UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

WILLIAM KENNEDY GOMES WANDERLEY

REDES BAYESIANAS: PREDIÇÃO DE TRAJETÓRIA ÓTIMA EM QUADROTORES

MONOGRAFIA DE TRABALHO DE CONCLUSÃO DE CURSO DE GRADUAÇÃO

CAMPINA GRANDE, PB
2024

WILLIAM KENNEDY GOMES WANDERLEY

REDES BAYESIANAS: PREDIÇÃO DE TRAJETÓRIA ÓTIMA EM QUADROTORES

Monografia de Trabalho de Conclusão de Curso de Graduação de Bacharelado submetida à Coordenadoria de Graduação em Engenharia Elétrica da Universidade Federal De Campina Grande - UFCG como parte dos requisitos necessários para obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Bruno Barbosa Albert, Dr

CAMPINA GRANDE, PB
2024

WILLIAM KENNEDY GOMES WANDERLEY

REDES BAYESIANAS: PREDIÇÃO DE TRAJETÓRIA ÓTIMA EM QUADROTORES

Monografia de Trabalho de Conclusão de Curso de Graduação de Bacharelado submetida à Coordenadoria de Graduação em Engenharia Elétrica da Universidade Federal De Campina Grande - UFCG como parte dos requisitos necessários para obtenção do grau de Bacharel em Engenharia Elétrica.

Orientador: Prof. Bruno Barbosa Albert, Dr

Aprovado em: _____ /_____ /_____

William Kennedy Gomes Wanderley
Orientando

Prof. Bruno Barbosa Albert, Dr
Orientador

CAMPINA GRANDE, PB
2024

Torna-te quem tu és (NIETZSCHE, Friedrich).

RESUMO

WANDERLEY, William. Redes Bayesianas: Predição de Trajetória Ótima em Quadrotores. 2024. 52 f. Monografia de Trabalho de Conclusão de Curso de Graduação – Curso de Bacharelado em Engenharia Elétrica, Universidade Federal De Campina Grande. Campina Grande, PB, 2024.

Este estudo buscou encontrar a trajetória ótima de quadrotores por meio de uma rede bayesiana, crucial em áreas como robótica e navegação autônoma. Adotando a metodologia CRISP-DM, foi possível analisar o problema de forma abrangente, demonstrando a viabilidade de empregar técnicas modernas de desenvolvimento. Os resultados obtidos permitiram a obtenção de dados sobre as trajetórias encontradas e a mensuração do tempo de execução dos componentes do sistema, essenciais para avaliar seu desempenho. Este trabalho contribui para o avanço do conhecimento na predição de trajetória ótima, fornecendo uma base sólida para pesquisas futuras e aplicações práticas em autonomia e robótica.

Palavras-chave: Quadrotores. Trajetória. Robótica.

ABSTRACT

WANDERLEY, William. Bayesian Networks: Optimum Trajectory Prediction in Quadrotors. 2024. 52 f. Monografia de Trabalho de Conclusão de Curso de Graduação – Curso de Bacharelado em Engenharia Elétrica, Universidade Federal De Campina Grande. Campina Grande, PB, 2024.

This study aimed to find the optimal trajectory of quadrotors using a Bayesian network, crucial in fields such as robotics and autonomous navigation. By adopting the CRISP-DM methodology, it was possible to comprehensively analyze the problem, demonstrating the feasibility of employing modern development techniques. The results obtained allowed for data acquisition on the trajectories found and the measurement of the execution time of system components, essential for evaluating its performance. This work contributes to advancing knowledge in optimal trajectory prediction, providing a solid foundation for future research and practical applications in autonomy and robotics.

Keywords: Quadrotors. Trajectory. Robotics.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ASCII	*American Standard Code for Information Interchange*
BIF	*Bayesian Interchange Format*
CRISP-DM	*Cross-Industry Standard Process for Data Mining*
DAG	*Directed Acyclic Graph*
DGM	*Directed Graphical Model*
ETL	*Extract, Transform, Load*
PGM	*Probabilistic Graphical Model*
PMF	*Probability Mass Function*
SQL	*Structured Query Language*
UAV	*Unmanned Aerial Vehicle*
UGM	*Undirected Graphical Model*

SUMÁRIO

1 INTRODUÇÃO

Quadrotor é uma aeronave que decola impulsionada por seus quatro motores. São aeronaves que se mantém no ar devido ao empuxo gerado pela rotação de suas hélices. No início da história da aviação as configurações de quadrotores já estavam presentes como solução para problemas de voo vertical. Uma série de protótipos surgiram entre 1920 e 1930, porém sofriam de problemas relacionados a baixo desempenho e dificuldade de pilotagem. Entretanto, tudo mudou com a introdução dos *Unmanned Aerial Vehicles* (UAVs) que se utilizam de sistemas eletrônicos de controle e sensores para estabilizar a aeronave.

Com relação à trajetória do UAV de pequeno porte e do tipo quadrotor, é necessário um controle complexo, pois ele é bastante sensível a ventos devido a seu tamanho pequeno e a baixa altitude de operação. Tanto o modelo de controle quanto o modelo dinâmico do veículo são necessários para modelagem da trajetória (XUE, 2017).

Tendo em vista que o mercado de drones de uso não militar espera chegar a 14,3 bilhões de dólares por ano até o final do ano de 2030, no ano de 2019 o valor esperado para o mercado era de 4,9 bilhões de dólares por ano e com chineses provendo cerca de $\frac{3}{4}$ do mercado com alto interesse dos norte americanos em expandir a produção (PIETSCH, 2019). Então, fazem-se necessários estudos e novas aplicações utilizando inovações para que este mercado possa se desenvolver cada vez mais.

Tal complexidade dos modelos torna interessante uma aplicação utilizando o aprendizado de máquina (HECKERMAN, 1998), que poderia simplificar bastante a abordagem fornecendo a melhor trajetória para o UAV com base em um processo de análise de dados. Portanto, esse trabalho utiliza dados que treinam uma rede bayesiana e com a rede calcula a trajetória ótima do UAV.

1.1 RESUMO DOS RESULTADOS

1.1.1 RESULTADOS GERAIS

O resultado principal obtido foi as trajetórias utilizando cinco mapas diferentes. Também foram obtidos dados que permitem a avaliação do tempo de execução dos principais componentes do sistema.

1.1.2 RESULTADOS ESPECÍFICOS

Obtiveram-se também resultados específicos, dentre eles:

I. Configurar as ferramentas para o desenvolvimento;
II. Escolha da rede para representar um ambiente simulado;
III. Teste e validação da rede e do modelo;
IV. Desacoplamento dos componentes de software;

1.2 METODOLOGIA

O projeto utiliza conceitos da ciência de dados e da dinâmica de voo, porém, como o projeto transita fortemente sobre os conceitos relacionados a ciência de dados, então faz-se necessário uma metodologia que tenha a ciência de dados como norte.

Tendo em vista o explicitado, utilizaremos a metodologia **CR**oss **I**ndustry **S**tandard **P**rocess for **D**ata **M**ining(**CRISP-DM**) que é amplamente reconhecido na indústria de dados, independente da aplicação específica do modelo. Consiste em seis fases, que abrangem desde a compreensão do negócio até a implementação. Essas fases são descritas na Tabela 1

Tabela 1 – Descrições do Processo **CRISP-DM**

Fase	Descrição
Business Understanding	Pesquisar os temas e buscar saber os recursos necessários, Delimitar o tipo de dado que se deseja obter e o objetivo a ser traçado é um passo importante nessa fase juntamente com uma condição de sucesso.
Data Understanding	Coletar dados de diversas fontes, explorar e descrever a qualidade são essenciais. Usar ferramentas estatísticas e determinar atributos e sua devida classificação.
Data Preparation	A seleção de dados ocorrera definindo um critério de inclusão e exclusão. Dados de má qualidade podem ser tratados através da limpeza dos dados. Dependendo do modelo definido no Business Understanding, atributos derivados têm que ser construídos. Todos esses métodos e passos são dependentes do modelo escolhido.
Modeling	Consiste em selecionar a técnica de modelagem, construir o caso teste e o modelo. Para construir o modelo parâmetros específicos devem ser determinados. Para avaliar o modelo, é necessário analisar os resultados do modelo contra o critério de avaliação e selecionar os melhores.
Evaluation	Os resultados são comparados com os objetivos definidos do Business Understanding, e então, ações necessárias são tomadas. O processo no geral também deve ser revisado.
Deployment	Relatório Final e Componente de Software são entregues.

Fonte: (SCHRÖER; KRUSE; GÓMEZ, 2021)

1.3 ORGANIZAÇÃO DO TRABALHO

No Capítulo 2 é feita uma introdução aos principais conceitos teóricos utilizados neste trabalho, no Capítulo 3 tratamos do escopo do trabalho e de decisões acerca do sistema projetado os resultados obtidos são analisados, no Capítulo 4 que nos fornece a explicação dos resultados obtidos e as principais observações, no Capítulo 5 é destacado possibilidades de futuros trabalhos e considerações finais.

2 REVISÃO DE LITERATURA

2.1 O DRONE

Os quadrotores utilizam quatro propulsores para voar. Cada propulsor gera empuxo, levantando e sustentando o quadrotor no ar, enquanto a alteração na velocidade de rotação de cada propulsor controla sua direção, inclinação e movimento. Esses ajustes são realizados automaticamente por sistemas de controle a bordo para manter o voo estável e controlado.

Observe a Figura 1. As setas vermelhas orientadas estão delimitados os propulsores responsáveis pelo empuxo do quadrotor, o termo *clockwise* (CW) significa que é um propulsor orientado no sentido horário e *counterclockwise* (CCW) significa que é no sentido anti-horário.

Figura 1 – Propulsores do UAV

Fonte: RC Drone Good

2.1.1 ROLL, PITCH, YAW E GUINADA VERTICAL

Esses quatro movimentos (*roll*, *pitch*, *yaw* e guinada vertical) são essenciais para controlar a orientação e a direção do UAV. A Figura 2 exibe um esquema visual desses três movimentos.

1. **Roll (Rolagem)**
   - **Definição:** *Roll* é a rotação em torno do eixo longitudinal da aeronave, que é perpendicular ao eixo lateral. Isso significa que o quadrotor gira para a esquerda ou para a direita.

- **Controle:** O *roll* envolve ajustar as velocidades dos motores esquerdo e direito para inclinar o quadrotor na direção desejada. Por exemplo, aumentar a velocidade do motor direito e diminuir a velocidade do motor esquerdo fará com que o quadrotor role para a direita.

2. **Pitch (Arfagem)**
   - **Definição:** *Pitch* é a rotação em torno do eixo lateral da aeronave, que é perpendicular ao eixo longitudinal. Isso significa que o quadrotor inclina-se para a frente ou para trás.
   - **Controle:** O *pitch* envolve ajustar as velocidades dos motores dianteiros e traseiros para inclinar o quadrotor na direção desejada. Por exemplo, Aumentar a velocidade dos motores dianteiros e diminuir a velocidade dos traseiros fará com que o quadrotor incline para a frente.
3. **Yaw (Guinada)**
   - **Definição:** *Yaw* é a rotação em torno do eixo vertical da aeronave. Isso significa que o quadrotor gira em torno de seu próprio eixo vertical, apontando para uma nova direção.
   - **Controle:** O *yaw* é realizado ajustando a velocidade de rotação dos motores em pares opostos. Por exemplo, aumentar a velocidade dos motores dianteiros direito e traseiro esquerdo fará com que o quadrotor gire para a direita.
4. **Guinada Vertical**
   - **Definição:** Guinada Vertical é o deslocamento no eixo z da aeronave. Isso significa que o quadrotor sobe ou desce em torno do plano cartesiano XYZ comum.
   - **Controle:** A Guinada Vertical é realizada ajustando a velocidade de rotação de todos os motores ao mesmo tempo. Por exemplo, aumentar a velocidade dos motores faz com que o quadrotor suba.

