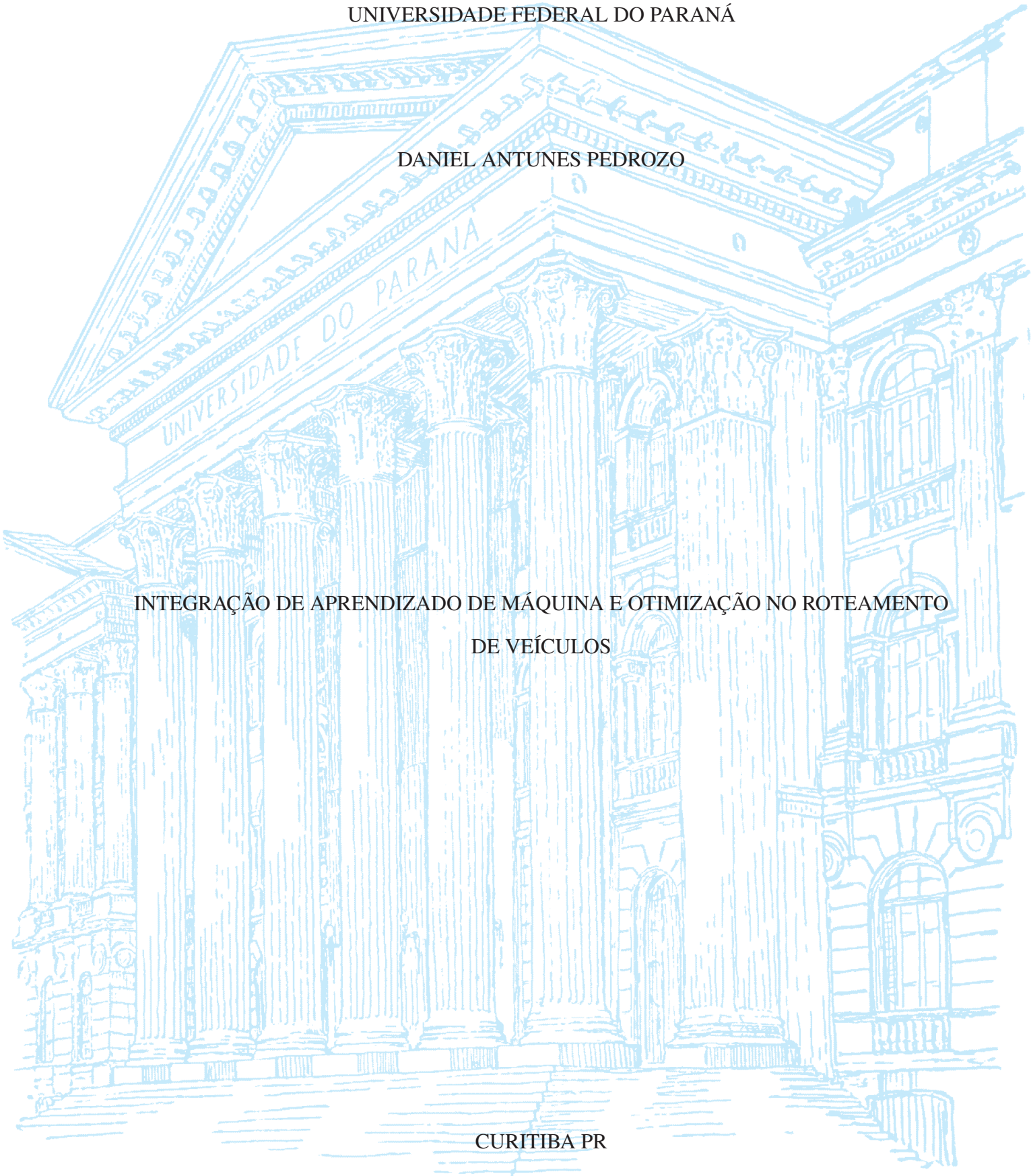
UNIVERSIDADE FEDERAL DO PARANÁ

DANIEL ANTUNES PEDROZO




# INTEGRAÇÃO DE APRENDIZADO DE MÁQUINA E OTIMIZAÇÃO NO ROTEAMENTO DE VEÍCULOS

CURITIBA PR

2024

DANIEL ANTUNES PEDROZO

# INTEGRAÇÃO DE APRENDIZADO DE MÁQUINA E OTIMIZAÇÃO NO ROTEAMENTO DE VEÍCULOS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação*.

Orientador: Fabiano Silva.

Coorientador: Jorge Augusto Meira.

CURITIBA PR

2024

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Pedrozo, Daniel Antunes
Integração de aprendizado de máquina e otimização no roteamento de veículos / Daniel Antunes Pedrozo. – Curitiba, 2024.
1 recurso on-line : PDF.

Dissertação (Mestrado) - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática.

Orientador: Fabiano Silva
Coorientador: Jorge Augusto Meira

1. Aprendizado do computador. 2. Sistemas inteligentes de transportes. 3. Otimização combinatória. 4. Veículos - Rotas. I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Informática. III. Silva, Fabiano. IV. Meira, Jorge Augusto. V. Título.

Bibliotecário: Elias Barbosa da Silva CRB-9/1894

# TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **DANIEL ANTUNES PEDROZO** intitulada: **INTEGRAÇÃO DE APRENDIZADO DE MÁQUINA E OTIMIZAÇÃO NO ROTEAMENTO DE VEÍCULOS**, sob orientação do Prof. Dr. FABIANO SILVA, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 04 de Novembro de 2024.

Assinatura Eletrônica
05/11/2024 15:21:32.0
FABIANO SILVA
Presidente da Banca Examinadora

Assinatura Eletrônica
05/11/2024 08:05:23.0
ANDRÉ LUIZ PIRES GUEDES
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
05/11/2024 15:28:59.0
MARIANE REGINA SPONCHIADO CASSENOTE
Avaliador Externo (UNIVERSIDADE POSITIVO)

Assinatura Eletrônica
04/11/2024 21:26:44.0
PAULO RICARDO LISBOA DE ALMEIDA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
07/11/2024 11:18:17.0
JORGE AUGUSTO MEIRA
Coorientador(a) (UNIVERSITé DU LUXEMBURG)

## AGRADECIMENTOS

Gostaria de expressar minha sincera gratidão a todas as pessoas que me apoiaram e contribuíram para a realização desta dissertação de mestrado.

Primeiramente, agradeço aos meus colegas de mestrado, Emerson Kojima e Adriel Martins, por toda a ajuda e suporte ao longo deste percurso. Emerson por sua fé inabalável e Adriel por se disponibilizar a me ajudar em momentos difíceis.

Agradeço também ao meu orientador, Fabiano Silva, e ao coorientador, Jorge Augusto Meira, por terem a coragem de orientar alguém de uma área distinta e por todo o suporte e apoio oferecido em cada etapa do projeto. Vocês me guiaram e inspiraram a seguir em frente, mesmo diante dos desafios.

Sou muito grato aos meus colegas do Banco Central, Ricardo Rezende e André Hollatz, sem os quais esta jornada não teria sequer começado. Obrigado pelo incentivo inicial e pelo esforço em ler este trabalho, sugerindo aperfeiçoamentos que o tornaram melhor.

Por fim, deixo o meu agradecimento mais importante: o suporte da minha família. Agradeço à minha esposa, Nayana, pelo incansável apoio e pela compreensão que me permitiram dedicar tempo aos estudos. E às minhas filhas, Clara Helena e Antônia Maria, por serem a minha maior motivação e a razão de todo este esforço.

A todos, meu muito obrigado.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES)

## RESUMO

O problema de roteamento de veículos capacitado é um problema fundamental de otimização combinatória e de grande interesse para as áreas de logística e de sistemas de transporte. Tradicionalmente, algoritmos exatos, heurísticas e metaheurísticas são utilizados para resolver o problema. Os métodos exatos, como o empregado pelo resolvedor SCIP, garantem soluções ótimas, porém são inviáveis para grandes instâncias do problema, devido à complexidade exponencial deste. Heurísticas e metaheurísticas obtêm soluções aproximadas em tempo razoável ao explorar o espaço de soluções de maneira eficiente. Entretanto, esses métodos necessitam, geralmente, de conhecimentos específicos acerca do problema em questão, não sendo facilmente generalizáveis. Avanços recentes em aprendizado de máquina oferecem caminhos promissores para superar essas limitações, embora ainda não tenham sido extensivamente explorados. Nesta dissertação, nós enfrentamos o problema de roteamento de veículos capacitado sob duas perspectivas: a do método exato e a do aprendizado profundo, a fim de comparar os seus resultados. O método exato utiliza um modelo em programação inteira elaborado na interface Python do resolvedor SCIP. Já a abordagem de aprendizado de máquina é baseada em uma Rede de Atenção em Grafos treinada sob o paradigma do aprendizado por reforço. Especificamente, analisamos três estratégias de treinamento para estes modelos: REINFORCE baseada em *Greedy Rollout*, o estimador de Monte Carlo Diferenciável (DiCE) e uma versão aprimorada da arquitetura utilizando a função de ativação Mish para tornar o DiCE mais eficiente ($DiCE_{Mish}$). Experimentos demonstram que, embora o modelo SCIP forneça soluções ótimas, ele é computacionalmente intensivo e inviável para grandes instâncias. Por outro lado, os modelos de redes neurais obtêm soluções quase-ótimas com um custo computacional reduzido. O modelo $DiCE_{Mish}$, em particular, aprimora o treinamento e a qualidade da solução ao empregar modificações na arquitetura da rede para permitir um maior fluxo dos gradientes. Esses resultados indicam que as redes de atenção em grafos treinadas sob técnicas de aprendizado por reforço oferecem uma alternativa viável e eficiente aos métodos tradicionais (exatos, heurísticos e metaheurísticos) para resolver o problema do roteamento de veículos capacitado com um menor custo computacional e com maior capacidade de generalização.



Palavras-chave: roteamento; otimização combinatória; scip; aprendizado de máquina; redes de atenção; reinforce; dice.

