anterior, e ao final da busca de caminho conectará os pontos selecionados para compor o caminho ótimo. Ou seja, cada r de cada nodo de busca selecionado pelo algoritmo é um nodo que pertence ao caminho final. Dessa forma, r é sempre o ponto de inflexão mais recente. (HARABOR et al., 2016) amplia o trabalho para além da definição teórica apresentada em (HARABOR; GRASTIEN, 2013). Esse traba-

lho discute considerações práticas e teóricas em torno do algoritmo Anya, fornecendo argumentos para demonstrar a otimização e completude do algoritmo.

Apesar de ser um algoritmo que encontra caminhos ótimos, o Anya apresenta uma implementação complexa. Em estruturas de representação de ambientes 2D, os intervalos são projetados de forma unidimensional. Em estruturas de representação de ambientes 3D, a implementação do algoritmo torna-se ainda mais complexa, na qual pesquisas sobre esse problema ainda precisam ser desenvolvidas. Contudo, há indícios de que os intervalos tenham necessidade de serem projetados de forma bidimensional, havendo mais aspectos a serem tratados em relação à implementação voltada ao contexto 2D.

Outro algoritmo recente na literatura e que pode fornecer caminhos de qualquer ângulo é o Grafo de SubGoals (URAS; KOENIG; HERNÁNDEZ, 2012). Para isso, é realizada uma busca prévia em um subgrafo construído a partir da identificação de bordas de obstáculos presentes no ambiente. Este algoritmo também implementa uma forma de busca hierárquica, visto que realiza a busca em duas etapas: a) uma busca superficial no subgrafo construído a partir das bordas dos obstáculos existentes no ambiente, e b) uma busca profunda entre os subgoals previamente selecionados. Para que o caminho final seja de qualquer ângulo, a etapa de busca refinada entre os subgoals deve ser realizada por um algoritmo classificado como *any-angle*. Essa técnica de busca em duas etapas, caracterizada como busca hierárquica, pode acelerar o processo de planejamento de caminhos, uma vez que limita o espaço de busca.

A construção do grafo de SubGoals surge da ideia de que caminhos ótimos sempre tocam as bordas de obstáculos encontrados no grid. Isso segue a ideia de que o caminho ótimo é o mais direto possível e que os obstáculos ao longo do mapa são as únicas ressalvas dessa regra. Em (URAS; KOENIG; HERNÁNDEZ, 2012), as conexões do grafo de subgoals são definidas em termos de "vizinhos alcançáveis", denominados *"h-reacheable"*, uma vez que lidam apenas com um tipo de vizinhança (a vizinhança octil). De acordo com Hormazábal et al. (2017), considerando um mapa $M = (G, O)$, tem-se um grafo de subgoals $S = (S, E)$ de forma que S representa o conjunto de subgoals do mapa M e E representa o conjunto de conexões tal que $E = \{(s_1, s_2) \mid s_1$ e s_2 são diretamente acessíveis $(h_8 - reacheable)\}$. O custo de cada conexão é dado por $w(s_1, s_2) = h_8(s_1, s_2)$.

Os trabalhos investigados demonstram propostas interessantes, muitas delas recentes na literatura. Todavia, algumas das propostas apresentadas são voltadas somente a ambientes 2D. Visto que transformar tais implementações em versões de algoritmos que tratem problemas em ambientes 3D não é uma tarefa trivial, algumas delas precisam mais

Tabela 3.1: Tabela comparativa das propostas que abordam suavização de caminhos apresentadas na literatura.

Referência	Algoritmo	Topologia	Suavização	Vizinhança
(Harabor & Grastien, 2011)	JPS	Grid 2D	Sim	8 vizinhos
(Nobes *et al.*, 2022)	JPS	Grid 3D	Sim	26 vizinhos
(Daniel *et al.*, 2007)	Theta*	Grid 2D	Sim	8 vizinhos
(Nash & Tovey, 2010)	Lazy Theta*	Grids 2D e 3D	Sim	8 ou 26 vizinhos
(Harabor & Grastien, 2013)	Anya	Grid 2D	Sim	Intervalos
(Uras *et al.*, 2012)	SubGoals	Grafo	Sim	Depende do algoritmo

Fonte: Autora

estudos para tornarem-se viáveis em problemas tridimensionais, gerando novas propostas de pesquisa. Portanto, os algoritmos *any-angle* Anya e SubGoals, cujas versões para ambientes 3D não foram exploradas, não foram implementados neste trabalho.

3.2.2 Trabalhos relacionados direcionados a problemas em contextos 3D

Recentemente, diferentes abordagens para tratar problemas de busca de caminhos em cenários 3D, e em muitos casos problemas de suavização em busca de caminhos também, têm sido propostas na literatura (como, por exemplo, em (RANTTILA, 2019), (BELOV et al., 2020), (KRAFFT, 2021), (CARSTEN; FERGUSON; STENTZ, 2006), (MANDLOI; ARYA; VERMA, 2021), (FARIA et al., 2019), (HAN, 2019), (PANDIT, 2017), (KOOPMAN, 2016) e (BEIG et al., 2019)). Embora existam propostas apresentadas na literatura, não é trivial adaptar um algoritmo originalmente idealizado para executar em um ambiente 2D para então ser executado em cenários 3D. Por exemplo, (RANTTILA, 2019) apresenta um estudo sobre o algoritmo JPS e propõe uma versão deste algoritmo para problemas em ambientes 3D. O trabalho discute adaptações realizadas na implementação do JPS 3D, para que a execução deste algoritmo seja viável em ambiente 3D. Além de ampliar as condições de salto para identificar os *jump points*, considerando os 26 vizinhos de um nodo em ambiente tridimensional, ajustes no processo de poda e limitação da profundidade da recursão ao realizar os saltos mostraram ser eficazes

para manter um bom desempenho.

Em (NOBES et al., 2022), os próprios autores do JPS descrevem e formalizam a versão 3D do algoritmo JPS. O trabalho aborda a ampliação nas condições de poda e condições de salto quando o algoritmo é executado em ambientes tridimensionais, apresentando provas que fundamentam as definições realizadas. Apesar de não apresentar uma aplicação prática, o trabalho estende um algoritmo de uma versão executável em ambiente 2D para execução em ambientes 3D, o que neste caso não é uma tarefa trivial. As novas regras e condições que viabilizam a execução do JPS em ambientes 3D desmontram complexidade para o desenvolvimento do algoritmo. Ainda, testes de performance são apresentados, avaliando os aspectos de tempo e nodos abertos durante execução do algoritmo. A apresentação formal do JPS em 3D garante a otimalidade dos caminhos encontrados pelo algoritmo.

Em (BELOV et al., 2020), o problema da busca de caminhos em ambiente 3D é abordado em um sistema multiagente. Com significativa relevância industrial, o trabalho aborda o problema de roteamento de tubulações. Esse problema visa distribuir tubos de um local inicial para um local destino em um ambiente 3D conhecido de forma a evitar colisões entre os tubos. A implementação proposta utiliza duas etapas. A primeira localiza os equipamentos no ambiente, encontrando as posições 3D para todos os equipamentos necessários da planta em relação a um conjunto de restrições operacionais. Na sequência, calcula uma aproximação dos custos das tubulações utilizando medidas aproximadas, por exemplo, por meio da distância de Manhattan. A segunda resolve diretamente o problema de *Pipe Routing*, encontrando rotas 3D para as tubulações que conectam as pontas de entrada e saída dos equipamentos já alocados.

Em (Krafft, 2021), a dinamicidade de um ambiente 3D é abordada visando a aplicação do algoritmo de busca de caminhos no desenvolvimento de jogos de computador. O trabalho busca melhorar o desempenho de algoritmos conhecidos, como A*, JPS, Theta*, D* Lite (Koenig & Likhachev, 2002) e Moving Target D* Lite(Sun, Yeoh & Koenig, 2010), em ambientes 3D dinâmicos. Por exemplo, no algoritmo A* foi identificada a necessidade de otimização da pesquisa e exclusão de itens das listas de nodos abertos e fechados, bem como o cálculo dos valores f dos nodos. No algoritmo Theta*, o ponto crítico para o desempenho é o cálculo dos valores f dos nodos (que inclui o cálculo do valor g), e que por sua vez requer uma verificação de campo de visão. Logo, otimizar as verificações de campo de visão é a principal necessidade para o algoritmo Theta*. Para o outro algoritmo avaliado no trabalho, D* Lite, obter o próximo nodo e atualizar os valores

rhs, valores de custos antecipados de uma etapa com base nos valores g, dos predecessores do próximo nodo são os processos mais demorados. Além de otimizações nestes algoritmos, o trabalho também propõe substituições que podem ser melhor sucedidas para ambientes 3D. Por exemplo, substituir o uso do algoritmo A* pelo uso de JPS. Voltando as implementações para o contexto de ambientes dinâmicos, o objetivo é que o algoritmo usado seja capaz de atualizar o grafo de busca e recalcular um caminho mais curto dentro do limite de 16 ms. Este tempo corresponde à duração de um quadro em um jogo com 60 FPS. Dessa forma, tratando a dinamicidade do ambiente, o trabalho também propõe um algoritmo que combina Theta* com Moving Target D* Lite.

(CARSTEN; FERGUSON; STENTZ, 2006) propõem uma abordagem para calcular caminhos em tempo real na solução de problemas em cenários 3D. O trabalho expande o algoritmo Field D* (apresentado em (FERGUSON; STENTZ, 2005), (FERGUSON; STENTZ, 2006) e (FERGUSON; STENTZ, 2007)) de forma a utilizar uma interpolação em 3D. O algoritmo Field D* intercala a busca A* com suavização do caminho por meio da interpolação entre os valores g de vértices para calcular os valores g de outros locais que não sejam vértices. Isso permite que o nodo pai de um vértice seja atribuído como nodo pai de qualquer localização da estrutura, seja ela um vértice ou não, desde que esteja ao longo da linha que conecta o vértice expandido e os vizinhos deste vértice ao calcular os valores através da interpolação. A diferença entre o algoritmo voltado para ambientes 2D em relação ao algoritmo voltado para o cenário 3D, é que as faces de um nodo são utilizadas ao invés de utilizar arestas para calcular os custos. Em 2D, o custo de caminho de um nodo n é calculado olhando para as arestas adjacentes do nodo e usando interpolação para fornecer caminhos através de pontos arbitrários em tais arestas. No cenário 3D, as faces adjacentes de n são consideradas, e caminhos através dessas faces são planejados, usando interpolação para fornecer caminhos através de pontos arbitrários nestas faces. Partindo de uma versão de Field D* aplicada a veículos terrestres, a ideia do trabalho foi suprir a falta de uma versão deste algoritmo para atender às necessidades de veículos que se movem no espaço tais como, por exemplo, submarinos e veículos aéreos. Essa versão 3D do algoritmo Field D* utiliza uma técnica de aproximação para obter desempenho em tempo real, produzindo caminhos menos custosos e mais diretos.

Em (MANDLOI; ARYA; VERMA, 2021), um estudo comparativo é realizado entre diferentes algoritmos de planejamento de caminhos visando ambientes 3D para rotas de veículos aéreos não tripulados (VANTs). O cenário do problema abordado é de um único VANT trafegando em ambiente 3D conhecido e com obstáculos estáticos. Os algo-

ritmos selecionados para desenvolver o estudo são A*, Theta* e Lazy Theta*. A escolha do algoritmo A* foi relizada devido a otimização e simplicidade do algoritmo em termos de implementação. O algoritmo garante encontrar o caminho mais curto em grafos. Contudo, não garante encontrar o caminho mais curto em um ambiente contínuo real. Os algoritmos Theta* e Lazy Theta* foram selecionados para suprir essa deficiência, já que retornam caminhos mais curtos e suavizados. Contudo, esse benefício é acompanhado de um aumento no tempo de computação do caminho. Os experimentos foram realizados em diferentes cenários 3D, com diferentes dimensões e complexidades. As métricas utilizadas para fins de comparação foram tempo computacional e comprimento do caminho final. O estudo concluiu que o A* possui o melhor desempenho, enquanto o Theta* fornece o melhor caminho em termos de custo (neste caso, o comprimento do caminho final). Por fim, o algoritmo que apresentou melhor relação custo-benefício entres os aspectos avaliados, tempo computacional e comprimento do caminho final, foi o algoritmo Lazy Theta*.

Em (FARIA et al., 2019), a busca de caminhos em ambientes 3D também é voltada para veículos aéreos não tripulados (VANTs). O trabalho melhora a eficiência da implementação do Lazy Theta*, algoritmo que vem sendo aplicado com sucesso em competições com VANTs multirotores. Como estrutura de representação, foi utilizado um grid irregular na descrição do ambiente, visando aumentar o desempenho do algoritmo de planejamento de caminhos. Especificamente, é utilizada uma grade de resolução esparsa na forma de uma octree. Isso permitiu organizar as medidas espacialmente, mesclando voxels quando estão no mesmo estado (livre, ocupado ou desconhecido). Como os voxels são mesclados conforme o estado no qual se encontram, obter um número variável de vizinhos para cada voxel se torna uma consequência. Dessa forma, à medida que o número de voxels mesclados aumenta, o número de vizinhos aumenta exponencialmente. O número total de vizinhos irá depender da configuração do espaço, portanto os vizinhos devem ser calculados de forma flexível, explorando a estrutura multi-resolução da octree. Essa representação é adequada para armazenar informações sobre grandes cenários, visto que requer pouca memória. Ainda, para evitar obstáculos, o conceito de corredor de voo foi introduzido para detectar obstáculos que estão ao redor da trajetória. Em resumo, o trabalho apresenta um algoritmo de planejamento de caminhos de qualquer ângulo que gera rotas tridimensionais com um corredor de voo livre de obstáculos em torno do trajeto, implementado sobre malhas esparsas. Além disso, o trabalho apresenta o refinamento da geração de vizinhança levando em conta a natureza multi-resolução da octree. O trabalho

é desenvolvido com foco em uma única aplicação, não generalizando a solução. Apesar de utilizar uma estrutura de representação mais complexa, porém que economiza memória, o trabalho explora uma vizinhança flexível que a estrutura oferece, formada a partir da configuração do espaço. Por exemplo, na base da resolução da árvore, um voxel com o tamanho da resolução tem 6 vizinhos. Subindo três níveis, a vizinhança de um voxel com apenas quatro vezes o tamanho da resolução do mapa acaba aumentando para 384 vizinhos. Salvos os casos em que trabalha-se com a base da resolução da árvore, o trabalho utiliza uma forma de ampliação da vizinhança. Todavia essa ampliação será uma consequência do espaço e distribuição dos estados no mapa, não sendo gerada de forma planejada a partir de uma técnica de vizinhança. Visto que a estrutura utilizada é irregular, para aplicar técnicas de vizinhança ampliada estratificada maiores estudos seriam necessários.

(HAN, 2019) propõe um método que diminui a complexidade do planejamento eficiente de caminho em 3D. O método COSPS (*Critical Obstacles and Surrounding Point Set*) determina um subconjunto de obstáculos em termos da importância destes para o planejamento de um caminho. O trabalho também encontra um subconjunto de pontos do grid que podem cercar totalmente esses obstáculos em 3D. O intuito é abranger somente pontos em áreas de interesse ao planejamento de caminhos, descartando regiões desnecessárias do mapa. Assim, o grafo é construído com um número menor de pontos, ao invés do número total de pontos que compõem a estrutura de representação do espaço 3D. Essa abordagem reduz o número de obstáculos e pontos, ao passo em que permite que o caminho seja gerado de forma eficiente. Nos experimentos, sobre o grafo gerado a partir do método proposto COSPS, o algoritmo de busca de caminhos Dijkstra foi utilizado. Tratando a grande complexidade de executar algoritmos em ambiente 3D, o método proposto demonstrou ser efetivo em encontrar caminhos viáveis para aplicações no cenário 3D. Todavia, o trabalho não esclarece o custo computacional da geração do grafo, construído de forma online após entrada de informações como conjunto de obstáculos, tamanho do grid (largura), ponto inicial e ponto destino. Isso indica que apenas o tempo computacional para realizar o planejamento de caminho entre os dois pontos informados sobre o grafo é avaliado.

Dois algoritmos de busca em ambiente tridimensional são propostos em (PANDIT, 2017). Um deles é um algoritmo de otimização de espaço de busca dinâmico. Este algoritmo pode ser aplicado para tesselar de forma desigual o espaço de busca 3D. O algoritmo pode ser usado com algoritmos populares de planejamento de caminhos usados

em jogos 3D. A técnica, novamente, auxilia na melhora de desempenho dos algoritmos de busca em ambientes 3D. O segundo algoritmo utiliza *ray-casting* ou campo de visão para gerar um caminho viável em tempo de execução, sem exigir a divisão do espaço de busca em uma grade 3D. Enquanto um possui a vantagem de poder ser implementado em conjunto com outros algoritmos, o outro possui como vantagem uma forma eficiente de evitar obstruções, sem exigir qualquer grade de pesquisa, visto que ele usa dados do campo de visão (ray-cast) para evitar colisões. Dessa forma, é possível obter caminhos mais diretos que remetam à ideia de suavização no espaço 3D.

(KOOPMAN, 2016) descreve um modelo de grid regular onde cada nodo ou voxel tem a forma de um cubo. A estrutura de representação pode ser constituída por uma estrutura de dados que suporte o armazenamento de voxels: elementos de coordenadas x, y e z. O trabalho é voltado para algoritmos que atuam em espaços fechados, ou seja, locais internos onde agentes navegam no interior desses espaços. O algoritmo deriva um subgrafo a partir da estrutura dada como entrada, permitindo realizar uma busca hierárquica. A construção desse subgrafo é realizada de forma diferente para cada tipo de agente. Baseado na forma de locomoção de cada agente (agente caminhável, dirigível ou voador, casos de drones), o grafo é construído a partir do que o trabalho nomeia como classes semânticas, as quais ajudam a determinar o espaço navegável. Estas classes são parâmetros para definir onde o movimento está ocorrendo, sendo elas: piso, escadas e obstáculo. Os rótulos que correspondem a essas classes são atribuídos para os voxels e nodos gerados. Visto que cada grafo é desenvolvido especificamente para cada tipo de agente, isso torna o método de planejamento de caminho adequado apenas para um único agente em um ambiente estático. Apesar de a técnica utilizada no trabalho aprimorar a performance do algoritmo de busca (o algoritmo A* é utilizado) em termos de tempo, a construção de cada subgrafo é um processo elaborado. Realizá-la para cada ator em ambientes estáticos pode ser uma desvantagem.

(BEIG et al., 2019) utilizam busca de caminhos em ambiente 3D apresentando a ideia de um renderizador de áudio espacial baseado em GPU para a engine de jogos Unity3D (UNITY-TECHNOLOGIES, 2022). O trabalho explora diferentes ideias de pesquisa em ambientes 3D. Uma delas é a aplicação de busca de caminhos voltada para áudio espacial ao invés de movimentação de agentes. Outra diferença é apresentar uma implementação baseada em GPU ao invés de usar CPU. A proposta é um método de busca de caminhos que reduz a oclusão e a obstrução, aumentando o desempenho no jogo. O trabalho desenvolve uma solução baseada na análise de áudio espacial para fornecer uma

aproximação perceptiva de pistas espaciais do ambiente, incluindo oclusão, reflexão e difração.

Tabela 3.2: Tabela comparativa das propostas apresentadas na literatura.

Referência	Algoritmo	Topologia	Suavização	Vizinhança
(Nobes *et al.*, 2022)	JPS	Grid 3D	Sim	26 vizinhos
(Ranttila, 2019)	JPS	Grid 3D	Sim	26 vizinhos
(Belov *et al.*, 2020)	PBS	Tree	Não	-
(Krafft, 2021)	A*, JPS, Theta*, D* Lite	Grafo	Sim	26 vizinhos
(Carsten *et al.*, 2006)	Field D*	Grid 3D	Não	6 vizinhos
(Mandloi *et al.*, 2021)	A*, Theta*, Lazy Theta*	Grid 3D	Sim	26 vizinhos
(Faria *et al.*, 2019)	Lazy Theta*	Octree	Sim	Vizinhança esparsa
(Han, 2019)	COSPS	Grid 3D	Não	26 vizinhos
(Pandit, 2017)	USQ, VCS	-	Sim	-
(Koopman, 2016)	A*	Grid 3D	Não	26 vizinhos
(Beig *et al.*, 2019)	G-SpAR	NavMesh	-	-

Fonte: Autora

Conforme argumentado na introdução do trabalho e observado nos trabalhos relacionados apresentados, pesquisas recentes têm sido direcionadas aos problemas em ambientes 3D, abordando diferentes contextos. Como pôde ser observado, a maioria das propostas voltadas ao planejamento de caminhos em ambientes 3D busca diminuir a complexidade do problema a fim de ganhar desempenho e melhorar a qualidade dos caminhos retornados. Essa característica pôde ser percebida em (KRAFFT, 2021) - identificando gargalos na execução dos algoritmos e utilizando recursos da Unity a fim de ganhar desempenho -, (HAN, 2019) - com a redução de um grafo abrangendo somente áreas de interesse -, em (BELOV et al., 2020) - com técnicas para solucionar problemas de *Pipe Routing (PR)* -, em (PANDIT, 2017) - utilizando tesselagem do espaço de busca e uso de *ray-casting* para simplificar o problema de busca e ganhar desempenho, dispensando o uso de uma estrutura de representação no segundo caso -, e em (BEIG et al., 2019) com uso

de parte do processamento em GPU para ganhar desempenho. Ainda, é possível verificar que alguns direcionam estudos para implementar versões 3D de algoritmos disponíveis na literatura somente em implementações 2D, suprindo a carência de tais algoritmos na resolução de problemas em ambientes tridimensionais. A exemplo disso, verificam-se os trabalhos de (NOBES et al., 2022), (RANTTILA, 2019) e (CARSTEN; FERGUSON; STENTZ, 2006), que restringem suas pesquisas à adaptação dos algoritmos JPS e Field D* para ambientes 3D, respectivamente.