2.1.2 OBSTÁCULOS

2.2 ESTATÍSTICA

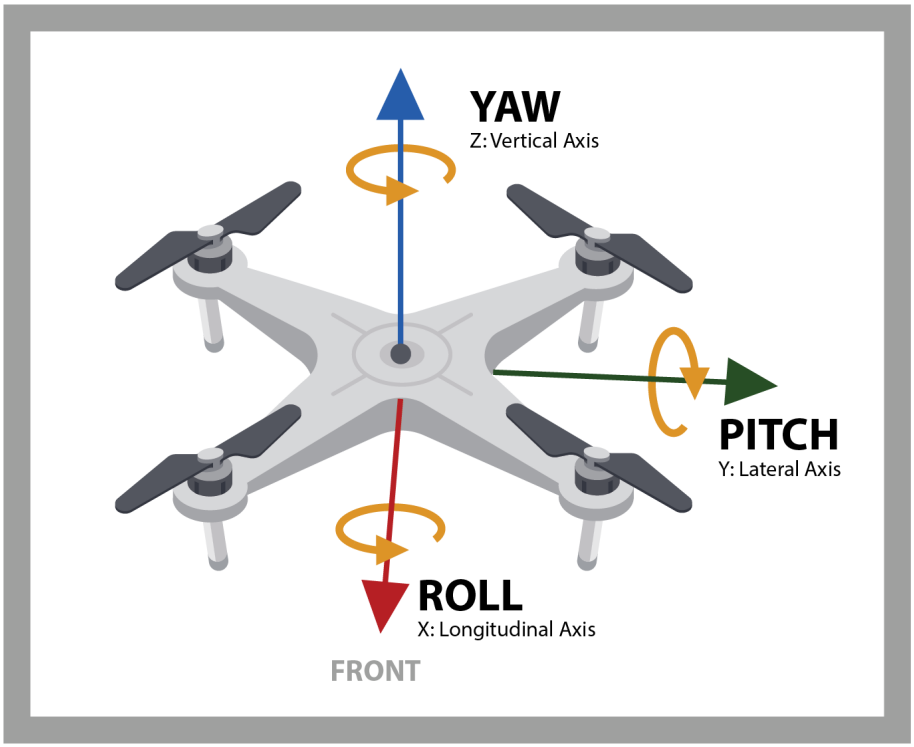
A probabilidade condicional e a independência condicional são conceitos fundamentais na teoria da probabilidade, desempenhando um papel crucial na análise de eventos em situações dependentes.(LEON-GARCIA, 2007)

2.2.1 PROBABILIDADE CONDICIONAL

A probabilidade condicional refere-se à probabilidade de que um evento ocorra, dado que outro evento já ocorreu. Seja A e B dois eventos em um espaço amostral S, a probabilidade condicional de A dado B, denotada por $P(A|B)$, é calculada pela fórmula:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Figura 2 – Movimentos do quadrotor.



Fonte: Immervision

em que:

- $P(A \cap B)$ é a probabilidade da interseção de A e B.
- $P(B)$ é a probabilidade de B.

Também pode-se concluir que:

$$P(A \cap B) = P(A|B)P(B)$$

2.2.2 INDEPENDÊNCIA CONDICIONAL

Dois eventos A e B são considerados independentes condicionalmente se a ocorrência ou não ocorrência de um deles não influenciar a probabilidade do outro. A independência condicional é expressa como:

$$P(A|B) = P(A)$$

Isso implica que a ocorrência (ou não ocorrência) de B não afeta a probabilidade de A e vice-versa.

Também pode-se concluir que:

$$P(A \cap B) = P(A|B)P(B) = P(A)P(B)$$

2.2.3 INTERPRETAÇÃO INTUITIVA

Como exemplos para os conceitos citados pode-se ter:

- **Probabilidade Condicional:** Imagine que você joga dois dados e sabe que a soma é maior que 9. A probabilidade condicional de obter um 6 no primeiro dado pode ser diferente da probabilidade geral, pois a informação sobre a soma afeta as chances.
- **Independência Condicional:** Se você está jogando cartas e sabe que a primeira carta retirada é um ás, a probabilidade de retirar um rei na próxima retirada não é afetada caso se coloque o ás de volta. Isso indica independência condicional.

2.2.4 TEOREMA DE BAYES

O Teorema de Bayes, em sua formulação geral, é uma ferramenta poderosa para atualizar probabilidades quando novas evidências estão disponíveis. Seja $\{A_1, A_2, \ldots, A_n\}$ um conjunto de eventos mutuamente exclusivos e exaustivos (ou seja, um desses eventos deve ocorrer), e seja B um evento com $P(B) > 0$. O Teorema de Bayes é expresso da seguinte forma:

$$S = A_1 \cup A_2 \cup \cdots \cup A_{n-1} \cup A_n$$

$$B = B \cap S = B \cap (A_1 \cup A_2 \cup \cdots \cup A_{n-1} \cup A_n)$$

$$B = (B \cap A_1) \cup (B \cap A_2) \cup \cdots \cup (B \cap A_n)$$

Então tem-se:

$$P(B) = \sum_{i=1}^{n} (P(B \cap A_i)) = \sum_{i=1}^{n} (P(B|A_i) \cdot P(A_i))$$

E, como sabe-se, segundo a probabilidade condicional que:

$$P(A_k|B) = \frac{P(A_k \cap B)}{P(B)}$$

Então:

$$P(A_k|B) = \frac{P(B|A_k) \cdot P(A_k)}{\sum_{i=1}^{n} P(B|A_i) \cdot P(A_i)}$$

em que:

- $P(A_k|B)$ é a probabilidade condicional de A_k dado B.
- $P(B|A_k)$ é a probabilidade condicional de B dado A_k.
- $P(A_k)$ é a probabilidade a priori de A_k.
- $P(B)$ é a probabilidade marginal de B.

Essa fórmula relaciona a probabilidade a posteriori de A_k dado B com a probabilidade condicional de B dado A_k, a probabilidade a priori de A_k, e a probabilidade marginal de B.

Interpretação Intuitiva:

- **Probabilidade a Priori ($P(A_k)$):** Representa a crença inicial ou probabilidade antes de considerar a nova evidência B.
- **Probabilidade Condicional de Evidência ($P(B|A_k)$):** Indica a probabilidade de observar a evidência B dado que o evento A_k ocorreu.
- **Probabilidade Marginal de Evidência ($P(B)$):** Refere-se à probabilidade total da evidência B ocorrer, levando em consideração todas as possíveis causas A_k.

Vantagens e importância:

- **Atualização Contínua:** O Teorema de Bayes permite uma atualização contínua de crenças à medida que novas evidências são apresentadas.
- **Versatilidade:** Sua formulação geral é aplicável a uma ampla gama de cenários, desde a medicina até a análise estatística.
- **Incorporação de Incerteza:** Lida efetivamente com a incerteza, fornecendo uma estrutura para atualizar probabilidades com base em informações adicionais.

2.2.5 VARIÁVEIS ALEATÓRIAS E DISTRIBUIÇÃO DE PROBABILIDADE CONJUNTA

Para variáveis aleatórias discretas $X_1, X_2, \ldots, X_n$, a distribuição de probabilidade conjunta é representada pela **Função Massa de Probabilidade** (PMF). A PMF atribui probabilidades a todas as combinações possíveis dos valores que as variáveis podem assumir.

1. **Função PMF:**
   - Seja $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$ a probabilidade conjunta de n variáveis aleatórias discretas $X_1, X_2, \ldots, X_n$ assumirem os valores $x_1, x_2, \ldots, x_n$, respectivamente.
   - A PMF deve satisfazer as seguintes propriedades:
     - $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \geq 0$ para todos os conjuntos possíveis de valores $(x_1, x_2, \ldots, x_n)$.
     - A soma de todas as probabilidades conjuntas deve ser igual a 1: $\sum_{x_1} \sum_{x_2} \ldots \sum_{x_n} P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = 1$.
   - A PMF descreve completamente a distribuição de probabilidade conjunta para as variáveis discretas.
2. **Independência:**
   - As variáveis aleatórias $X_1, X_2, \ldots, X_n$ são independentes se, para todos os conjuntos de valores $x_1, x_2, \ldots, x_n$:

$$\begin{aligned} &P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \\ &= P(X_1 = x_1) \cdot P(X_2 = x_2) \cdot \ldots \cdot P(X_n = x_n) \end{aligned}$$

3. **Distribuições Marginais:**
   - As distribuições marginais podem ser obtidas somando sobre todos os possíveis valores das outras variáveis. Por exemplo, para X_1:

$$P(X_1 = x_1) = \sum_{x_2} \sum_{x_3} \cdots \sum_{x_n} P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n).$$

Essas definições são específicas para variáveis aleatórias discretas e formam a base para a análise de sistemas que envolvem múltiplas variáveis aleatórias discretas.

Suponha que temos duas variáveis aleatórias discretas, X e Y, com os seguintes resultados e probabilidades associadas:

	$Y = 1$	$Y = 2$	$Y = 3$
$X = 1$	0.2	0.1	0.1
$X = 2$	0.1	0.3	0.2
$X = 3$	0.1	0.2	0.0

Neste exemplo:

- X representa a primeira variável aleatória.
- Y representa a segunda variável aleatória.
- As probabilidades estão nas células correspondentes.

Esta tabela representa a distribuição de probabilidade conjunta para os valores possíveis de X e Y. Por exemplo, a probabilidade de $P(X = 2, Y = 3) = 0.2$, indicando a probabilidade de ocorrerem esses valores simultaneamente.

2.3 REDES BAYESIANAS

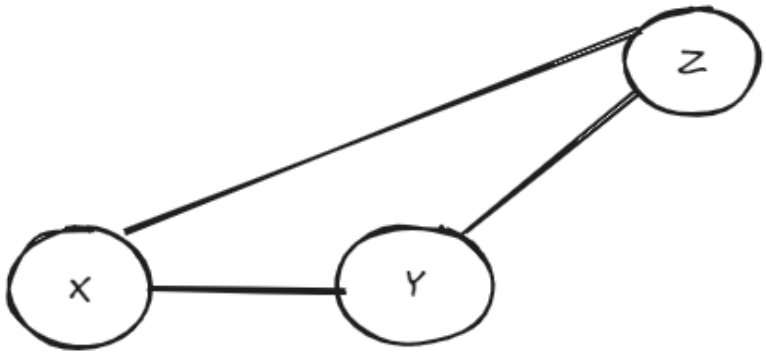
Um modelo gráfico, também chamado de modelo gráfico probabilístico(PGM) ou modelo probabilístico estruturado, é uma forma de representar as relações entre variáveis aleatórias usando um grafo. Nesse tipo de modelo probabilístico, o grafo é utilizado para mostrar como as variáveis dependem condicionalmente umas das outras. Esses modelos são comumente usados em teoria da probabilidade, estatísticas (especialmente estatísticas bayesianas) e aprendizado de máquina. Em resumo, a ideia central desses modelos é capturar e visualizar as interações condicionais entre as variáveis aleatórias por meio de um esquema gráfico. (JENSEN; NIELSEN, 2007)

Os PGMs podem ser classificados em dois tipos principais com base na direção das arestas no grafo:

- *Undirected Graphical Models* (UGMs), Observe a Figura 3.
- *Directed Graphical Models* (DGMs), Observe a Figura 4 (Também pode ser conhecido como *Directed Acyclic Graphs* (DAGs).

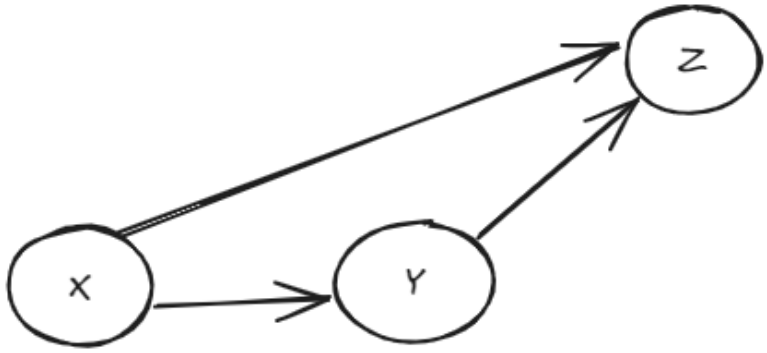
Nas redes Bayesianas, as variáveis e suas inter-relações são expressas por meio da Teoria dos Grafos. Nesse contexto, as variáveis são representadas como nós, enquanto as conexões

Figura 3 – UGM



entre elas são indicadas por setas, formando um grafo direcionado e acíclico, conhecido como DAG, Um exemplo é dado na Figura 5.

Figura 4 – DGM



Um DAG é uma estrutura matemática composta por nós e arestas, em que cada aresta tem uma direção específica, e não há ciclos, ou seja, não é possível percorrer um caminho fechado seguindo as direções das arestas. Essa propriedade torna os DAGs particularmente úteis em representar relações e dependências direcionadas entre elementos, sem o risco de laços infinitos. Eles são comumente empregados em ciência da computação, matemática e modelagem de sistemas complexos.