## ABSTRACT

The Capacitated Vehicle Routinf Problem (CVRP) is a essential problema in combinatorial optimization and of great interest to the logistic and transportantion systems fields. Traditionally, exact algorithms, heuristics and metaheuristics have been used to solve it. Exact methods, like the one employed by SCIP, guarantee optimal solutions but are impratical for larger instances due to its exponential time complexity. Heuristics and metaheuristics provide aproximate solutions in a reasonable time by exploring the solution space efficiently. However, they often require problem-specific tuning, limiting their generalization capacities. Recent developments in machine learning offer promising avenues to overcome this limitations, while not being exaustively explored. In this dissertation, we tackle the CVRP by using two distinct approaches to compare their results: one will use the exact method by implementing a model using the SCIP framework in python; the other will use deep learning in the form of a Graph Attention Network (GAT) trained under the reinforcement learning paradigm. Specifically, we implement and evaluate three training strategies for the GAT models: REINFORCE with a greedy rollout baseline, the differentiable Monte Carlo estimator (DiCE), and an enhanced architecture incorporating the Mish activation function to make DiCE more efficient ($DiCE_{Mish}$). Our experiments show that while the SCIP model delivers optimal solutions, it is computationally intensive and impratical for larger instances. In contrast, the GAT models achieve near optimal solutions with reduced computational cost and in less time. The $DiCE_{Mish}$, in particular, improves training and solution quality through architectural modifications that enhance gradient flow. These results indicate that GATs trained with reinforcement learning techniques offer a viable and efficient alternative to the traditional exact, heuristic and metaheuristic methods for solving the CVRP with a reduced computational cost and improved generalization capacities.



Keywords: routing; combinatorial optimization; scip; graph attention network; reinforce; dice.

## LISTA DE FIGURAS

## LISTA DE ACRÔNIMOS

| | |
|---|---|
| VRP | Vehicle Routing Problem |
| TSP | Traveling Salesman Problem |
| CVRP | Capacitaded Vehicle Routing Problem |
| LP | Linear Programming |
| IP | Integer Programming |
| MIP | Mixed Integer Programming |
| MH | Metaheurística |
| GNN | Graph Neural Networks |
| RNN | Recurrent Neural Networks |
| PN | Pointer Networks |
| RL | Reinforcement Learning |
| AC | Ator-Crítico |
| FC | Fully Connected Layer |
| BC | Batch Normalization |
| MHA | Multi-head Attention Mechanism |

**SUMÁRIO**

# 1 INTRODUÇÃO

Este trabalho aborda o problema de roteamento de veículos (VRP, na sigla em inglês) sob duas perspectivas: a do método exato, utilizando o resolvedor SCIP e a do aprendizado de máquina, fazendo uso de uma rede neural de atenção em grafos. Busca-se comparar as soluções apresentadas em cada método em relação ao tempo de execução e ao valor apresentado. O problema de roteamento de veículos busca otimizar as rotas percorridas por uma frota de veículos responsáveis por satisfazer a demanda de um conjunto de consumidores de modo a minimizar uma determinada função de custo – geralmente uma função da distância percorrida, do tempo dispendido ou do combustível gasto.

A otimização de rotas é um problema tradicional na área de logística, pesquisa operacional e ciência de computação. Sua primeira formulação data de 1959, com o trabalho seminal de Dantzig e Ramser (Dantzig e Ramser, 1959). No artigo em questão, os autores estudaram como otimizar o despacho de caminhões a fim de minimizar a distância total percorrida. Desde então, o campo de estudo do VRP deu origem a diversas ramificações, originando um campo de pesquisa que propõe diversas abordagens para a sua solução, a depender da instância utilizada.

Apesar das décadas de estudo, o VRP ainda possui significativa relevância. O problema continua a fomentar o desenvolvimento de novas estratégias para a sua solução, algoritmos e novas percepções, desempenhando papel importante na indústria do transporte, onde os algoritmos de otimização podem impactar positivamente a estrutura de custos. Otimização é o fator chave, uma vez que uma solução que leva à melhor rota em cada cenário pode ser encontrada minimizando funções objetivos modeladas matematicamente.

O processo de otimização pode ser dividido em quatro estágios: i. formulação do problema; ii. elaboração do modelo do problema; iii. otimização do problema; e iv. implementação da solução. Em outras palavras, o processo de otimização compreende o modelo matemático de um problema em termos de funções objetivos e restrições relacionadas (Talbi, 2009). No caso do VRP, o problema pode ser modelado como uma função que minimiza variáveis como o consumo de combustível, tempo de viagem e/ou distância; e as restrições podem ser vistas como a necessidade de partir e retornar a um depósito específico, a capacidade de carga dos veículos utilizados, a janela de tempo para a entrega, dentre outros fatores. Uma vez que o problema seja corretamente modelado, este poderá ser resolvido por diferentes técnicas. Ao longo do tempo, essas técnicas foram agrupadas em três grandes grupos: resolvedores exatos, soluções heurísticas e soluções metaheurísticas. Os métodos exatos, como a programação linear inteira, programação dinâmica e métodos de busca em árvore (Asghari e Mirzapour Al-e hashem, 2021), podem encontrar a solução ótima para uma determinada instância do VRP, no entanto, eles são limitados pelo número de nós (como o número de consumidores em uma rota) da instância em questão (Sabet e Farooq, 2022). Tal limitação decorre da natureza combinatória do VRP, este é considerado um problema NP-completo. Como tal, a sua complexidade aumenta exponencialmente à medida que o número de nós aumenta, tornando computacionalmente inviável encontrar uma solução exata em um tempo viável para grandes instâncias. Um exemplo de resolvedor exato é o software CPLEX, desenvolvido pela IBM. Ele usa algoritmos $Branch - and - bound$ e $Branch - and - cut$ para resolver modelos de programação linear inteira mista.

Em termos gerais, quando não é possível se utilizar de resolvedores exatos para solucionar cenários reais, nos quais o número de nós tende a ser grande e a aumentar ao longo do tempo, uma alternativa é usar algoritmos heurísticos. Um algoritmo heurístico faz uso de

um processo iterativo para encontrar soluções sub-ótimas dentro de um tempo de execução viável (polinomial). Entretanto, o desenho de algoritmos heurísticos para resolver instâncias de problemas de otimização requer conhecimento especializado sobre aquela determinada instância, pouco aproveitando as características comuns aos problemas e dificultando a sua generalização (Talbi, 2009). Os algoritmos heurísticos podem ser divididos em construtivos – que se utilizam da construção de rotas de maneira serial ou paralela para indicar uma solução inicial – e de melhoria – que varrem uma determinada vizinhança de soluções até que nenhuma melhoria na solução apresentada possa ser encontrada (Asghari e Mirzapour Al-e hashem, 2021).

Com o intuito de melhorar a eficiência dos algoritmos heurísticos, seja para torná-los mais generalistas ou aumentar o seu desempenho, pode-se usar algoritmos metaheurísticos. Em Talbi (2009), o autor afirma que os algoritmos metaheurísticos podem ser utilizados em problemas de otimização, especificamente nos problemas de otimização combinatória. Esses algoritmos podem ser interpretados como um conjunto de regras (normas ou boas práticas) que orientam o processo de elaboração de um algoritmo heurístico para encontrar soluções aproximadamente ótimas.

De maneira geral, os algoritmos metaheurísticos podem ser subdivididos em (Talbi, 2021): i. metaheurísticas de única solução – são técnicas de otimização que buscam melhorar uma solução única. Esses métodos exploram vizinhanças ou buscam por trajetórias dentro do espaço de solução do problema alvo. Um exemplo desse tipo de técnica é o algoritmo Adaptative Large Neighbourhood Search (ALNS); e ii. metaheurísticas baseadas em população – são algoritmos que, por meio de um processo de iteração, buscam refinar a população de soluções até que um critério de parada seja atingido. Exemplos desses algoritmos são os algoritmos genéticos (GA, em inglês) e o algoritmo Particle Swarm Optimization (PSO).

O desenvolvimento de metaheurísticas para a solução do VRP possui uma longa história, com inúmeras contribuições realizadas nas últimas décadas (Caceres Cruz et al., 2014). Entretanto, cumpre ressaltar que a integração de aprendizado de máquina com o problema de roteamento de veículos e as técnicas supracitadas acabou sendo, com poucas exceções, negligenciada (Sabet e Farooq, 2022). As técnicas de aprendizado de máquina podem ser utilizadas em várias etapas da implementação de heurísticas e metaheurísticas, como na seleção de algoritmos, avaliação da função objetivo, ajuste de parâmetros, melhorando o seu desempenho (Karimi-Mamaghan et al., 2022).

As últimas contribuições nesse campo propõem integrar aprendizado de máquina e algoritmos de otimização por meio de Redes Neurais de Grafos (GNN, em inglês). Essa metodologia pode ser implementada de diversas maneiras: redes neurais de grafos tradicionais, redes de atenção em grafos (GAT, na sigla em inglês) e GAT estendidas, que incorporam camadas neurais customizadas, como a descrita em (Lei et al., 2022). Essas abordagens apresentam grande potencial para aumentar o desempenho, como o tempo gasto para encontrar uma solução aproximada, na resolução de problemas de roteamento. Diferentemente dos métodos convencionais, como os algoritmos heurísticos – que necessitam de conhecimento especializado sobre o problema -, as GNN podem produzir algoritmos mais flexíveis e eficientes.

Redes neurais de grafos são objeto de estudo há décadas (Gori et al., 2005). Em 2018, pesquisadores da DeepMind e do Google Brain publicaram um artigo apresentando uma estrutura para a construção de redes de grafos como se estes fossem blocos independentes chamada de Graph Networks (Battaglia et al., 2018). A grande vantagem dessa estrutura é a possibilidade de modelar as dependências presentes nas informações estruturadas em forma de grafos, a fim de que essas redes possam ser usadas para resolver problemas combinatórios, como o VRP. Em outras palavras, as GNN podem discernir as regras que governam a relação entre os nós de um grafo. Desse modo, essas redes conseguem predizer rotas que poderiam minimizar a função

custo enquanto aderem às restrições do problema, como a capacidade de carga do veículo. Essa forma de elaborar redes neurais de grafos, proposta pela DeepMind pode ser usada para construir redes complexas, aumentando a sua capacidade representacional.

As GAT podem ser vistas como uma evolução das GNN por introduzirem mecanismos de atenção à arquitetura, de modo que esta possa atribuir valores relativos à importância de determinados nós para a solução do problema em questão. A técnica de aplicar mecanismos de atenção a redes de grafos foi primeiramente apresentada em 2017 por Veličković et al. no artigo Graph Attention Networks (Velickovic et al., 2018). A técnica permite que o foco aplicado a determinada parte do grafo possa variar, permitindo que a rede atribua os pesos corretos para indicar a importância do nó em questão para a solução do problema, otimizando a predição de rotas, aumentando a acurácia da solução apresentada. A capacidade de uma GAT avaliar a importância de diferentes partes do grafo acaba por refinar a predição das rotas, tornando o algoritmo mais responsivo à instância do VRP trabalhada ao indicar dinamicamente a quais partes da informação contida no grafo o algoritmo deverá prestar mais atenção.