Alguns dos trabalhos também foram desenvolvidos tendo em vista somente uma aplicação. É o caso de (FARIA et al., 2019) e (MANDLOI; ARYA; VERMA, 2021), direcionando pesquisas a aplicações de busca de caminhos para VANTs, e (KOOPMAN, 2016), que trata problemas de busca de caminhos de acordo com o agente informado por parâmetro. Verificou-se interesse por parte dos pesquisadores no uso do algoritmo Lazy Theta* para solucionar problemas de VANTs, visto a qualidade dos caminhos retornados por esse algoritmo devido à suavização que o algoritmo promove. No presente trabalho, o algoritmo Lazy Theta* também foi selecionado para ser avaliado nos experimentos devido à sua característica de realizar suavização e crescente uso em aplicações 3D. Em vários trabalhos a técnica de suavização foi requisitada, como em (KRAFFT, 2021), (MANDLOI; ARYA; VERMA, 2021), (FARIA et al., 2019) e (PANDIT, 2017). Em (CARSTEN; FERGUSON; STENTZ, 2006), o uso de interpolação pelo algoritmo Field D* também promove suavização, pois não limita o caminho aos vértices da estrutura.

Apesar de haver variação nos contextos dos trabalhos voltados a ambientes 3D apresentados, nenhum trabalho direcionado à ampliação da vizinhança para compor implementações de algoritmos de busca em ambientes 3D foi encontrado. Tal como investigado neste trabalho, uma alternativa para simplificar o problema da complexidade dos algoritmos em 3D é voltada à exploração de uma vizinhança ampliada. Em geral, técnicas voltadas à ampliação da vizinhança possibilitam obter caminhos de melhor qualidade associadas a implementações simples de algoritmos de planejamento de caminhos. Em (FARIA et al., 2019) o uso de octree pode promover ampliação da vizinhança. Como a seleção dos vizinhos é baseada na configuração do espaço 3D, de acordo com a resolução da estrutura a quantidade de vizinhos pode aumentar. Contudo, a vizinhança padrão na base da resolução da árvore é de 6 vizinhos. Neste caso, o aumento da vizinhança se torna uma consequencia em função da resolução da estrutura, não ocorrendo de forma planejada. Dessa forma, nota-se que o presente trabalho é um dos pioneiros nas pesquisas em torno de ampliação da vizinhança para algoritmos de busca de caminhos em ambientes

3D. Em conjunto à expansão da vizinhança, o trabalho também explora o uso de algoritmos *any-angle*, tendo em vista a suavização dos caminhos a fim de obter soluções de melhor qualidade. Diferentemente dos trabalhos revisados, as métricas de avaliação neste trabalhos não são restritas ao tempo de execução e ao comprimento do caminho. Aspectos de suavização, avaliada por meio do número de nodos no caminho final, e de consumo de memória, avaliado por meio do número de nodos abertos, também são avaliados nos experimentos. Refinando a avaliação dos dados, este trabalho também utiliza métodos estatísticos para análise dos resultados experimentais.

3.2.3 Trabalhos relacionados sobre uso de grandes vizinhanças

A maioria dos trabalhos voltados ao planejamento de caminhos em ambientes 3D busca diminuir a complexidade do problema a fim de ganhar desempenho e melhorar a qualidade dos caminhos retornados. Tal como investigado neste trabalho, uma alternativa para simplificar tal problema é voltada para a exploração de uma vizinhança expandida. Em geral, técnicas voltadas à exploração da vizinhança possibilitam obter caminhos de melhor qualidade associadas a implementações simples de algoritmos de planejamento de caminhos.

(RIVERA; HERNÁNDEZ; BAIER, 2017) propõem a exploração de uma vizinhança ampliada, por meio da chamada *"Vizinhança* 2^k*"*. Essa solução transcende a limitação de vizinhança convencional, que considera somente 4 ou 8 vizinhos. O trabalho também propõe uma heurística a ser utilizada juntamente com a vizinhança, a qual é idealmente aplicável em estruturas livres de obstáculos. Os autores exploram ambientes 2D, construindo implementações básicas (como A* e A* canônico, por exemplo) competitivas com algoritmos *any-angle*, cujas implementações geralmente são mais complexas. Em (HORMAZÁBAL et al., 2017), a ideia da vizinhança é aplicada a grafos de subgoals. O grafo é pré-processado (o que normalmente apresenta um custo adicional de tempo de pré-processamento e armazenamento do grafo em memória) e, apesar de ser originalmente voltado para redes conectadas, é adaptado para uso de tal vizinhança. Ainda, o trabalho apresenta uma heurística iterativa 2^k, linear em k. Em (RIVERA et al., 2020), esse tópico de pesquisa é estendido, testando a vizinhança 2^k com o emprego de diferentes algoritmos, além de apresentar uma definição formal deste conceito de vizinhança. Neste trabalho, os vizinhos não são definidos pelas coordenadas centrais dos nodos, pois são definidos pelos vértices que definem os limites do nodo.

(SHAW, 1998) introduz o chamado *Large Neighborhood Search* (LNS), um método de busca local para problemas de otimização de solução, utilizado de forma aplicada para resolver problemas de roteirização de veículos. Um problema de roteirização de veículos (VRP) consiste em definir rotas para um conjunto de veículos (inicialmente estacionados em um ponto central do grafo) a fim de visitar um conjunto de clientes (dispostos no grafo), considerando as restrições dos veículos, clientes, motoristas, entre outros. O objetivo é produzir um plano de roteirização de baixo custo, partindo do ponto central para os pontos de demanda, de modo que a soma dos comprimentos das rotas seja minimizada. Uma maneira de aplicar LNS a um VRP é definindo um movimento para ser a remoção e reinserção de um conjunto I de visitas aos clientes. Uma medida de "relacionamento"entre as visitas de clientes é definida e utilizada como base para escolher o conjunto I em cada etapa. A heurística LNS pertence à classe de heurísticas conhecidas como algoritmos de busca de vizinhança de grande escala. O LNS explora uma grande vizinhança da solução atual, selecionando um número de visitas de clientes "relacionados"para remover do conjunto de rotas planejadas e reinserindo essas visitas usando uma pesquisa em árvore baseada em restrições. O processo de reinserção usa heurística e propagação de restrições: o LNS faz movimentos como a pesquisa local, mas usa uma pesquisa baseada em árvore com propagação de restrição para avaliar o custo e a legalidade da mudança. O LNS apresentou bom desempenho médio, além de produzir novas e melhores soluções para os problemas utilizados nos experimentos realizados. (PISINGER; ROPKE, 2010) revisa a heurística LNS e suas extensões, além de explicar brevemente os conceitos centrais em VLSN (*Very Large Scale Neighborhood Search*)(AHUJA; ORLIN; SHARMA, 1998). Segundo este trabalho, uma solução inicial é gradualmente melhorada no LNS, alternando entre a destruição e reparação da solução. Todos os algoritmos VLSN são baseados na ideia de que realizar a pesquisa em uma grande vizinhança resulta em encontrar ótimos locais de alta qualidade. Portanto, algoritmos VLSN podem retornar soluções de melhor qualidade em termos de custo da solução.

(DEMIR; BEKTAŞ; LAPORTE, 2012) propõem uma extensão do algoritmo Adaptive Large Neighborhood Search (ALNS) para o Problema de Roteamento de Poluição (PRP). PRP é uma extensão de um problema clássico de roteamento de veículos com janelas de tempo (VRPTW). PRP consiste em roteirizar um número de veículos para atender a um conjunto de clientes dentro de janelas de tempo pré-definidas e determinar as velocidades de tais veículos em cada segmento de rota. Isso implica em minimizar uma função que inclui custos de combustível, emissões e motorista. O algoritmo integra o es-

quema clássico de ALNS (PISINGER; ROPKE, 2007)(ROPKE; PISINGER, 2006) com um algoritmo especializado de otimização de velocidade. Esse algoritmo de otimização calcula velocidades ótimas em um determinado caminho para minimizar o consumo de combustível.

(LI et al., 2021) abordam o uso de LNS em problemas de sistemas multiagentes. A seleção de bons vizinhos a serem explorados, de acordo com o critério da vizinhança - como no caso de LNS adaptativo as vizinhanças devem ser ortogonais, no sentido de serem formadas muito diferentes, para serem bem sucedidas -, é uma etapa fundamental para determinar se o algoritmo LNS deve ser bem sucedido. Tendo um parâmetro N como valor restringente da vizinhança, o trabalho apresenta três diferentes cálculos de heurísticas para definição da vizinhança a ser utilizada no problema de busca de caminhos multiagente. Uma das heurísticas é baseada nos agentes do sistema e seus caminhos, buscando o menor caminho sem obstrução e selecionando o respectivo agente para formular a heurística. A segunda é baseada na topologia do mapa utilizado, na qual os vértices de interesse são aqueles frequentados por agentes mais de uma vez. Uma vez que se tenha um vértice com grau maior que dois, uma ordem diferente dos agentes para atravessar um vértice de interseção pode levar a soluções de qualidades diferentes. Esta vizinhança é ortogonal a partir da vizinhança baseada no agente. A terceira heurística consiste em selecionar N vizinhos de forma aleatória. Essa forma de definição aleatória de vizinhança é uma boa linha de base usada em muitas abordagens LNS (DEMIR; BEKTAŞ; LAPORTE, 2012)(SONG et al., 2020). Embora seja uma abordagem aparentemente simples, é muito eficaz para *instâncias congestionadas* (instâncias com uma alta proporção de número de agentes sobre o número de vértices no grafo). (LI et al., 2022) estende o trabalho (LI et al., 2021) propondo o algoritmo MAPF-LNS2, que pode encontrar de forma eficiente uma solução (ao invés de melhorar uma dada solução) para uma instância de busca de caminho multiagente.

Alguns trabalhos direcionados à investigação e ao uso de grandes vizinhanças são propostos na literatura, porém muitos dos trabalhos encontrados são voltados a problemas de otimização. Abordagens LNS são exemplos disso e podem ser a aplicadas em diferentes contextos, porém não é uma técnica específica que possa ser acoplada a um algoritmo de busca de caminhos. A proposta de *Vizinhança* 2^k demonstra ser uma técnica interessante a ser aplicada em algoritmos de busca caminhos, porém é voltada somente a ambientes e algoritmos 2D. Para ambientes 3D, nenhuma abordagem foi proposta, oportunizando novas pesquisas nesse sentido.

4 BUSCA DE CAMINHOS EM AMBIENTE 3D UTILIZANDO A *VIZINHANÇA* 3^K

O planejamento de caminhos em ambientes 3D é uma tarefa fundamental para a resolução de problemas do mundo-real. Contudo, esse planejamento é mais complexo que a busca de caminhos em ambientes 2D. Além de ter de tratar um espaço de busca ampliado, o acréscimo de uma nova dimensão no processo de busca requer o desenvolvimento de algoritmos adaptados para 3D. Em termos práticos, esses algoritmos possuem um maior número de condições e verificações, visto que ampliam as vizinhanças dos nodos e direções de busca.

Ao acrescenter uma terceira dimensão no processo de busca de caminhos, também é possível analisar os impactos nos caminhos computados. Segundo Nash and Koenig (2013), A* pode encontrar caminhos mais curtos em grids, uma vez que sejam caminhos mais curtos restritos à estrutura dos nodos do grid. Porém, um caminho mais curto que seja restrito à estrutura dos nodos do grid pode não ser o mais curto possível. A* em grids pode encontrar caminhos que são no máximo cerca de 8% mais longos que os caminhos mais curtos em grids 2D, considerando expansão de 8 vizinhos com vértices posicionados nos cantos das células do grid. Em um problema no cenário 3D, cujo ambiente seja representado por um grid cúbico, os caminhos resultantes são pelo menos 13% mais longos que os caminhos mais curtos computados em grids 3D (NASH; KOENIG, 2013). Nesses espaços de busca ampliados, cada nodo possui 26 vizinhos, e os vértices dos cantos das células do grid são considerados, ao invés de considerar os vértices centrais de cada nodo. Quando executado sobre grafos de visibilidade((LEE, 1978), (LOZANO-PÉREZ; WESLEY, 1979)), o algoritmo A* encontra caminhos mais curtos em mapas 2D com obstáculos poligonais. Contudo, não há garantia de encontrar caminhos mais curtos em mapas 3D com obstáculos poliédricos (CHOSET et al., 2005). Além de impactar no comprimento dos caminhos resultantes, a mudança de 2D para o espaço 3D também impacta de forma significativa nos tempos de execução dos algoritmos de busca. Enquanto em um mapa 2D com obstáculos poligonais os caminhos mais curtos podem ser encontrados em tempos polinomiais, o tempo para encontrar esses caminhos é considerado NP-difícil em mapas 3D com obstáculos poliédricos (CANNY; REIF, 1987).

Para o ambiente 3D, certos algoritmos de busca de caminhos também encontram desafios de implementação consideráveis. Para o algoritmo Lazy Theta* (NASH; KOENIG; TOVEY, 2010), por exemplo, visto que ele é adaptado para funcionar em grids

cúbicos, não é necessário realizar alterações na implementação para execução no espaço 3D. Por outro lado, o algoritmo JPS (HARABOR; GRASTIEN, 2011) requer ajustes, além de otimizações para obter um melhor desempenho. Além de ajustes relacionados à necessidade de implementar mais testes condicionais devido à ampliação das direções de movimento, técnicas de otimização, como limitar as chamadas recursivas da função *jump*, podem ser necessárias para viabilizar a execução dos algoritmos na resolução de diferentes problemas de aplicação 3D. Essa necessidade deve-se ao fato de as tarefas de busca em ambientes 3D terem um impacto maior em tempo e consumo de memória. Os algoritmos *any-angle*, conforme discutido em (NASH; KOENIG, 2013) e (URAS; KOENIG, 2015), podem computar caminhos verdadeiramente mais curtos - apesar de não terem essa garantia (NASH; KOENIG, 2013). Porém, esses algoritmos exigem maior esforço computacional especialmente no espaço 3D. Dessa forma, desenvolver técnicas como, por exemplo, ampliação ou melhor exploração da vizinhança usada pelo algoritmo de busca torna-se uma proposta de pesquisa relevante.

4.1 Ampliando a exploração da vizinhança

(RIVERA; HERNÁNDEZ; BAIER, 2017) propõem a ampliação na exploração de vizinhos por meio de uma expansão 2^k vizinhos. Em particular, essa proposta foi demonstrada em grids regulares representando ambientes 2D. A expansão foi implementada com o uso de coordenadas dos centros dos nodos (Figura 4.1). Em (RIVERA et al., 2020), os autores expandem o trabalho mencionado, permitindo atingir os vértices nos cantos dos nodos, ao invés de ter como objetivo o ponto central de cada nodo. Maior realismo no planejamento de rotas é alcançado com essa proposta, o que fornece maior benefício em caso de nodos maiores - onde existe pouco refinamento da estrutura de representação do ambiente/mapa. Dessa forma, as possibilidades de direção de movimentação são ampliadas, e o uso de vizinhança 2^k permite que o algoritmo de planejamento de caminhos apresente comportamento análogo a um algoritmo *any-angle*.

A vizinhança 2^k pode ser definida em termos de séries (Q_0, Q_1, ...), as quais são compostas pelos movimentos a serem realizados no primeiro quadrante, definidos em Q_i. Para calcular os pontos a serem expandidos, somas de vetores são realizadas, partindo dos movimentos ortogonais iniciais (1, 0) e (0, 1). Neste caso, o primeiro elemento da série contém os movimentos *4-neighborhood* no primeiro quadrante (quadrante cujo sentido é positivo para todos os eixos). As séries seguintes são compostas pela inserção da soma

de elementos consecutivos entre cada elemento que representam as parcelas da soma. Ou seja, para os elemenos consecutivos a_i e a_{i+1} (que são as parcelas da soma), a soma de tais elementos a_i + a_{i+1} é inserida entre estes dois vetores na série. Por exemplo, os primeiros três elementos de uma série Q são:

$$Q_0 = <(1, 0), (0, 1)>$$

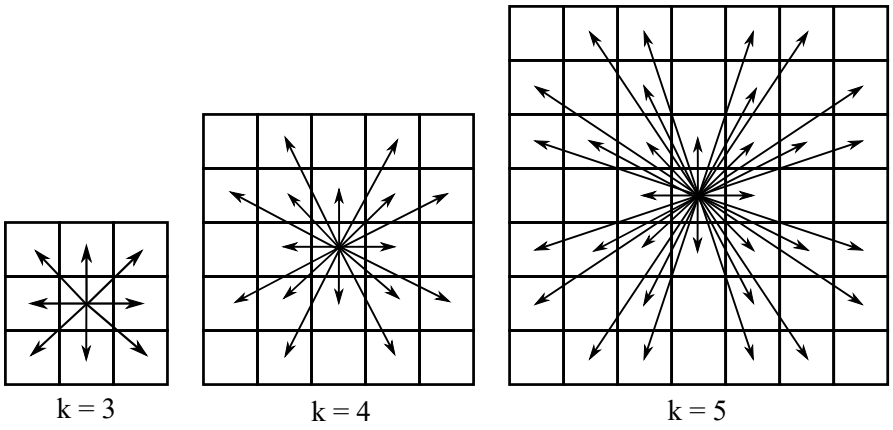
$$Q_1 = <(1, 0), (1, 1), (0, 1)>$$

$$Q_2 = <(1, 0), (2, 1), (1, 1), (1, 2), (0, 1)>$$

Cada movimento descrito para o primeiro quadrante é mapeado para os demais quadrantes, possibilitando executar um movimento em diferentes direções. Para os movimentos ortogonais, a direção oposta é mapeada, no sentido negativo, gerando os vetores (-1, 0) e (0, -1). Para cada movimento não ortogonal, três novos movimento são gerados, correspondendo à troca de sinais dos elementos x e y. Assim, é possível descrever o passo de mapeamento para as demais direções da seguinte forma:

$$(kx, k'y) \mid (x, y) \in Q_i \; e \; k, k' \in \{-1, 1\} \tag{4.1}$$

Figura 4.1: Grids cujos nodos apresentam 8, 16 e 32 vizinhos, respectivamente.



Fonte: Adaptado de (RIVERA; HERNÁNDEZ; BAIER, 2017).

A qualidade do caminho resultante de algoritmos utilizando exploração 2^k vizinhos pode ser comparada a qualidade dos caminhos resultantes de algoritmos *any-angle*, os quais buscam maior realismo ao planejar rotas. (RIVERA; HERNÁNDEZ; BAIER, 2017) e (RIVERA et al., 2020) descrevem a expansão de vizinhança 2^k e o desenvolvimento de uma função de distância para 2^k. Essa função pode ser utilizada como heurística

admissível adaptada à vizinhança. Tal função heurística foi desenvolvida considerando ser aplicável idealmente em grids livres de obstáculos. Dessa forma, como o próprio trabalho sugere, espaços de pesquisa oportunizando novas investigações e trabalhos voltados a ambientes 3D e tratamento em ambientes com obstáculos demonstram potencial.

4.2 Ampliando a exploração da vizinhança em ambientes 3D: Expansão 3^k vizinhos

A expansão 2^k vizinhos foi investigada somente em grids regulares bidimensionais (Rivera *et al.*, 2017)(Rivera *et al.*, 2020), o que é relevante para a execução de busca de caminhos em ambientes 2D. Em contraste, a pesquisa desenvolvida neste trabalho focaliza a exploração de vizinhos em estruturas regulares usadas na representação de ambientes 3D. Similar a expansão 2^k vizinhos, a expansão proposta, denominada 3^k vizinhos, ocorre de acordo com a geração sucessiva de vetores cujas coordenadas indicam o centro dos nodos. Enquanto a expansão 2^k vizinhos requer como parâmetro k um valor mínimo de k = 2, a expansão 3^k explora um valor mínimo de k = 3 para o parâmetro k. O valor mínimo 3 atribuído à variável k deve-se ao fato de fornecer uma vizinhança de 26 vizinhos, valor de vizinhança padrão em grids cúbicos quando movimentos diagonais são inclusos.

A distribuição de pontos, que representam os vizinhos, no plano ocorre somente pela soma de dois vetores consecutivos. De forma diferente, a geração de pontos no espaço é realizada tanto a partir da soma de dois vetores quanto a partir da soma de três vetores. Estes são denominados vetores geradores. No plano, o vetor resultante em uma série é somado a um vetor base: vetor inicial com variação de apenas um eixo. No espaço, cada vetor resultante é somado a outros dois vetores base, onde alguns vetores são somados aos pares. Os vetores somados aos pares permitem computar movimentos em diagonais envolvendo dois eixos. Dessa forma, a geração dos vizinhos ocorre de modo que a disposição desses vizinhos seja equilibrada, formando uma estrutura espacial bem distribuída. Essa distribuição espacial remete à distribuição gerada ao utilizar a técnica de expansão da vizinhança usada em ambientes 2D.

Similar a versão 2^k, para cada movimento ortogonal (1, 0, 0), (0, 1, 0) e (0, 0, 1), os vetores para movimento em direção oposta são gerados. Por exemplo, os vetores (-1, 0, 0), (0, -1, 0) e (0, 0, -1). Para cada vetor não ortogonal gerado pela soma sucessiva entre vetores, 7 novos movimentos representando as demais direções são mapeados no espaço. Além das orientações cardeais (vertical e horizontal), existe a necessidade de também trabalhar com a profundidade. Assim, é possível compreender tais direções de

movimento a partir da ideia de um cubo seccionado em 8 regiões. Neste cubo, a seção cujos elementos são todos positivos (onde as coordenadas x, y e z sempre são positivas) é tomada como primeiro octante ou octante positivo. As demais direções são mapeadas da seguinte forma:

$$(kx, k'y, k''z) \mid (x, y, z) \in Q_i \; e \; k, k', k'' \in \{-1, 1\} \tag{4.2}$$

Neste trabalho, a vizinhança 3^k é denotada por N_{3^k}. Definimos N_{3^k} para todo $k \geq 3$ em termos de séries Q_0, Q_1, ..., onde Q_i é a sequência que possui os movimentos referentes ao primeiro octante. Para definir os movimentos do primeiro quadrante e posteriormente mapear esses movimentos para os demais quadrantes, iniciamos com a soma dos três vetores base (ortogonais): $vI_1 = (1, 0, 0)$, $vI_2 = (0, 1, 0)$ e $vI_3 = (0, 0, 1)$. Para melhor execução do algoritmo, esses vetores ortogonais são os primeiros elementos a serem adicionados ao que chamamos de vetores intermediários, abordado nas primeiras linhas do código apresentado no Algoritmo 4. Esses vetores iniciais geram um vetor resultante $vR = (1, 1, 1)$, como mostrado na linha 9 do Algoritmo 4. Neste ponto do algoritmo sempre ocorre a soma que resulta em um movimento envolvendo as coordenadas dos três eixos. Os vetores geradores são somados par a par, formando os *vetores intermediários*. Os vetores que somam coordenadas que envolvem variações em dois eixos permitem que a vizinhança não somente seja expandida para diagonais cujos 3 eixos estejam variando em um movimento. Dessa forma, esses vetores também permitem a expansão de vizinhos em diagonais de dois eixos, ampliando a variação dos ângulos de movimentação.