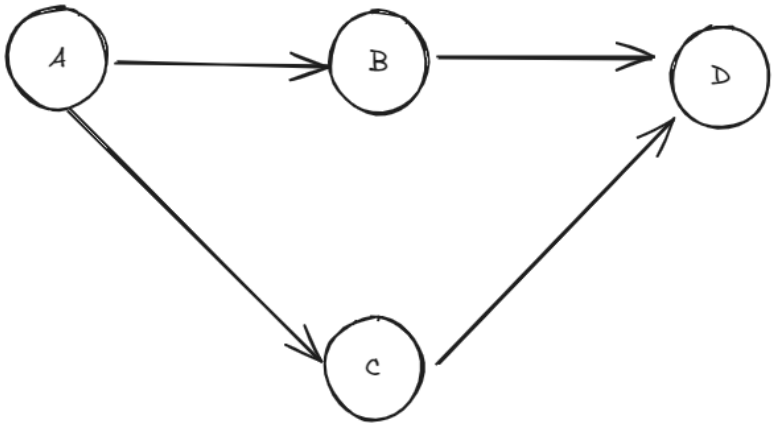
Na Engenharia Elétrica, os DAGs encontram aplicação na modelagem de sistemas complexos, como circuitos digitais e redes elétricas. No projeto de processadores, por exemplo, blocos funcionais como ALUs(Arithmetic Logic Unit) e registradores podem ser representados como nós em um DAG, onde as arestas indicam dependências de dados entre esses componentes. Além disso, em análises de redes elétricas, os DAGs são úteis para representar fluxos de energia, auxiliando na compreensão das relações e na otimização da eficiência em sistemas elétricos. O uso de DAGs na engenharia elétrica proporciona uma abordagem visual e eficaz para modelar e analisar sistemas interdependentes e hierárquicos.

1. **DAG:**
    - Uma rede bayesiana é representada por um grafo acíclico dirigido, o que significa que as conexões entre as variáveis formam um grafo sem ciclos. Cada nó no grafo

representa uma variável, e as setas indicam as dependências probabilísticas entre elas.

2. **Nós representam variáveis aleatórias:**
   - Cada nó na rede representa uma variável aleatória, que pode ser observável ou latente. Essas variáveis podem ter estados discretos ou contínuos, e a rede modela as relações de dependência condicional entre elas.
3. **Parâmetros de probabilidade condicional:**
   - Cada conexão na rede está associada a uma tabela de probabilidade condicional que descreve a relação probabilística entre a variável representada pelo nó de chegada e as variáveis representadas pelos nós de entrada direta. Essas tabelas capturam como a probabilidade de uma variável muda dado o estado de seus nós pais.
4. **Inferência e atualização de crenças:**
   - As redes Bayesianas são usadas para realizar inferência probabilística. Dado um conjunto de observações, a rede pode ser usada para atualizar as crenças sobre outras variáveis não observadas. Isso é feito calculando a distribuição de probabilidade posterior, que reflete a probabilidade das variáveis não observadas dadas as observadas.

Figura 5 – DAG



2.3.1 DEFININDO A REDE

O processo de definição de uma rede bayesiana consiste em dois passos:

I Topologia (Nós, Arcos);

II Distribuições de Probabilidade;

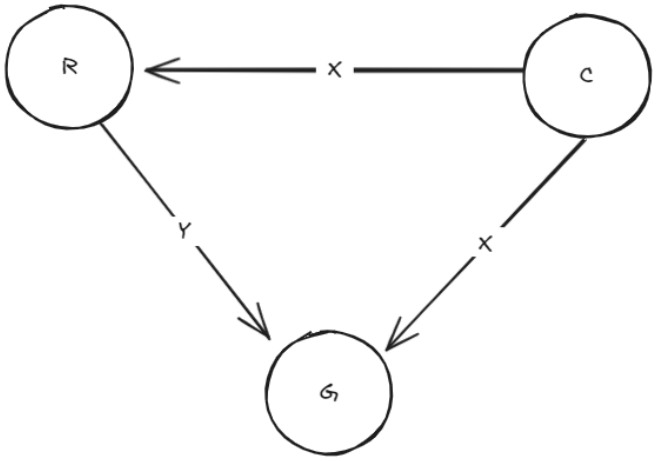
Para definir a topologia primeiro precisamos entender um sistema de irrigação que usaremos de exemplo.

- C: Chuva
- R: Regador
- G: Grama Molhada

Dois eventos(arcos) podem tornar a grama molhada. Respectivamente a chuva e o regador ativo, chamemos de x. A chuva também possui um efeito sobre o uso do regador, já que quando chove não faz sentido utilizar regador, chamemos de y

Logo modelando temos a seguinte estrutura da Figura 26

Figura 6 – DAG do Sistema de Irrigação



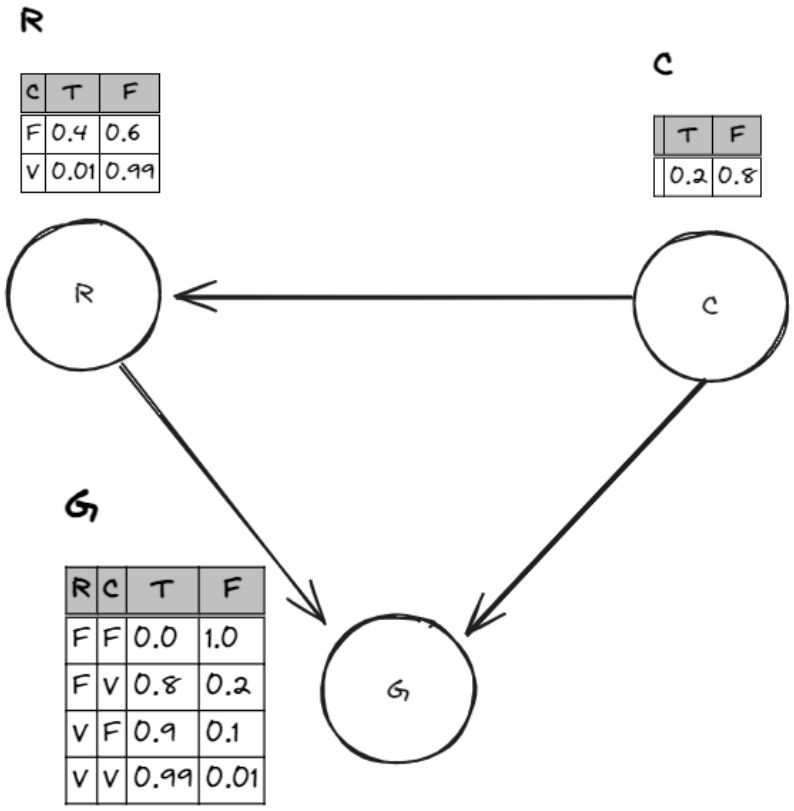
Para definir as Distribuições de Probabilidade como na Figura 7 precisamos das tabelas de probabilidade condicional de cada nó calculada, é possível obter a distribuição de probabilidade conjunta e consequentemente inferir qualquer evidência sobre o domínio.

2.3.2 APRENDIZAGEM

A aprendizagem das Redes Bayesianas geralmente envolve dois aspectos principais:

1. **Aprendizagem Estrutural:**
   - **Algoritmos de Crescimento e Poda (Grow-Shrink):** Começa com um grafo vazio e cresce a estrutura adicionando arcos de acordo com algum critério. Posteriormente, realiza a poda de arcos que não são considerados úteis.
   - **Algoritmos Baseados em Restrições (Constraint-Based):** Utilizam testes estatísticos para determinar relações entre variáveis. Algoritmos como o PC (Peter and Clark) e o FCI (Fast Causal Inference) são exemplos.
   - **Algoritmos Baseados em Pontuação (Score-Based):** Atribuem uma pontuação a diferentes estruturas e escolhem aquela com a pontuação mais alta. O algoritmo de K2 é um exemplo.
2. **Aprendizagem de Parâmetros:**
   - Após a estrutura ser determinada, os parâmetros das distribuições condicionais das variáveis são estimados.
   - Isso pode envolver a contagem de frequências nos dados observados ou o uso de métodos mais sofisticados, dependendo do tipo de variável (categórica ou contínua) e da quantidade de dados disponíveis.

Figura 7 – Definição da Rede



Os desafios na aprendizagem de Redes Bayesianas incluem lidar com a complexidade computacional associada à busca de estruturas ótimas, especialmente quando o número de variáveis é grande. Além disso, a escolha de algoritmos e critérios adequados pode afetar significativamente os resultados.

2.4 ALGORITMOS DE BUSCA

Um algoritmo de busca em grafos é um método utilizado para explorar ou percorrer um grafo de maneira sistemática, a fim de encontrar determinadas informações, como um caminho entre dois vértices, verificar a conectividade do grafo, encontrar ciclos, entre outros objetivos.

Os grafos podem ser classificados em dois tipos principais: não direcionados(Figura 8) e direcionados(Figura 9). Em um grafo direcionado, as arestas têm uma direção associada, indicando uma relação direcional entre os vértices. Por outro lado, em um grafo não direcionado, as arestas representam uma conexão bidirecional entre os vértices, sem uma direção específica.

Figura 8 – Grafo não direcionado

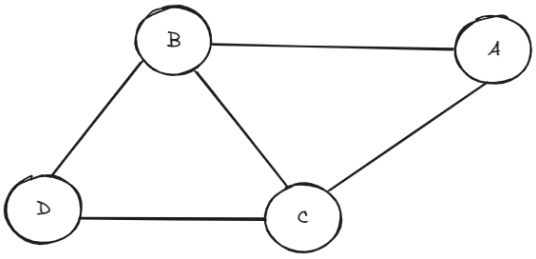
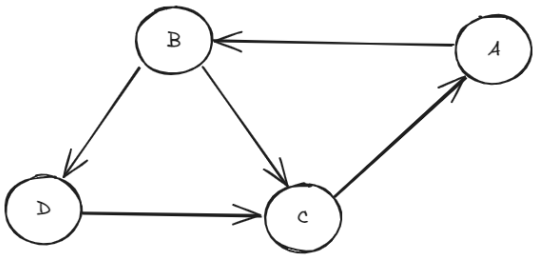


Figura 9 – Grafo direcionado



2.4.1 DIJKSTRA

O algoritmo de Dijkstra é um algoritmo clássico usado para encontrar o caminho mais curto entre dois vértices em um grafo ponderado, onde os pesos das arestas representam as distâncias entre os vértices. (CANDRA; BUDIMAN; HARTANTO, 2020)

Explicação do Algoritmo:

I. **Inicialização**: comece definindo um vértice de origem e atribuindo uma distância inicial de zero para ele. Para todos os outros vértices, atribua uma distância inicial de infinito (ou um valor muito grande) e marque todos os vértices como não visitados.
II. **Seleção do vértice**: escolha o vértice não visitado com a menor distância até agora. Na primeira iteração, esse será o vértice de origem, pois sua distância inicial foi definida como zero.
III. **Atualização das distâncias**: para cada vértice adjacente ao vértice selecionado, calcule a distância acumulada até esse vértice através do vértice selecionado. Se essa distância acumulada for menor do que a distância atualmente atribuída ao vértice adjacente, atualize a distância para esta menor distância. Este passo garante que estamos sempre mantendo a menor distância conhecida para cada vértice.
IV. **Marcação do vértice selecionado**: após atualizar as distâncias de todos os vértices adjacentes, marque o vértice selecionado como visitado para garantir que não seja selecionado novamente.
V. **Repetição**: repita os passos II a IV até que todos os vértices tenham sido visitados.
VI. **Fim do algoritmo**: uma vez que todos os vértices tenham sido visitados, o algoritmo termina e você terá a menor distância conhecida de cada vértice até o vértice de origem.

Exemplo:

O nó de início é A e queremos encontrar o caminho mais curto até F. Cada aresta do grafo tem um custo de movimento associado.

I. **Inicialização**: comece definindo um vértice de origem e atribuindo uma distância inicial de zero para ele. Para todos os outros vértices, atribua uma distância inicial de infinito (ou um valor muito grande) e marque todos os vértices como não visitados.