Em sua maioria, as tecnologias citadas buscam representar as informações estruturadas em grafo utilizando-se dos atributos dos nós (como as demandas dos consumidores, por exemplo) e das informações sobre o conjunto de arestas de saída e o conjunto de arestas de chegada dos vértices (chamado também de *edge index*, em inglês). Entretanto, as arestas podem conter informações relevantes acerca da instância em questão, como as distâncias de um nó ao outro. Para fazer uso de tais informações, novas redes neurais de grafos, as quais chamaremos de GAT estendidas, fazem uso de camadas de processamento customizadas que visam integrar as representações das informações das arestas àquelas fornecidas pelos atributos dos vértices, para então serem objeto de análise pelo mecanismo de atenção (Lei et al., 2022). Tal extensão pode fornecer uma representação mais detalhada da informação contida no grafo, proporcionando um entendimento mais aprofundado da instância do problema de roteamento em questão, contribuindo para o desenvolvimento de soluções mais precisas.

O treinamento da GAT elaborada neste trabalho foi realizado sob o paradigma do aprendizado por reforço, utilizando o algoritmo REINFORCE (Williams, 1992) para realizar a estimativa dos gradientes. Nesse contexto, os parâmetros da rede neural são ajustados com base na expectativa das recompensas associadas às rotas geradas pela GAT. A principal estratégia adotada foi o *Greedy Rollout*, na qual o modelo compara as rotas geradas pela GAT com aquelas que seriam obtidas por essa mesma rede neural em um modo determinístico, calculando o que é conhecido como vantagem. O gradiente estimado é então dado pela multiplicação da vantagem pela probabilidade logarítmica da rota gerada. Além disso, como uma alternativa para a estimação dos gradientes, foi empregado o método DiCE (Foerster et al., 2018). Esse método oferece uma abordagem mais flexível e potencialmente menos custosa computacionalmente.

De maneira resumida, a comparação entre métodos exatos e técnicas baseadas em aprendizado de máquina é de grande importância no âmbito do problema de roteamento de veículos. Enquanto os métodos exatos garantem soluções ótimas, o seu uso em instâncias maiores pode se tornar impraticável devido à escalabilidade e ao tempo de execução. Os modelos de aprendizado de máquina, por outro lado, como as GAT, apesar de oferecerem soluções sub-ótimas, são mais flexíveis e capazes de enfrentar instâncias maiores do problema dentro de um tempo razoável. Comparar essas abordagens nos permite não apenas identificar a relação de custo-benefício entre a precisão, em termos da qualidade da solução encontrada, e o desempenho, o tempo dispendido para encontrar a solução, mas também identificar novos caminhos para pesquisas que integrem o melhor de cada técnica, avançando o nosso entendimento na otimização de problemas de roteamento.

A principal hipótese desse trabalho é a de que as arquiteturas de Redes de Atenção em Grafos podem oferecer uma alternativa eficaz aos resolvedores exatos para encontrar soluções eficientes, em termos da distância total percorrida, na solução do problema de roteamento de veículos capacitado. A segunda hipótese avaliada é a de que o estimador DiCE representa uma alternativa viável e computacionalmente mais eficiente ao uso do estimador REINFORCE com baseline para o treinamento das Redes de Atenção em Grafos.

A verificação destas hipóteses envolveu o treinamento de uma GAT em um ambiente de aprendizado por reforço utilizando dois estimadores de gradientes diferentes, REINFORCE e DiCE, comparando o seu desempenho com um método exato implementado na estrutura fornecida pelo resolvedor SCIP. Ambos os modelos, SCIP e GAT, foram avaliados no mesmo conjunto de testes, garantindo uma comparação justa em termos da distância percorrida e do tempo necessário para se encontrar uma solução. Os resultados apresentados corroboram as hipóteses levantadas. Primeiramente, as GAT provaram ser uma alternativa promissora para o problema de roteamento de veículos capacitado (CVRP), alcançando soluções próximas ao ótimo (com valores apenas $4,3\%$ superiores a este, no caso do REINFORCE com *Greedy Rollout*, e $5,3\%$, no caso da implementação com o estimador DiCE) e com um tempo de processamento significativamente menor, mesmo em instâncias maiores. Além disso, os experimentos demonstraram que o estimador DiCE é uma alternativa viável e computacionalmente mais eficiente em comparação com o REINFORCE com *Greedy Rollout* para o treinamento das GAT. A implementação com o DiCE apresentou um tempo de treinamento cerca de 30% menor, sem comprometer a qualidade das soluções geradas. Assim, as abordagens baseadas em aprendizado profundo, especialmente com o uso do estimador DiCE, mostraram-se eficazes e escaláveis, confirmando o potencial das GAT para resolver o CVRP em cenários onde o uso de métodos exatos se torna inviável devido às limitações de tempo ou recursos computacionais.

As próximas seções deste trabalho estão organizadas de maneira a aprofundar os conceitos e técnicas citados. No **Capítulo 2**, realizamos uma explanação teórica dos conceitos, como a definição do problema e as técnicas utilizadas para a sua solução, como os métodos exatos, as heurísticas, as metaheurísticas e o aprendizado de máquina. O **Capítulo 3** apresenta alguns dos trabalhos mais importantes realizados na área de roteamento de veículos. O **Capítulo 4** detalha os materiais e métodos, descrevendo a implementação do modelo exato na estrutura do resolvedor SCIP e as implementações das Redes de Atenção em Grafos. Já no **Capítulo 5**, apresentamos os experimentos realizados, incluindo o treinamento das redes neurais, a comparação do método exato com o método de aprendizado profundo e a comparação entre os estimadores REINFORCE e DiCE na resolução do problema de roteamento de veículos capacitado. Por último, no **Capítulo 6** apontamos as conclusões e as direções para trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Esse capítulo apresenta o fundamento teórico que provê o arcabouço sobre o qual o experimento será construído. Busca-se explicar os conceitos envolvidos e as ferramentas utilizadas. Primeiramente, define-se o problema de roteamento de veículos, o qual será a base dos esforços de otimização. Em seguida, apresentam-se as bases da programação linear, inteira e mista, as quais são necessárias para o correto entendimento do problema em questão. Na sequência, importantes conceitos de aprendizado de máquina são explicados para que as seções seguintes, que versam sobre redes neurais, possam ser compreendidas. Por último, aborda-se o método de treinamento de redes neurais "*REINFORCE com baseline*" o qual será utilizado para o treinamento da Rede de Atenção em Grafos apresentada.

## 2.1 OTIMIZAÇÃO MATEMÁTICA

A Otimização Matemática é uma importante área dentro do campo da matemática aplicada, na qual o foco está em encontrar a solução mais favorável para determinado problema dentro de um conjunto de alternativas viáveis respeitando determinadas restrições. De maneira geral, busca-se formular problemas nos quais uma função objetivo, ou custo, deve ser maximizada, ou minimizada, enquanto uma serie de condições, formuladas como um sistema de equações e inequações, é respeitada, atuando como um limitação às soluções viáveis - que são aquelas que respeitam todas as restrições, ainda que não sejam aquelas que maximizem ou minimizem a função objetivo (João Pedro Pedroso e Muramatsu, 2012). Dessa forma, o problema deve ser modelado de forma matemática, permitindo, assim, um melhor entendimento das suas características e dos seus limites. A melhor solução, chamada de solução ótima, é aquela que maximiza ou minimiza a função objetivo dados os limites impostos pelas restrições. A otimização matemática é muito utilizada nos campos da engenharia, economia, logística, entre outros.

### 2.1.1 Programação Linear

A Programação Linear - Linear Programming (LP), em inglês -, é um subconjunto do campo de otimização matemática no qual os problemas apresentam funções objetivo e restrições lineares (Matousek e Gärtner, 2006). Dito de outra forma, na LP a relação entre as variáveis, função objetivo e restrições podem ser representadas por retas ou planos, a depender da dimensão da instância trabalhada. O termo programação, ao contrário do que aparenta, não advém da sua relação com ciência de computação, e sim das suas origens nos problemas de planejamento logístico militar da década de 1950. Em sua forma geral, um problema de LP pode ser descrito da seguinte forma:

$$\begin{aligned} \textbf{Maximize} \quad & c^T x \\ \textbf{sujeito a} \quad & Ax \leq b, \\ & x \in \mathbb{R}^n \end{aligned} \tag{2.1}$$

na qual $\mathbf{A}$ é uma matriz $m \times n$, $\mathbf{c} \in \mathbb{R}^n$ e $\mathbf{b} \in \mathbb{R}^n$ são vetores dados. A relação estabelecida por '$\leq$' implica uma comparação elemento a elemento dos vetores apresentados (Matousek e Gärtner, 2006).

Na LP, uma solução pode ser vista como um conjunto específico de valores atribuídos às variáveis de decisão que compõem o problema. As soluções são categorizadas de acordo com

a sua viabilidade e do comportamento da função objetivo em relação ao conjunto de restrições do problema. Os principais tipos de solução são as viáveis, as inviáveis, as ilimitadas e as ótimas. Uma solução viável satisfaz todas as restrições da instância em questão, encontrando-se, portanto, dentro do espaço de viabilidade estabelecido por essas restrições - também chamado de espaço de soluções. Por outro lado, as soluções inviáveis falham em cumprir com todas as restrições, não apresentando, por conseguinte, intersecção com o espaço de soluções. Uma solução ilimitada ocorre quando a função objetivo pode assumir valores infinitamente grandes, sejam eles positivos ou negativos, fazendo com que não haja um limite para a otimização. Por último, a solução ótima é aquela que, além de viável, representa o melhor valor possível a ser alcançado pela função objetivo na instância em questão.

### 2.1.2 Programação Linear Inteira

A programação linear inteira- Integer Linear Programming (ILP), em inglês -, é um tipo de otimização na qual a função objetivo e as restrições são lineares, mas as variáveis de decisão são restritas a valores inteiros, diferentemente da programação linear, na qual as variáveis podem assumir qualquer valor dentro de um período contínuo. A necessidade de se utilizar ILP surge em aplicações práticas em que determinadas instâncias de problemas, como na alocação de recursos, não admitem que suas variáveis de decisão assumam valores fracionários (por exemplo, não faz sentido alocar "meio" trabalhador em uma linha de produção) (Matousek e Gärtner, 2006). Pode acontecer, também, do problema exigir que algumas variáveis sejam inteiras enquanto outras podem assumir valores reais. Nesse caso, trata-se de um problema de Programação Linear Inteira Mista (MILP, na sigla em inglês), a ser resolvida, frequentemente, pelas mesmas técnicas empregadas na ILP.

$$\begin{aligned} \textbf{Maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \textbf{sujeito a} \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbb{Z}^n \end{aligned}$$

A principal diferença entre LP e ILP surge da natureza das soluções que essas podem apresentar. Na LP, o espaço de soluções pode ser visto como um objeto multidimensional convexo, como um poliedro, que admite valores fracionários para as suas variáveis de decisão. Já na ILP, o espaço de soluções consiste de pontos discretos correspondentes aos valores aos quais as variáveis de decisão podem assumir, fazendo com que o espaço de soluções viáveis seja não-convexo. Tal diferença faz com que instâncias de problemas de programação inteira sejam muito mais complexos que instâncias lineares, demandando mais capacidade computacional para a sua resolução.