Seguindo os vetores ortogonais préviamente adicionados aos vetores intermediários, as somas de tais elementos são adicionadas: $vI_4 = vI_1 + vI_2$, $vI_5 = vI_1 + vI_3$ e $vI_6 = vI_2 + vI_3$. Essas somas dos vetores intermediários, as quais somam variações de duas coordenadas e resultam em movimentos de dois eixos, são apresentas nas linhas 13-15 do Algoritmo 4. O vetor resultante vR é então somado a dois dos vetores geradores, ou seja, a cada um dos vetores intermediários recém calculados. Isso resulta em mais três somas que utilizam os vetores par a par: $v1 = vR + vI_4$, $v2 = vR + vI_5$ e $v3 = vR + vI_6$, processo que ocorre no laço da linha 18 à linha 20 no Algoritmo 4.

Algorithm 4: Função *Find3kNeighbors* para expansão da vizinhança

```
1  Find3kNeighbors (n, k)
2      if k mod 2 ≠ 0 then
3          sizeQ = ⌊(3^k+6)/8⌋;
4      else
5          sizeQ = ⌈(3^k+6)/8⌉;
6      end
7      vI = {(1, 0, 0), (0, 1, 0), (0, 0, 1)};
8      while (sizeof(vet) + sizeof(vI) < sizeQ) do
9          vet.Add(vI[sizeof(vI) - 3] + vI[sizeof(vI) - 2] + vI[sizeof(vI) - 1]);
10         if sizeof(vet) + sizeof(vI) ≥ sizeQ then
11             break;
12         end
13         vTemp.Add(vI[sizeof(vI) - 3] + vI[sizeof(vI) - 2]);
14         vTemp.Add(vI[sizeof(vI) - 3] + vI[sizeof(vI) - 1]);
15         vTemp.Add(vI[sizeof(vI) - 1] + vI[sizeof(vI) - 2]);
16         vI.AddRange(vTemp);
17         lasIndex = sizeof(vet);
18         for i ← 3 to 1 do
19             vet.Add(vet[lastIndex] + vI[sizeof(vI) - i]);
20         end
21         vTemp.Clear();
22     end
23     vet.AddRange(vI);
24     finalVet = vet + MapToOtherQuads(vet);
25     finalVetNeig = MapToNodesCoordinates(finalVet, n);
26     return finalVetNeig;
27 end
```

Função que calcula os nodos da vizinhança 3^k de um nodo n.

Data:

n: Current node.

k: K parameter indicates the size of neighborhood.

Reiniciando o laço de repetição, os três últimos vetores adicionados aos vetores

intermediários são somados, gerando um vetor resultante: $vRI = vI_4 + vI_5 + vI_6$. Ou seja, os últimos três índices do conjuntos dos vetores intermediários são somados gerando um novo vetor que realiza um movimento diagonal no qual três eixos variam. Assim como na primeira iteração do laço, essa etapa ocorre na linha 9 do Algoritmo 4. Os vetores que compõem a soma do novo vetor resultante são somados aos pares entre si, para novamente serem somadas ao vetor resultante: $v4 = vRI + (vI_4 + vI_5)$, $v5 = vRI + (vI_4 + vI_6)$ e $v6 = vRI + (vI_5 + vI_6)$. Cada vetor computado a cada etapa (inclusive os considerados "intermediários", que são somados aos pares) é adicionado à vizinhança do nodo. Além disso, eles têm suas diferentes direções mapeadas para os demais octantes.

O processo de soma de vetores apresentado é repetido até que o número previsto por 3^k, dado um parâmetro k, seja satisfeito. Dessa forma, a distribuição dos nodos vizinhos fica homogênea. Assim como em (RIVERA; HERNÁNDEZ; BAIER, 2017)(RIVERA et al., 2020), a suavização dos movimentos em direções diagonais, tanto de três eixos quanto de dois eixos, é priorizada. Nas orientações horizontal, vertical e profundidade, mantém-se apenas os vizinhos consecutivos. As maiores expansões ocorrem nos demais (e variados) ângulos. Por exemplo, os primeiros elementos de uma série Q em 3^k, representando os movimentos no primeiro octante (positivo em todos os eixos), são:

$$Q_0 = <(1, 0, 0), (0, 1, 0), (0, 0, 1)>$$

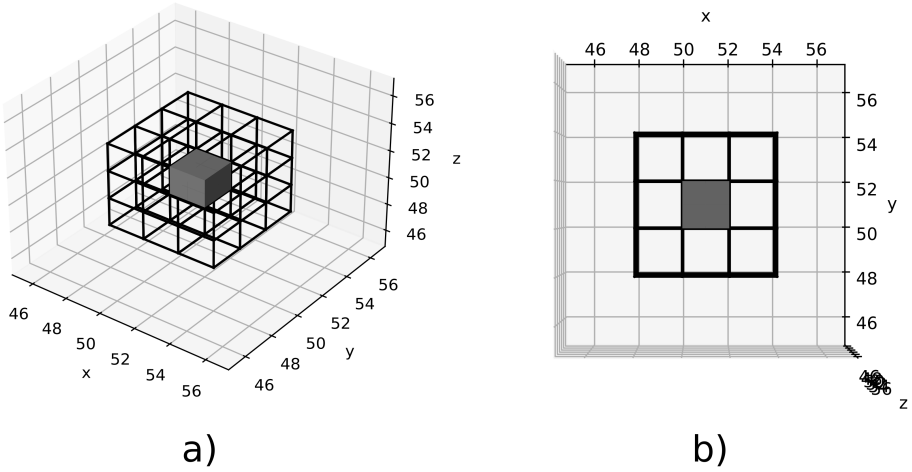
$$Q_1 = <(1, 0, 0), (0, 1, 0), (0, 0, 1), (1, 1, 1)>$$

$$Q_2 = <(1, 0, 0), (0, 1, 0), (0, 0, 1), (1, 1, 1), (1, 0, 1),$$
$$(1, 1, 0), (0, 1, 1), (2, 1, 2), (2, 2, 1), (1, 2, 2), (2, 2, 2) >$$

Avaliando o elemento Q_2 nesta série, o primeiro vetor resultante é o elemento número 4 em Q_2 (considerando indíces a partir de 1), cujas coordenadas são $(1, 1, 1)$, denotado por a_4. Os elementos a_5, a_6 e a_7 são os *vetores intermediários*. Estes vetores somam dois a dois os vetores de a_1 a a_3 ($a_5 = a_1 + a_3$, $a_6 = a_1 + a_2$, $a_7 = a_2 + a_3$). Esse processo é realizado para que diagonais de dois eixos também sejam visitadas, buscando melhor explorar a região em torno do nodo.

Os elementos de a_8 a a_{10} representam os resultados das somas de a_4 com os vetores de a_5 a a_7, respectivamente. O elemento a_{11} é a soma de todos os vetores intermediários de a_5 a a_7, a exemplo de a_4 que surge da soma dos vetores iniciais em Q_0 ($a_1 + a_2 + a_3$). Os vetores apresentados neste exemplo refletem o processo de somas de vetores anteriormente descrito, e que pode ser compreendido por meio do Algoritmo 4, até

Figura 4.2: a) Exemplo de disposição de vizinhos 3^k para k=3 na forma de nodos de um grid regular; b) Visão plana da disposição dos nodos.



Fonte: Autora.

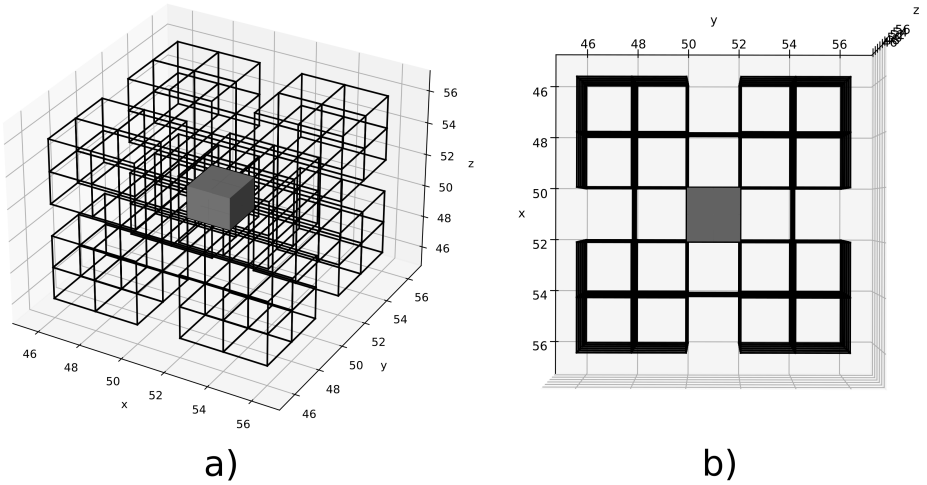
o passo em que realiza a soma para gerar vRI. Partindo disso, a sequência de passos é repetida até que, dado um valor atribuído ao parâmetro k, a série do octante satisfaça a cardinalidade esperada para atingir o valor 3^k vizinhos, valor que é atingido ao mapear as coordenadas para os demais octantes. Ou seja, um cálculo é realizado para identificar qual o valor limite do tamanho da série que representa os movimentos no primeiro octante, como apresentado nas linhas 2-8 do Algoritmo 4. Para saber o número de elementos em cada série, ou seja, a cardinalidade da série, utilizamos a seguinte função composta:

$$\begin{cases} \lfloor (3^k + 6)/8 \rfloor \mid k \mod 2 \neq 0 \\ \\ \lceil (3^k + 6)/8 \rceil \mid k \mod 2 = 0 \end{cases} \tag{4.3}$$

Computando movimentos diagonais que envolvem tanto dois eixos quanto movimentos diagonais de três eixos, uma melhor distribuição da vizinhança é realizada. Após computados os movimentos do primeiro octante, os mesmos são mapeados para as demais direções (linha 24 do Algoritmo 4). Em seguida, os vetores que indicam direções e variação da vizinhança de forma generalizada são mapeados para as coordenadas do nodo atual (linha 25 do Algoritmo 4), indicando os vizinhos deste nodo.

Os resultados da distribuição de nodos da vizinhança podem ser vistos nas Figuras 4.2 e 4.3. Quando $k = 3$, temos a distribuição padrão de vizinhança em grids cúbicos, que seleciona todos os nodos vizinhos em torno de um nodo expandido. Dessa forma, torna-se

Figura 4.3: a) Exemplo de disposição de vizinhos 3^k para k=4 na forma de nodos de um grid regular; b) Visão plana da disposição dos nodos.



Fonte: Autora.

necessário determinar um valor mínimo de $k = 3$. Representando a vizinhança na forma de nodos em um grid, a disposição é apresentada na Figura 4.3. O nodo central, nodo expandido e cujas coordenadas centrais são $n(51, 51, 51)$, é representado na cor cinza. Os vizinhos são representados somente pelas bordas de um cubo sem preenchimento. Figura "a)"apresenta uma visão em perspectiva, enquanto a Figura "b)"apresenta a visão superior no plano.

A cardinalidade da vizinhança N_{3^k} é 3^k, para todo $k \geq 3$. Ao gerarmos as séries que representam os movimentos no primeiro octante para cada vizinhança em função de k, a cardinalidade da vizinhança pode ser descrita em função da cardinalidade da série referente ao seu primeiro octante:

$$N_{3^k} = 8 \cdot \mid Q_i \mid -6 \tag{4.4}$$

Na Equação 4.4, como cada movimento não ortogonal possui 8 direções diferentes, multiplica-se a série por 8. O valor 6 nesta Equação remete aos seis movimentos ortogonais no espaço 3D. Por exemplo, as seis faces de um cubo (nodo em um grid cúbico).

Nota-se um padrão ao resultado da Equação 4.4: quando k é um valor ímpar, o valor da vizinhança é 3^k - 1; quando k é um valor par, o valor da vizinhança é 3^k + 1. Visto que 3^k sempre resulta um valor ímpar, não é possível ter o valor exato de 3^k distribuído entre os vizinhos, já que são 8 regiões direcionais diferentes em um cubo. Ou seja, mapeando um vetor na direção do primeiro octante para as demais direções, resulta

em um total de 8 vetores representando diferentes sentidos e direções. Portanto, um valor par é necessário. Por exemplo, para $|Q_1|$ = 4, a Equação 4.4 resulta em $8 \cdot 4 - 6$ = 26, que é o valor de 3^3 - 1. Para $|Q_2|$ = 11, a Equação resulta em $8 \cdot 11 - 6$ = 82, que é o valor de 3^4 + 1. Esse padrão é repetido nas séries subsequentes.

Assim como na versão 2^k para ambientes 2D, a expansão de vizinhança 3^k também é aplicável idealmente em grids livres de obstáculos quando não há tratamento para possíveis obstáculos entre vizinhos não adjascentes. A ideia de ampliar a vizinhança por si só expande nodos mais distantes do nodo atual. Contudo, essa expansão não prediz a existência de obstáculos entre os vizinhos selecionados para atingir diretamente um vizinho que esteja mais próximo à periferia do raio de expansão. O raio de expansão é dado pela distância do nodo mais distante, ou seja, o nodo que tiver maior coordenada entre os vizinhos.

$$raio(N) = max\{max\{\mid x \mid, \mid y \mid, \mid z \mid\} \mid (x, y, z) \in N\} \tag{4.5}$$

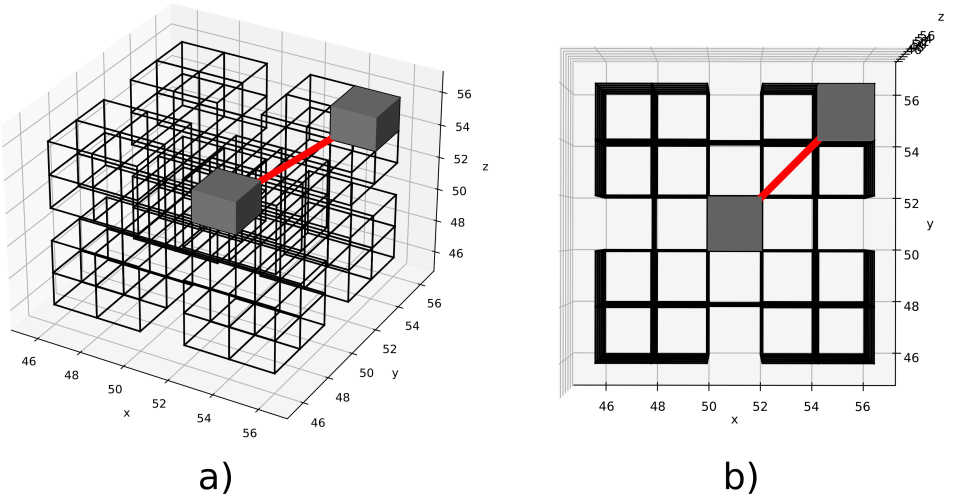
Ao aplicar a técnica de expansão 3^k em mapas com obstáculos[1], alguns tratamentos são necessários. Isso é necessário porque a técnica expande vizinhos mais distantes, além dos adjacentes, e em diferentes direções, de forma bem distribuída. Logo, é possível que haja bloqueios na linha reta entre o nodo que está sendo expandido e o vizinho não adjacente a ser atingido. Para isso, é necessário verificar se há visibilidade entre o nodo vizinho e o nodo atual. Para resolver esse problema, a técnica de campo de visão é utilizado a fim de verificar se há obstrução no caminho entre esses dois nodos. Essa ideia é introduzida por este trabalho, visto que a técnica de campo de visão para tratar possíveis obstáculos na vizinhança não é abordada em (RIVERA; HERNÁNDEZ; BAIER, 2017) e (RIVERA et al., 2020). Com isso, em caso de obstrução, outro vizinho é selecionado. Vale ressaltar que essa verificação é necessária quando $k > 3$, pois os vizinhos abertos são adjascentes ao nodo quando $k = 3$. Portanto, é necessário apenas verificar se o vizinho está bloqueado ou não. Algoritmos que não fazem uso de campo de visão como, por exemplo, os algoritmos A* e JPS, implementam essa etapa para realizar a verificação de campo de visão entre vizinhos não adjascentes.

Um exemplo de campo de visão não obstruído é ilustrado na Figura 4.4. A linha vermelha que conecta o nodo central representando o nodo atual expandido e o nodo vizinho, candidato a sucessor, demonstra que não há obstáculos entre esses dois nodos.

[1] https://github.com/carolchagas/PathFinding3DSpace

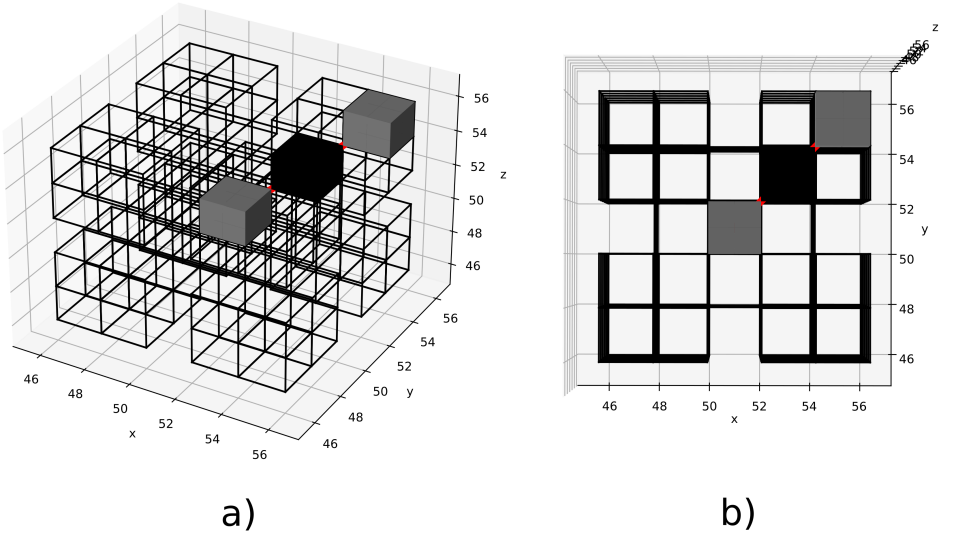
Portanto, a linha de visão é livre, permitindo a travessia. Por outro lado, a Figura 4.5 exemplifica uma situação onde um nodo intermediário é bloqueado, representado pelo nodo em preto. Nesse caso, o campo de visão é obstruído, bloqueando a travessia. Quando isso acontece, o vizinho avaliado não é selecionado como potencial nodo para compôr o caminho a partir do nodo atual. Dado o obstáculo, o movimento deve ser redirecionado para desviar esse obstáculo.

Figura 4.4: a) Visão em perspectiva do campo de visão não obstruído entre nodo expandido e um de seus vizinhos; b) Visão plana do campo de visão não obstruído.



Fonte: Autora.

Figura 4.5: a) Visão em perspectiva do campo de visão obstruído entre nodo expandido e um de seus vizinhos; b) Visão plana do campo de visão obstruído.



Fonte: Autora.

Algorithm 5: Função *FindPruned3kNeighbors* para expansão da vizinhança 3^k utilizando poda.

```
1  FindPruned3kNeighbors (n, k, tDx, tDy, tDz)
2      if k mod 2 ≠ 0 then
3          sizeQ = ⌊(3^k+6)/8⌋;
4      else
5          sizeQ = ⌈(3^k+6)/8⌉;
6      end
7      vI = {(tDx, 0, 0), (0, tDy, 0), (0, 0, tDz)};
8      vet = ComputeSerie(vI);
9      vet.AddRange(vI);
10     finalVetNeig = MapToNodesCoordinates(vet, n);
11     finalVetNeig.AddRange(FindForcedNeig(n));
12     return finalVetNeig;
13 end
```

Função que calcula os nodos da vizinhança 3^k com poda de um nodo n.

Data:

n: Current node.

k: K parameter indicates the size of neighborhood.

tDx : x coordinate of vector $\vec{d}$.

tDy : y coordinate of vector $\vec{d}$.

tDz : z coordinate of vector $\vec{d}$.

Além do tratamento de obstáculos, alguns algoritmos podem exigir o processo de poda da vizinhança, como é o caso do algoritmo JPS. Neste caso, a expansão da vizinhança 3^k também apresenta algumas modificações. Diferentemente da expansão completa da vizinhança, quando utilizada a poda apenas os vizinhos dispostos na direção em que ocorre o movimento são selecionados. Dessa forma, a quantidade de vizinhos expandidos será o valor equivalente ao de uma série (cujo cálculo foi apresentado anteriormente), não sendo mapeados os vizinhos nas demais direções referentes demais octantes. Assim, a vizinhança é expandida a partir do vetor $\vec{d}$. Após selecionar os vizinhos promissores da vizinhança 3^k de acordo com a direção, são verificados os vizinhos forçados do nodo atual. O Algoritmo 5 apresenta como a vizinhança 3^k é selecionada utilizando a poda. As somas iniciais ocorrem a partir dos vetores $(tDx, 0, 0)$, $(0, tDy, 0)$ e $(0, 0, tDz)$, e na sequência os cálculos para identificar os vizinhos da série ocorrem normalmente con-

forme as linhas 8-22 do Algoritmo 4, processo indicado na linha 8 do Algoritmo 5. Os vizinhos selecionados nessa etapa são considerados os *vizinhos naturais*. Na linha 24 os vetores computados são mapeados para as coordenadas dos respectivos nodos. A partir disso, são computados os vizinhos forçados do nodo atual (linha 10 do Algoritmo 5).