Figura 10 – Selecionando A

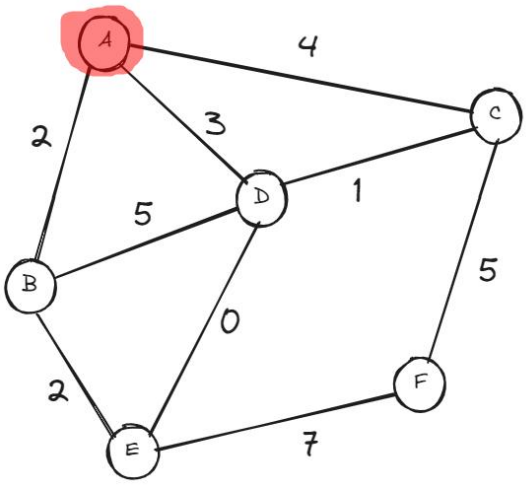
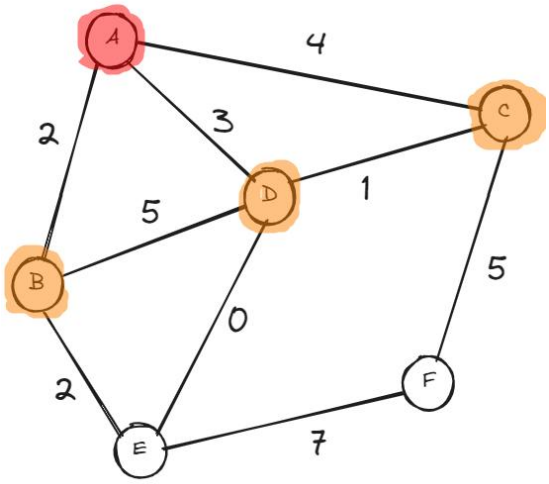


Figura 11 – Vértices Adjacentes de A



II. **Seleção do vértice**: selecionamos o vértice com a menor distância ainda não visitada, que é o vértice A.
III. **Atualização das distâncias**: para os vértices adjacentes à A (B, C e G).
IV. **Marcação do vértice selecionado**: vértice A.

Tabela 2 – (a) Distâncias a partir do vértice A

Vertice	Distancia de A
A	0
B	2
C	4
D	3
E	∞
F	∞

Fonte: Autoria Própria.

Tabela 3 – (b) Vértices visitados

Vertice	Visitado
A	True
B	False
C	False
D	False
E	False
F	False

Fonte: Autoria Própria.

II. **Seleção do vértice**: selecionamos o vértice com a menor distância ainda não visitada, que é o vértice B.
III. **Atualização das distâncias**: para os vértices adjacentes a B (D, E).
IV. **Marcação do vértice selecionado**: vértice B.

Figura 12 – Vértices Adjacentes de B

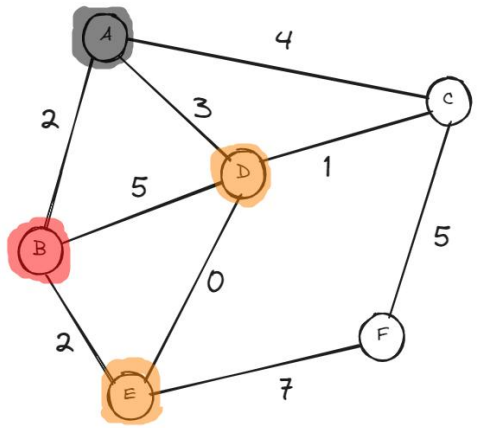


Tabela 4 – (a) Distâncias a partir do vértice A

Vertice	Distancia de A
A	0
B	2
C	4
D	3
E	4
F	∞

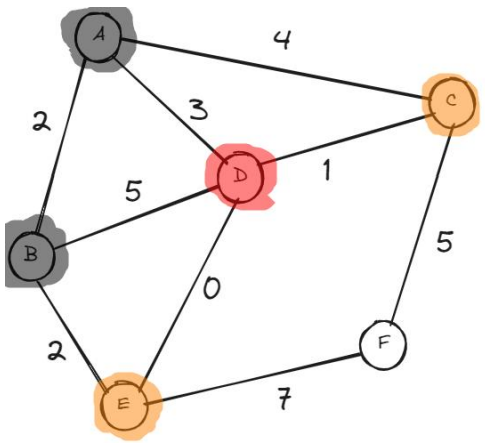
Fonte: Autoria Própria.

Tabela 5 – (b) Vértices visitados

Vertice	Visitado
A	True
B	True
C	False
D	False
E	False
F	False

Fonte: Autoria Própria.

Figura 13 – Vértices Adjacentes de D



II. **Seleção do vértice**: selecionamos o vértice com a menor distância ainda não visitada, que é o vértice D.
III. **Atualização das distâncias**: para os vértices adjacentes a D (C, E)
IV. **Marcação do vértice selecionado**: vértice D

Tabela 6 – (a) Distâncias a partir do vértice A

Vertice	Distancia de A
A	0
B	2
C	4
D	3
E	3
F	∞

Fonte: Autoria Própria.

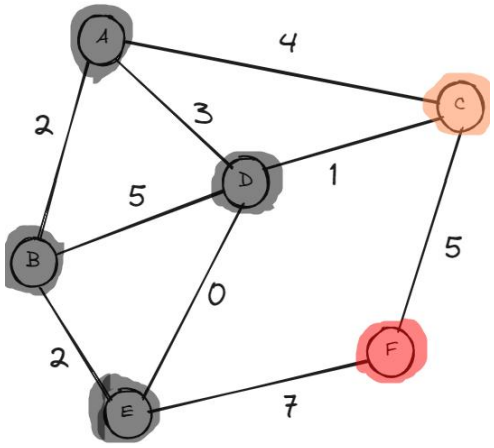
Tabela 7 – (b) Vértices visitados

Vertice	Visitado
A	True
B	True
C	False
D	True
E	False
F	False

Fonte: Autoria Própria.

Continuando obtemos no penúltimo passo:

Figura 14 – Vértices Adjacentes de F



II. **Seleção do vértice**: selecionamos o vértice com a menor distância ainda não visitada, que é o vértice F.
III. **Atualização das distâncias**: para os vértices adjacentes a F (C).
IV. **Marcação do vértice selecionado**: vértice F.

E realizando mais um passo que é visitando o nodo C obtemos as seguintes tabelas atualizando o custo do nodo F para 9 pelo caminho passando pelo nodo C.

Tabela 8 – (a) Distâncias a partir do vértice A

Vertice	Distancia de A
A	0
B	2
C	4
D	3
E	3
F	10

Fonte: Autoria Própria.

Tabela 9 – (b) Vértices visitados

Vertice	Visitado
A	True
B	True
C	False
D	True
E	True
F	True

Fonte: Autoria Própria.

Tabela 10 – (a) Distâncias a partir do vértice A

Vertice	Distancia de A
A	0
B	2
C	4
D	3
E	3
F	9

Fonte: Autoria Própria.

Tabela 11 – (b) Vértices visitados

Vertice	Visitado
A	True
B	True
C	True
D	True
E	True
F	True

Fonte: Autoria Própria.

Portanto, para traçar o menor caminho de A até F basta percorrer os menores custos em cada nodo anterior que para o nosso caso seria:

A -> C -> F

2.4.2 A*

O algoritmo de Dijkstra e o algoritmo A* são ambos algoritmos populares de busca de caminho em grafos, mas diferem em como eles determinam qual nó explorar a seguir durante a busca pelo caminho mais curto de um nó inicial para um nó objetivo.

No algoritmo A*, além de considerar o custo acumulado do nó inicial até o nó atual, também é levado em conta uma estimativa do custo restante do nó atual até o nó objetivo. Isso é feito através da utilização de uma função heurística, que estima a distância entre o nó atual e o nó objetivo. O próximo nó a ser explorado é escolhido com base na soma do custo acumulado e da estimativa heurística. Esta soma é dada por $f(n)$, onde $f(n) = g(n) + h(n)$, em que $g(n)$ é o custo acumulado do nó inicial até o nó atual e $h(n)$ é a estimativa do custo do nó atual até o nó objetivo.

Uma estimativa muito utilizada é a distância pitagórica do nodo atual até o final da busca.

$$h(n) = \sqrt{(x_f - x_n)^2 + (y_f - y_n)^2}$$

Note que se tornando $h(n) = 0$ temos a seleção da mesma forma do algoritmo de Dijkstra.

A* é especialmente útil quando há um conhecimento heurístico disponível sobre a localização do objetivo. É amplamente utilizado em aplicações de jogos, robótica e planejamento de rotas.

O Dijkstra é utilizado quando não se tem nenhuma informação adicional sobre a localização do objetivo ou quando se quer encontrar o caminho mais curto em termos de custo total: $g(n)$.

3 DESENVOLVIMENTO

3.1 DELIMITAÇÃO DO ESCOPO

Delimitação de escopo é determinar os limites do problema. Isso pode incluir funcionalidades específicas do software, áreas de aplicação, plataformas de implantação e quaisquer requisitos de integração com sistemas existentes (SOMMERVILLE, 2016).

Aqui, devido ao fato do objetivo ser especifico, trata-se de delimitar em torno de dois aspectos:

I Dados de Entrada: Um arquivo com estados, que indicam o grau de perigo de um determinado ponto no espaço.

II Dados de Saída: Medições armazenadas em um banco de dados e gráfico de calor da trajetória.

3.1.1 DADOS DE ENTRADA

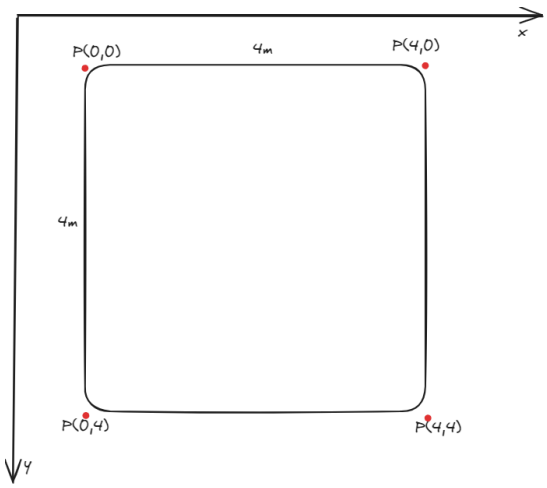
Um dos primeiros passos é entender o problema que desejamos resolver para poder delimitar o dado de entrada. Dada uma sala de formato retangular mXn, o objetivo é mapear em um plano cartesiano suas características.

3.1.1.1 PARÂMETROS

Uma decisão do trabalho foi escolher a orientação do plano cartesiano e o $P(0,0)$. Foi escolhido o canto superior esquerdo como $P(0,0)$ do sistema de coordenadas.

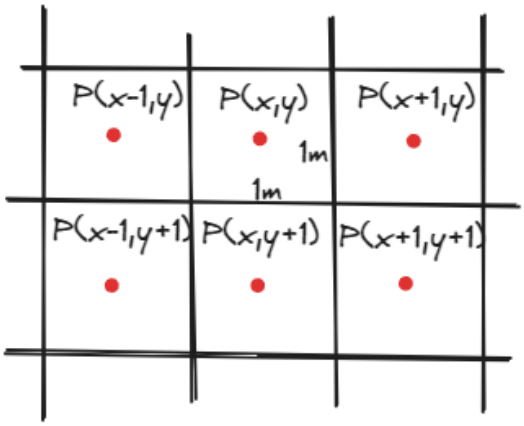
Na Figura 15, uma sala com $m = 4$ e $n = 4$ é usada para exemplificar.

Figura 15 – Representação de uma sala 4m X 4m



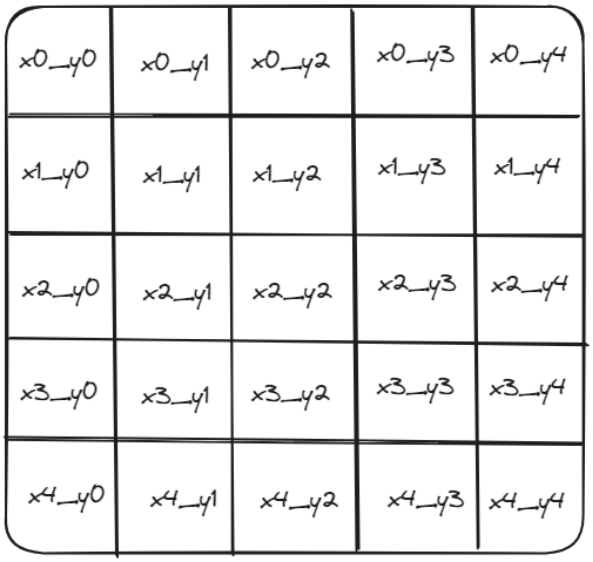
Com o plano cartesiano definido, precisamos escolher a representação para os nodos de acordo com uma definição. Aqui escolhemos discretizar o espaço contínuo em quadrados de $1mX1m$. Sendo assim, o `Nodo(X2, Y2)` da rede bayesiana equivale à posição no espaço do centro geométrico desse quadrado (o ponto vermelho na Figura 16).

Figura 16 – Processo de discretização da sala 4m X 4m



Como resultado da discretização, podemos representar a sala $4mX4m$ como a seguinte rede da Figura 17:

Figura 17 – Representação da rede Bayesiana da sala 4m X 4m



Uma das formas de representar os dados obtidos pelos sensores de um drone é representando cada nodo por um processo estocástico dependente de uma variável temporal.