## 2.2 O PROBLEMA DE ROTEAMENTO DE VEÍCULOS

Para definir o problema de roteamento de veículos - Vehicle Routing Problem (VRP), em inglês -, é necessário, primeiramente, definir o problema do caixeiro viajante - traveling salesman problem (TSP). Os dois são problemas fundamentais no campo da otimização combinatória, com profundas implicações reais nos campos da logística, cadeias de suprimento, planejamento, entre outros. Embora ambos os problemas busquem encontrar uma rota ótima, o TSP, em sua formulação tradicional, busca apenas encontrar o menor caminho possível passando uma única vez pelos elementos de um conjunto de localidades e retornando ao seu ponto de origem. O

VRP, por outro lado, apresenta um conjunto de veículos que saem de um depósito para servir à demanda de um conjunto de consumidores e então retornar. Dessa forma, pode-se observar que o VRP é uma extensão do TSP, adicionando complexidades, na forma de restrições, ao problema geral.

Apresentado pela primeira vez, de maneira formal, em 1959 por George Dantzig e John Ramser em seu seminal artigo "The Truck Dispatching Problem"(Dantzig e Ramser, 1959), o VRP, ao longo das décadas seguintes, foi formulado das mais diversas maneiras, a depender do conjunto de restrições que ele incorpora. Uma das suas variantes mais conhecidas é a do roteamento de veículos capacitado (CVPR, na sigla em inglês). Nessa sua forma, o problema adiciona uma restrição referente à capacidade de carga do veículo utilizado para atender à demanda dos consumidores, não podendo esta ser superior àquela no planejamento da rota. O objetivo continua o mesmo, seja ele o de minimizar o tempo, distância ou custo da rota, mas a complexidade do problema aumenta devido à nova restrição.

De acordo com Toth et al. (2014), de maneira geral, o problema de roteamento de veículos pode ser descrito da seguinte forma: dado um conjunto de requisições de transporte e uma frota de veículos, o problema consiste em elaborar um planejamento para que todas as requisições sejam atendidas de maneira a minimizar o custo - que pode ser visto, por exemplo, como a distância percorrida, o tempo despendido ou a energia consumida. No fundo, busca-se determinar qual dos veículos suprirá qual requisição de modo que todas as rotas empreendidas pelos veículos sejam viáveis. O problema tratado ao longo deste trabalho será o do roteamento de veículos na sua forma CAPACITADA (CVRP).

O CVRP busca otimizar a distribuição de mercadorias a partir de um único depósito, geralmente definido como o vértice 0, a um conjunto de consumidores representados pelo conjunto $\mathbf{N} = \{1, 2, ..., n\}$. Cada consumidor $i \in \mathbf{N}$ possui uma demanda dada por $q_i > 0$, que expressa a quantidade de mercadoria a ser entregue. A frota de veículos, dada por $\mathbf{M} = \{1, 2, ..., |M|\}$, é homogênea, indicando que todos os veículos possuem os mesmos custos e a mesma capacidade de carga, dada por $\mathbf{Q}_m > 0$. Cada veículo deverá suprir um subconjunto de consumidores $\mathbf{S} \subseteq \mathbf{N}$, partindo do depósito e a este retornando ao fim da rota.

Matematicamente, o CVRP pode ser apresentado como uma estrutura em grafo. Seja $G = (V, A)$ um grafo direcionado no qual $V$ é o conjunto dos vértices dado pela união entre o conjunto dos consumidores, $\mathbf{N} = \{1, 2, ..., n\}$ e o depósito: $V = \{N\} \cup \{0\}$. O conjunto $A$ contém os arcos $(i, j)$, sendo $i, j \in V$, os quais representam os possíveis caminhos entre os vértices e possuem um custo associado $c_{ij}$. A demanda do consumidor $i$ é dada por $q_i$ - depósito possui demanda igual a zero ($q_0 = 0$). A variável de decisão do problema é apresentada como $x_{ij}$, uma variável binária responsável por indicar se um determinado arco $(i, j)$ foi utilizado na solução do problema:

$$x_{ij} = \begin{cases} 1, & \text{se o arco } (i, j) \text{ for utilizado na rota;} \\ 0, & \text{caso contrário.} \end{cases}$$

O CVRP pode ser genericamente representado da seguinte forma (Borcinova, 2017):

$$\textbf{Minimize} \quad \sum_{m=1}^{M} \sum_{i=0}^{n} \sum_{\substack{j=0 \\ i \neq j}}^{n} c_{ij} \cdot x_{ijm} \tag{2.2}$$

$$\textbf{Subject to} \quad \sum_{m=1}^{M} \sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ijm} = 1, \qquad \forall j \in \mathbf{V} \tag{2.3}$$

$$\sum_{j=1}^{n} x_{0jm} = 1, \qquad \forall m \in \mathbf{M} \tag{2.4}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ijm} = \sum_{i=0}^{n} x_{jim}, \qquad \forall j \in \mathbf{V}, \forall m \in \mathbf{M} \tag{2.5}$$

$$\sum_{i=0}^{n} \sum_{\substack{j=1 \\ i \neq j}}^{n} q_j \cdot x_{ijm} \leq Q, \qquad \forall m \in \mathbf{M} \tag{2.6}$$

$$\sum_{m=1}^{M} \sum_{i \in S} \sum_{\substack{j \in S \\ i \neq j}} x_{ijm} \leq |S| - 1, \qquad \forall S \subseteq \mathbf{N} \tag{2.7}$$

$$x_{ijm} \in \{0, 1\}, \qquad \forall m \in \mathbf{M}, \forall i, j \in \mathbf{V}, i \neq j \tag{2.8}$$

No modelo acima, (2.2) representa a função objetivo: minimizar o custo da rota - novamente, esse custo pode ser dado pela distância percorrida, energia consumida, tempo dispendido, entre outros. A restrição (2.3) garante que cada consumidor será visitado por apenas um veículo, enquanto que a restrição (2.4) assegura que os veículos deixem o depósito uma única vez. A restrição dada em (2.5) é a restrição da conservação de fluxo: o número de veículos que chegam a determinado consumidor ou ao depósito deve ser igual ao número de veículos que saem de um consumidor ou do depósito. A restrição de controle da capacidade é apresentada em (2.6), garantindo que a demanda total dos consumidores visitados em uma determinada rota deverá ser inferior à capacidade de carga do veículo utilizado. A restrição (2.7) certifica que não haverá rotas não conectadas ao depósito, eliminando a possibilidade de subciclos. Em (2.8) teremos que a variável de decisão será binária (Borcinova, 2017).

Nas suas formulações mais simples, o VRP pode ser facilmente solucionado por resolvedores lineares, como o **SCIP**, primeiramente por sua natureza linear e, também, por apresentar instâncias pequenas, resultando em um espaço de soluções reduzido. Dessa forma, ele pode ser implementado por restrições lineares que culminam em uma função objetivo também linear. Todavia, a viabilidade dessa abordagem diminui ao considerarmos vertentes mais complexas do problema, como o CVRP, ou instâncias maiores, com centenas de vértices, as quais sofrem do fenômeno da explosão combinatória, no qual o número de possíveis rotas aumenta exponencialmente em virtude do aumento do número de vértices, fazendo com que o tempo de execução possa tornar-se inviável.

Consequentemente, instâncias mais complexas e maiores necessitam de métodos de otimização que possam varrer o espaço de busca de modo mais eficiente. Nesse caso, heurísticas e metaheurísticas são boas alternativas por oferecerem aproximações que podem rapidamente convergir para soluções parcialmente ótimas.

## 2.3 MÉTODOS DE OTIMIZAÇÃO

A depender da complexidade do problema enfrentado - nesse caso, o CVRP -, alguns métodos de otimização, como aqueles apresentados na figura 2.1, podem ser utilizados. Talbi (2009) os divide em dois grandes grupos: métodos exatos e métodos de aproximação. Dentro deste último grupo, os mais importantes são os algoritmos heurísticos e as metaheurísticas. Os métodos exatos garantem que as soluções encontradas sejam ótimas, mas atuam em tempo não-polinomial, ou seja, grandes instâncias podem ser intratáveis. As heurísticas e metaheurísticas buscam encontrar boas soluções dentro de um tempo razoável, mas não garantem que a solução ótima seja encontrada.

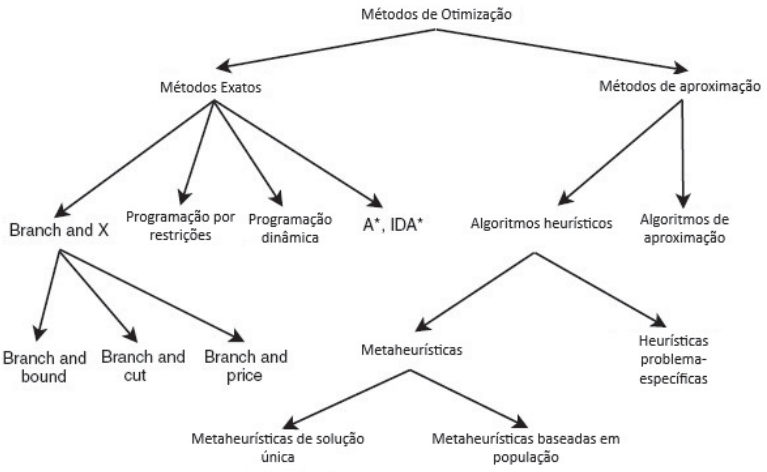



Figura 2.1: Métodos de otimização. Fonte: Talbi (2009)

### 2.3.1 Métodos Exatos

Os métodos exatos são o caminho fundacional para a resolução de problemas de otimização. Dentre eles, pode-se citar os algoritmos de *branch-and-bound* e *branch-and-cut*, a programação dinâmica, a programação por restrições, entre outros. O que caracteriza esse método é a sua atuação de forma a exaustivamente explorar o espaço de soluções viáveis para encontrar a solução ótima (Caceres Cruz et al., 2014). O método Simplex, criado por George Dantzig em 1947, é o caso mais emblemático da utilização de um método exato para a solução de programas lineares. Superficialmente, o seu funcionamento é facilmente explicado: a partir de uma solução inicial, o algoritmo visa navegar entre os vértices do espaço de soluções viáveis, iterativamente quantificando o resultado da função objetivo. A cada passo do algoritmo, também chamado de pivoteamento, o algoritmo garante a manutenção ou melhora da solução encontrada, até que o ótimo seja alcançado. Já a programação dinâmica, por exemplo, busca dividir o problema em subproblemas menores, buscando implementar medidas que reduzam o espaço de busca, eliminando áreas que não podem conter a solução ótima, para melhorar a eficiência computacional (Talbi, 2009).