A vizinhança 3^k pode ser aplicada a algoritmos que não implementem uma técnica própria de expansão de vizinhança. Por exemplo, algoritmos como Anya (HARABOR; GRASTIEN, 2013)(HARABOR et al., 2016) e Field D* (CARSTEN; FERGUSON; STENTZ, 2006), que possuem técnicas próprias de expansões de vizinhança, não seriam adeptos à expansão 3^k. Anya expande a vizinhança por meio de intervalos selecionados com base em um nodo $root$. Por sua vez, o Field D* seleciona a vizinhança por meio de interpolação entre pontos próximos. Sendo assim, uma maior investigação desses algoritmos é necessária para verificar a viabilidade de utilizar a vizinhança proposta neste trabalho. Todavia, por ser uma expansão mais ampla, a principal ideia é que seja aplicada a implementações menos complexas (como, por exemplo, o algoritmo A*, bem explorado na literatura e em inúmeras aplicações práticas), de forma a produzir caminhos de melhor qualidade, resultados que podem ser próximos aos de implementações de algoritmos de planejamento de caminhos mais complexos. Ainda, a vizinhança 3^k é desenvolvida idealmente para grids regulares, visto que sua construção é dada a partir das coordenadas centrais de nodos próximos.

Por meio da aplicação da vizinhança 3^k ao algoritmo A* pretende-se melhorar a qualidade dos caminhos computados, ampliando as possibilidades de movimentação em um tempo de execução reduzido em relação a algoritmos *any-angle* de implementação complexa. Dessa forma, essa técnica pode reduzir significativamente a sinuosidade dos caminhos retornados pelo algoritmo A*, mantendo a otimalidade deste algoritmo. Com o algoritmo JPS, apesar de ele já ser um algoritmo que realiza suavização devido aos *jump points* serem selecionados de acordo com a disposição dos obstáculos no mapa, a expansão 3^k também amplia a possibilidade de ângulos de movimentação. Além dos obstáculos influenciarem na angulação do caminho e, consequentemente, eleição dos sucessores, a distribuição e organização dos nodos na nova proposta de vizinhança também passa a influenciar tal processo. Uma sequência de saltos em busca dos sucessores, que tradicionalmente é iniciada a partir de um vizinho adjascente, passa a ter a possibilidade de ser iniciada partindo de um vizinho mais distante da vizinhança 3^k, influenciando na angulação da movimentação. O Lazy Theta*, por sua vez, tende a apresentar diferenças menores quanto à qualidade do caminho retornado ao comparar-se o uso do algoritmo

com vizinhança padrão e vizinhança 3^k, visto que já é um algoritmo de busca classificado como *any-angle*.

5 EXPERIMENTOS E RESULTADOS

Os experimentos realizados neste trabalho visam analisar o uso da vizinhança 3^k em algoritmos propostos pela literatura. Para a realização dos experimentos, três mapas em grids 3D foram desenvolvidos a fim de representar três ambientes 3D diferentes. Cada ambiente possui um grau de complexidade diferente em termos de quantidade e distribuição de obstáculos. Inicialmente, foi realizado um conjunto de experimentos nos três mapas 3D, a fim de avaliar os impactos promovidos com o incremento da variável k. Por meio destes experimentos, referenciados no trabalho como "experimentos iniciais", foi avaliado qual valor máximo parametrizado pela variável k permite a viabilidade do uso da *Vizinhança* 3^k em implementações que possam ser aplicadas. Na sequência, foram conduzidos os chamados "experimentos finais", que foram realizados aplicando a vizinhança 3^k a algoritmos que realizam técnicas de suavização, inclusive no estado da arte, e também ao tradicional A*. A saber, o trabalho utiliza os algoritmos A*, JPS e Lazy Theta*, cujas justificativas de suas escolhas encontram-se na seção 5.3. Para estes experimentos, foram utilizados os valores de k considerados viáveis para aplicações futuras, como jogos, sistemas de simulação e sistemas de veículos aéreos não tripulados. Os algoritmos implementados neste trabalho foram desenvolvidos utilizando a *Game Engine Unity 3D* (UNITY-TECHNOLOGIES, 2022), ambiente de desenvolvimento no qual os experimentos foram conduzidos.

Esta seção está dividida da seguinte forma: a subseção 5.1 descreve como os dados foram descritos e avaliados por meio do uso de métodos estatísticos em ambos experimentos; a subseção 5.2 apresenta uma análise dos impactos no incremento da variável k, a fim de definir a faixa de valores como parâmetro para esta variável que tornem a vizinhança aplicável a outras implementações para os experimentos finais; a subseção 5.3 apresenta o protocolo de testes, descrevendo mapas desenvolvidos, algoritmos selecionados e os detalhes para realização dos experimentos finais; a subseção 5.4 apresenta os resultados dos experimentos finais para os três mapas desenvolvidos, obtidos por meio dos métodos estatísticos; e a subseção 5.5 apresenta uma análise dos resultados obtidos, comparando-os entre os três mapas no qual foram executados.

5.1 Refinamento dos dados brutos para avaliação dos resultados

Modelos de regressão linear generalizados (MCCULLAGH; NELDER, 1989) foram utilizados para apoiar a análise dos resultados dos experimentos. A distribuição Gamma foi explorada na construção dos modelos de regressão, visto os valores dos resultados obtidos nos experimentos realizados. Essa distribuição é a que melhor representa valores reais positivos, os quais são obtidos para tempo de execução da busca de caminho, número de nodos abertos durante a execução dos algoritmos de busca, número de nodos nos caminhos resultantes e comprimento dos caminhos resultantes.

Os modelos de regressão utilizados para analisar e comparar os tempos de execução da busca de caminhos e o número de nodos do caminho resultante dos algoritmos testados no experimento inicial são definidos por g(mu) = $Beta_0$ + $Beta_1$ * X + $Beta_2$ * D1 + $Beta_3$ * D2, onde g é a função logarítmica. Na prática, este modelo representa os respectivos valores de número de nodos do caminho encontrado e o tempo de execução do algoritmo de busca, ambos como uma função de distâncias entre as posições de origem e destino no mapa 3D, onde os valores de distância são representados pela variável X do modelo. Para fazer as comparações das diferentes técnicas com o método base (o algoritmo A* com uso de vizinhança $3^{k=3}$), variáveis dummy D1 - D2 desse modelo estatístico são usados para representar cada algoritmo. Portanto, os modelos de regressão descrevem o tempo de execução e número de nodos no caminho resultante do algoritmo A* com vizinhança $3^{k=3}$ quando D1 = D2 = 0. Os respectivos valores das variáveis Dummy de acordo com o algoritmo avaliado no modelo são apresentados na Tabela 5.1.

Tabela 5.1: Configuração das variáveis Dummy nos experimentos iniciais.

Variáveis Dummy	**Vizinhança de acordo com valores de** k
$D_1 = 0, D_2 = 0$	$3^{k=3}$
$D_1 = 1, D_2 = 0$	$3^{k=4}$
$D_1 = 0, D_2 = 1$	$3^{k=5}$

Fonte: Autora.

No experimento final, os modelos utilizados para comparar número de nodos no caminho resultante, números de nodos abertos, comprimento do caminho resultante e tempos de execução dos algoritmos testados são definidos por g(mu) = $Beta_0$ + $Beta_1$ * X + $Beta_2$ * D1 + ... + $Beta_9$ * D8. Para fazer as comparações no experimento final das diferentes técnicas com o método base, tomado neste trabalho como o algoritmo A* padrão, as chamadas variáveis dummy D1 - D8 desse modelo estatístico são usados para representar cada algoritmo. Portanto, os modelos de regressão descrevem o tempo

de execução, número de nodos no caminho resultante, nodos abertos ou o comprimento do caminho do algoritmo A* quando D1 = D2 = D3 = D4 = D5 = D6 = D7 = D8 = 0. As respectivas configurações das variáveis para cada algoritmo avaliado no modelo são apresentadas na Tabela 5.2. Isto é, para obter os resultados para cada um desses algoritmos, é necessário atribuir o valor 1 à variável D_i, a qual representa o i-*ésimo* algoritmo, enquanto outras variáveis D devem ter o valor = 0.

Tabela 5.2: Configuração das variáveis Dummy nos experimentos finais.

Variáveis Dummy	Algoritmo
D_1, ..., $D_8 = 0$	A*
$D_1 = 1$, D_2, ..., $D_8 = 0$	A* $3^{k=3}$
$D_1 = 0$, $D_2 = 1$, D_3, ..., $D_8 = 0$	A* $3^{k=4}$
D_1, $D_2 = 0$, $D_3 = 1$, D_4, ..., $D_8 = 0$	JPS
D_1, ..., $D_3 = 0$, $D_4 = 1$, D_5, ..., $D_8 = 0$	JPS $3^{k=3}$
D_1, ..., $D_4 = 0$, $D_5 = 1$, D_6, ..., $D_8 = 0$	JPS $3^{k=4}$
D_1, ..., $D_5 = 0$, $D_6 = 1$, D_7, $D_8 = 0$	Lazy Theta*
D_1, ..., $D_6 = 0$, $D_7 = 1$, $D_8 = 0$	Lazy Theta* $3^{k=3}$
D_1, ..., $D_7 = 0$, $D_8 = 1$	Lazy Theta* $3^{k=4}$

Fonte: Autora.

Comparando os algoritmos A*, A* 3^k com $k = 3$, A* 3^k com $k = 4$, JPS, JPS 3^k com $k = 3$, JPS 3^k com $k = 4$, Lazy Theta* (nos gráficos é referenciado como LT*), Lazy Theta* 3^k com $k = 3$ e Lazy Theta* 3^k com $k = 4$, as estimativas estatísticas representam os valores médios para (i) o tempo de execução e (ii) o número de nodos que compõem o caminho resultante (indicativo de suavização), (iii) quantidade de nodos abertos durante a execução do algoritmo, (iv) o comprimento do caminho resultante em relação aos diferentes valores de distância entre os pontos inicial e destino. A função de ligação utilizada nos modelos de regressão é a função log, pois os resultados desta função no método estatístico podem representar qualquer valor real. Para desenvolver a análise estatística, foi definido um nível de significância de 0,01 (1%). Assim, duas hipóteses H_0 e H_1 foram definidas com $Beta_i$ = 0 ou $Beta_i \neq 0$, respectivamente, para i = 1, 2, ..., 9. Então, o valor alfa é comparado com o valor p. Se alfa $< P - valor$, então H_0 é rejeitado; caso contrário, H_0 não é rejeitado. $Beta_1$ significa que distâncias maiores entre as posições de origem e destino implicam em tempos de execução mais longos para os algoritmos retornarem caminhos nessas distâncias. Para $Beta_i$, com i assumindo os valores 2-9, $Beta_i$ = 0 significa que o método D_i é equivalente ao método base. $Beta_i >$ 0 significa que o método ao qual D_i se refere é mais lento que o método base. $Beta_i < 0$ significa que o método representado por D_i é mais rápido que o método base.

Os resultados de busca de caminhos obtidos para as técnicas testadas, como mos-

trado nas Figs. 5.6 e 5.7, e nas Tabelas 5.10-5.21, os valores de P obtidos com os modelos de regressão são praticamente todos próximos de zero e abaixo do nível alfa considerado de 0,01; portanto, quase todos os resultados são estatisticamente significativos. A exceção foi o aspecto de comprimento do caminho, que obteve resultados muito próximos, não apresentando diferenças significativas. De forma geral, os modelos de regressão mostram que existem diferenças significativas entre as técnicas comparadas. Todos os valores de tempo, nodos abertos e quantidade de nodos ao longo do caminho para as distâncias do caminho entre as posições inicial e final com as técnicas A*, as técnicas que utilizam JPS e as técnicas que utilizam Lazy Theta* são diferentes da técnica A* padrão, tomada como parâmetro base.

Como exemplo da análise dos resultados estatísticos obtidos nos experimentos, o valor do tempo de execução do algoritmo de busca para D2 = A* 3^k para $k = 4$ na Tabela 5.15 é 1,7090. Como Exp(1,7090) = 5,52 (aqui a função exponencial é usada devido ao uso da função logarítmica utilizada como função de ligação), pode-se observar que a técnica A* 3^k (para $k = 4$) requer 5,52 vezes mais tempo computacional para calcular caminhos do que a técnica base A* requer para as mesmas distâncias de percurso. Interpretações semelhantes podem ser feitas inspecionando os outros resultados mostrados nas demais Tabelas 5.10-5.21. Para as estimativas de tempo de execução na Tabela 5.15 cujos valores são positivos, os modelos de regressão mostram que, nestes casos, a busca de caminho com o A* ainda resulta em um menor tempo de execução em relação aos respectivos algoritmos (ocorrendo quando utilizados os algoritmos A* com vizinhança 3^k e o algoritmo Lazy Theta*, tanto para versão padrão quanto para versão que utiliza vizinhança 3^k). Quando as estimativas apresentam valores negativos, como nos casos em que o algoritmo JPS é utilizado (tanto na versão padrão quanto na versão que utliza vizinhança 3^k), demonstra que o algoritmo obtém tempos menores em relação ao algoritmo A*, tomado como método base.

5.2 Análise de impacto do incremento do valor k no algoritmo A*

Visto que a cardinalidade da vizinhança 3^k pode aumentar de forma acentuada a cada incremento da variável k, uma avaliação é realizada sobre os valores desta variável. tomando como parâmetros os valores de 3, 4 e 5. Lembrando que às condições de cardinalidade da vizinhança 3^k, tais parâmetros geram vizinhanças de tamanhos $3^{k=3}$ - 1 = 26, $3^{k=4}$ + 1 = 82 e $3^{k=5}$ - 1 = 242. Tomando o parâmetro $k = 6$, a cardinalidade da vizi-

nhança resultaria em $3^{k=6}$ + 1 = 730. Utilizar tal tamanho de vizinhança é inviável, pois resultaria em valores de tempos de execução extremamente altos. Na prática, tal técnica não teria aplicação factível, como possíveis usos em jogos e simulações. Portanto, valores para $k \geq 6$ não foram testados.

Este experimento foi realizado somente com o algoritmo A*, considerando o bom desempenho deste algoritmo (HART; NILSSON; RAPHAEL, 1968)(NILSSON N. J., 1998), que é essencial quando $k > 4$. Além disso, é um algoritmo muito explorado na literatura, que produz caminhos ótimos em grafos, o que permite utilizá-lo como base para comparações. Testes com as versões do A* para vizinhança com diferentes valores de k foram realizados nos 3 mapas tridimensionais com diferentes níveis de bloqueios. Nas avaliações estatísticas realizadas, o algoritmo com menor parâmetro de k foi tomado como método base; sendo assim, definido pelo algoritmo A* com vizinhança 3^k para $k = 3$.

Ao utilizar uma vizinhança composta por uma grande quantidade de nodos com valores de k a partir de 5, os tempos de execução apresentaram resultados muito elevados. Neste caso, uma maior suavização do caminho é promovida. Porém, quando comparada com o tempo necessário para as execuções do algoritmo, não há uma boa relação custo-benefício. Ao comparar as versões do algoritmo A* que utilizam a vizinhança 3^k para $k = 3$ e para $k = 4$, as diferenças em tempos de execução, apesar de estatisticamente significativas, não são acentuadas. Por exemplo, no mapa A (Figura 5.3), a versão do algoritmo que utiliza vizinhança $3^{k=4}$ apresentou tempo de execução 3,9 vezes maior, em relação a versão que utiliza vizinhança $3^{k=3}$ (Figura 5.1(a)). No mapa B (Figura 5.4), a versão do A* com vizinhança $3^{k=4}$ demonstrou obter um tempo de execução 5,22 vezes maior. Em relação ao mapa C (Figura 5.5), esse valor adicional de tempo foi de 4,35 vezes. É possível observar que os valores adicionais de tempo nos 3 mapas 3D para o algoritmo A* com vizinhança $3^{k=4}$ em relação ao método base ficaram muito próximos. Os tempos obtidos com o uso de vizinhança 3^k para $k = 4$ tornam-se viáveis na prática, diferente de quando o parâmetro $k = 5$ é utilizado. As diferenças discrepantes nos tempos de execução podem ser observadas nos gráficos apresentados na Figura 5.1. No mapa A, o algoritmo A* com vizinhança $3^{k=5}$ apresentou ser 41,38 vezes mais lento em relação ao método base. Por sua vez, no mapa B, esse valor se torna cerca de 59,32 vezes mais lento. No caso do mapa C, o algoritmo A* utilizando o valor 5 como parâmetro para a variável k exigiu 42,6 vezes mais tempo para retornar um caminho, em relação ao método base. Apesar de obter classificação dos valores obtidos dentre as vizinhanças e resultados

Tabela 5.3: Dados estatísticos para avaliar impacto de k no mapa A: Tempos de execução.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,6790	0,02624	178,31	<2e-16
A* $3^{k=3}$	0,0014	3,549e-05	40,55	<2e-16
A* $3^{k=4}$	1,3660	0,0198	68,98	<2e-16
A* $3^{k=5}$	3,7230	0,0201	184,40	<2e-16

Fonte: Autora.

Tabela 5.4: Dados estatísticos para avaliar impacto de k no mapa B: Tempos de execução.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,3390	0,0272	159,14	<2e-16
A* $3^{k=3}$	0,0016	3,512e-05	47,60	<2e-16
A* $3^{k=4}$	1,6530	0,0199	82,81	<2e-16
A* $3^{k=5}$	4,0830	0,0211	193,38	<2e-16

Fonte: Autora.

similares nos três mapas 3D, é possível observar por meio da Figura 5.1 que o crescimento das curvas dos gráficos se torna mais acelerado à medida em que a complexidade dos mapas é aumentada (isto é, ao passo em que a quantidade de obstáculos no ambiente é aumentada). Entende-se por "classificação dos valores obtidos"que a ordem dos valores dos resultados de comparação para as 3 vizinhanças é a mesma nos diferentes mapas.

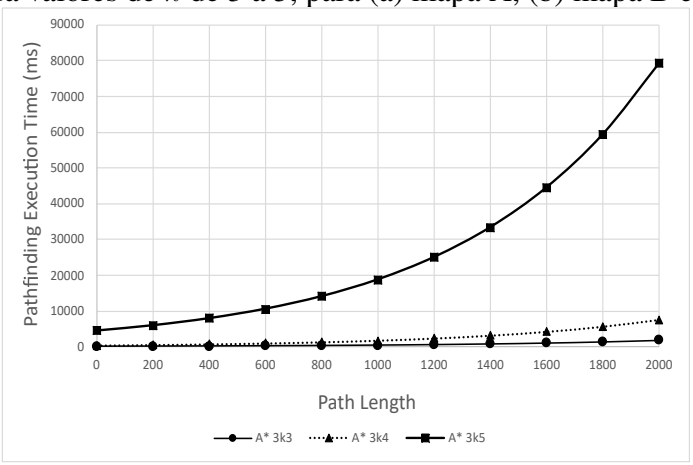
Tratando-se de suavização do caminho, é possível observar que a eficiência do uso dessa técnica pelo algoritmo aumenta ao passo que o valor de k é incrementado , sendo esses dois fatores diretamente proporcionais. Ainda, percebe-se que a cada incremento do valor k a proporção do efeito da suavização é praticamente mantida em relação ao efeito sobre o caminho utilizando a vizinhança com valor de k anterior. Isso significa que a razão do efeito de suavização sobre o uso de vizinhança $3^{k=4}$ em relação ao uso de vizinhança $3^{k=3}$ é próxima da razão do efeito de suavização sobre o uso de vizinhança $3^{k=5}$ em relação ao uso de vizinhança $3^{k=4}$. Por exemplo, no mapa A a quantidade de nodos do caminho retornado utilizando vizinhança $3^{k=4}$ é 12,82% menor que a quantidade de nodos do caminho resultante utilizando vizinhança $3^{k=3}$. A proporção é mantida na relação entre

Tabela 5.5: Dados estatísticos para avaliar impacto de k no mapa C: Tempos de execução.

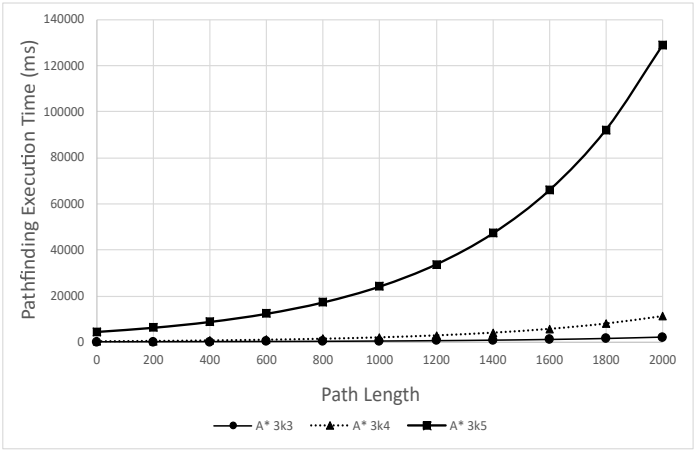
Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	3,1690	0,0305	103,76	<2e-16
A* $3^{k=3}$	0,0039	4,517e-05	87,02	<2e-16
A* $3^{k=4}$	1,4700	0,0191	76,95	<2e-16
A* $3^{k=5}$	3,7520	0,0191	195,49	<2e-16

Fonte: Autora.

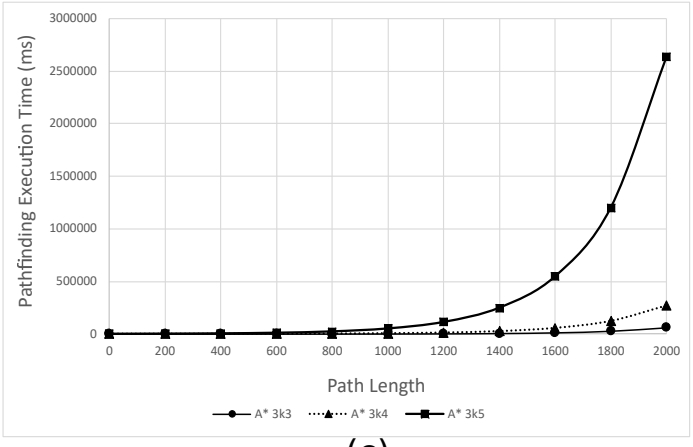
Figura 5.1: Gráficos referentes aos tempos obtidos com as versões do algoritmo A* com vizinhança 3^k para valores de k de 3 a 5, para (a) mapa A, (b) mapa B e (c) mapa C.