Então, escolhe-se o processo estocástico Estado do Nodo(x,y):

$$ES_{x,y}(t)$$

A partir dos dados de $ES_{x,y}(t)$, então treinamos parametros da rede bayesiana que representa a sala. Outra escolha é o valor esperado:

$$0 \leq \mathbb{E}[ES_{x,y}(t)] \leq 5 \quad \forall x, y, t$$

Tabela 12 – Especificação dos Estados Esperados.

Significado	**Valores**
Rápido	$0 < \mathbb{E}[ES_{x,y}(t)] < 1$
Atenção	$1 < \mathbb{E}[ES_{x,y}(t)] < 2$
Lento	$2 < \mathbb{E}[ES_{x,y}(t)] < 3$
Cuidado	$3 < \mathbb{E}[ES_{x,y}(t)] < 4$
Obstaculo	$4 < \mathbb{E}[ES_{x,y}(t)] < 5$

Fonte: Autoria Própria.

Então, obtêm-se a tabela Tabela 13 como dado de entrada:

Tabela 13 – Dados para treinamento dos parâmetros dos nodos.

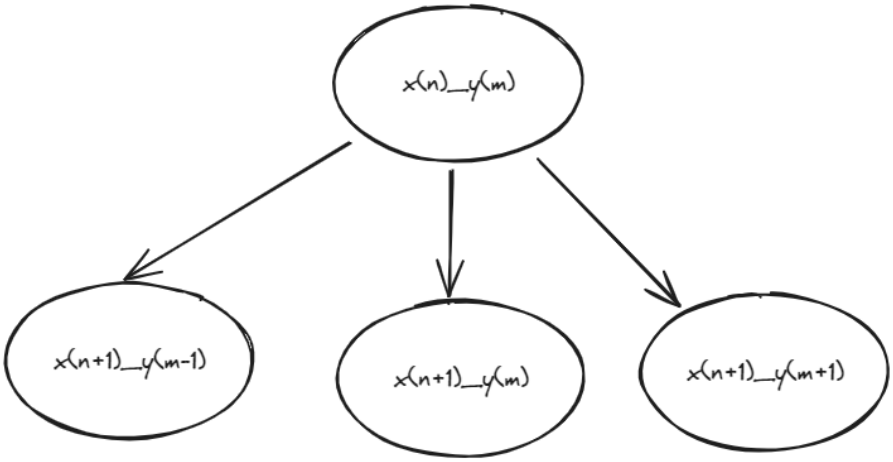
	x0_y0	x0_y1	x0_y2	x0_y3	...	x4_y2	x4_y3	x4_y4
0	1.684	-2.156	-1.407	-0.843	...	-0.031	1.441	-1.219
1	-1.944	1.504	1.328	-1.618	...	-2.394	-0.536	4.337
2	-2.564	-0.624	2.592	0.314	...	-0.018	0.647	0.607
3	0.870	-0.813	2.593	-0.267	...	2.901	-1.013	1.502
4	-0.272	-0.479	1.451	1.601	...	0.659	2.280	0.376
⋮	⋮	⋮	⋮	⋮	⋱	⋮	⋮	⋮
95	0.536	-2.378	-1.670	-1.390	...	-0.762	0.536	-0.059
96	1.122	-1.120	0.657	-2.502	...	-0.287	-2.328	1.206
97	-2.319	-3.120	-0.219	2.548	...	0.217	0.189	2.104
98	2.904	0.039	1.496	-4.613	...	0.642	0.675	0.500
99	-2.679	-0.846	1.140	-2.967	...	-1.222	2.158	-0.069

Fonte: Autoria Própria.

3.1.1.2 ESTRUTURA

Outro ponto que se faz necessário para treinar a rede é o aspecto estrutural (nodos e arcos). Com os nodos já definidos podemos arbitrar a estrutura dos arcos. Para simplificar o problema, escolhe-se a configuração da Figura 18. Também se faz interessante poder escolher essa estrutura de forma flexível.

Figura 18 – Arcos partindo de um nodo.



3.1.2 DADOS DE SAÍDA

Como dado de saída, é fundamental saber como se comportam as soluções de treinamento das redes e do encontro do caminho mais seguro.

Portanto, queremos como resultados os seguintes pontos:

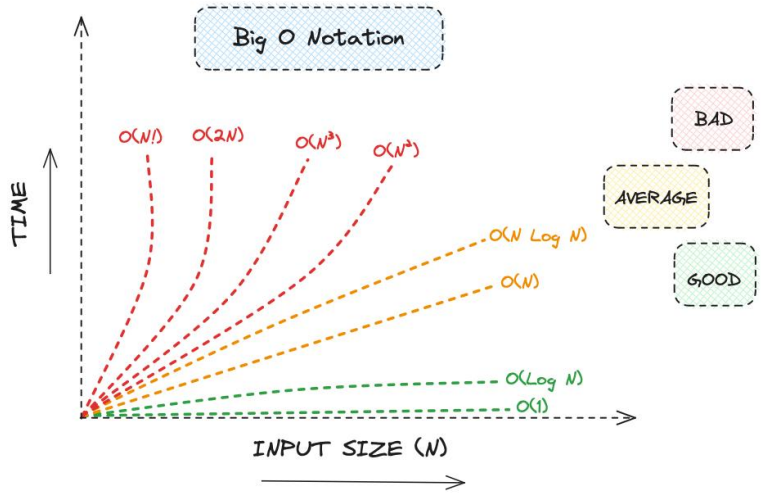
- Complexidades
  - Treinamento da Rede;
  - Encontro do caminho ótimo;
- Mapa de Calor com o caminho;

3.1.2.1 ANALISE DE COMPLEXIDADE

Outro ponto que se faz necessário é avaliar a execução de duas funções, obtendo o gráfico da complexidade, como na Figura 19, para cada um dos casos:

- Treinamento da Rede Bayesiana;
- Calculo da Trajetória ótima;

Figura 19 – Gráfico de Complexidade



Fonte: (HLFDEV, 2023)

3.1.2.2 MAPA DE CALOR

O mapa de calor serve como uma representação visual do resultado obtido do começo até o fim da execução, é uma forma de verificar se a solução obtida para cada mapa tem sentido lógico e prático.

O mapa pode ser representado como na Figura 20.

Figura 20 – Mapa de Calor com a Trajetória

3.2 ESPECIFICAÇÃO DO SISTEMA

A ideia geral é ser um sistema que tenha processamento serial e seja altamente extensível e flexível. De início, os dados provenientes dos sensores serão mocados (simulados).

3.2.1 DETERMINAÇÃO DE REQUISITOS

Na engenharia de software, requisitos referem-se às funções e características que um sistema deve possuir para atender às necessidades e expectativas dos seus usuários, clientes ou outras partes interessadas. Esses requisitos são a base para o *design*, desenvolvimento, implementação e teste de um sistema de software.

3.2.1.1 REQUISITOS DE USUÁRIO

Os requisitos do usuário descrevem as necessidades, expectativas e funcionalidades específicas que os usuários finais ou clientes desejam que o sistema tenha. Eles se concentram na experiência do usuário e nas funcionalidades importantes para os usuários alcançarem seus objetivos.

Como requisitos de usuário, tem-se:

I Deve ser possível treinar;
II Deve ser possível encontrar o caminho ótimo utilizando uma rede treinada como entrada;
III Deve ser possível gerar entradas fictícias para simular os dados de um drone;
IV Deve existir uma forma de visualização do caminho ótimo;

3.2.1.2 REQUISITOS DE SISTEMA

Os requisitos do sistema descrevem as funções, características e restrições que o sistema todo deve possuir para atender aos objetivos globais do projeto. Eles se concentram nos aspectos técnicos, de infraestrutura e de arquitetura do sistema.

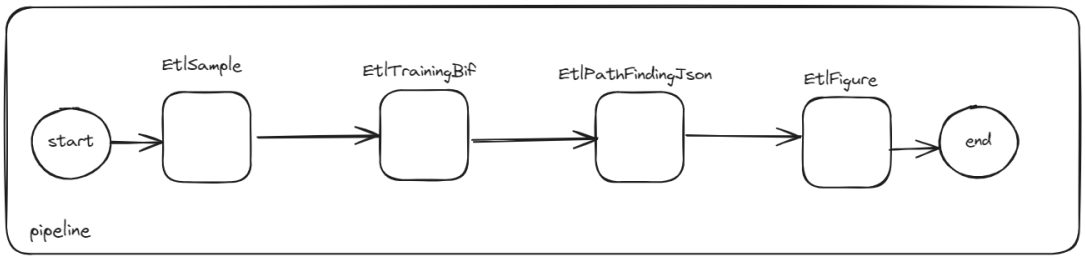
Como requisitos de sistema, tem-se:

I O sistema deve ter um processo que gere dados de entrada para teste;
II O sistema deve interagir gerando arquivos e lendo do formato `BIF`;
III Os dados devem ser armazenados apenas como arquivos;
IV Devem ser armazenados dados de performance de treinamento da rede;
V Devem ser armazenados dados de performance do calculo da trajetória ótima;

3.2.2 ARQUITETURA

A escolha inicial da arquitetura é do padrão pipeline, sendo o pipeline composto por 4 tasks.

Figura 21 – Arquitetura Pipeline



A explicação de cada etapa é a seguinte:

Quadro 1 – Descrição das tarefas.

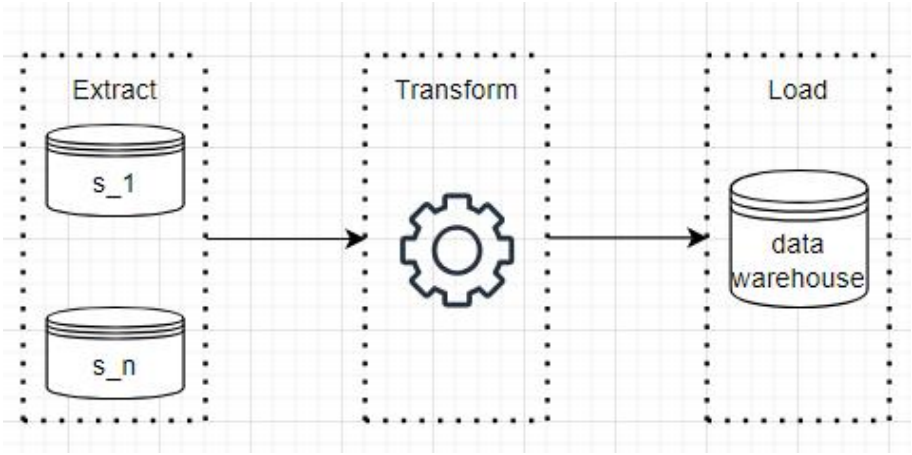
Task	**Descrição**
`EtlSample`	Gera os dados de entrada para treinamento da rede.
`EtlTrainingBif`	Com base nos dados de entrada treina a rede e gera um arquivo `BIF`.
`EtlPathFindingJson`	Com base no arquivo `BIF` e utilizando o algoritmo AStar modificado encontra o caminho ótimo e retorna um arquivo `JSON` com informações para plotar.
`EtlFigure`	Com base no `JSON` gera um arquivo `PNG`

Fonte: Autoria Própria.

3.2.2.1 ETL

O padrão ETL (Extract, Transform, Load) é uma abordagem comum na área de engenharia de dados e integração de dados, usada para mover e transformar dados de uma fonte para um destino.

Figura 22 – Padrão ETL



Extract (Extração):

- Nesta etapa, os dados são coletados de uma ou mais fontes de dados, que podem incluir bancos de dados, arquivos, APIs, entre outros.
- Os dados podem ser extraídos em sua forma bruta ou em algum formato intermediário, dependendo das necessidades do processo de transformação.

Transform (Transformação):

- Durante esta etapa, os dados extraídos são processados e transformados segundo as regras de negócio, requisitos de qualidade de dados e esquemas de destino.
- As transformações podem incluir limpeza de dados, normalização, agregação, enriquecimento com dados de outras fontes, padronização de formatos, entre outras operações.

Load (Carregamento):

- Após a transformação, os dados são carregados no destino, que geralmente é um Data Warehouse, Data Lake, banco de dados relacional ou qualquer outro sistema de armazenamento de dados.
- Durante este processo, os dados transformados são persistidos no destino de forma que possam ser facilmente acessados e consultados para análise, relatórios e outras aplicações.