Apesar de garantir que uma solução ótima possa ser encontrada, os métodos exatos sofrem com a natureza exponencial dos problemas combinatórios - como o VRP. Dada uma certa instância de um problema de roteamento, o espaço das soluções potenciais é dado por uma função exponencial daquela, ou seja, a medida que a instância cresce (a quantidade de consumidores, por exemplo), o espaço de soluções viáveis cresce exponencialmente, o que pode gerar a chamada "explosão combinatória". A aplicação de métodos que buscam encontrar a

solução ótima procurando-a exaustivamente dentro do espaço de soluções pode ser inviável devido aos recursos computacionais necessários para encontrá-la em um tempo razoável.

### 2.3.2 Heurísticas

As heurísticas são uma classe de algoritmos elaborados para encontrar boas soluções para problemas complexos em um tempo razoável. Em sua maioria, são utilizadas para lidar com grandes instâncias de problemas nos quais os métodos exatos são muito lentos ou computacionalmente caros ou mesmo inviáveis para encontrar uma solução. Apesar de não garantir uma solução ótima, as soluções encontradas por uma heurística podem ser consideradas aceitáveis, aproximadamente ótimas.

Em geral, para elaborar uma heurística para um determinado problema de otimização é necessário um conhecimento especializado sobre este. Esse conhecimento especializado permite a implementação de heurísticas capazes de entender padrões específicos do problema, gerando algoritmos mais eficientes e rápidos. Entretanto, essa dependência de conhecimento especializado é uma de suas maiores limitações, pois, muitas vezes, pequenas mudanças nas características da instância em questão podem comprometer o funcionamento ideal do algoritmo. O conhecimento específico, em suma, impede que as heurísticas sejam generalizadas, restringindo o seu foco de atuação (Talbi, 2009).

De acordo com Sabet e Farooq (2022), as heurísticas voltadas à solução do VRP podem ser divididas em dois grandes grupos: construtivas e de melhoria. Heurísticas construtivas buscam prover uma solução inicial construindo rotas de maneira serial ou paralela. As heurísticas de melhoria realizam uma busca iterativa no espaço de soluções na "vizinhança" da solução inicial, buscando encontrar alguma outra solução que seja superior à anterior. A busca termina quando não se realizam novas melhorias na solução objeto do processo, esta é chamada, então, de ótimo local.

### 2.3.3 Metaheurísticas

De forma simplificada, as metaheurísticas (MH) podem ser vistas como uma estrutura que provê regras e direcionamentos, de maneira independente relativamente ao problema enfrentado, para o desenvolvimento de algoritmos heurísticos de otimização. Assim como as heurísticas, as MH buscam encontrar soluções aceitáveis dentro de um tempo razoável. A falta de exatidão é compensada por sua maior flexibilidade e adaptabilidade. A construção de uma MH depende do equilibrio entre duas estratégias: a exploração do espaço de busca e o refinamento das soluções encontradas. A exploração visa investigar regiões do espaço de soluções para evitar permanecer em um ótimo local. Já o refinamento busca observar a vizinhança da solução encontrada com o intuito de encontrar alguma solução melhor (Talbi, 2009).

As MH podem ser classificadas de diversas maneiras, a depender das características em questão. Duas das principais classificações dizem respeito ao foco de busca da MH, se voltado a uma única solução ou a uma população delas, e à sua inspiração, se baseada em processos naturais ou não. As metaheurísticas de busca de solução única visam manipular e transformar a solução encontrada para melhorá-la, um processo de refinamento. Por outro lado, as MH de busca baseada em população utilizam amostras de soluções, aumentando o espaço de busca, em um processo de exploração. Quando inspirada em processos naturais, a MH pode ser baseada em processos biológicos ou evolutivos, como uma MH que busca soluções simulando o comportamento de uma colônia de abelhas (Talbi, 2009).

De maneira geral, as MH podem ser definidas como um processo de criação de uma heurística (visto como um conjunto de normas, boas práticas) para guiar outra heurística, de

forma a combinar estratégias e garantir tanto uma melhor eficiência na busca quando uma maior generalização entre problemas. Essa definição alinha-se aos paradigmas modernos de aprendizado de máquina, que também buscam padrões e soluções ótimas dentro de grandes conjuntos de dados. As aplicações de aprendizado de máquina na elaboração de algoritmos de otimização pode ser vista como um processo metaheurístico. Por conseguinte, um algoritmo "aprendido" por uma rede neural que vise solucionar problemas de otimização ou combinatórios pode ser classificado como uma metaheurística.

## 2.4 APRENDIZADO DE MÁQUINA

Segundo Russell e Norvig (2021) , quando um agente experiencia eventos acerca do mundo e essa experiência o leva a um ganho de desempenho, há aprendizado. Caso o agente em questão seja um computador, diz-se que há aprendizado de máquina. No aprendizado de máquina, um computador é alimentado com dados e os usa para construir um modelo que se adeque ao contexto dos dados. O modelo será usado, então, para fazer inferências sobre a realidade objetiva e para resolver problemas.

Quando o modelo é utilizado para se fazer inferências acerca de um conjunto finito de valores – como, sim/não, spam/não-spam ou positivo/negativo/neutro, por exemplo -, o problema a ser resolvido é chamado de classificação. Caso o problema busque uma resposta quantitativa, seja ela na forma de um inteiro ou um número real, afirma-se que se trata de um problema de regressão.

Já de acordo com Domingos (2012), modelos de aprendizado de máquina são sistemas que aprendem determinadas representações do mundo real a partir de dados. O autor afirma que um dos problemas de maior interesse em aprendizado de máquina é o da classificação. Nesse contexto, o programa é elaborado para receber uma entrada de dados – discretos ou contínuos – referentes às características de um determinado objeto e produzir uma saída discreta referente à classe à qual esse objeto pertence, ou seja, o modelo o classifica em um determinado rótulo.

Um sistema de aprendizado é basicamente composto por três grandes componentes: (i) representação, (ii) avaliação e (iii) otimização. A (i) representação diz respeito à escolha da linguagem formal a ser utilizada. A seleção é de suma importância para o modelo, pois ela define aquilo que é chamado de espaço de hipóteses – o conjunto de possíveis soluções (funções) que um modelo pode considerar para resolver determinado problema - do sistema. Se um classificador não se encontra no espaço definido, o modelo não poderá aprendê-lo; já a (ii) avaliação determina a função que será utilizada para separar os classificadores com base em sua qualidade. Por último, a (iii) otimização busca estabelecer um método para selecionar os melhores classificadores. É um conceito chave que influencia na eficiência do modelo e nas características do classificador produzido (Domingos, 2012).

Em seu livro Deep Learning (2016), Goodfellow afirma que aprendizado de máquina é uma forma de estatística aplicada com uso intensivo de computadores para estimar funções complexas, as quais serão utilizadas para fazer predições sobre determinado conjunto de dados. Segundo ele, a maior parte dos problemas que envolvem aprendizado de máquina pode ser dividida em aprendizado supervisionado e aprendizado não supervisionado, a depender do tipo de experiência que eles podem ter no processo de aprendizado (Goodfellow et al., 2016).

No aprendizado supervisionado, busca-se, com base em um conjunto de treino estruturado na forma de pares entrada-saída - $(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$ -, descobrir uma função $h$ que se aproxime de uma função $f$, dita verdadeira, responsável por gerar os pares de treino. A função $h$ é uma hipótese sobre o contexto dos dados selecionada a partir de um espaço de hipóteses que contém todas as funções possíveis. Ou seja, o algoritmo de aprendizado supervisionado

experiencia um conjunto de dados de treinamento, contendo exemplos já rotulados. A função $h$ é, portanto, um modelo aprendido a partir dos dados, pois é a representação final aprendida pelo algoritmo a partir do conjunto de treinamento.

Em último caso, o modelo será tão bom quanto a hipótese escolhida. A questão se torna como escolher uma boa hipótese. Em Russell e Norvig (2021) (tradução nossa):

> "(...) Nós podemos esperar por uma hipótese consistente: uma função $h$ para a qual cada $x_i$ presente no conjunto de treinamento tenha $h(x_i) = y_i$. Com saídas de dados na forma contínua, não se pode esperar uma relação exata com o real valor de $y_i$; desta forma, nós procuramos pela função que produz, para cada $h(x_i)$, o valor mais próximo à $y_i$.[1]"

No fim, a avaliação do modelo não será feita sobre os dados de treino, e sim sobre dados que o modelo ainda não experienciou, chamado conjunto de teste. Visa-se avaliar o quão bem a função $h$ generaliza sobre dados inéditos. Segundo Domingos (2012), generalizar a função para além do conjunto de treino é o objetivo fundamental de qualquer sistema de aprendizado de máquina, pois é extremamente improvável que os mesmos dados utilizados para treinar o modelo sejam encontrados posteriormente, independentemente do tamanho do conjunto de dados disponível.

Já no aprendizado não-supervisionado, de acordo com Goodfellow (2016), os algoritmos de aprendizado de máquina buscam aprender as características estruturais de um dataset ao experienciar os seus exemplos. Geralmente, busca-se encontrar, explícita ou implicitamente, a distribuição de probabilidades que gerou o conjunto de dados. Em outras palavras, aprendizado não-supervisionado envolve experienciar dados de um vetor aleatório e, implícita ou explicitamente, descobrir a sua distribuição de probabilidades ou alguma outra característica interessante.

Murphy (2012) afirma que no aprendizado não supervisionado não se sabe de antemão qual a saída de dados esperada para cada entrada de dados, transformando a tarefa em questão em um problema de estimar a função de densidade dos dados, ou seja, a função que descreve a distribuição de probabilidade de uma variável aleatória contínua. Enquanto que no aprendizado supervisionado busca-se estimar uma função de densidade condicional, no aprendizado não-supervisionado, a função que se pretende estimar é incondicional.

No entanto, além desses dois paradigmas, existe também o aprendizado por reforço. Nesse tipo de aprendizado, o agente aprende a tomar decisões ao interagir com o ambiente no qual se insere buscando maximizar um sinal chamado de recompensa. O aprendizado por reforço será apresentado com maior profundidade na seção 2.6, na qual abordaremos os seus principais conceitos, como a política do agente e a estimação de gradientes.

### 2.4.1 Avaliação de desempenho

Em um primeiro momento, um modelo de aprendizado de máquina aprende sobre os dados de treino, buscando gerar uma função que se adéque aos dados observados, de forma a apresentar um baixo erro de treino. Essa situação representa, tão somente, um problema de otimização. Entretanto, um modelo de aprendizado de máquina é elaborado, de maneira geral, para maximizar a sua capacidade de generalização. Por generalização, entende-se a capacidade do modelo em estimar ou classificar corretamente dados ainda não observados.