(a)



(b)



(c)

Fonte: Autora.

Tabela 5.6: Dados estatísticos para avaliar impacto de k no mapa A: Número de nodos no caminho resultante.

Algoritmo	**Estimativa**	**Erro padrão**	**Valor T**	**Pr(>|t|)**
Intercept	4,3370	0,02867	151,27	<2e-16
A* $3^{k=3}$	0,0017	4,712e-05	37,03	<2e-16
A* $3^{k=4}$	-0,1372	0,0060	-22,71	<2e-16
A* $3^{k=5}$	-0,2795	0,0060	-46,08	<2e-16

Fonte: Autora.

o uso das vizinhanças $3^{k=5}$ e $3^{k=4}$. Essa propriedade foi confirmada em todos os mapas 3D. Diferente do que ocorre com os tempos, observa-se que a suavização aumenta de forma gradual a cada incremento do parâmetro k.

No mapa A, o algoritmo A* que utiliza vizinhança $3^{k=4}$ demonstrou obter um valor 12,82% menor de nodos no caminho final em relação ao número de nodos resultante do método base, enquanto o algoritmo que utiliza vizinhança $3^{k=5}$ apresentou um valor 24,38% menor para o mesmo aspecto. No que diz respeito à propriedade de manter uma proporção na suavização a cada incremento de k, o algoritmo que utiliza vizinhança $3^{k=5}$ demonstrou obter um valor 13,26% menor de nodos no caminho final do que o algoritmo que utiliza a mesma vizinhança tendo como parâmetro $k = 4$.

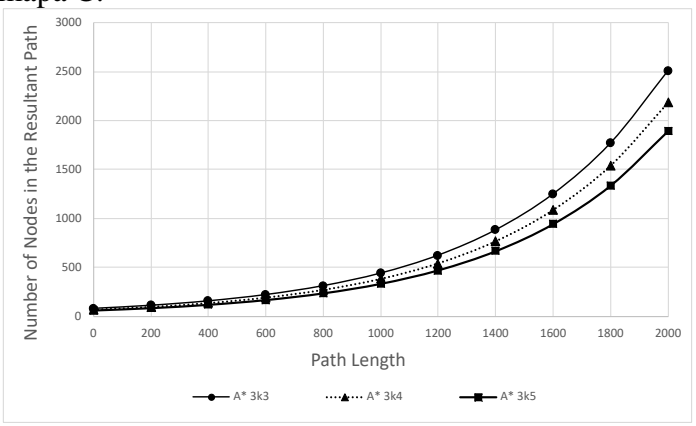
No mapa B, o uso de vizinhança $3^{k=5}$ resultou em menos 24,12% de nodos no caminho final do que o método base com parâmetro $k = 3$. Com o uso de vizinhança $3^{k=4}$, a quantidade de nodos no caminho final é 12,87% menor que a quantidade de nodos no caminho obtido pelo método base. Ao comparar o uso de vizinhança $3^{k=5}$ com o uso de vizinhança $3^{k=4}$, o primeiro apresenta uma quantidade 12,91% menor de nodos no caminho resultante.

No mapa C, a quantidade de nodos do caminho retornado utilizando vizinhança $3^{k=4}$ é 13,21% menor que a quantidade de nodos do caminho retornado com vizinhança $3^{k=3}$. Ao compararmos vizinhança $3^{k=5}$ em relação à vizinhança $3^{k=3}$, o primeiro produz caminhos com 24,78% menos nodos. Mantendo a proporção nos resultados entre valores consecutivos atribuídos à variável k, o uso de vizinhança $3^{k=5}$ reduz o número de nodos no caminho resultante em 13,33% quando comparado ao uso de vizinhança $3^{k=4}$.

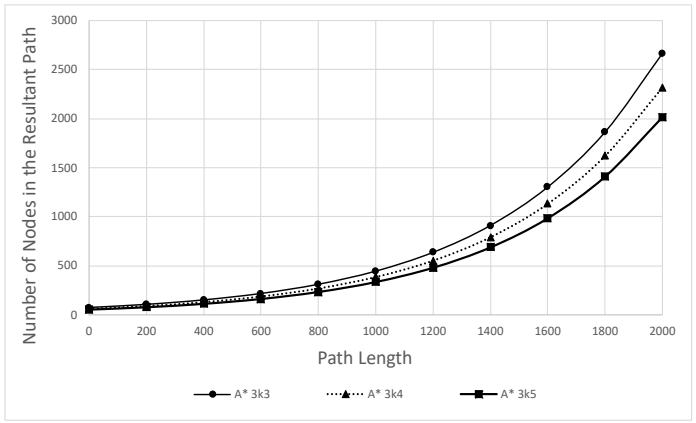
Embora o crescimento das curvas de tempos de execução que refletem os resultados dos testes apresentem diferentes velocidades de crescimento para os três mapas 3D, quando a suavização é abordada as velocidades de crescimento das curvas são mantidas, conforme apresentado na Figura 5.2.

Partindo dos resultados de execução do algoritmo A* para o parâmetro k, definiu-

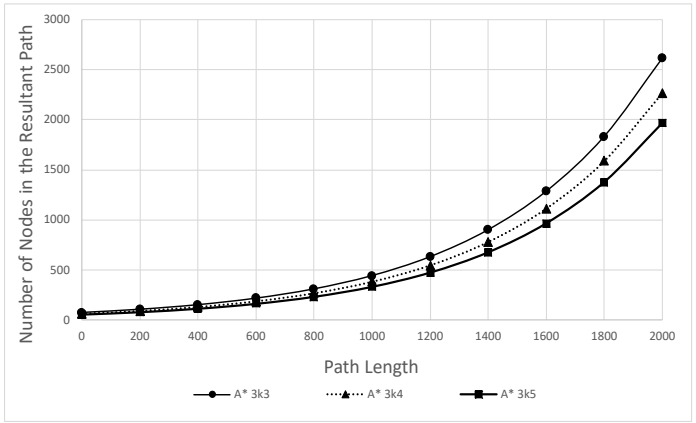
Figura 5.2: Gráficos referentes às quantidades de nodos nos caminhos obtidos com as versões do algoritmo A* com vizinhança 3^k para valores de k de 3 a 5, para (a) mapa A, (b) mapa B e (c) mapa C.



(a)



(b)



(c)

Fonte: Autora.

Tabela 5.7: Dados estatísticos para avaliar impacto de k no mapa B: Número de nodos no caminho resultante.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,3120	0,0234	184,01	<2e-16
A* $3^{k=3}$	0,0017	3,869e-05	46,20	<2e-16
A* $3^{k=4}$	-0,1378	0,0059	-23,02	<2e-16
A* $3^{k=5}$	-0,2761	0,0059	-46,02	<2e-16

Fonte: Autora.

Tabela 5.8: Dados estatísticos para avaliar impacto de k no mapa C: Número de nodos no caminho resultante.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,3200	0,0210	204,85	<2e-16
A* $3^{k=3}$	0,0017	3,4500e-05	51,44	<2e-16
A* $3^{k=4}$	-0,1420	0,0060	-23,30	<2e-16
A* $3^{k=5}$	-0,2850	0,0060	-46,69	<2e-16

Fonte: Autora.

se o valor máximo de $k = 4$ a ser usado nos experimentos finais de maior complexidade envolvendo os demais algoritmos selecionados para os testes. Para valores de k a partir de 5, a vizinhança se torna muito grande, resultando em tempos de execução altíssimos para grandes distâncias, sendo inviável de utilização em possíveis aplicações futuras. Em implementações mais complexas de algoritmos de busca de caminhos, que muitas vezes realizam mais verificações, como o JPS sem realizar a poda de vizinhança e, principalmente, Lazy Theta*, os tempos de computação de caminhos seriam ainda maiores. Ainda, o valor 4 para o parâmetro k apresenta uma importante relação custo-benefício entre suavização e tempo de execução, o que é relevante para muitas aplicações do mundo real.

5.3 Protocolo de testes

Conforme citado no início deste capítulo, três mapas em grids 3D foram desenvolvidos para representar ambientes 3D. Para a geração dos mapas utilizados, foram criados obstáculos em torno de pontos aleatoriamente distribuídos no espaço. Durante a geração dos mapas 3D para testes houve uma disposição estratificada dos obstáculos em diferentes regiões do espaço. Tal ambiente foi divido em 8 sub-áreas, representando os 8 octantes do volume total. Internamente a cada octante, um ponto aleatório foi selecionado na posição central do octante, formando o agrupamento de obstáculos em torno do ponto escolhido. A disposição dos obstáculos entre os octantes que seccionam o mapa 3D pode ser ob-

servada nos histogramas referentes a cada mapa desenvolvido, apresentados nas Figuras 5.3(b), 5.4(b) e 5.5(b). As características dos mapas 3D e o protocolo de testes são apresentados na Tabela 5.9.

Da mesma forma, os pontos inicial e destino usados nas computações de caminhos dos testes realizados foram selecionados de forma estratificada. No total, 4.480 pares de pontos foram utilizados. Desse valor, cada octante posicionou 560 pontos aleatórios de origem. Essa quantia foi dividida por 7, permitindo distribuir 80 pontos de destino para cada um dos 7 octantes restantes. Internamente ao octante, os 80 pontos também foram distribuídos aleatoriamente. Dessa forma, os testes utilizaram pares de pontos para permitir gerar caminhos com diferentes distâncias e direções.

As métricas selecionadas para avaliação dos algoritmos testados foram i) comprimento do caminho, ii) suavização, iii) tempo de execução e iv) consumo de memória. A suavização, neste caso, foi medida pela quantidade de nodos do caminho. O consumo de memória, foi medido por meio da quantidade de nodos abertos. Ainda, uma análise em torno da expansão da vizinhança foi realizada, a fim de avaliar o impacto da variação do valor de K no caminho final computado.

Para a execução dos experimentos, três algoritmos foram selecionados: A*, JPS e Lazy Theta*. Dentre estes, Lazy Theta* é classificado como *any-angle*. Ao ampliar a exploração da vizinhança, o algoritmo que utiliza essa técnica também realiza comportamento semelhante ao de um algoritmo *any-angle*, independentemente se essa característica é da natureza do algoritmo selecionado ou não. O algoritmo A*, tal como em (MANDLOI; ARYA; VERMA, 2021), foi escolhido devido à vasta exploração em contextos de jogos e simulações, oferecendo bons parâmetros base para fins de comparação, visto a completude e otimalidade deste algoritmo. Ainda, o algoritmo A* possui uma implementação simplificada que fornece bons resultados para a qualidade do caminho retornado quando a expansão de vizinhança 3^k é usada. Na prática, isso resulta em uma suavização do caminho retornado que é associada a baixos tempos de computação.

O algoritmo JPS foi selecionado por ser um algoritmo que também realiza suavização à medida que a busca é executada. Desenvolvido a partir do tradicional A*, ele encontra caminhos expandindo menos nodos e retornando-os em tempos de execução satisfatórios, superando o desempenho do algoritmo A* enquanto mantém a otimalidade dos caminhos retornados por este algoritmo.

O algoritmo Lazy Theta* foi selecionado por produzir caminhos ótimos. Diferente do Theta* padrão, este algoritmo otimiza o processo de verificação de linha de

Tabela 5.9: Características dos mapas 3D e protocolo de testes.

Informação	**Mapa A**	**Mapa B**	**Mapa C**
Dimensões do mapa 3D em nodos	360 X 360 X 360		
Número total de nodos	46.656.000		
Estrutura de representação do mapa	Grid regular cúbico (3D)		
Percentual de área bloqueada	30%	45%	65%
Métricas de avaliação	Comprimento do caminho resultante Tempo de execução Número de nodos abertos Número de nodos no caminho resultante		
Algoritmos de busca de caminhos	A* com 3^k vizinhos A* com vizinhança normal Lazy Theta* com 3^k vizinhos Lazy Theta* com vizinhança normal JPS com 3^k vizinhos JPS com vizinhança normal		
Função heurística	Distância de Manhattan		
Conjunto de dados de teste	4.480 pares de pontos na estrutura de representação		

Fonte: Autora.

visão, reduzindo de forma considerável o tempo de execução da busca, apresentando bom desempenho em mapas 3D. Quando comparado ao algoritmo A*, este último encontra o menor caminho em grafos, porém não garante encontrar o caminho mais curto em um ambiente contínuo real. Tal como em (MANDLOI; ARYA; VERMA, 2021), os algoritmos Theta* e Lazy Theta* foram selecionados para suprir essa deficiência, já que retornam caminhos mais curtos e suavizados.

O computador utilizado para realizar os experimentos tem a seguinte configuração: CPU: AMD Ryzen 5 3600 6-Core 3.60GHz; RAM: 16 GB; Video Card: NVIDIA GeForce GTX 1660 SUPER; Windows 10 Pro (v. 21H2, Windows Feature Experience Pack 120.2212.4170.0).

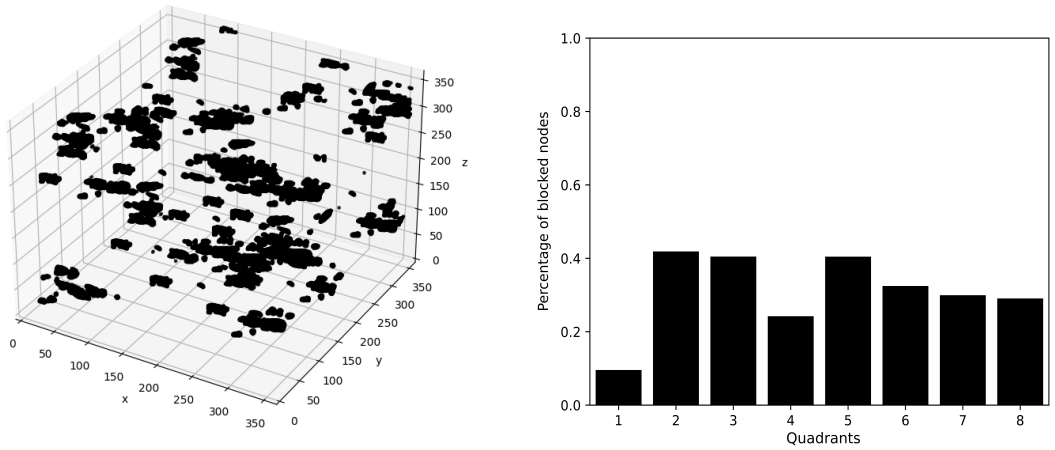
5.4 Resultados dos experimentos

Três diferentes mapas tridimensionais foram planejados para a execução dos experimentos, com diferentes complexidades em termos de obstáculos estáticos dispostos nos cenários de testes. Dessa forma, os mapas 3D desenvolvidos possuem 30%, 45% e 65% de espaço ocupado por obstáculos distribuídos em todas as dimensões. Neste contexto, entende-se por vizinhança padrão a tradicional vizinhança de 26 cubos vizinhos adjascentes em um grid 3D. Visto que a versão de vizinhança padrão dos algoritmos possui mesma distribuição que a vizinhança 3^k quando $k = 3$, espera-se resultados similares entre algoritmos essencialmente iguais, porém resultados baseados em uma das duas abordagens de vizinhança exploradas neste trabalho.

O mapa A possui 30% de nodos bloqueados. É um mapa com espaços mais abertos para os algoritmos explorarem, permitindo que as técnicas de suavização tenham melhores resultados e caminhos menos sinuosos sejam gerados. Neste caso, os obstáculos também foram distribuídos de forma a preencher diferentes áreas do mapa, buscando certa homogeneidade.

No mapa A, as diferenças no comprimento dos caminhos resultantes para mesmos algoritmos, onde cada um possui uma vizinhança diferente (vizinhanças padrão, 3^k com $k = 3$ e 3^k com $k = 4$), não foram significativas, como mostra o gráfico apresentado na Figura 5.6(b). Apenas o algoritmo Lazy Theta* apresenta resultados distintos dos demais devido ao processo de suavização usado, o qual corta muitos nodos ao longo da busca. Entretanto, as variações de k na vizinhança não alteraram os comprimentos finais das diferentes versões dos algoritmos implementados. Apesar de não apresentarem discre-

Figura 5.3: a) Mapa A, com 30% de nodos bloqueados; b) Histograma de distribuição de obstáculos no mapa A.



Fonte: Autora.

pâncias em termos de comprimento do caminho resultante, os resultados de suavização demonstram relevância (Figura 5.6(a)). Naturalmente, as versões que utilizam vizinhança padrão e vizinhança 3^k para $k = 3$ mantém resultados similares. No algoritmo A*, o uso de vizinhança 3^k com $k = 4$ reduziu os nodos no caminho resultante em 13% em relação aos nodos obtidos com A* padrão e A* com vizinhança 3^k para $k = 3$, promovendo a suavização deste caminho. No algoritmo JPS, o uso de vizinhança 3^k com $k = 4$ reduziu os nodos no caminho resultante em 47% em relação ao caminho encontrado pelo JPS padrão. Comparando os algoritmos A* com vizinhança 3^k para $k = 4$ e JPS com vizinhança 3^k para $k = 4$, o segundo apresenta caminhos cerca de 65% mais suavizados. A mesma versão de JPS se comparada ao A* padrão promove uma redução de 70% dos nodos no caminho resultante. No algoritmo Lazy Theta*, a suavização ocorre de forma ainda mais acentuada: apenas a versão padrão do algoritmo já apresenta uma redução em 88% da quantidade de nodos em relação ao A* padrão. A versão de Lazy Theta* com vizinhança 3^k para $k = 4$ apresenta uma redução em 93% na quantidade de nodos no caminho final em relação ao A* padrão. Comparado ao algoritmo JPS com vizinhança 3^k para $k = 4$, a quantidade de nodos nos caminhos retornados pelo Lazy Theta* padrão contém 39% da quantidade de nodos no caminho final da versão do JPS com vizinhança 3^k, para $k = 4$. Quando Lazy Theta* com vizinhança 3^k para $k = 4$ é comparado ao JPS de vizinhança padrão, a quantidade de nodos no caminho resultante de Lazy Theta* com vizinhança 3^k para $k = 4$ é reduzida em cerca de 87% em relação ao JPS. Ao comparar as versões com vizinhança 3^k para $k = 4$ dos algoritmos JPS e Lazy Theta*, a quantidade de nodos

Tabela 5.10: Dados estatísticos do mapa A: Número de nodos no caminho resultante.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,1690	0,0150	276,39	<2e-16
A*	0,0020	1,8970e-05	107,90	<2e-16
A* $3^{k=3}$	-2,441e-15	0,0142	0,0	1
A* $3^{k=4}$	-0,1355	0,0143	-9,48	<2e-16
JPS	-0,5545	0,0143	-38,79	<2e-16
JPS $3^{k=3}$	-0,5545	0,0143	-38,79	<2e-16
JPS $3^{k=4}$	-1,192	0,0143	-83,38	<2e-16
LT*	-2,130	0,0147	-144,68	<2e-16
LT* $3^{k=3}$	-2,129	0,0147	-144,65	<2e-16
LT* $3^{k=4}$	-2,661	0,0146	-181,56	<2e-16

Fonte: Autora.

Tabela 5.11: Dados estatísticos do mapa A: Comprimentos dos caminhos resultantes.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	5,1850	0,0019	2642,946	<2e-16
A*	0,0021	2,815e-06	752,222	<2e-16
A* $3^{k=3}$	-1,084e-13	0,0018	0,000	1
A* $3^{k=4}$	-0,0005	0,0018	-0,318	0,750
JPS	0,0021	0,0018	1,147	0,251
JPS $3^{k=3}$	0,0021	0,0018	1,147	0,251
JPS $3^{k=4}$	-0,0003	0,0018	-0,199	0,842
LT*	-0,0475	0,0019	-24,698	<2e-16
LT* $3^{k=3}$	-0,0475	0,0019	-24,704	<2e-16
LT* $3^{k=4}$	-0,0470	0,0019	-24,550	<2e-16

Fonte: Autora.

nos caminhos retornados pelo Lazy Theta* para $3^{k=4}$ representam 23% da quantidade de nodos no caminho final da versão do JPS para $3^{k=4}$ (uma redução de 76%).

Apesar do algoritmo Lazy Theta* e suas versões com diferentes vizinhanças apresentarem resultados superiores em relação aos demais algoritmos testados no que diz respeito à suavização do caminho, os algoritmos Lazy Theta* demonstram grande desvantagem em tempo de computação (Figura 5.7(a)). Em sistemas voltados para jogos e simulações, este aspecto pode ser de extrema importância. O tempo de execução do algoritmo Lazy Theta* padrão necessita cerca de 72 vezes o tempo de execução exigido pelo algoritmo A* padrão. Quando associado ao uso de vizinhança 3^k para $k = 4$, o Lazy Theta* exige um tempo de execução 60 vezes maior para retornar caminhos. Percebe-se que nos mapas com menos obstáculos (tais como os mapas A e B), ou seja, mais espaços livres para movimentação, as versões Lazy Theta* com vizinhança 3^k para $k = 4$ apresentam melhor desempenho que as versões padrão deste algoritmo com vizinhança $3^{k=3}$.

Tabela 5.12: Dados estatísticos do mapa A: Tempo de execução.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,2050	0,0270	155,322	<2e-16
A*	0,0022	3,251e-05	67,846	<2e-16
A* $3^{k=3}$	0,0384	0,0265	1,451	0,147
A* $3^{k=4}$	1,374	0,0265	51,746	<2e-16
JPS	-1,615	0,0265	-60,902	<2e-16
JPS $3^{k=3}$	-1,169	0,0265	-44,107	<2e-16
JPS $3^{k=4}$	-0,306	0,0266	-11,487	<2e-16
LT*	4,2800	0,0274	155,799	<2e-16
LT* $3^{k=3}$	4,2660	0,0274	155,343	<2e-16
LT* $3^{k=4}$	4,0980	0,0273	149,940	<2e-16

Fonte: Autora.