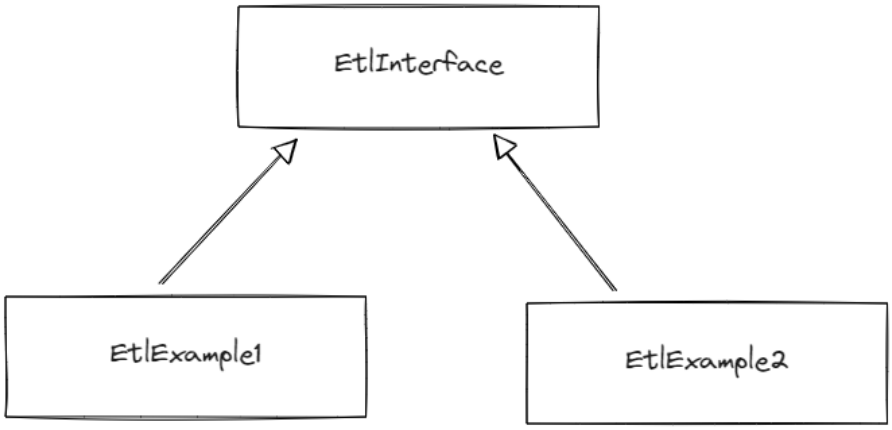
O padrão ETL é fundamental em todas as fases do sistema devido ao requisito de desacoplamento e flexibilidade. Na Figura 23, isso é demonstrado. Vemos que, com base na interface `EtlInterface`, pode-se criar quantos ETLs forem necessários no sistema.

3.2.2.2 CORE

Dentro do sub-pacote `core`, tem-se as principais classes do sistema.

Uma importante é a classe `NetworkTable`, que se comunica com a tabela Tabela 13 armazenada e é fundamental na task `EtlTrainningBif`.

Figura 23 – Implementação de ETL



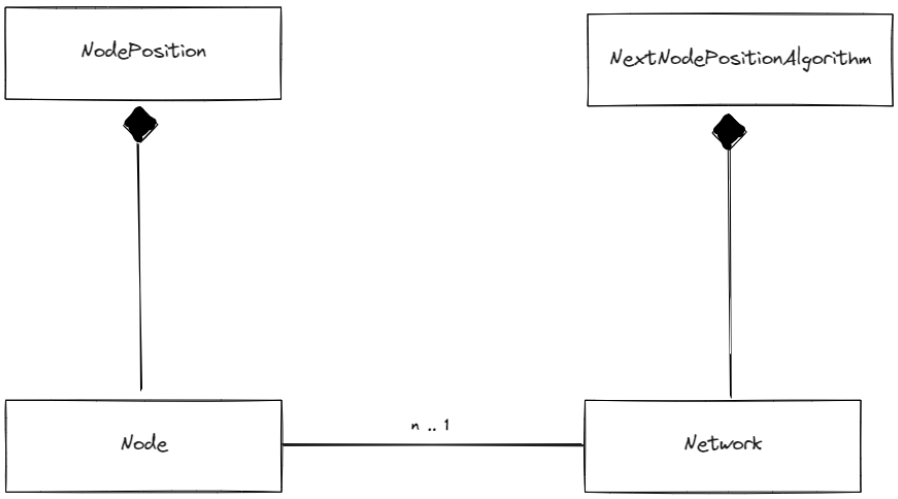
Outra classe do sub-pacote é `Network`, que é também fundamental na task `EtlTrainningBif`. Sua função é representar a estrutura de rede para poder se comunicar com a biblioteca `PGMPY`.

A arquitetura da classe `Network` é bem mais complexa que da `NetworkTable` podendo ser representada pelo diagrama da Figura 24. Nela, tem-se `Node` e `NodePosition` sendo respectivamente responsáveis por representar o Nodo e a Posição (x, y) do Nodo.

Temos também `NextNodePositionAlgorithm`, que utiliza o padrão *Strattegy* para fornecer uma família genérica de algoritmos, que retorna para a classe `Network` a estrutura dos arcos da rede.

Uma implementação que fizemos nesse trabalho de `NextNodePositionAlgorithm` está representada na Figura 18.

Figura 24 – Relações da Classe Network



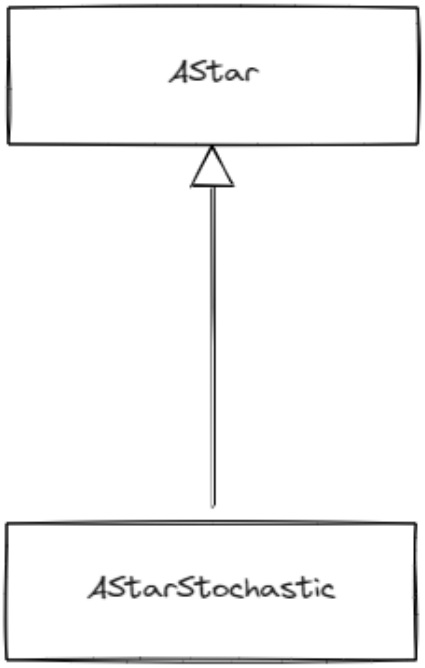
3.2.2.3 PATH FINDING

Outra classe muito importante é a classe `Astar`. Com base nessa classe, é calculado o caminho ótimo. Vale notar que `AStar` é uma classe abstrata e, portanto, não pode-se utilizá-la

diretamente.

Assim, definimos uma implementação de `AStar` chamada de `AStarStochastic`. O diagrama da Figura 25 mostra que `AStarStochastic` herda de `AStar`.

Figura 25 – Implementação de AStar



Os métodos abstratos de `AStar` estão definidos no Quadro 2. Portanto, definindo esses métodos em qualquer implementação, poderemos instanciar um objeto e utilizar a implementação específica para encontrar o caminho usando o algoritmo implementado.

Quadro 2 – Métodos abstratos de AStar.

Nome	**Descrição**
`heuristic_cost_estimate`	Calcula a distância estimada (aproximada) entre um nó e o objetivo.
`distance_between`	Fornece a distância real entre dois nós adjacentes n1 e n2, onde n2 é garantido pertencer à lista retornada pela chamada a `neighbors(n1)`.
`neighbors`	Para um nó dado, retorna (ou produz) a lista de seus vizinhos.

Fonte: Autoria Própria.

3.2.3 AMBIENTE DE DESENVOLVIMENTO

Utilizando o The Hitchhiker's Guide To Python, sabe-se que dois fatores em desenvolvimento Python são importantes:

1. A escolha do ambiente de desenvolvimento;
2. Escrever o código;

3.2.3.1 ESTRUTURA DO PROJETO

A estrutura do projeto é uma parte essencial para a arquitetura do projeto. Um exemplo de arquitetura para o projeto `sample` pode ser mostrado no Quadro 3.

Quadro 3 – Estrutura de um Projeto Python

Nome do Arquivo	**Descrição**
`README.md`	Arquivo com informações sobre o projeto.
`LICENSE`	Arquivo contendo os termos da licença do projeto.
`setup.py`	Script de configuração do pacote Python.
`requirements.txt`	Lista de dependências do projeto.
`sample/`	Diretório contendo o pacote "sample".
`sample/__init__.py`	Arquivo para inicialização do pacote "sample".
`sample/core.py`	Módulo principal contendo a lógica central.
`sample/helpers.py`	Módulo contendo funções auxiliares.
`docs/`	Diretório contendo a documentação do projeto.
`docs/conf.py`	Configurações para a documentação Sphinx.
`docs/index.rst`	Página inicial da documentação.
`tests/`	Diretório contendo os testes do projeto.
`tests/test_basic.py`	Testes básicos para o projeto.
`tests/test_advanced.py`	Testes mais avançados para o projeto.

Fonte: Adaptado de (REITZ; SCHLUSSER, 2016).

3.2.3.2 DEPENDÊNCIAS

Durante o desenvolvimento, ocorreram problemas devido à não utilização de ambientes virtuais.

Figura 26 – Problema com PyParsing

```
C:\Users\willi\OneDrive\Documentos\Programas\Python\study\tcc>pip show pyparsing
Name: pyparsing
Version: 2.4.7
Summary: Python parsing module
Home-page: https://github.com/pyparsing/pyparsing/
Author: Paul McGuire
Author-email: ptmcg@users.sourceforge.net
License: MIT License
Location: c:\users\willi\appdata\roaming\python\python39\site-packages
Requires:
Required-by: matplotlib, pgmpy
```

Há uma incongruência no manuseio do `pyparsing`. Conforme o arquivo `matplotlib/requirements/testing/minver.txt`, é exigido que o `pyparsing >= 2.3.1`. No entanto, o arquivo `pgmpy/requirements/runtime.txt` especifica que o `pyparsing >= 3.0`. Teoricamente, isso não seria um problema se a condição `pyparsing >= 3.0` fosse atendida. Entretanto, ao tentar instalar a biblioteca usando o comando `pip install pgmpy`, o gerenciador de pacotes `pip` detecta que já existe uma versão da dependência `pyparsing` instalada, especificamente a versão `2.4.7`. Como resultado, o `pip` não baixa a versão mais recente do `pyparsing`, levando a uma incompatibilidade de dependência na biblioteca `pgmpy`.

3.2.3.3 AMBIENTE VIRTUAL

A instalação e gerenciamento de ambientes virtuais são práticas essenciais ao desenvolver projetos em Python, proporcionando um ambiente isolado para cada projeto, evitando conflito de dependências. Se você ainda não tem o pacote `virtualenv` instalado, pode fazê-lo executando o seguinte comando no terminal:

```
pip install virtualenv
```

Após a instalação do `virtualenv`, navegue até o diretório do seu projeto utilizando o comando `cd`. Dentro desse diretório, crie um ambiente virtual com o seguinte comando, adaptado conforme seu sistema operacional:

- **No Windows:**

  ```
  python -m venv venv
  ```

- **No Linux/Mac:**

```
python3 -m venv venv
```

Ative o ambiente virtual no terminal:

- **No Windows:**

```
venv\Scripts\activate.bat
```

- **No Linux/Mac:**

```
source venv/bin/activate
```

O ambiente virtual ativado será indicado no prompt do terminal.

Agora, com o ambiente virtual ativo, use o `pip` para instalar as dependências do seu projeto. Por exemplo, para instalar a biblioteca `pgmpy`:

```
pip install pgmpy
```

Quando concluir o trabalho no projeto, desative o ambiente virtual:

```
deactivate
```

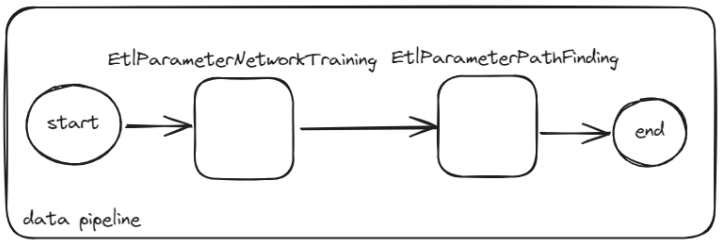
Uma dica adicional é adicionar o diretório do ambiente virtual ao arquivo `.gitignore` para evitar que seja incluído no controle de versão (como Git). Adicione a linha `venv/` ao arquivo `.gitignore`.

Esses passos garantem um ambiente Python isolado e facilitam a gestão de dependências, contribuindo para um desenvolvimento mais consistente e organizado.

3.3 AQUISIÇÃO DE PARÂMETROS

Para a aquisição de parâmetros já medidos, a arquitetura pipeline é o ideal, além de já ter sido implementada na Subseção 3.2.2. A figura Figura 27 demonstra como se configura a pipeline.

Figura 27 – Arquitetura Data Pipeline



As tarefas que compõe a pipeline de parâmetros estão aqui expostas no Quadro 4.

Quadro 4 – Descrição das tarefas.

Task	**Descrição**
`EtlParameterNetworkTraining`	Gera os gráficos de complexidade para network training.
`EtlParameterPathFinding`	Gera os gráficos de complexidade para path finding.

Fonte: Autoria Própria.

3.3.1 DATABASE

Como database, escolhe-se um banco de dados relacional por fornecer as seguintes características. (ELMASRI, 2011)

I **Estrutura de Dados Organizada:** Bancos de dados relacionais organizam os dados em tabelas, que consistem em linhas e colunas. Essa estrutura tabular pode ser mais intuitiva e fácil de entender, especialmente para pessoas familiarizadas com conceitos de tabelas e consultas SQL.

II **Integridade de Dados:** Os bancos de dados relacionais são projetados para garantir a integridade dos dados, implementando regras de integridade referencial e restrições de chave estrangeira. Isso ajuda a manter a consistência e a precisão dos dados, prevenindo a inserção de informações inconsistentes ou inválidas.

III **Flexibilidade e Poder de Consulta:** A linguagem SQL (Structured Query Language) é amplamente utilizada em bancos de dados relacionais, oferecendo uma linguagem poderosa e flexível para consultar e manipular dados. Com SQL, os usuários podem realizar operações complexas de consulta, agregação e junção de dados de forma eficiente.