[1] "(. . . ) We could hope for a consistent hypothesis: an $h$ such that each $x_i$ in the training set has $h(xi) = yi$. With continuous-valued outputs we can't expect an exact match to the ground truth; instead we look for a best-fit function for which each $h(x_i)$ is close to $y_i$."

Ao se avaliar um modelo de aprendizado de máquina, deve-se levar em consideração dois fatores: primeiro, se o modelo reduz o erro de treino a níveis suficientemente baixos; segundo, se a diferença entre o erro de treino e o erro de generalização, também chamado de erro de teste, é considerada pequena para o contexto. De acordo com Goodfellow (2016), da consideração desses dois fatores surgem dois conceitos de suma importância para a avaliação dos modelos de aprendizado, *underfitting* e *overfitting*.

Russel e Norvig (2021) definem *overfitting* como a situação na qual o modelo acaba por se sobreajustar ao conjunto de dados de treino, o que o leva a ter um baixo desempenho ao experienciar novos dados. Uma das explicações para este fenômeno é a de que o modelo acaba por capturar, além dos padrões encontrados na distribuição, a variação decorrente do ruído presente nos dados de entrada (Murphy, 2013). Por outro lado, um modelo é considerado *underfitted* quando falha em encontrar padrões no conjunto de dados experienciados. Tal fato pode ser observado quando o modelo não consegue reduzir o erro de treino a níveis aceitáveis.

Por sua vez, para que um algoritmo de aprendizado de máquina seja avaliado quanto à magnitude do erro de treino e do erro de teste, é necessário que alguma medida quantitativa de desempenho seja estabelecida. Usualmente, essa medida de desempenho está intimamente ligada à tarefa à qual o modelo de aprendizado se propõe a realizar. No caso de um sistema de classificação, pode-se usar a acurácia ou a taxa de erros. Na acurácia, mede-se a proporção de exemplos para os quais o algoritmo atribuiu a correta classificação (correta positiva e correta negativa); já a taxa de erros seria basicamente o inverso: a proporção de exemplos que o algoritmo incorretamente classifica.

Quando um classificador avalia um exemplo como pertencente a determinada classe, esse exemplo possui classificação positiva para essa classe e negativa para as demais. Caso essa avaliação tenha sido feita de forma correta, será uma classificação verdadeira. Do contrário, será uma classificação falsa. Dessas premissas surgem os seguintes conceitos de classificação: verdadeira-positiva, verdadeira-negativa, falsa-positiva e falsa-negativa. Utilizando-se dessas definições, pode-se calcular, por exemplo, a métrica $F1$ (Han et al., 2012).

## 2.5 REDES NEURAIS DE GRAFOS

Aprendizado profundo é o nome dado a um conjunto de técnicas de aprendizado de máquina que apresentam modelos que possuem várias camadas de processamento entre a entrada de dados e a respectiva saída, ou seja, as entradas de dados passam por vários passos de processamento antes de serem apresentadas como saídas de dados. A ideia por trás do conceito é a de aumentar o caminho do processamento de dados, de modo que estes possam interagir entre si para que padrões mais complexos possam emergir. Nesse caso, ao contrário dos métodos lineares, os dados deixam de ser independentes em relação à saída de dados (Russell e Norvig, 2021).

As redes neurais são uma implementação do aprendizado profundo com o intuito de aumentar o número de etapas entre a entrada de dados e a saída destes, buscando superar as limitações de técnicas menos complexas, como a regressão logística, que produzem apenas funções lineares. O termo rede neural advém da tentativa de modelar os circuitos cerebrais por meio de modelos computacionais. Dessa forma, as redes neurais possibilitam aos algoritmos de aprendizado profundo obter representações hierárquicas dos dados, por exemplo, as primeiras camadas podem aprender características mais simples, como bordas em imagens, enquanto que as camadas posteriores podem aprender padrões mais complexos, como formas ou objetos, possibilitando que aqueles possam aprender automaticamente importantes características do

conjunto de dados sem a necessidade de intervenção humana para pré-processá-los (LeCun et al., 2015).

As redes neurais utilizadas no contexto de aprendizado profundo buscam representar informações como vetores ou matrizes, porém, muitas vezes, a informação utilizada apresenta uma relação topológica entre os seus elementos, sendo melhor representada por um grafo. O problema em se processar dados estruturados em grafos é como transformá-los em vetores. Uma das alternativas seria a de "achatar" o grafo para representá-lo como um vetor. Entretanto, essa estratégia pode ocasionar perda de informação, principalmente aquela referente aos relacionamentos entre os elementos.

Para superar esse obstáculo, Gori et al. (2005) introduziu o conceito de Redes Neurais de Grafos (Graph Neural Networks - GNN). As GNN podem ser vistas como uma extensão da aplicação de Redes Neurais Recorrentes (RNN, em inglês), utilizando dados estruturados como grafos na entrada de dados. O aspecto fundamental de uma GNN é o de encapsular objetos ou elementos como os vértices de um grafo e os relacionamentos entre esses objetos como arestas.

Dado um grafo $G(V, E)$, no qual $V$ representa o conjunto de vértices e $E$ o conjunto de arestas que conectam esses vértices, cada vértice $v \in V$ possui um conjunto de outros vértices a ele conectado por arestas dado por ne$[v]$ - chamados de vizinhos de $v$. Cada vértice $v$ possui também um rótulo $l_v \in \mathbb{R}^q$, que representa os seus atributos, e um vetor de estado $\boldsymbol{x_v} \in \mathbb{R}^s$ que condensa a representação desse elemento. O estado do vértice $v$ é determinado pelo seu rótulo, $l_v$ e pelos estados e rótulos dos vértices vizinhos, uma dependência modelada por uma função $f_w$, tal que:

$$\boldsymbol{x_v} = f_w(l_v; x_{\text{ne}[v]}; l_{\text{ne}[v]}), \quad \forall v \in V \tag{2.9}$$

Além do vetor de estado, cada vértice possui um vetor de saída $\boldsymbol{o_v} \in \mathbb{R}^m$ que é uma função do seu estado $\boldsymbol{x_v}$ e do seu rótulo $\boldsymbol{l_v}$ modelada por $g_w$:

$$\boldsymbol{o_v} = g_w(x_v; l_v), \quad \forall v \in V \tag{2.10}$$

### 2.5.1 Redes de Atenção em Grafos

As redes de atenção em grafos (Graph Attention Networks - GAT), são uma arquitetura de redes neurais que processam dados estruturados em grafos utilizando-se do mecanismo de atenção múltipla proposto por Vaswani et al. (2023). A essência de uma GAT encontra-se em uma camada de atenção em grafos que atua nos atributos dos vértices, $\mathbf{h} = \{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_V\}$, onde $V$ é o conjunto de vértices e cada $\mathbf{h}_i \in \mathbb{R}^F$ representa os atributos do vértice $i$. Essa camada é responsável por transformar os atributos dos vértices em um novo conjunto de atributos $\mathbf{h}' = \{\mathbf{h}'_1, \mathbf{h}'_2, ..., \mathbf{h}'_V\}$, possivelmente em uma dimensionalidade diferente $F'$.

Inicialmente, aplica-se uma transformação linear, compartilhada entre os vértices, parametrizada por uma matriz de pesos $W \in \mathbb{R}^{F' \times F}$, para iniciar o processo de transformação dos dados (Velickovic et al., 2018).

O núcleo da camada GAT é o mecanismo de *self-attention* - um mecanismo de atenção compartilhado entre os elementos, $a : \mathbb{R}^{F'} \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$, utilizado para calcular os coeficientes de atenção:

$$e_{ij} = a(W\mathbf{h}_i, W\mathbf{h}_j) \tag{2.11}$$

Os coeficientes de atenção medem a importância relativa dos atributos de um vértice $j$ para um vértice $i$. Como a informação de entrada é estruturada em forma de grafo, o cálculo da atenção é realizado levando em consideração apenas vértices $j$ que se encontrem na vizinhança $\mathcal{N}$ do vértice $i$ ($e_{ij} : j \in \mathcal{N}_i$), preservando, assim, a topologia do grafo de origem no aprendizado.

Para facilitar a comparação dos valores, o resultado do processo citado passa por uma camada de normalização por meio da função *softmax*, resultando em uma distribuição de probabilidades:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \tag{2.12}$$

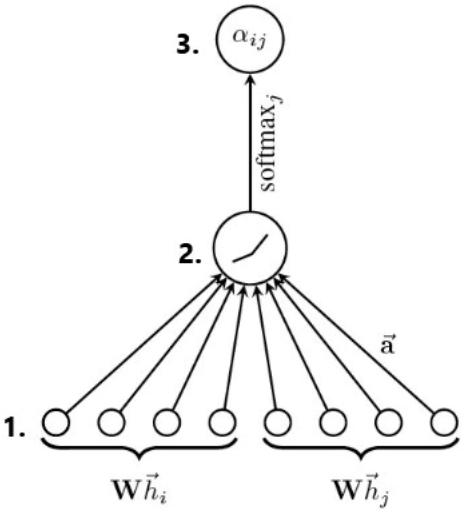



Figura 2.2: Cálculo do coeficiente de atenção: 1. representação vetorial dos atributos dos vértices; 2. aplicação de uma função de ativação não-linear após o mecanismo de atenção - dado por $\vec{a}$; e 3. coeficiente de atenção normalizado após a aplicação da função softmax à saída da função de ativação. Fonte: Velickovic et al. (2018).

A figura 2.2 apresenta de uma generalização do processo do cálculo do coeficiente de atenção. A normalização dos coeficientes de atenção atua de modo a facilitar o reconhecimento da importância relativa de determinado vértice, permitindo que o modelo adapte dinamicamente a influência de tal vértice no contexto da aprendizagem. Na sequência, os coeficientes de atenção são usados para realizar uma combinação linear dos atributos dos seus vértices, resultando na apresentação dos atributos de saída de cada vértice - a utilização de uma função não-linear, $\sigma$ na equação abaixo, permite que o modelo capte maiores nuances presentes nos dados:

$$\vec{h}'_i = \sigma\left(\sum_{j \in V_i} \alpha_{ij} W \vec{h}_j\right) \tag{2.13}$$

$$\sigma = \mathit{LeakyReLU} = \begin{cases} x, & \text{se } x \geq 0 \\ \alpha x, & \text{se } x < 0 \end{cases} \tag{2.14}$$

Inspirando-se em Vaswani et al. (2023), percebe-se que aplicar multiplas camadas de atenção (*multi-head attention*) melhora a capacidade de aprendizagem e a estabilidade de determinados modelos. Essa abordagem consiste em aplicar $K$ mecanismos de atenção independentes de forma paralela, cada um calculando os atributos de saída dos vértices. O resultado de cada mecanismo será então concatenado:

$$\vec{h}'_i = \sigma\left(\frac{1}{K}\sum_{k=1}^{K}\sum_{j \in \mathcal{N}_i} \alpha_{ij}^k W^k \vec{h}_j\right) \tag{2.15}$$

para as camadas intermediárias do modelo. Geralmente, para a camada final, opta-se por uma medida de agregação, como a média, ao invés da concatenação. Sobre os dados resultantes, aplica-se uma nova função não-linear. A aplicação de multiplos mecanismos de atenção aumenta a expressividade do modelo, pois permite que este tenha acesso aos dados por perspectivas diferentes (Velickovic et al., 2018).