Isso ocorre pois com mais espaços livres, as presunções otimistas de que os campos de visão entre nodo expandido e vizinhos são livres acabam por satisfeitas, o que resulta em menos revisões realizadas para corrigir os casos em que uma presunção não é satisfeita (caso em que o campo de visão préviamente assumido como livre, seria na verdade obstruído, exigindo uma nova computação dos custos e do $parent$). Caso a etapa precise ser revista, o campo de visão é novamente verificado e os vizinhos do nodo expandido são visitados em busca do vizinho que o expandiu em um passo anterior. Com $k = 4$, o algoritmo realiza saltos mais longos entre nodos consecutivos que compõem o caminho, além de abrir menos nodos.

Em relação ao algoritmo JPS, o Lazy Theta* apresenta um tempo de execução cerca de 363 vezes maior. Quando comparado ao JPS com viznhança 3^k para $k = 4$, é necessário um tempo de execução 98 vezes maior. Ao utilizarmos a vizinhança 3^k para $k = 4$ em ambos os algoritmos, a execução do Lazy Theta* com vizinhança 3^k para $k = 4$ exige um tempo de execução 81 vezes maior para retornar um caminho.

Entre os algoritmos JPS com vizinhança 3^k para $k = 4$ e A* com vizinhança 3^k para $k = 4$, o segundo exige um tempo de execução cerca de 5,36 vezes maior. Se comparado ao o JPS com vizinhança 3^k para $k = 4$, o A* padrão necessita um tempo de execução 1,35 vezes maior. A versão padrão do JPS também apresenta menor tempo de execução que o algoritmo A* com vizinhança 3^k para $k = 4$, sendo que o último exige tempo de execução 19,86 vezes maior.

Em respeito à quantidade de nodos expandidos, o JPS padrão e o JPS com vizinhança 3^k apresentaram os melhores resultados, cerca de 50% menor que a quantidade de nodos usada pelo algoritmo base (Figura 5.7(b)). Na sequência, o algoritmo JPS com vizinhança 3^k, para $k = 4$, apresentou o menor valor para nodos abertos em relação aos

Tabela 5.13: Dados estatísticos do mapa A: Número de nodos abertos.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,3290	0,0143	301,473	<2e-16
A*	0,0022	1,848-05	121,815	<2e-16
A* $3^{k=3}$	2,516e-05	0,0136	0,002	0,999
A* $3^{k=4}$	-0,2727	0,0136	-19,984	<2e-16
JPS	-0,6365	0,0136	-46,635	<2e-16
JPS $3^{k=3}$	-0,6365	0,0136	-46,635	<2e-16
JPS $3^{k=4}$	-0,5858	0,0137	-42,527	<2e-16
LT*	-0,2079	0,0140	-14,799	<2e-16
LT* $3^{k=3}$	-0,2086	0,0140	-14,845	<2e-16
LT* $3^{k=4}$	-0,3640	0,0139	-26,019	<2e-16

Fonte: Autora.

demais algoritmos, sendo em torno de 41% menor em relação ao algoritmo A* padrão.

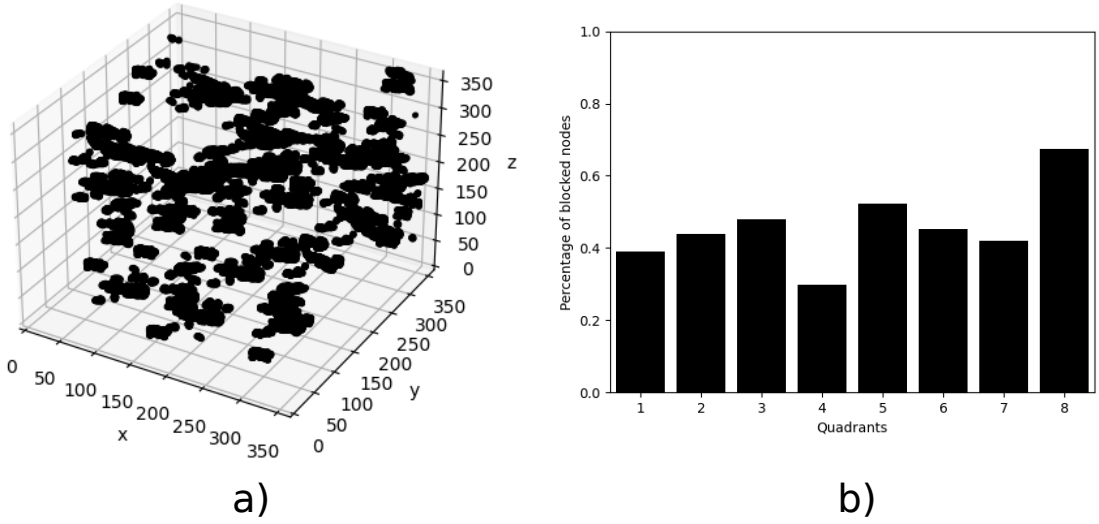
O algoritmo Lazy Theta* padrão obteve uma quantidade de nodos abertos menor que os algoritmos A* padrão e A* com vizinhança 3^k, para $k = 3$, obtendo 30% menos nodos abertos. Em relação ao A* com vizinhança 3^k para $k = 4$, o algoritmo Lazy Theta* padrão obteve 8% menos nodos abertos. A versão com vizinhança 3^k para $k = 4$ do Lazy Theta* apresentou menos nodos abertos que suas versões padrão e com vizinhança 3^k para $k = 3$, sendo este valor 40% menor que o resultado obtido pelo método base (valores de A* padrão e de A* com vizinhança 3^k, para $k = 3$). Ainda, o Lazy Theta* com vizinhança 3^k para $k = 4$ obteve 21% menos nodos abertos que o algoritmo A* com vizinhança 3^k, para $k = 4$.

O algoritmo JPS para $3^{k=4}$ apresentou uma quantidade de nodos abertos 2,4% menor que a quantidade de nodos obtida pelo algoritmo Lazy Theta* com $3^{k=4}$. Em relação à versão padrão do Lazy Theta*, JPS com $3^{k=4}$ obteve 16% menos nodos abertos. Tratando-se da versão padrão do JPS, este algoritmo obteve 18% menos nodos abertos que Lazy Theta* com $3^{k=4}$. Em comparação aos algoritmos Lazy Theta* padrão e Lazy Theta* com $3^{k=3}$, o JPS padrão e com vizinhança $3^{k=3}$ apresentaram uma diferença de cerca de 30% de nodos abertos.

O mapa B, com 45% de volume ocupado por obstáculos, permitiu executar uma avaliação que não estressou o algoritmo de busca. A ideia do mapa B foi manter uma boa distribuição dos obstáculos, mantendo espaços de travessias menores que os usados no mapa A e maiores que os usados na construção do mapa C, que é o que possui maior quantidade de nodos bloqueados entre todos os mapas. No mapa B, espera-se que os caminhos computados sejam menos sinuosos do que aqueles caminhos computados no mapa C. Na Figura 5.4(a), é possível observar que o mapa 3D já apresenta maior vo-

lume de obstáculos em relação ao mapa A. A distribuição destes obstáculos ao longo dos octantes do mapa é demonstrada pelo histograma da Figura 5.4(b).

Figura 5.4: a) Mapa B, com 45% de nodos bloqueados. b) Histograma de distribuição de obstáculos no mapa B.



Fonte: Autora.

No mapa B (Figura 5.4), os comprimentos dos caminhos retornados pelos algoritmos ficaram muito próximos, como apresentado na Figura 5.6(d). Isso pôde ser observado pelo indicador de significância dos dados estatísticos. Entre os algoritmos A*, A* 3k (com k=3), A* 3k (com k=4), JPS, JPS 3k (com k=3) e JPS 3k (com k=4), não foram observadas diferenças significativas. As versões dos algoritmos que utilizam k=4 forneceram caminhos que são menos de 1% mais curtos do que os caminhos obtidos pelas versões dos mesmos algoritmos quando utilizam k=3. Quando as versões do algoritmo Lazy Theta* (padrão, com k=3 e k=4) foram comparadas com os demais algoritmos dos experimentos e suas respectivas versões, os algoritmos Lazy Theta* apresentaram resultados com diferenças significativas em relação aos demais, já que este tipo de algoritmo realiza maior suavização, fornecendo caminhos de maior qualidade. Entre as versões de Lazy Theta* testadas, com uso das diferentes vizinhanças, não houve diferenças significativas nos comprimentos dos caminhos resultantes. Contudo, o algoritmo Lazy Theta* e suas variações apresentaram grande desvantagem em relação aos demais quando os tempos de execução foram avaliados (Figura 5.7(c)). Por exemplo, os algoritmos Lazy Theta* padrão e com vizinhança 3^k para $k = 3$ exigiram cerca de 84 vezes mais tempo quando comparados ao algoritmo A* padrão, tomado como parâmetro base. Isso pode ser verificado por meio da estimativa estatística Exp(4,43) = 84,1835. Já o Lazy Theta* com vizinhança 3^k para $k = 4$ exigiu cerca de 74 vezes mais tempo de execução que o

Tabela 5.14: Dados estatísticos do mapa B: Comprimentos dos caminhos resultantes.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	5,1600	0,0021	2442,598	<2e-16
A*	0,0021	3,019e-06	717,506	<2e-16
A* $3^{k=3}$	-2,606e-10	0,0020	0,000	1
A* $3^{k=4}$	-0,0017	0,0020	-0,839	0,401
JPS	0,0027	0,0020	1,347	0,178
JPS $3^{k=3}$	0,0027	0,0020	1,347	0,178
JPS $3^{k=4}$	-0,0008	0,0020	-0,420	0,675
LT*	-0,0446	0,0020	-21,573	<2e-16
LT* $3^{k=3}$	-0,0455	0,0020	-21,969	<2e-16
LT* $3^{k=4}$	-0,0440	0,0020	-21,329	<2e-16

Fonte: Autora.

Tabela 5.15: Dados estatísticos do mapa B: Tempo de execução.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	3,8380	0,0292	131,44	<2e-16
A*	0,0024	3,439e-05	70,82	<2e-16
A* $3^{k=3}$	0,0461	0,0270	1,71	0,0873
A* $3^{k=4}$	1,6960	0,0270	62,82	<2e-16
JPS	-1,2430	0,0270	-45,88	<2e-16
JPS $3^{k=3}$	-0,7704	0,0270	-28,44	<2e-16
JPS $3^{k=4}$	-0,5136	0,0271	-18,90	<2e-16
LT*	4,4330	0,0276	160,18	<2e-16
LT* $3^{k=3}$	4,4330	0,0276	160,08	<2e-16
LT* $3^{k=4}$	4,3060	0,0276	155,87	<2e-16

Fonte: Autora.

algoritmo A* padrão. Em relação ao algoritmo JPS padrão, este algoritmo demonstrou ser cerca de 291 vezes mais rápido que o Lazy Theta* padrão e sua versão com vizinhança 3^k, para $k = 3$. Se considerarmos $k = 4$ na vizinhança do Lazy Theta*, o JPS padrão foi cerca de 256 vezes mais rápido. Quando usamos a vizinhança 3^k com $k = 4$ ao JPS, ele demonstrou ser cerca de 140 vezes mais rápido que o Lazy Theta* padrão e 123 vezes mais rápido que o Lazy Theta* com vizinhança 3^k, para $k = 4$. Quando comparado ao algoritmo A* padrão, o algoritmo JPS foi 3,4 vezes mais rápido. Associando o JPS à expansão 3^k com $k = 4$, o tempo de execução apresentado passa a ser 1,67 vezes mais rápido em relação ao A* padrão. Quando comparados A* com vizinhança 3^k para $k = 4$ e JPS com vizinhança 3^k para $k = 4$, o algoritmo A* apresentou tempo de execução 9,11 vezes maior.

Para nodos abertos (Fig. 5.7(d)), as versões dos três algoritmos testados que utilizaram vizinhança 3^k com $k = 4$ obtiveram valores menores em relação às versões padrão

Tabela 5.16: Dados estatísticos do mapa B: Número de nodos abertos.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,5130	0,0144	312,309	<2e-16
A*	0,0019	1,828e-05	105,046	<2e-16
A* $3^{k=3}$	0,0005	0,0133	0,041	0,967
A* $3^{k=4}$	-0,2186	0,0133	-16,374	<2e-16
JPS	-0,5390	0,0133	-40,361	<2e-16
JPS $3^{k=3}$	-0,5390	0,0133	-40,361	<2e-16
JPS $3^{k=4}$	-0,8208	0,0133	-61,450	<2e-16
LT*	-0,2256	0,0135	-16,618	<2e-16
LT* $3^{k=3}$	-0,2274	0,0135	-16,745	<2e-16
LT* $3^{k=4}$	-0,3790	0,0135	-28,010	<2e-16

Fonte: Autora.

e com vizinhança 3^k para $k = 3$. Realizando saltos maiores entre vizinhos, menos nodos diretamente conectados são expandidos. Dentre os três algoritmos testados, o algoritmo JPS expandiu menor número de nodos em relação aos demais. Ainda, dentre as implementações testadas, o JPS associado à vizinhança 3^k com $k = 4$ obteve o menor valor de nodos abertos entre todos os demais algoritmos. Em relação ao A* padrão, o algoritmo JPS com vizinhança 3^k para $k = 4$ resultou em 41% menos nodos abertos. O JPS com vizinhança 3^k para $k = 4$ obteve um número de nodos abertos 45% menor que o A* com vizinhança 3^k e $k = 4$. A quantidade de nodos abertos pelo algoritmo JPS com vizinhança 3^k para $k = 4$ foi 25% menor do total de nodos abertos da versão padrão deste algoritmo. O JPS com vizinhança 3^k para $k = 4$ abriu 45% menos nodos em relação ao total de nodos abertos dos algoritmos Lazy Theta* padrão e Lazy Theta* com vizinhança 3^k para $k = 3$. Já o JPS com vizinhança 3^k para $k = 4$ abriu 35% menos nodos em relação ao total de nodos abertos da versão de Lazy Theta* com vizinhança 3^k para $k = 4$.

Em relação à suavização promovida pelos algoritmos testados (Fig. 5.6(a)), os resultados demonstram que o incremento de k na vizinhança resulta em uma suavização significativa. Visto que a técnica de suavizar um caminho consiste em retirar nodos desnecessários destes caminhos, o uso de vizinhança 3^k para valores de k a partir de 4 promoveram suavização quando os caminhos computados com essa técnica foram comparados aos caminhos resultantes das versões padrão dos algoritmos. O uso de vizinhança 3^k para $k = 4$ no algoritmo A* reduziu em 13% a quantidade de nodos que compõem o caminho retornado pela versão padrão do A*, que utiliza os tradicionais 26 vizinhos adjascentes. No algoritmo JPS, o uso da vizinhança 3^k para $k = 4$ promoveu uma redução de 47% do número de nodos ao longo do caminho retornado pela implementação padrão deste algoritmo, também utilizando os 26 vizinhos adjascentes. Em relação ao A* pa-

Tabela 5.17: Dados estatísticos do mapa B: Número de nodos no caminho resultante.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,1060	0,0146	279,561	<2e-16
A*	0,0021	1,845e-05	117,282	<2e-16
A* $3^{k=3}$	2,920e-10	0,0141	0,00	1
A* $3^{k=4}$	-0,1365	0,0141	-9,675	<2e-16
JPS	-0,5668	0,0141	-40,163	<2e-16
JPS $3^{k=3}$	-0,5668	0,0141	-40,163	<2e-16
JPS $3^{k=4}$	-1,199	0,0141	-84,978	<2e-16
LT*	-1,9850	0,0143	-138,533	<2e-16
LT* $3^{k=3}$	-1,9810	0,0143	-138,152	<2e-16
LT* $3^{k=4}$	-2,5370	0,0142	-177,581	<2e-16

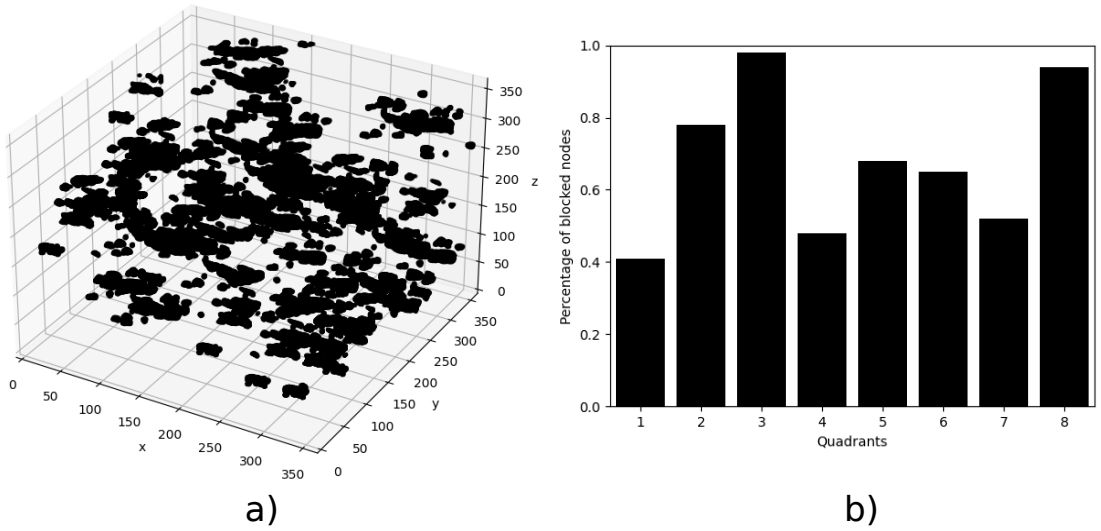
Fonte: Autora.

drão, tomado como método base, o uso do algoritmo JPS em conjunto com a vizinhança 3^k para $k = 4$ reduziu a quantidade de nodos do caminho resultante em 70% para as mesmas distâncias percorridas. O algoritmo Lazy Theta* quando associado à expansão de vizinhança 3^k para $k = 4$ reduziu os nodos do caminho resultante em 42% quando comparado à sua implementação padrão utilizando 26 vizinhos adjascentes. Em relação ao A* padrão, tomado como algoritmo base para comparação, o uso do algoritmo Lazy Theta* associado à vizinhança 3^k para $k = 4$ reduziu em 92% a quantidade de nodos do caminho resultante para as mesmas distâncias percorridas. Apesar de obter os tempos de execução mais elevados em relação aos demais algoritmos, o algoritmo Lazy Theta* com vizinhança 3^k para $k = 4$ apresenta caminhos de maior qualidade em relação aos demais algoritmos, apresentando boa suavização do caminho resultante.

O mapa C apresenta menores espaços livres entre os obstáculos, já que estes obstáculos ocupam 65% do volume deste mapa 3D (Figura 5.5(a)). Essa característica pode acabar resultando em mais curvas e desvios nos caminhos computados pelos algoritmos. Um mapa com maior número e maior distribuição de obstáculos se torna desafiador para tais algoritmos de busca, ainda mais quando eles buscam suavizar o caminho produzido. Neste sentido, a suavização pode ser menor em caminhos com muitos obstáculos, visto que o caminho final pode apresentar maior número de pontos de inflexão. A distribuição dos obstáculos entre os oito quadrantes no mapa C é caracterizada pelo histograma na Figura 5.5(b).

No mapa C (5.5), os dados de comprimento de caminhos obtidos foram muito próximos entre os algoritmos testados (Figura 5.6(f)). Contudo, quanto trata-se de suavização, as diferenças entre os resultados foram estatisticamente significativas (Figura 5.6(e)). Entre as versões do A*, comparando a versão padrão deste algoritmo (que se

Figura 5.5: a) Mapa C, com 65% de nodos bloqueados. b) Histograma de distribuição de obstáculos no mapa C.



Fonte: Autora.

Tabela 5.18: Dados estatísticos do mapa C: Comprimentos dos caminhos resultantes.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	5,1710	0,0019	2611,845	<2e-16
A*	0,0021	2,820e-06	760,441	<2e-16
A* $3^{k=3}$	7,577e-14	0,0019	0,000	1
A* $3^{k=4}$	-0,0008	0,0019	-0,463	0,6326
JPS	0,0047	0,0019	2,465	0,0221
JPS $3^{k=3}$	0,0047	0,0019	2,465	0,0221
JPS $3^{k=4}$	0,0005	0,0019	0,293	0,3540
LT*	-0,0428	0,0019	-22,089	<2e-16
LT* $3^{k=3}$	-0,0416	0,0019	-21,440	<2e-16
LT* $3^{k=4}$	-0,04234	0,0019	-21,843	<2e-16

Fonte: Autora.

iguala ao uso de vizinhança 3^k quando $k = 3$) com a versão que utiliza vizinhança 3^k para $k = 4$, ocorreu uma redução de 13% na quantidade de nodos do caminho final computado. Comparando a versão padrão do JPS, igualada à versão 3^k com $k = 3$, com a versão que utiliza vizinhança 3^k para $k = 4$, ocorreu uma redução de 46% nos nodos que compõem o caminho final. Já o algoritmo Lazy Theta* que utiliza vizinhança 3^k com $k = 4$ reduziu em cerca de 43% a quantidade de nodos que compõem o caminho final em relação às suas versões padrão e com vizinhança 3^k com $k = 3$. A característica de suavização do algoritmo promoveu uma maior retirada de nodos considerados desnecessários no caminho. Quando comparado ao algoritmo A* padrão, o Lazy Theta* com vizinhança 3^k para $k = 4$ demonstrou uma suavização de 91% no caminho. Quando comparado ao A* com vizinhança 3^k para $k = 4$, o Lazy Theta* com vizinhança 3^k para $k = 4$ pro-

Tabela 5.19: Dados estatísticos do mapa C: Número de nodos no caminho resultante.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,1150	0,0140	292,08	<2e-16
A*	0,0021	1,756e-05	122,28	<2e-16
A* $3^{k=3}$	4,317e-14	0,0137	0,00	1
A* $3^{k=4}$	-0,1384	0,0137	-10,10	<2e-16
JPS	-0,5638	0,0137	-41,16	<2e-16
JPS $3^{k=3}$	-0,5638	0,0137	-41,16	<2e-16
JPS $3^{k=4}$	-1,1930	0,0137	-87,07	<2e-16
LT*	-1,8650	0,0137	-135,26	<2e-16
LT* $3^{k=3}$	-1,8630	0,0138	-135,91	<2e-16
LT* $3^{k=4}$	-2,4310	0,0137	-176,50	<2e-16

Fonte: Autora.

moveu uma redução de 89% da quantidade de nodos do caminho final. Em relação ao algoritmo JPS, Lazy Theta* com vizinhança 3^k com $k = 4$ reduziu em 84% a quantidade de nodos do caminho final. Comparando o algoritmo Lazy Theta* com vizinhança 3^k para $k = 4$ com o JPS com vizinhança 3^k para $k = 4$, o primeiro obteve caminhos com 71% menos nodos que o segundo. O JPS com vizinhança 3^k para $k = 4$ promoveu uma redução de 65% no número de nodos obtido pelo A* com vizinhança 3^k para $k = 4$. Comparando o A* padrão em relação ao algoritmo JPS com vizinhança 3^k para $k = 4$, a versão do JPS testada reduziu em 70% a quantidade de nodos do caminho final.