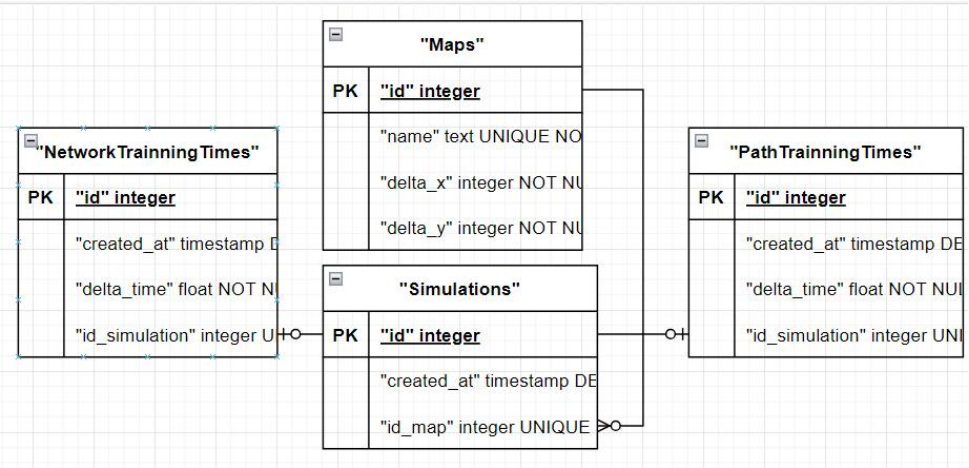
IV **Suporte para Transações ACID:** Os bancos de dados relacionais geralmente garantem a consistência e a confiabilidade das transações por meio do suporte ao ACID (Atomicidade, Consistência, Isolamento, Durabilidade). Isso significa que as operações de banco de dados são executadas de forma confiável, mesmo em caso de falha do sistema ou erro humano.

V **Escalabilidade Vertical:** Embora os bancos de dados relacionais possam enfrentar limitações de escalabilidade vertical (adicionar mais recursos a um único servidor), eles são adequados para muitos casos de uso que não exigem escala massiva, especialmente quando a consistência dos dados é crucial.

VI **Modelagem de Dados Complexos:** Para dados altamente estruturados e com relacionamentos complexos entre entidades, os bancos de dados relacionais oferecem um modelo de dados robusto e flexível, permitindo a representação eficaz de esquemas complexos.

Assim, O modelo físico encontra-se definido na Figura 28.

Figura 28 – Modelo Físico da Database



Pode-se obter ainda todos os dados utilizando a seguinte SQL Query. A Execução da Query deu-se por um cliente em Python utilizando ORM.

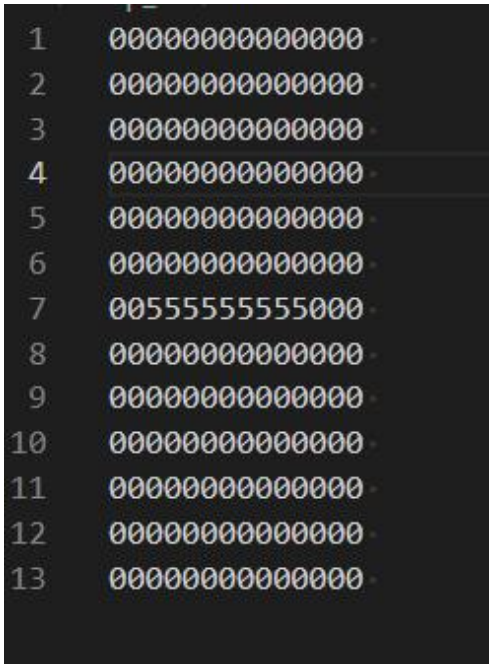
Listing 3.1 – SQL Query

```
SELECT * FROM "maps"
        JOIN simulations
          ON maps.id = simulations.id_map
        JOIN networktrainingtimes
          ON simulations.id = networktrainingtimes.id_simulation
        JOIN pathfindingtimes
          ON simulations.id = pathfindingtimes.id_simulation
```

4 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Os arquivos de entrada são mapas em `ascii` do tipo `.txt`, em que 0 significa sem obstaculo e 5 é o grau mais alto de obstaculo. Um exemplo de arquivo é o da Figura 29.

Figura 29 – ASCII do Mapa 1



Os outros mapas `ascii` usados no projeto estão descritos na tabela Tabela 14.

Tabela 14 – Dados dos Mapas.

name	delta_x	delta_y	nodes
map_1	14	13	182
map_2	19	18	342
map_3	24	23	552
map_4	29	28	812
map_5	34	33	1122

Fonte: Autoria Própria.

Os resultados obtidos são divididos em:

- Complexidade dos Componentes
  - Treinamento da Rede
  - Tempo de Busca da Trajetória
- Trajetórias

4.1 COMPLEXIDADE DOS COMPONENTES

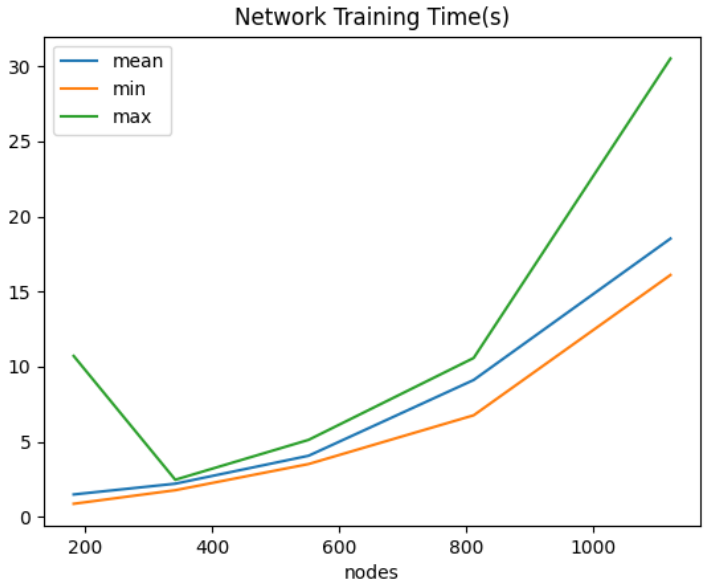
Ambos os gráficos obtidos na complexidade dos componentes constam de:

- Eixo Y: Tempo em segundos;
- Eixo X: Numero de nodos;

4.1.1 TREINAMENTO DA REDE

O comportamento da task `EtlTrainingBif` é descrito na Figura 30.

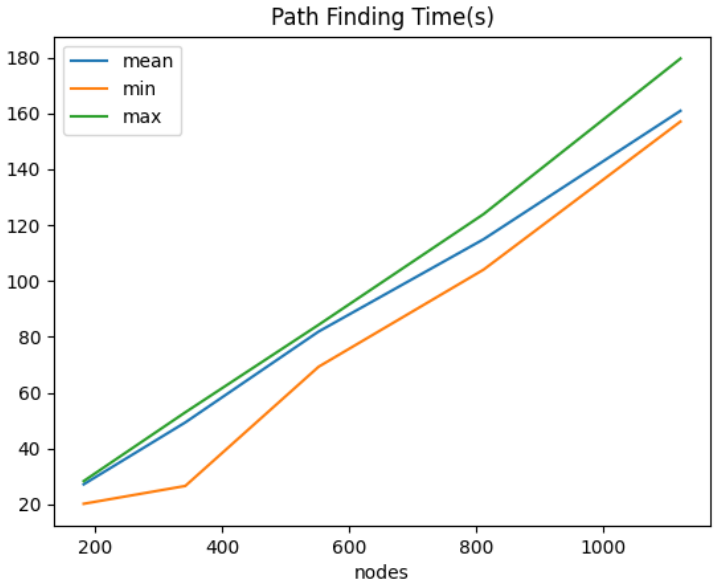
Figura 30 – Complexidade do treinamento da rede



4.1.2 CALCULO DA TRAJETÓRIA

O comportamento da task `EtlPathFindingJson` é descrito na Figura 31.

Figura 31 – Complexidade do cálculo da trajetória



4.2 TRAJETÓRIAS

O resultado do projeto está demonstrado nas Figura 32 e Figura 33. Nela, podemos perceber que se trata de um mapa de calor, na qual o estado dos nodos variam de $0{,}0$ até $5{,}0$. O caminho está delimitado pelas setas em vermelho.

Figura 32 – Trajetória Mapa 1

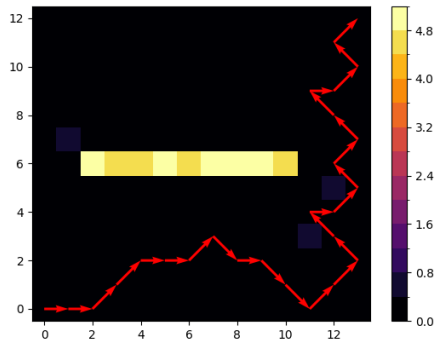
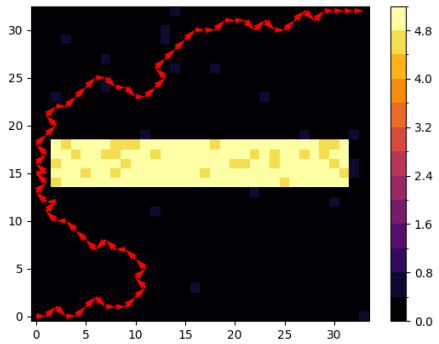


Figura 33 – Trajetória Mapa 5



5 CONCLUSÃO

5.1 TRABALHOS FUTUROS

Para aprimorar nosso projeto, planejamos implementar as seguintes melhorias:

- Integrar drones reais como entrada, ampliando as capacidades do sistema e tornando-o mais próximo das condições reais de operação;
- Refinar o software existente, visando aprimorar sua usabilidade, desempenho e robustez;
- Aprimorar os cálculos no algoritmo AStar, buscando otimizar a eficiência e precisão das rotas traçadas pelos drones;

Essas medidas visam fortalecer a funcionalidade e eficácia do sistema, contribuindo para sua aplicação em diversos cenários e contextos.

5.2 CONSIDERAÇÕES FINAIS

Desenvolver software não é tarefa simples, ainda mais quando se busca escalabilidade e aderência às melhores práticas. Neste projeto, atribuímos uma enorme importância a esse aspecto.

Como demonstrado, conseguimos efetivamente obter resultados e desenvolver o sistema de forma flexível, o que o torna passível de ser estendido no futuro. Essa abordagem não apenas atende às necessidades atuais, mas também prepara o terreno para futuras iterações e expansões, garantindo sua relevância e utilidade a longo prazo.

Referências

CANDRA, A.; BUDIMAN, M. A.; HARTANTO, K. Dijkstra's and a-star in finding the shortest path: a tutorial. In: **2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)**. [S.l.]: IEEE, 2020. Citado na página 14.

ELMASRI, R. **Fundamentals of database systems**. [S.l.]: Addison-Wesley, 2011. 1172 p. ISBN 9780136086208. Citado na página 33.

HECKERMAN, D. A tutorial on learning with bayesian networks. In: **Learning in Graphical Models**. [S.l.]: Kluwer Academic Publishers, 1998. p. 302–354. Citado na página 1.

HLFDEV. **Algorithms discover the power of big O notation**. 2023. Disponível em: <https://medium.com/@hlfdev/algorithms-discover-the-power-of-big-o-notation-17a367bd62a>. Citado na página 24.

JENSEN, F. V.; NIELSEN, T. D. **Bayesian Networks and Decision Graphs (Information Science and Statistics)**. [S.l.]: Springer, 2007. 448 p. ISBN 9780387682815. Citado na página 9.

LEON-GARCIA, A. **Probability and Random Processes For Electrical Engineering (3rd Edition)**. [S.l.]: Prentice Hall, 2007. 608 p. ISBN 9780131471221. Citado na página 5.

PGMPY Documentation. 2023. Disponível em: <https://pgmpy.org/>. Citado na página 41.

PIETSCH, B. **Global drone market estimated to reach $14 billion over next decade: study**. [S.l.], 2019. Disponível em: <https://www.reuters.com/article/us-usa-security-drones-idUSKCN1UC2MU>. Acesso em: 10 de Março de 2023. Citado na página 1.

REITZ, K.; SCHLUSSER, T. **The Hitchhiker's Guide To Python**: Best practices for development. [S.l.]: O'Reilly Media, 2016. 338 p. ISBN 9781491933176. Citado na página 30.

SCHRÖER, C.; KRUSE, F.; GÓMEZ, J. M. A systematic literature review on applying crisp-dm process model. **Procedia Computer Science**, v. 181, p. 526–534, 2021. Citado na página 2.