### 2.5.2 Pointer Networks

As *Pointer Networks* (PN) representam uma abordagem dedicada a contornar o problema do dicionário de saída de uma rede neural, mais especificamente em problemas no qual a saída é composta de elementos do conjunto de entrada. Redes neurais usadas em processamento de linguagem natural possuem um dicionário de saída de tamanho fixo, composto pelas palavras do idioma, por exemplo, o que constitui um obstáculo para lidar com problemas de otimização combinatória, nos quais as saídas são permutações dos elementos do conjunto de entrada, como o VRP. Nesse caso, o tamanho do dicionário de saída é dependente do tamanho da entrada. Para contornar esse entrave, as PN utilizam-se do mecanismo de atenção para apontar para a posição ocupada pelo elemento de entrada, que será também o elemento de saída.

A principal inovação decorrente das PN encontra-se na sua abordagem do mecanismo de atenção. Modelos anteriores utilizavam o mecanismo de atenção para criar vetores de contexto - vetores que representam a informação apreendida pelo modelo. As PN, por outro lado, usam da atenção para apontar qual posição ocupa o elemento do conjunto de entrada que deverá ser a saída da função. Dessa forma, o modelo pode lidar com conjuntos de entrada de tamanhos diferentes e com conjuntos de saída que sejam dependentes destes tamanhos (Vinyals et al., 2017).

Matematicamente, o mecanismo de atenção nas Pointer Networks é definido pelas seguintes expressões:

$$u^i_j = v^T \tanh(W_1 e_j + W_2 d_j), \quad j \in \{1, \ldots, n\} \tag{2.16}$$

$$p(C_i|C_1, \ldots, C_{i-1}, P) = \text{softmax}(u^i) \tag{2.17}$$

Na primeira equação, $u_{i,j}$ é o coeficiente de atenção entre o $i$-ésimo termo do decoder e o $j$-ésimo termo do encoder; $v$ é um vetor com os pesos aprendidos e $W_1$ e $W_2$ são matrizes de pesos que projetam a dimensão dos elementos para um espaço comum. A função de ativação *tanh* retorna a tangente hiperbólica do resultado da operação, de modo a introduzir não-linearidade ao modelo para permitir que este capture relações complexas entre a entrada e a saída de dados.

Na segunda equação, $p(C_i|C_1, \ldots, C_{i-1}, P)$ representa a probabilidade condicional de selecionar o próximo elemento de saída $C_i$, dado o histórico dos elementos anteriormente selecionados $C_1, \ldots, C_{i-1}$ e a sequência completa de entrada $P$. A função $softmax(u_i)$ normaliza os coeficientes de atenção em uma distribuição de probabilidade sobre todos os elementos da entrada para indicar a probabilidade da seleção do próximo elemento na sequência de saída.

## 2.6 APRENDIZADO POR REFORÇO

O aprendizado por reforço (Reinforcement Learning - RL) é um paradigma de aprendizado de máquina no qual um agente aprende a tomar decisões ao realizar ações dentro de um ambiente no qual o objetivo é maximizar um sinal chamado recompensa. Em outras palavras, o agente aprende ao interagir com o seu ambiente - uma das ideias recorrentes em teorias sobre o aprendizado e inteligência. Para Sutton e Barto (2018), o RL é um sistema de aprendizado de

máquina diferente do aprendizado supervisionado ou não-supervisionado. Sistemas de RL são caracterizados pela sua natureza de laço fechado (*closed-loop nature*), no qual o agente realiza ações que alteram os dados de entrada, pela ausência de instruções diretas sobre qual ação tomar e pela presença de efeitos imediatos e futuros das ações nas recompensas auferidas.

Quatro elementos compõem a estrutura de RL: a política $\pi$, a recompensa $\mathcal{R}$, a função de valor $\mathcal{V}$ e, opcionalmente, um modelo. A política mapeia a percepção do ambiente às ações, ditando o comportamento do agente; a recompensa provê feedback imediato e sugere eventual modificação da política adotada; e a função de valor informa qual caminho de estados levará a um acúmulo maior de recompensas (Sutton e Barto, 2018). A cada período $t$, o agente, dado o o estado do ambiente, $s_t$, realiza uma ação, $a_t$, pertencente a um conjunto de ações possíveis, $\mathcal{A}$, e recebe uma recompensa, $r_t$, e um novo estado do ambiente $s_{t+1}$. O objetivo do agente é o de maximizar o acúmulo de recompensas, $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, onde $\gamma \in (0, 1]$ é um valor que busca equilibrar a importância de recompensas imediatas e futuras. Essa composição é representada por uma política ($\pi$) que visa maximizar o retorno auferido em cada estado (Mnih et al., 2016).

O problema do roteamento de veículos capacitado é um problema de otimização combinatória cujo objetivo é encontrar uma rota para os veículos que minimize uma determinada função objetivo. No contexto do aprendizado por reforço, isso envolve aprender uma política $\pi_\theta(a|s)$, parametrizada por $\theta$, que maximize a recompensa esperada. Os gradientes da política (ator) em questão, dados por (Sutton e Barto, 2018):

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[ \sum_{t=0}^{T} G_t \nabla_\theta \log \pi_\theta(A_t|S_t) \right] \tag{2.18}$$

atuam de modo a ajustar os parâmetros $\theta$ para maximizar a recompensa ao longo das iterações. $\nabla_\theta J(\theta)$ é o gradiente da função objetivo $J(\theta)$, dado pela expectativa sob a política $\pi$, $\mathbb{E}_\pi$, da recompensa acumulada, $G_t$ representa a recompensa obtida na iteração $t$, e $\log \pi_\theta(A_t|S_t)$ é a probabilidade logarítimica da ação $A_t$, dado o estado $S_t$, ser escolhida em $t$.

Como no CVRP uma determinada solução corresponde a um conjunto de vértices que formam uma determinada rota dentre um número finito de combinações possível, o seu espaço de solução é discreto. Dessa forma, utilizam-se, dentro do aprendizado por reforço, amostras (rotas) obtidas da distribuição de probabilidades dada pela política do ator para estimar o gradiente esperado. A política $\pi_\theta(a|s)$ do ator define a distribuição de probabilidade sobre as possíveis ações (rotas) e métodos Monte Carlo são usados para realizar a amostragem e obter as recompensas associadas. Um dos algoritmos mais tradicionais para estimar gradientes utilizando de métodos Monte Carlo é o algoritmo REINFORCE.

### 2.6.1 REINFORCE

O algoritmo REINFORCE, apresentado por Williams (1992), é um dos principais métodos para otimizar a política de um determinado agente no RL. Usando uma estratégia Monte Carlo, o REINFORCE otimiza a política do ator ao estimar o gradiente necessário para ajustar os parâmetros deste por meio da expectativa da recompensa calculada diretamente de amostras das decisões tomadas pelo agente. O objetivo é o de maximizar a recompensa acumulada ao ajustar os parâmetros $\theta$ por meio da técnica do gradiente ascendente.

De maneira geral, em aprendizado por reforço, busca-se otimizar a política $\pi_\theta(a|s)$ que mapeia os estados $s \in S$ às ações $a \in A$ de forma a maximizar a recompensa acumulada. Formalmente, o objetivo é o de maximizar a recompensa esperada $J(\theta) = \mathbb{E}_\pi[G_t]$, onde $G_t$ é a

recompensa auferida na iteração $t$. O gradiente do objetivo $J(\theta)$ em relação aos parâmetros da política (ator) $\theta$ é dado pelo *policy gradient theorem*:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[ \sum_{t=0}^{T} G_t \nabla_\theta \log \pi_\theta(A_t|S_t) \right] \tag{2.19}$$

no qual $G_t$ é a recompensa e $\nabla_\theta \log \pi_\theta(A_t|S_t)$ o gradiente da probabilidade logarítmica da ação escolhida na iteração $t$ (Sutton e Barto, 2018). De posse do gradiente acima calculado, o algoritmo REINFORCE atualiza os parâmetros da política da seguinte forma:

$$\theta_{t+1} = \theta_t + \alpha G_t \nabla_\theta \log \pi_\theta(A_t \mid S_t) \tag{2.20}$$

A expressão acima mostra que o REINFORCE atua de maneira a aumentar a probabilidade das ações que levem a recompensas mais altas ao direcionar os parâmetros $\theta$ na direção do gradiente. O gradiente é escalonado pela recompensa $G_t$ e pela constante $\alpha$ (que controla o tamanho do ajuste pretendido), indicando que as ações que resultarem em uma maior recompensa terão uma maior influência na atualização dos parâmetros.

### 2.6.2 REINFORCE com baseline

O algoritmo REINFORCE fornece um método direto para otimização da política do agente, entretanto, por utilizar o valor da recompensa $G_t$ no calculo da estimativa do gradiente, ele apresenta grande variância nas suas atualizações. Para reduzir essa variância, uma das técnicas mais adotadas é a de utilizar uma *baseline* que não afete o valor do gradiente estimado, mas reduza a magnitude da variância. De acordo com Sutton e Barto (2018), a *baseline* pode ser uma função qualquer, até mesmo uma variável aleatória, caso aquela não varie em função da ação tomada. Uma generalização do *policy gradient theorem* com o uso de uma baseline $b(S_t)$ pode ser elaborada da seguinte forma:

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi \left[ \sum_{t=0}^{T} (G_t - b(S_t)) \nabla_\theta \log \pi_\theta(A_t \mid S_t) \right] \tag{2.21}$$

e a função de atualização dos parâmetros:

$$\theta_{t+1} = \theta_t + \alpha (G_t - b(S_t)) \nabla_\theta \log \pi_\theta(A_t \mid S_t) \tag{2.22}$$

A introdução de uma *baseline* atua de modo a diminuir a variância sem introduzir viés, proporcionando um aprendizado mais estável, principalmente em problemas como o roteamento de veículos, no qual a variação na distância total percorrida entre as rotas pode ser significativa.