Apesar de melhores resultados em termos de suavização e pequena melhoria em termos de menor comprimento do caminho, o Lazy Theta* retornou resultados piores em termos de tempo de execução (Figura 5.7(e)). Em relação ao método base A* padrão, o Lazy Theta* necessitou 39 vezes mais tempo de execução. Em relação ao algoritmo A* com vizinhança 3^k para $k = 4$, o Lazy Theta* com vizinhança 3^k para $k = 4$ necessitou um tempo 7,21 vezes maior para retornar um caminho. Quando comparado ao algoritmo JPS, este algoritmo também apresenta desvantagem em termos de tempo de execução, sendo 182 mais lento. Em relação ao JPS com vizinhança 3^k para $k = 4$, a versão do Lazy Theta* com essa mesma vizinhança foi 32,9 vezes mais demorada para retornar um caminho. Avaliando os dados do JPS em relação ao método base, a versão padrão do JPS exigiu menor tempo de execução, sendo 4,65 vezes mais rápido. Quando trata-se da versão com vizinhança 3^k para $k = 4$ de ambos algoritmos, o JPS demonstra semelhante vantagem em desempenho. Neste caso, a versão com vizinhança 3^k para $k = 4$ do algoritmo A* apresentou um tempo de execução 4,5 vezes maior que JPS com vizinhança 3^k para $k = 4$. Comparando os tempos de execução entre JPS com vizinhança 3^k para $k = 4$ e o algoritmo a* padrão, os valores ficaram muito próximos, não apresentando diferenças

Tabela 5.20: Dados estatísticos do mapa C: Tempo de execução.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,1170	0,0288	142,520	<2e-16
A*	0,0022	3,318e-05	67,922	<2e-16
A* $3^{k=3}$	0,0465	0,0270	1,721	0,0852
A* $3^{k=4}$	1,506	0,0270	55,691	<2e-16
JPS	-1,537	0,0270	-56,715	<2e-16
JPS $3^{k=3}$	-1,101	0,0270	-40,648	<2e-16
JPS $3^{k=4}$	-0,0117	0,0274	-0,429	0,6683
LT*	3,6680	0,0275	132,934	<2e-16
LT* $3^{k=3}$	3,4830	0,0275	126,616	<2e-16
LT* $3^{k=4}$	3,4820	0,0275	126,524	<2e-16

Fonte: Autora.

Tabela 5.21: Dados estatísticos do mapa C: Número de nodos abertos.

Algoritmo	Estimativa	Erro padrão	Valor T	Pr(>|t|)
Intercept	4,5860	0,0150	305,529	<2e-16
A*	0,0022	1,816e-05	129,870	<2e-16
A* $3^{k=3}$	-0,0001	0,0146	-0,012	0,99
A* $3^{k=4}$	-0,2679	0,0146	-18,321	<2e-16
JPS	-0,7106	0,0146	-48,588	<2e-16
JPS $3^{k=3}$	-0,7106	0,0146	-48,588	<2e-16
JPS $3^{k=4}$	-0,5358	0,0146	-36,620	<2e-16
LT*	-0,3533	0,0146	-24,100	<2e-16
LT* $3^{k=3}$	-0,3506	0,0146	-23,917	<2e-16
LT* $3^{k=4}$	-0,5110	0,0146	-34,865	<2e-16

Fonte: Autora.

significativas.

Novamente, o algoritmo JPS destacou-se com o menor valor de nodos abertos (Figura 5.7(f)). Com um acréscimo de 16% em relação às implementações padrão deste algoritmo e com vizinhança 3^k para $k = 3$, o JPS com vizinhança 3^k para $k = 4$ também obteve os menores valores em relação aos demais algoritmos testados. Em relação ao algoritmo A* padrão, o JPS com vizinhança 3^k para $k = 4$ apresentou 41% menos nodos abertos para encontrar caminhos de mesmas distâncias. Em relação ao A* com vizinhança 3^k para $k = 4$, este algoritmo exigiu um número de nodos abertos 23% menor. O algoritmo A* com vizinhança 3^k para $k = 4$ também apresentou desvantagem no que diz respeito aos nodos abertos em relação à implementação padrão do algoritmo JPS, demonstrando obter 35% mais nodos abertos. O algoritmo Lazy Theta* foi o algoritmo que ficou entre os outros dois (A* e JPS) na classificação dos resultados obtidos para nodos abertos. Em relação ao JPS com vizinhança 3^k para $k = 4$, as versões padrão e versão

3^k com $k = 3$ de Lazy Theta* obtiveram 16% mais nodos abertos. Quando comparados JPS e Lazy Theta*, ambos em suas versões 3^k com $k = 4$, o Lazy Theta* apresentou 2% mais nodos abertos. Quando o Lazy Theta* com vizinhança 3^k com $k = 4$ é comparado às versões padrão e com 3^k para $k = 3$ do algoritmo JPS, o primeiro demonstrou um acréscimo de 18% de nodos abertos em relação às versões citadas do JPS. Comparando Lazy Theta* com vizinhança 3^k para $k = 4$ com o algoritmo base A* padrão, o primeiro apresentou 40% menos nodos abertos. Comparando as versões que utilizaram a vizinhança 3^k para $k = 4$ de ambos os algoritmos (Lazy Theta* e A*), a versão do algoritmo Lazy Theta* obteve 21% menos nodos abertos. Essa é uma diferença menor do que a encontrada quando as versões padrão dos dois algoritmos foram comparadas: neste caso, o Lazy Theta* apresentou 30% menos nodos abertos.

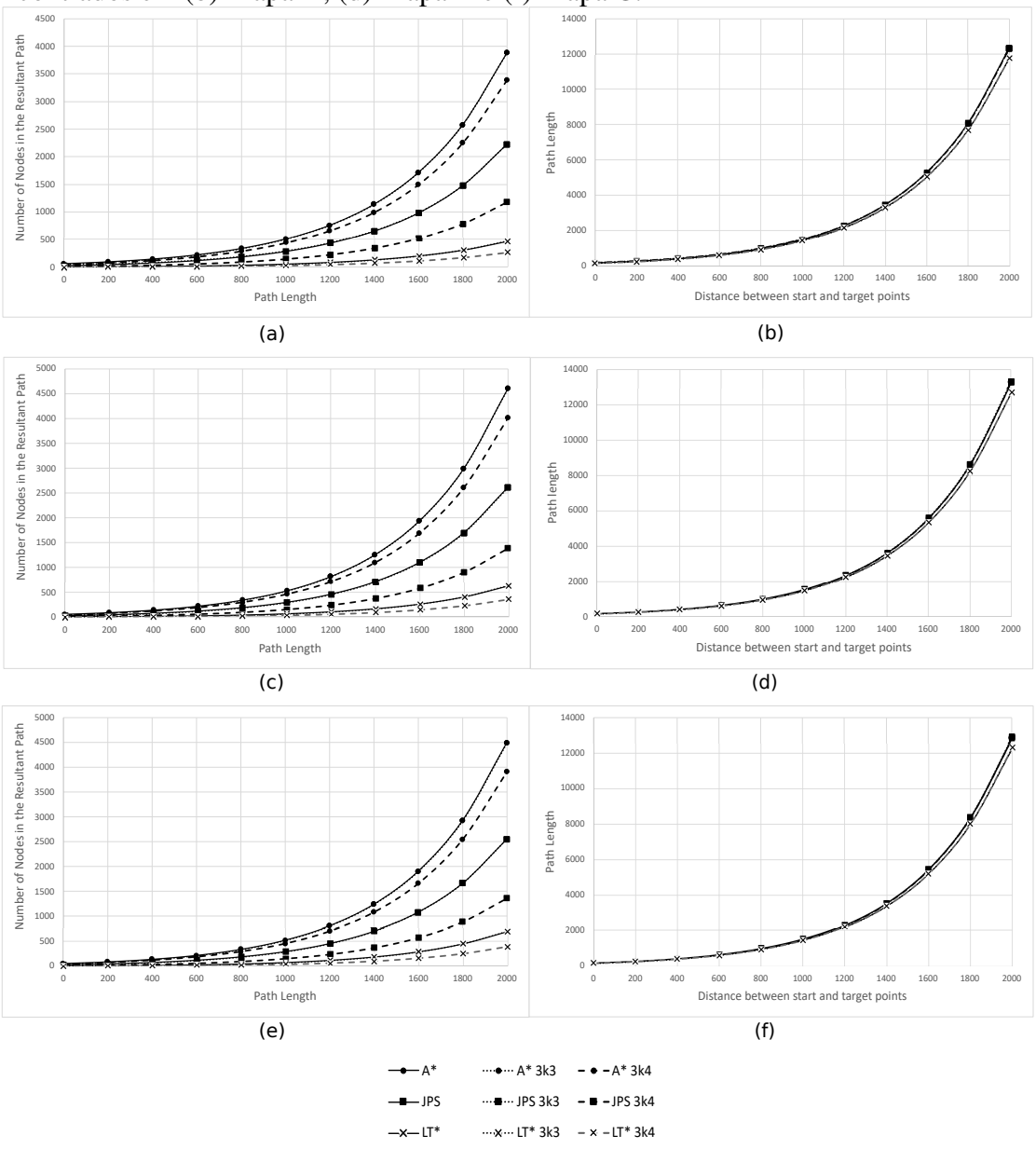
5.5 Análise dos resultados para os algoritmos testados nos três mapas

Os testes iniciais que avaliam os impactos no incremento de k demonstram que os tempos de execução se tornam relativamente altos a partir do valor 5 como parâmetro para a variável k, impactando negativamente o uso dessa abordagem. Isso deve-se ao fato da vizinhança se tornar realmente grande, visto que a sua cardinalidade cresce exponencialmente. Ainda, acaba não apresentando boa relação custo-benefício entre tempo de execução e suavização do caminho. De forma geral, os experimentos indicam que o valor 4 atribuído para k pode oferecer um relevante custo-benefício entre aspectos de tempo de execução, nodos abertos e suavização. Mesmo assim, uma heurística específica para a vizinhança 3^k pode ser elaborada e proposta em trabalhos futuros, a fim de melhorar o desempenho dos algoritmos que utilizam essa vizinhança. Com essa possibilidade, uma ampliação dos valores atribuídos a k pode ser novamente analisada.

Com exceção dos comprimentos dos caminhos resultantes, os resultados dos experimentos finais com os algoritmos A*, JPS e Lazy Theta* apresentam diferenças estatisticamente significativas. Somente os resultados entre as versões padrão e com vizinhança 3^k de cada algoritmo não demonstram essas diferenças, visto que as seleções de vizinhos usada por esses algoritmos são similares.

Em termos de suavização, resultados obtidos nos três mapas 3D demonstram que os algoritmos testados apresentam a mesma ordem de classificação quando o menor número de nodos no caminho resultante é analisado. As versões do algoritmo Lazy Theta* apresentam os menores valores de número de nodos no caminho retornado, seguidas pelas

Figura 5.6: Gráficos referentes aos números de nodos no caminho resultante para (a) Mapa A, (c) Mapa B e (e) Mapa C; e gráficos referentes aos comprimentos dos caminhos encontrados em (b) Mapa A, (d) Mapa B e (f) Mapa C.



Fonte: Autora.

versões do JPS e, por fim, as versões do algoritmo A*. Os algoritmos que utilizam vizinhança $3^{k=4}$ apresentam valores de nodos no caminho resultante abaixo das versões que utilizam vizinhança $3^{k=3}$ e vizinhança padrão para todos os algoritmos. No mapa A, para distâncias de cerca de 1km, a quantidade de nodos ao longo do caminho resultante dos algoritmos fica na faixa de 34 nodos (para o Lazy Theta* com vizinhança $3^{k=4}$, que obtém o menor resultado) a 500 nodos (para o A* padrão e com vizinhança $3^{k=3}$, os quais apresentam os maiores resultados). Nesta faixa de valores, o algoritmo Lazy Theta* obteve menor valor em nodos ao longo do caminho (aproximando seus valores de 0 no gráfico Fig. 5.7(b)) visto que consegue realizar uma boa suavização. Por sua vez, o algoritmo A* obteve valores mais próximos à faixa de 500 nodos, enquanto o JPS obteve valores intermediários entre os dois algoritmos (A* e Lazy Theta*). No mapa B, para distâncias de cerca de 1km, a quantidade de nodos ao longo do caminho resultante da execução dos algoritmos fica entre 34, novamente para Lazy Theta* com vizinhança $3^{k=4}$, e 528, para as versões de A* padrão e com vizinhança $3^{k=3}$. No mapa C, a faixa de valores de nodos no caminho retornado fica mais próxima, já que ambos têm mais obstáculos, apresentando o valor máximo de 524 nodos no caminho resultante para caminhos de 1km, para as versões do algoritmo A* padrão e com vizinhança $3^{k=3}$. Neste mapa, o qual possui mais obstáculos, aumentando a complexidade do ambiente, o valor mínimo de nodos no caminho resultante é de 46, para Lazy Theta* com vizinhança $3^{k=4}$, o qual é o menor valor obtido para essa métrica.

Em termos de tempo de execução e nodos abertos, no mapa A, os algoritmos que apresentam maior número de nodos abertos são o A* padrão e A* com vizinhança $3^{k=3}$. Ainda assim, os algoritmos A* não obtêm os maiores tempos de execução. Por sua vez, o JPS apresenta os menores valores para nodos abertos, demonstrando menor consumo de memória. Outra grande vantagem do algoritmo foi seu desempenho em termos de tempo de execução, obtendo os menores valores em relação aos algoritmos testados. No aspecto de nodos abertos, as versões de Lazy Theta* apresentam resultados intermediários entre as versões dos outros dois algoritmos testados (A* e JPS), mostrando-se próximas do algoritmo A* com vizinhança $3^{k=4}$ no que diz respeito a nodos abertos. Entretanto, os gráficos dos tempos de execução das versões do algoritmo Lazy Theta* apresentam uma curva exponencial de rápido crescimento, apresentando valores muito elevados em relação aos demais algoritmos testados (Figura 5.7(a)). Em algumas aplicações, o bom desempenho pode ser um fator primordial, permitinddo reconsiderar o uso do algoritmo em grandes ambientes tridimensionais.

Quando o uso do mapa A é comparado ao uso do mapa B, os tempos de execução de alguns algoritmos no mapa B são reduzidos em aproximadamente 50% de acordo com o algoritmo e vizinhança utilizada. A ordem de classificação dos tempos de execução dos algoritmos também é mantida. No mapa B, o algoritmo Lazy Theta* mantém a característica de aumentar rapidamente os tempos de execução com o aumento das distâncias percorridas. Em termos de nodos abertos, de forma geral, neste mapa B também foram obtidos valores menores em relação ao mapa A.

No mapa C, os tempos obtidos para as versões do algoritmo A* mantêm resultados próximos aos resultados obtidos com o uso do mapa B. As versões padrão e com vizinhança $3^{k=3}$ do algoritmo JPS apresentam valores de tempo de execução muito semelhantes nos terrenos B e C, enquanto a versão com vizinhança $3^{k=4}$ do algoritmo, quando executada no terreno C, apresentou quase o dobro do tempo para encontrar caminhos de mesmas distâncias em relação ao mapa B. Por sua vez, o algoritmo Lazy Theta* e versões deste apresentam os resultados de tempo de execução reduzidos em quase 50% neste mapa C. Ainda assim, os tempos de execução são muito elevados em relação aos demais algoritmos (mais de 87% maiores). Em termos de nodos abertos, os valores obtidos no mapa C ficaram muito próximos aos valores obtidos com o mapa B para todos os algoritmos. Dessa forma, o ranqueamento de performance dos algoritmos testados foi mantido nos testes realizados no mapa C.

Analisando a acurácia e qualidade dos caminhos encontrados pelos algoritmos testados ao utilizar a vizinhança 3^k, o comprimento e suavização do caminho foram avaliados. Como algoritmo base para as análises realizadas, o algoritmo A* padrão é utilizado. Para um algoritmo apresentar caminhos ótimos, ele deve apresentar resultados menores ou iguais ao algoritmo base. Para verificar a qualidade do caminho, é necessário que além de comprimentos dentro dos limites do método base, o algoritmo também apresente menores valores para número de nodos no caminho resultante. Conforme apresentado na seção 5.4, as diferenças entre os resultados de comprimento dos caminhos resultantes não demonstraram ser estatisticamente significativas para a maioria dos algoritmos testados. Na Tabela 5.22, os comprimentos do caminho são iguais para os algoritmos A* (método base), A* com vizinhança $3^{k=3}$, A* com vizinhança $3^{k=4}$, JPS, JPS com vizinhança $3^{k=3}$ e JPS com vizinhança $3^{k=4}$. As variações do algoritmo Lazy Theta* apresentam diferenças significativas em relação ao algoritmo base A*, em todos os mapas. Na Tabela 5.22, o comprimento do caminho retornado pelo algoritmo Lazy Theta* é 5% menor em relação ao comprimento do caminho obtidos pelo método base no mapa A. Nos mapas B e C, os

Tabela 5.22: Percentuais de comprimentos dos caminhos resultantes dos algoritmos testados em relação ao método base.

Algoritmo	Mapa "A"	Mapa "B"	Mapa "C"
A*	100%	100%	100%
A* $3^{k=3}$	100%	100%	100%
A* $3^{k=4}$	100%	100%	100%
JPS	100%	100%	100%
JPS $3^{k=3}$	100%	100%	100%
JPS $3^{k=4}$	100%	100%	100%
LT*	95%	96%	96%
LT* $3^{k=3}$	95%	96%	96%
LT* $3^{k=4}$	95%	96%	96%

Fonte: Autora.

comprimentos dos caminhos obtidos pelo emprego das versões de Lazy Theta* são 4% menores em relação ao método base. Devido à menor quantidade de obstáculos no mapa A, os comprimentos dos caminhos retornados pelas versões do algoritmo Lazy Theta* são menores que os valores obtidos nos mapas B e C.

Visto que os algoritmos garantem o retorno de caminhos ótimos, esse aspecto pode ser analisado em conjunto com a suavização do caminho, que é avaliada por meio da redução de nodos no caminho final (Tabela 5.23). Naturalmente, o algoritmo A* com vizinhança $3^{k=3}$ obtém resultados iguais aos do método base, o que se mantém em todos os mapas. Por sua vez, o algoritmo A* com vizinhança $3^{k=4}$ apresenta uma redução em 13% no número de nodos no caminho resultante em relação ao método base. Neste caso, o algoritmo A* com vizinhança $3^{k=4}$ melhora a qualidade do caminho em relação ao A*. Os algoritmos JPS e JPS com vizinhança $3^{k=3}$ também apresentam resultados similares. Ambos demonstram uma redução em 43% no número de nodos do caminho retornado em relação ao método base. O algoritmo JPS com vizinhança $3^{k=4}$ reduz a quantidade de nodos do caminho retornado pelo algoritmo A* em mais de 50%. Dessa forma, o número de nodos no caminho resultante do JPS com vizinhança $3^{k=4}$ apresenta 30% da quantidade de nodos no caminho resultante do método base nos mapas A, B e C, indicando uma melhora significativa na qualidade do caminho em todos os mapas. A versão de JPS que utiliza vizinhança $3^{k=4}$ também melhora a qualidade do caminho em relação à versão padrão de JPS. Quanto às versões de Lazy Theta*, elas promovem uma suavização considerável no caminho, reduzindo de forma acentuada a quantidade de nodos no caminho resultante da execução deste algoritmo. Em relação ao método base, as versões de Lazy Theta* padrão e com vizinhança $3^{k=3}$ reduzem a quantidade de nodos no caminho final em 88% no mapa A, em 86% no mapa B, e em 84% no mapa C. A versão

Tabela 5.23: Percentuais de número de nodos no caminho resultante dos algoritmos testados em relação ao método base.