SOMMERVILLE, I. **Software engineering**. [S.l.]: Pearson, 2016. 810 p. ISBN 9781292096131. Citado na página 20.

XUE, M. Uav trajectory modeling using neural networks. **American Institute of Aeronautics and Astronautics**, 2017. Citado na página 1.

Apêndices

APÊNDICE A – BIBLIOTECA PGMPY

pgmpy é uma biblioteca em Python para trabalhar com Modelos Gráficos Probabilísticos (PGMPY..., 2023). É uma ferramenta poderosa para modelagem, inferência e aprendizado de Redes Bayesianas e outros modelos gráficos probabilísticos.

A.1 DAGs

A classe `pgmpy.base.DAG` representa um grafo direcionado acíclico, que é a estrutura fundamental para redes Bayesianas em pgmpy.

No Quadro 5, estão definidas as funções principais da classe `pgmpy.baseDAG`.

Quadro 5 – Métodos da classe DAG.

Função	**Descrição**
`pgmpy.base.DAG.add_node(node)`	Adiciona um nó ao DAG.
`pgmpy.base.DAG.add_nodes_from(nodes)`	Adiciona vários nós ao DAG.
`pgmpy.base.DAG.add_edge(edge)`	Adiciona uma aresta (relação direcionada) entre os nós no DAG.
`pgmpy.base.DAG.add_edges_from(edges)`	Adiciona várias arestas ao DAG.
`pgmpy.base.DAG.remove_node(node)`	Remove um nó do DAG.
`pgmpy.base.DAG.remove_edge(edge)`	Remove uma aresta do DAG.
`pgmpy.base.DAG.nodes()`	Retorna uma lista de todos os nós no DAG.
`pgmpy.base.DAG.edges()`	Retorna uma lista de todas as arestas no DAG.
`pgmpy.base.DAG.parents(node)`	Retorna uma lista dos pais (nós predecessores) do nó especificado.
`pgmpy.base.DAG.children(node)`	Retorna uma lista dos filhos (nós sucessores) do nó especificado.

Fonte: Adaptado de (PGMPY..., 2023)

Essas funções são comumente utilizadas em bibliotecas de grafos, como `NetworkX` em Python, especialmente quando se trabalha com representações de grafos direcionados acíclicos, como os `DAGs`.

Esses métodos são utilizados para manipular e explorar a estrutura de um Grafo Direcionado Acíclico (`DAG`) na biblioteca `pgmpy`.

A.2 REDES BAYESIANAS

A PGMPY suporta a criação e manipulação de Redes Bayesianas, que são modelos gráficos probabilísticos que representam relações de dependência entre variáveis por meio de um grafo acíclico direcionado (DAG).

Criar uma Rede Bayesiana é bem simples. Usando a Figura 5, temos o seguinte código Python:

Listing A.1 – Criação da Rede

```
from pgmpy.models import BayesianModel

model = BayesianModel(
    [('A', 'B'), ('B', 'D'), ('A', 'C'), ('C', 'D')]
)
```

A.3 ARQUIVO BIF

Nem sempre se faz necessário criar as redes bayesianas na biblioteca. Pode-se fazer o uso do Arquivo *Bayesian Interchange Format* (BIF).

O BIF é uma notação usada para representar redes bayesianas de uma maneira textual. Considere uma rede bayesiana simples com duas variáveis, A e B:

1. A seção "network"define o nome da rede bayesiana.
2. As seções "variable"definem as variáveis A e B, indicando que ambas são discretas com dois estados possíveis ("True"e "False").
3. A seção "probability (A)"especifica a distribuição de probabilidade marginal para a variável A.
4. A seção "probability (B | A)"especifica a distribuição condicional de probabilidade para B dado A.

Listing A.2 – modelo.bif

```
network "Exemplo" {
}

variable "A" {
   type discrete [2] { "True", "False" };
}

variable "B" {
   type discrete [2] { "True", "False" };
}
```

```
probability (A) {
   table 0.6, 0.4;
}

probability (B | A) {
   (True) 0.2, 0.8;
   (False) 0.7, 0.3;
}
```

Essencialmente, o arquivo BIF organiza as informações em seções que descrevem a estrutura da rede e as probabilidades associadas a cada variável, permitindo a representação compacta e legível de relações probabilísticas complexas.

Para ler um arquivo BIF, basta fazer:

Listing A.3 – Importando arquivo BIF

```
from pgmpy.externals import import_bif

# Leia um modelo bayesiano do formato BIF
model_bif = import_bif.read_bif('modelo.bif')
```

Pode-se também armazenar as redes que forem criadas usando a biblioteca:

Listing A.4 – Exportando arquivo BIF

```
from pgmpy.models import BayesianModel
from pgmpy.estimators import ParameterEstimator
from pgmpy.externals import BayesianModel as BNet
from pgmpy.externals import export_bif

# Crie um modelo bayesiano
model = BayesianModel([('A', 'B')])

# Adicione dados ao modelo
#(isso pode variar dependendo dos seus dados reais)
data = pd.DataFrame(
    data={'A': [0, 1, 0, 1], 'B': [0, 0, 1, 1]}
)
model.fit(data)

# Exporte o modelo para o formato BIF
export_bif.write_bif(model, 'modelo.bif')
```

A.4 EXEMPLO DE INFERÊNCIA DE PROBABILIDADES

A biblioteca oferece ferramentas para realizar inferência probabilística em Redes Bayesianas, permitindo calcular probabilidades condicionais e realizar inferências de máxima verossimilhança.

Listing A.5 – Calculo de Probabilidade

```
from pgmpy.inference import VariableElimination

inference = VariableElimination(model)
result = inference.query(
    variables=['B'], evidence={'A': 1, 'C': 0}
)
print(result)
```

A.5 APRENDIZADO DE ESTRUTURA E PARÂMETROS

A pgmpy suporta tanto o aprendizado de parâmetros quanto o aprendizado de estrutura em Redes Bayesianas a partir de dados.

Listing A.6 – Aprendizado de Estrutura

```
from pgmpy.estimators import (
    ParameterEstimator,
    StructureEstimator
)
from pgmpy.data import data

data = data.Titanic()
model = StructureEstimator(data).estimate()
```

Para fornecer um exemplo de aprendizagem de parâmetros em redes Bayesianas utilizando a biblioteca pgmpy, primeiro, precisamos criar uma estrutura de rede bayesiana e, em seguida, aprender os parâmetros com base em dados observados.

Vamos criar um exemplo simples de uma rede bayesiana com duas variáveis: "Chuva"e "Guarda-chuva". A ideia é modelar a probabilidade de alguém pegar um guarda-chuva com base na presença ou ausência de chuva. Aqui está o código:

Listing A.7 – Aprendizado de Parâmetros

```
# Instale a biblioteca pgmpy se ainda nao tiver instalado
# pip install pgmpy

from pgmpy.models import BayesianModel
from pgmpy.estimators import ParameterEstimator
from pgmpy.estimators import MaximumLikelihoodEstimator
import pandas as pd

# Criar o dataframe de dados de treinamento
data = pd.DataFrame(
    data = {
        'Chuva': [1, 1, 0, 0, 1, 0, 1, 0],
        'Guarda_chuva': [1, 1, 0, 0, 1, 0, 1, 0]
    }
)

# Criar a estrutura da rede bayesiana
model = BayesianModel([('Chuva', 'Guarda_chuva')])
```

```
# Aprender os parametros usando Estimativa de Maxima
# Verossimilhanca (MLE)
model.fit(data, estimator=MaximumLikelihoodEstimator)

# Imprimir a Tabela de Probabilidades Condicional
# (CPD) aprendida
for cpd in model.get_cpds():
    print("CPD", cpd.variable)
    print(cpd)
```

Neste exemplo, `data` é um DataFrame que contém observações de duas variáveis: "Chuva"e "Guarda-chuva". A rede bayesiana é definida com uma aresta direcionada de "Chuva"para "Guarda-chuva", indicando que a variável "Guarda-chuva"depende da variável "Chuva".

A função `fit` é usada para aprender os parâmetros do modelo a partir dos dados de treinamento. Neste caso, usamos o Estimador de Máxima Verossimilhança (Maximum Likelihood Estimator - MLE) para estimar os parâmetros.

Finalmente, imprimimos as Tabelas de Probabilidades Condicional (CPDs) aprendidas para cada variável na rede bayesiana.

A.6 OUTROS MODELOS PROBABILÍSTICOS

Além de Redes Bayesianas, a `pgmpy` oferece suporte a outros modelos gráficos probabilísticos, como Modelos de Markov.

Listing A.8 – Criação de Modelo de Markov

```
from pgmpy.models import MarkovModel

model = MarkovModel([('A', 'B'), ('C', 'B')])
```

APÊNDICE B – Implementação da Interface ETL

Listing B.1 – etl_interface.py

```
import abc


class EtlInterface(metaclass = abc.ABCMeta):
    @classmethod
    def __subclasshook__(cls, subclass):
        return (
                hasattr(subclass, 'extract') and
                callable(subclass.extract) and

                hasattr(subclass, 'transform') and
                callable(subclass.transform) and

                hasattr(subclass, 'load') and
                callable(subclass.load) and

                hasattr(subclass, 'etl') and
                callable(subclass.etl) or

                NotImplemented
                )

    @abc.abstractmethod
    def extract(self):
        """Load in the data set"""
        raise NotImplementedError

    @abc.abstractmethod
    def transform(self):
        """Extract text from the data set"""
        raise NotImplementedError

    @abc.abstractmethod
    def load(self):
```

```
        """Extract text from the data set"""
        raise NotImplementedError

    @abc.abstractmethod
    def etl(self):
        """Extract text from the data set"""
        raise NotImplementedError
```

Anexos

ANEXO A – Implementação do Algoritmo AStar

Listing A.1 – a_star_algorithm.py

```
class AStar(ABC, Generic[T]):
    __slots__ = ()

    @abstractmethod
    def heuristic_cost_estimate(self, current: T, goal: T) ->
        float:
        """
        Computes the estimated (rough) distance between a node
            and the goal.
        The second parameter is always the goal.
        This method must be implemented in a subclass.
        """
        raise NotImplementedError

    @abstractmethod
    def distance_between(self, n1: T, n2: T) -> float:
        """
        Gives the real distance between two adjacent nodes n1
            and n2 (i.e n2
        belongs to the list of n1's neighbors).
        n2 is guaranteed to belong to the list returned by the
            call to neighbors(n1).
        This method must be implemented in a subclass.
        """
        raise NotImplementedError

    @abstractmethod
    def neighbors(self, node: T) -> Iterable[T]:
        """
        For a given node, returns (or yields) the list of its
            neighbors.
        This method must be implemented in a subclass.
        """
        raise NotImplementedError
```

```
def is_goal_reached(self, current: T, goal: T) -> bool:
    """
    Returns true when we can consider that 'current' is the
       goal.
    The default implementation simply compares `current ==
       goal`, but this
    method can be overwritten in a subclass to provide more
       refined checks.
    """
    return current == goal

def reconstruct_path(self, last: SearchNode, reversePath=
   False) -> Iterable[T]:
    def _gen():
        current = last
        while current:
            yield current.data
            current = current.came_from

    if reversePath:
        return _gen()
    else:
        return reversed(list(_gen()))

def astar(
    self, start: T, goal: T, reversePath: bool = False
) -> Union[Iterable[T], None]:
    if self.is_goal_reached(start, goal):
        return [start]

    openSet: OpenSet[SearchNode[T]] = OpenSet()
    searchNodes: SearchNodeDict[T] = SearchNodeDict()
    startNode = searchNodes[start] = SearchNode(
        start, gscore=0.0, fscore=self.
           heuristic_cost_estimate(start, goal)
    )
    openSet.push(startNode)
```

```
        while openSet:
            current = openSet.pop()

            if self.is_goal_reached(current.data, goal):
                return self.reconstruct_path(current,
                    reversePath)

            current.closed = True
            for neighbor in map(lambda n: searchNodes[n], self.
                neighbors(current.data)):
                if neighbor.closed:
                    continue

                tentative_gscore = current.gscore + self.
                    distance_between(
                    current.data, neighbor.data
                )

                if tentative_gscore >= neighbor.gscore:
                    continue

                neighbor_from_openset = neighbor.in_openset

                if neighbor_from_openset:
                    # we have to remove the item from the heap,
                        as its score has changed
                    openSet.remove(neighbor)

                # update the node
                neighbor.came_from = current
                neighbor.gscore = tentative_gscore
                neighbor.fscore = tentative_gscore + self.
                    heuristic_cost_estimate(
                    neighbor.data, goal
                )

                openSet.push(neighbor)

        return None
```