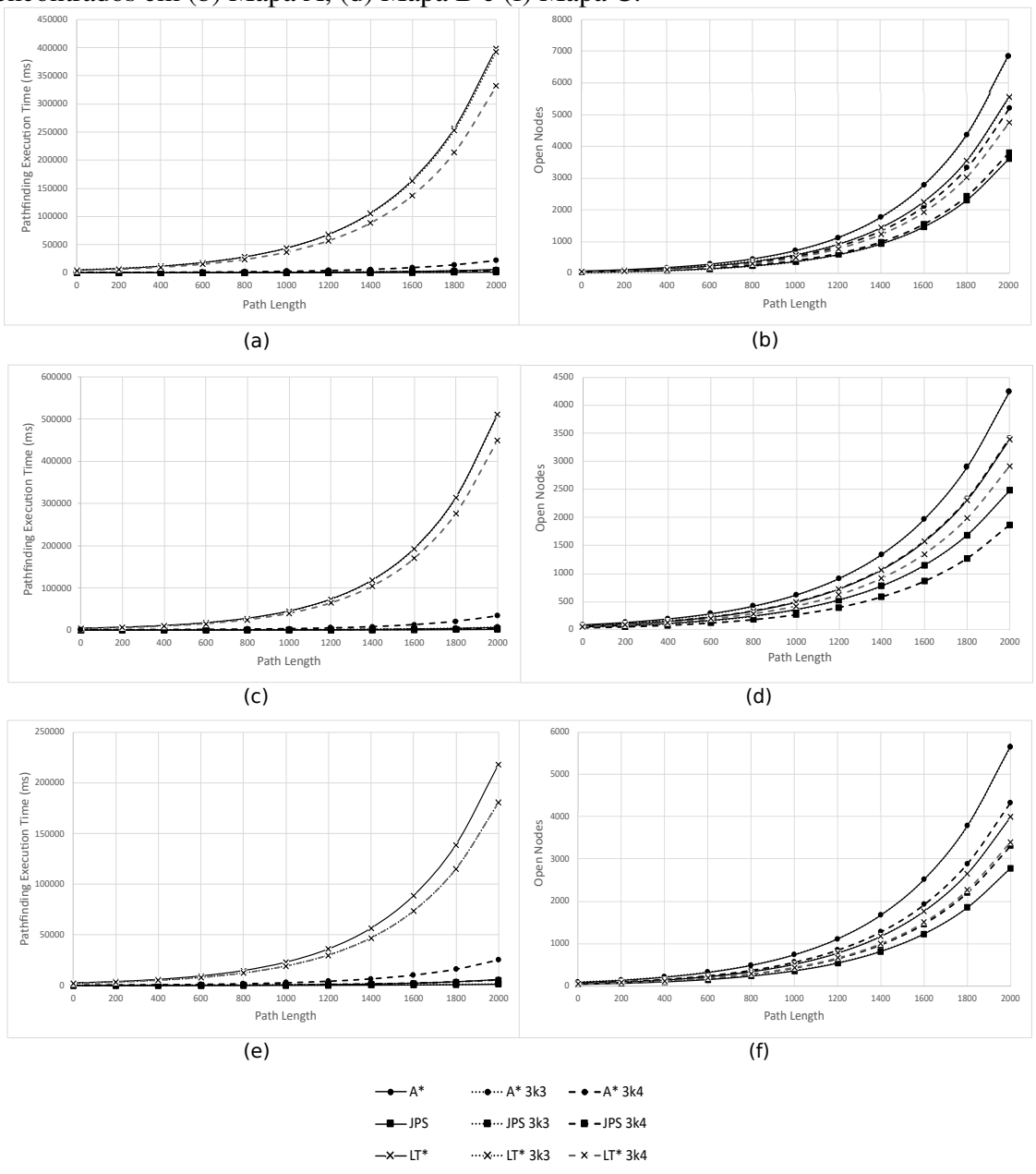
Algoritmo	Mapa "A"	Mapa "B"	Mapa "C"
A*	100%	100%	100%
A* $3^{k=3}$	100%	100%	100%
A* $3^{k=4}$	87%	87%	87%
JPS	57%	57%	57%
JPS $3^{k=3}$	57%	57%	57%
JPS $3^{k=4}$	30%	30%	30%
LT*	12%	14%	16%
LT* $3^{k=3}$	12%	14%	16%
LT* $3^{k=4}$	7%	8%	9%

Fonte: Autora.

de Lazy Theta* com vizinhança $3^{k=4}$ reduz em 93% a quantidade de nodos no caminho final quando comparado a quantidade de nodos no caminho final do método base no mapa A. No mapa B essa redução ocorre em 92%, enquanto no mapa C a redução é de 91% na quantidade de nodos no caminho final em relação ao método base. As versões de Lazy Theta* melhoram de forma considerável a qualidade dos caminhos nos três mapas, melhorando tanto a suavização do caminho quanto o comprimento do caminho. Com o uso de vizinhança 3^k, tendo $k = 4$, os resultados das métricas de avaliação da qualidade dos caminhos retornados são ainda melhores.

Diante dos resultados experimentais obtidos neste trabalho, prós e contras relacionados ao uso de cada algoritmo foram analisados. As versões de A* apresentam ótimo desempenho, porém apresentam os maiores valores em nodos abertos. O uso de vizinhança $3^{k=4}$ promove uma suavização significativa, mas não tão eficaz quanto aquela obtida com o emprego dos outros algoritmos testados. O algoritmo Lazy Theta* apresenta uma suavização muito eficaz e reduzida quantidade em nodos abertos. Entretanto, os tempos de execução deste algoritmo são altíssimos, inviabilizando o Lazy Theta* em contextos como, por exemplos, o desenvolvimento de sistemas de simulação, onde o fator tempo de execução é primordial para a fluência das simulações realizadas em ambientes virtuais que devem ser o mais realistas possível. As versões de JPS apresentam uma relação custo-benefício positiva considerando os aspectos avaliados. Especialmente, o algoritmo JPS com vizinhança $3^{k=4}$ demonstra benefícios em todos os aspectos: além de encontrar caminhos ótimos, conclui esta tarefa com o melhor desempenho dentre os algoritmos testados, além do menor consumo de memória (visto que dentre os algoritmos ele obtém menor número de nodos abertos) e suavização do caminho.

Figura 5.7: Gráficos referentes aos tempos de execução para (a) Mapa A, (c) Mapa B e (e) Mapa C; e gráficos referentes números de nodos abertos durante algoritmo de busca encontrados em (b) Mapa A, (d) Mapa B e (f) Mapa C.



Fonte: Autora.

6 CONCLUSÃO

Este trabalho investiga a ampliação da exploração de vizinhança em algoritmos de busca de caminhos, dada uma estrutura de representação de um ambiente 3D. Devido a necessidade de explorar estruturas e algoritmos voltados à resolução de problemas de busca em espaços tridimensionais em diferentes aplicações, o trabalho contribui de diferentes formas para a pesquisa nesta área. Nos últimos anos, por exemplo, tais ambientes tridimensionais têm recebido relevante atenção, visto o crescente uso destes em aplicações voltadas à jogos, sistemas de simulação, robótica, entre outras.

Um aspecto ainda fracamente explorado na literatura, principalmente quando trata-se de problemas de busca de caminhos, é o uso de grandes vizinhanças. Em algoritmos de busca de caminhos, a carência de literatura abordando exploração em grandes vizinhanças é ainda maior. Além disso, os poucos trabalhos abordando o assunto são majoritariamente voltados a ambientes 2D. A exemplo disso, trabalhos (RIVERA; HERNÁNDEZ; BAIER, 2017) (RIVERA et al., 2020) motivadores para a nossa pesquisa são apresentados, os quais introduzem o uso de *"Vizinhança* 2^k*"* para busca de caminhos em grids. Entre outros aspectos, esses trabalhos demonstram que a ampliação da vizinhança proporciona uma melhoria na qualidade dos caminhos resultantes. Partindo desta ideia, o presente trabalho buscou aprofundar este tópico de pesquisa tanto na área de ampliação de vizinhança em algoritmos de busca de caminhos, quanto na exploração de ambientes 3D, aliando ambos aspectos na investigação e proposta de novas soluções.

Uma das principais contribuições deste trabalho é apresentar a *"Vizinhança* 3^k*"*, uma proposta de ampliação de vizinhança em relação à vizinhança convencional para grids cúbicos. Por meio de somas de vetores consecutivos, a expansão da vizinhança ocorre tanto em diagonais de duas coordenadas quanto em diagonais de três coordenadas, tendo em vista obter uma boa distribuição dos nodos vizinhos, a ser explorada durante o processo de computação de caminhos. Além disso, o trabalho desenvolvido mantém a possibilidade de utilizar os vizinhos adjascentes, inclusos na vizinhança expandida.

Ao utilizar técnicas que ampliem a expansão de vizinhos, a possibilidade de caminhos é ampliada. Além disso, a movimentação torna-se mais direta, consequentemente, mais realista, o que é desejável em muitas aplicações de jogos e sistemas de simulação. Isso possibilita ao algoritmo de busca realizar uma forma de suavização do caminho computado. Transcendendo a exploração convencional de 6 vizinhos, para movimentos cardeais, ou 26 vizinhos, considerando a inclusão de movimentos diagonais, a pesquisa

em torno de ampliação de vizinhança desenvolvida neste trabalho permite implementar tal técnica em diferentes algoritmos de busca voltados para ambientes 3D.

Os resultados experimentais apresentados neste trabalho demonstram que a vizinhança proposta pode ser aplicada em diferentes algoritmos de busca de caminhos, de forma a contribuir no processo de suavização dos caminhos retornados. Isso implica na melhora da qualidade dos caminhos resultantes (neste caso, caminhos com menor número de curvas), observada em todos os algoritmos testados nos experimentos. Ainda, tal técnica pode promover suavização de caminhos mesmo quando aplicada a implementações simplificadas e amplamente utilizadas na indústria de desenvolvimento de jogos e sistemas de simulação, como o tradicional A*, que essencialmente não possui o processo de suavização. Em resumo, o trabalho apresenta:

1. Proposta de grande vizinhança 3^k a ser aplicada em estruturas de representação de ambientes tridimensionais;
2. Implementação que possibilita suavização do caminho mesmo em algoritmos de planejamento de caminhos cujas implementações são mais simplificadas, os quais não possuem ou possuem processos simplificados de suavização como característica;
3. Adaptação dos algoritmos de planejamento de caminhos para viabilizar o uso da vizinhança em mapas 3D com obstáculos, implementando o uso do campo de visão em estrutura tridimensional.

Durante o primeiro ano de mestrado, estudos na área de busca de caminhos foram desenvolvidos, voltados às necessidades do sistema de simulação SIS-ASTROS GMF (SIS-ASTROS. . . , 2020). Esses estudos e implementações desenvolvidas culminaram na publicação de um artigo abordando navegação terrestre considerando informações topográficas sobre os terrenos utilizados no projeto (CHAGAS et al., 2022). Concluindo os requisitos da navegação terrestre, surgiu a necessidade de incluir novos agentes e novos algoritmos para a navegação de tais agentes. O projeto ampliou seus recursos para utilizar agentes aéreos, demonstrando a necessidade de exploração e navegação do espaço aéreo no sistema de simulação. Para isso, estudos de trabalhos e aplicações 3D foram conduzidos, e também algoritmos e técnicas aplicáveis em ambientes 3D foram investigados, resultando no presente trabalho.

A proposta de grandes vizinhanças para ambientes 3D apresentada neste trabalho pode ser expandida em diferentes direções, oportunizando o desenvolvimento de traba-

lhos futuros. Uma dessas possibilidades é o desenvolvimento de funções heurísticas especialmente voltadas e otimizadas para o emprego da vizinhança 3^k. Ainda, diferentes otimizações para o uso da *Vizinhança* 3^k podem ser realizadas em diferentes algoritmos de planejamento de caminhos a partir desta proposta. Por exemplo, o emprego de técnicas de aprendizado de máquina que permitam o uso otimizado de valores maiores de k nas computações realizadas. Outra possibilidade é realizar maiores avaliações das contribuições apresentadas neste trabalho em diferentes aplicações, utilizando ambientes 3D que modelam problemas do mundo real, como na busca de caminhos para VANTs utilizando medidas de coordenação do espaço aéreo. Além da análise de outras questões práticas, uma análise teórica e formalização da *Vizinhança* 3^k pode ser desenvolvida como um trabalho futuro, apresentando estudos mais teóricos em diferentes frentes. Por fim, o uso da técnica de *Vizinhança* 3^k pode ser investigada e ampliada para aplicação em grids com custo não uniformes e grids representados hierarquicamente (irregulares), condições que permitem otimizar e ampliar os algoritmos de busca de caminhos em novos problemas de aplicação.

REFERÊNCIAS

ABAR, S. et al. Agent based modelling and simulation tools: A review of the state-of-art software. **Computer Science Review**, Elsevier, v. 24, p. 13–33, 2017.

AHUJA, R. K.; ORLIN, J. B.; SHARMA, D. New neighborhood search structures for the capacitated minimum spanning tree problem. Sloan School of Management, Massachusetts Institute of Technology, 1998.

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

BAST, H. et al. Route planning in transportation networks. In: **Algorithm engineering**. [S.l.]: Springer, 2016. p. 19–80.

BEIG, M. et al. G-spar: Gpu-based voxel graph pathfinding for spatial audio rendering in games and vr. In: IEEE. **2019 IEEE Conference on Games (CoG)**. [S.l.], 2019. p. 1–8.

BELOV, G. et al. From multi-agent pathfinding to 3d pipe routing. In: **Thirteenth Annual Symposium on Combinatorial Search**. [S.l.: s.n.], 2020.

BOTEA, A.; MÜLLER, M.; SCHAEFFER, J. Near optimal hierarchical path-finding. **J. Game Dev.**, Citeseer, v. 1, n. 1, p. 1–30, 2004.

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

BRONDANI, J. R. et al. Semi-autonomous navigation for virtual tactical simulations in the military domain. **SIMULTECH**, p. 443–450, 2018.

CANNY, J.; REIF, J. New lower bound techniques for robot motion planning problems. In: IEEE. **28th Annual Symposium on Foundations of Computer Science (sfcs 1987)**. [S.l.], 1987. p. 49–60.

CARSTEN, J.; FERGUSON, D.; STENTZ, A. 3d field d: Improved path planning and replanning in three dimensions. In: IEEE. **2006 IEEE/RSJ international conference on intelligent robots and systems**. [S.l.], 2006. p. 3381–3386.

CHAGAS, C. et al. Hierarchical and smoothed topographic path planning for large-scale virtual simulation environments. **Expert Systems with Applications**, Elsevier, v. 189, p. 116061, 2022.

CHOSET, H. et al. **Principles of robot motion: theory, algorithms, and implementations**. [S.l.]: MIT press, 2005.

DANIEL, K. et al. Theta*: Any-angle path planning on grids. **Journal of Artificial Intelligence Research**, v. 39, p. 533–579, 2010.

DEMIR, E.; BEKTAŞ, T.; LAPORTE, G. An adaptive large neighborhood search heuristic for the pollution-routing problem. **European journal of operational research**, Elsevier, v. 223, n. 2, p. 346–359, 2012.

FARIA, M. et al. Efficient lazy theta* path planning over a sparse grid to explore large 3d volumes with a multirotor uav. **Sensors**, MDPI, v. 19, n. 1, p. 174, 2019.

FERGUSON, D.; STENTZ, A. The field d* algorithm for improved path planning and replanning in uniform and non-uniform cost environments. **Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-05-19**, 2005.

FERGUSON, D.; STENTZ, A. Using interpolation to improve path planning: The field d* algorithm. **Journal of Field Robotics**, Wiley Online Library, v. 23, n. 2, p. 79–101, 2006.

FERGUSON, D.; STENTZ, A. Field d*: An interpolation-based path planner and replanner. In: **Robotics research**. [S.l.]: Springer, 2007. p. 239–253.

HAN, J. An efficient approach to 3d path planning. **Information Sciences**, Elsevier, v. 478, p. 318–330, 2019.

HARABOR, D.; GRASTIEN, A. Online graph pruning for pathfinding on grid maps. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2011. v. 25, n. 1, p. 1114–1119.

HARABOR, D.; GRASTIEN, A. An optimal any-angle pathfinding algorithm. In: **Proceedings of the International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2013. v. 23, p. 308–311.

HARABOR, D.; GRASTIEN, A. Improving jump point search. In: **Proceedings of the International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2014. v. 24, p. 128–135.

HARABOR, D. D. et al. Optimal any-angle pathfinding in practice. **Journal of Artificial Intelligence Research**, v. 56, p. 89–118, 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**, IEEE, v. 4, n. 2, p. 100–107, 1968.

HORMAZÁBAL, N. et al. Fast and almost optimal any-angle pathfinding using the 2k neighborhoods. In: **Tenth Annual Symposium on Combinatorial Search**. [S.l.: s.n.], 2017.

HUNG, C.-M.; HE, R. Pathfinding in 3d space: A*, theta*, lazy theta* in octree structure. 2016.

KOOPMAN, M. 3d path-finding in a voxelized model of an indoor environment. 2016.

KRAFFT, C. **Optimisation of pathfinding in a dynamic 3D space.** [S.l.]: Master Thesis. HAW Hamburg. Hamburg, Germany., 2021.

LEE, D.-T. **Proximity and reachability in the plane.** [S.l.]: Ph.D. Dissertation, Coordinated Science Lab. University of Illinois at Urbana-Champaign, Campaign, IL., 1978.

LI, J. et al. Anytime multi-agent path finding via large neighborhood search. In: **International Joint Conference on Artificial Intelligence (IJCAI)**. [S.l.: s.n.], 2021.

LI, J. et al. Mapf-lns2: Fast repairing for multi-agent path finding via large neighborhood search. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2022.

LOZANO-PÉREZ, T.; WESLEY, M. A. An algorithm for planning collision-free paths among polyhedral obstacles. **Communications of the ACM**, ACM New York, NY, USA, v. 22, n. 10, p. 560–570, 1979.

MANDLOI, D.; ARYA, R.; VERMA, A. K. Unmanned aerial vehicle path planning based on a* algorithm and its variants in 3d environment. **International Journal of System Assurance Engineering and Management**, Springer, v. 12, n. 5, p. 990–1000, 2021.

MCCULLAGH, P.; NELDER, J. A. Monographs on statistics and applied probability. **Generalized linear models**, Chapman & Hall, v. 37, 1989.

MILLINGTON, I.; FUNGE, J. Artificial intelligence for games. 2009. **Cité en**, p. 63, 2009.

MITCHELL, J. S.; MOUNT, D. M.; PAPADIMITRIOU, C. H. The discrete geodesic problem. **SIAM Journal on Computing**, SIAM, v. 16, n. 4, p. 647–668, 1987.

NASH, A. et al. Thetaˆ*: Any-angle path planning on grids. In: **AAAI**. [S.l.: s.n.], 2007. v. 7, p. 1177–1183.

NASH, A.; KOENIG, S. Any-angle path planning. **AI Magazine**, v. 34, n. 4, p. 85–107, 2013.

NASH, A.; KOENIG, S.; TOVEY, C. Lazy theta*: Any-angle path planning and path length analysis in 3d. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2010. v. 24, n. 1, p. 147–154.

NEISSE C., S. J. L. S. L. A. L.; FREITAS E. P., C. Investigating deep neural networks as heuristic functions for path planning with topographic terrain characteristics in agent-based simulation. In: **To appear at: Proceedings of the 31st IEEE International Symposium on Industrial Electronics (ISIE).** [S.l.: s.n.], 2022.

NILSSON N. J., . N. N. J. **Artificial intelligence: a new synthesis**. [S.l.]: Morgan Kaufmann, 1998.

NOBES, T. K. et al. The jps pathfinding system in 3d. In: **Proceedings of the International Symposium on Combinatorial Search**. [S.l.: s.n.], 2022. v. 15, n. 1, p. 145–152.

PANDIT, D. 3d pathfinding and collision avoidance using uneven search-space quantization and visual cone search. **arXiv preprint arXiv:1706.01320**, 2017.

PISINGER, D.; ROPKE, S. A general heuristic for vehicle routing problems. **Computers & operations research**, Elsevier, v. 34, n. 8, p. 2403–2435, 2007.

PISINGER, D.; ROPKE, S. Large neighborhood search. In: **Handbook of metaheuristics**. [S.l.]: Springer, 2010. p. 399–419.

PISINGER, D.; ROPKE, S. Large neighborhood search. In: **Handbook of metaheuristics**. [S.l.]: Springer, 2019. p. 99–127.

POZZER C. T., M. J. B. F. L. M. S. L. A. d. L. R. M. N. R. C.; FREITAS, E. P. de. Sis-astros: An integrated simulation system for the artillery saturation rocket system (astros). In: **To appear at: Proceedings of the 12^{th} International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)**. [S.l.: s.n.], 2022.

RANTTILA, P. **JPS Algorithm Adaptation and Optimization to Three-dimensional Space**. 2019.

RIVERA, N.; HERNÁNDEZ, C.; BAIER, J. Grid pathfinding on the 2^k neighborhoods. In: **Proceedings of the AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2017. v. 31, n. 1.

RIVERA, N. et al. The 2^k neighborhoods for grid path planning. **Journal of Artificial Intelligence Research**, v. 67, p. 81–113, 2020.

ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. **Transportation science**, Informs, v. 40, n. 4, p. 455–472, 2006.

SHAW, P. Using constraint programming and local search methods to solve vehicle routing problems. In: SPRINGER. **International conference on principles and practice of constraint programming**. [S.l.], 1998. p. 417–431.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. **Introduction to autonomous mobile robots**. [S.l.]: MIT press, 2011.

SIS-ASTROS GMF Project: 3.07.0073 Agreement: 898347/2020. 2020. Brazil: Federal University of Santa Maria, Education Ministry.

SONG, J. et al. A general large neighborhood search framework for solving integer linear programs. **Advances in Neural Information Processing Systems**, v. 33, p. 20012–20023, 2020.

SOUISSI, O. et al. Path planning: A 2013 survey. In: IEEE. **Proceedings of 2013 International Conference on Industrial Engineering and Systems Management (IESM)**. [S.l.], 2013. p. 1–8.

THORPE, C.; MATTHIES, L. Path relaxation: Path planning for a mobile robot. In: IEEE. **OCEANS 1984**. [S.l.], 1984. p. 576–581.

UNITY-TECHNOLOGIES. **Software Unity®**. 2022. Available from Internet: <unity3d.com>.

URAS, T.; KOENIG, S. An empirical comparison of any-angle path-planning algorithms. In: **International Symposium on Combinatorial Search**. [S.l.: s.n.], 2015. v. 6, n. 1.

URAS, T.; KOENIG, S.; HERNÁNDEZ, C. Subgoal graphs for eight-neighbor gridworlds. In: **International Symposium on Combinatorial Search**. [S.l.: s.n.], 2012. v. 3, n. 1.

YANG, L. et al. Survey of robot 3d path planning algorithms. **Journal of Control Science and Engineering**, Hindawi, v. 2016, 2016.

YAP, P. K. Y. et al. Any-angle path planning for computer games. In: **Seventh artificial intelligence and interactive digital entertainment conference**. [S.l.: s.n.], 2011.