A figura 2.3 apresenta um exemplo do fluxo de atuação do algoritmo REINFORCE com baseline no contexto do aprendizado por reforço usando uma rede de atenção em grafos para escolher as ações a cada iteração. Uma instância do problema dará entrada na rede neural que possui a tarefa de construir uma rota ao selecionar os vértices que a compõem por amostragem. De posse da rota selecionada, a sua probabilidade de ocorrência e respectiva recompensa podem ser calculadas e então utilizadas para atualizar os parâmetros da rede neural.

$$\nabla \mathcal{L}(\theta \mid S_t) = \mathbb{E}_{p_\theta(\hat{\pi}|S_t)} \left[ (D(\hat{\pi}_t) - b(S_t)) \nabla \log p_\theta(\hat{\pi}_t \mid S_t) \right]$$

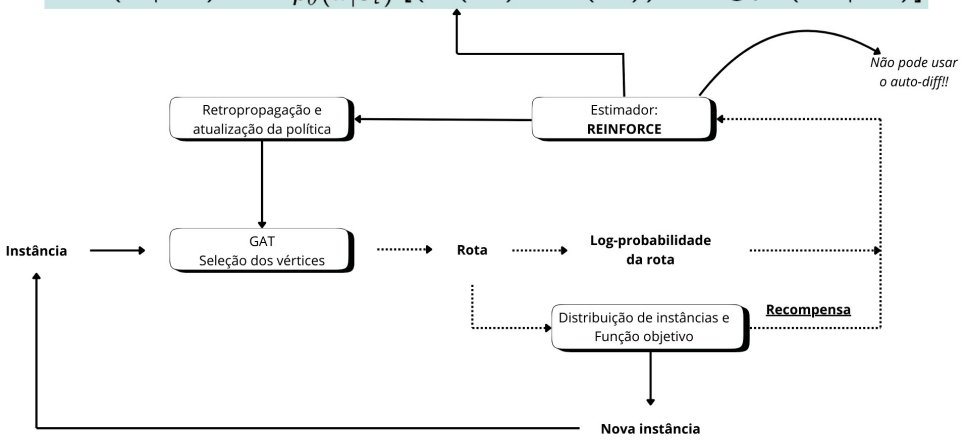



Figura 2.3: Fluxo de atuação do algoritmo REINFORCE com baseline: $\hat{\pi}_t$ é a rota construída por amostragem; $D(\hat{\pi}_t)$ é a recompensa obtida com a rota; $b(S_t)$ é a baseline utilizada para reduzir a variância; e $\nabla \log p_\theta(\hat{\pi}_t \mid S_t)$ corresponde ao gradiente da probabilidade de ocorrência da rota selecionada.

Uma outra limitação do algoritmo REINFORCE é a impossibilidade de se utilizar o paradigma da auto-diferenciação no seu processo de treinamento. O paradigma da auto-diferenciação é muito utilizado em bibliotecas de aprendizado profundo (como a pytorch e a tensorflow) para calcular os gradientes para a retro-propagação automaticamente por meio da diferenciação de funções definidas programaticamente (a função objetivo, por exemplo) (Harrison, 2021). Após realizar a amostragem da rota, o REINFORCE acaba por tratá-la como fixa, quebrando a dependência desta com a função objetivo, impedindo que os gradientes sejam retro-propagados.

### 2.6.3 DiCE

O estimador diferenciável Monte Carlo (DiCE) (Foerster et al., 2018) apresenta uma outra abordagem para estimar os gradientes utilizando-se do conceito de *Stochastic Computation Graphs* (SCG) (Schulman et al., 2016), uma ferramenta utilizada para modelar processos de decisão que envolvem operações determinísticas e estocásticas em contextos de aprendizado por reforço.

O algoritmo REINFORCE calcula os gradientes por meio de uma amostra das rotas possíveis e do valor das probabilidades das ações, dadas por $\nabla_\theta \log \pi_\theta(A_t \mid S_t)$. O método DiCE, por outro lado, busca prover um estimador não viesado que seja infinitamente diferenciável. Essa propriedade garante que os gradientes possam ser propagados pela rede neural eficientemente, ou seja, com menos ruído, variância.

Para tanto, Foerster et al. (2018) introduz o operador matemático MAGICBOX $\boxdot$. Esse operador apresenta duas importantes propriedades:

$$1. \quad \boxdot(\mathcal{W}) \rightarrow 1, \tag{2.23}$$

$$2. \quad \nabla_\theta \boxdot(\mathcal{W}) = \boxdot(\mathcal{W}) \sum_{w \in \mathcal{W}} \nabla_\theta \log p(w; \theta). \tag{2.24}$$

sendo W um conjunto de vértices estocásticos no SCG que dependem de $\theta$, a política que está sendo otimizada, e influenciam a função objetivo; $w$ um vértice estocástico individualmente

considerado; e $p(w;\theta)$ a probabilidade, parametrizada por $\theta$, de ocorrência do vértice estocástico $w$. A expressão 2.23 indica que, durante o *forward pass* (representado pela seta $\rightarrow$), $\square$(W) será avaliado para o valor 1. Já a expressão 2.24 mostra que o gradiente de $\square$(W) em relação aos parâmetros $\theta$ é dado por $\square$(W) multiplicado pela soma sobre todos os vértices estocásticos $w \in \mathcal{W}$ de $\nabla_\theta \log p(w;\theta)$. Essa propriedade garante que, durante a retro-propagação, o gradiente contenha a informação de como os parâmetros $\theta$ influenciam a probabilidade dos vértices estocásticos, possibilitanto que o DiCE construa uma função objetivo que, sendo diferenciável, produza o gradiente correto à retro-propagação.

O operador garante que as dependências entre a amostra de ações (rotas) e a função objetivo sejam preservadas. Assegura-se, dessa forma, que os gradientes possam ser retro propagados corretamente. Tendo em vista que o problema do CVRP possui um espaço de solução discreto, a capacidade do DiCE de gerenciar as dependências entre os vértices da rede neural leva a um aumento da estabilidade do treinamento realizado por meio de atualizações dos gradientes.

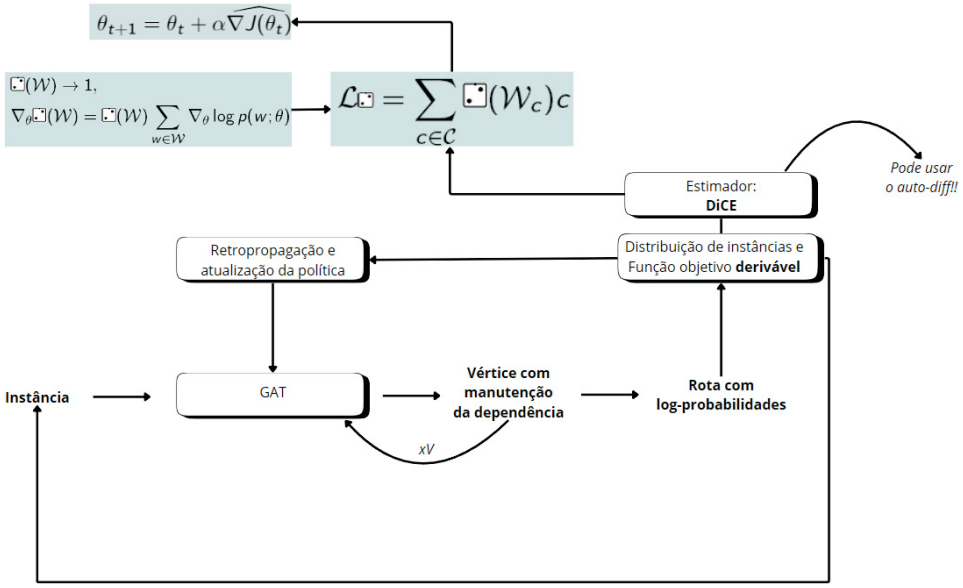



Figura 2.4: Fluxo de atuação do estimador DiCE. $\mathcal{W}$ são os nós estocásticos; $\mathcal{W}_c$ são os nós estocásticos que influenciam a função objetivo; e $c$ é a função objetivo.

A figura 2.4 apresenta o fluxo de treinamento utilizando o estimador DiCE. Como se pode observar, o DiCE também busca estimar os gradientes para a atualização dos parâmetros da rede neural ao construir rotas por meio da amostragem dos seus vértices. A grande diferença do método é a manutenção da dependência entre os parâmetros da rede neural e a função objetivo, permitindo que o paradigma da auto-diferenciação possa ser utilizado.

## 2.7 CONSIDERAÇÕES

Com base nos conceitos apresentados no capítulo 2, fundamentação teórica, utilizamos a programação linear (LP) e a programação linear inteira (ILP), subseções 2.1.1 e 2.1.2, respectivamente, para modelar matematicamente o Problema de Roteamento de Veículos Capacitado (CVRP), utilizando o modelo apresentado na seção 2.2 como referência. A formulação do CVRP como um problema de ILP nos permite implementar o método exato por meio de um modelo SCIP para obter soluções ótimas. Embora tais soluções sejam computacionalmente custosas para serem obtidas, elas servem como um importante referencial para avaliar a qualidade das soluções obtidas por métodos aproximados.

Para mitigar os custos envolvidos na obtenção de soluções ótimas por métodos exatos, integramos os conceitos de Redes Neurais de Grafos (GNN) e Redes de Atenção em Grafos (GAT), seção 2.5 e subseção 2.5.1, respectivamente. A aplicação de uma Pointer Network, subseção 2.5.2, nos permite enfrentar o problema do dicionário variável de saída no CVRP, mapeando diretamente os vértices de entrada aos vértices de saída. O treinamento da GAT elaborada é realizado sob o paradigma do aprendizado por reforço, utilizando tanto o algoritmo REINFORCE com baseline, presente em 2.6.2, quanto o estimador DiCE, definido em 2.6.3.

# 3 REVISÃO DA LITERATURA

O problema CVRP é um dos problemas fundamentais no campo da otimização combinatória, o qual visa determinar qual a rota mais eficiente para uma frota de veículos suprir a demanda de um conjunto de consumidores respeitando as restrições de capacidade dos veículos. O VRP pode ser entendido como uma extensão do TSP, elevando o grau de complexidade deste ao adicionar restrições, como a capacidade de carga dos veículos (Toth et al., 2014).

Christofides (1976) propôs um algoritmo heurístico capaz de encontrar soluções para um TSP dentro de um fator de distância do ótimo de apenas 1,5 vezes. O resolvedor Concorde (Applegate et al., 2006) é considerado um dos melhores softwares para encontrar soluções exatas para o TSP, utilizando-se de estratégias como *cutting planes e Branch-and-bound* para iterativamente resolver versões relaxadas do problema e diminuir o espaço de busca. No caso de uma heurística de procura, a *Lin-Kernighan-Helsgaun* (Helsgaun, 2000) é considerada o estado da arte das heurísticas de procura utilizadas em TSP simétricos. O resolvedor da Google (Google, 2023) demonstra como heurísticas desenhadas à mão, com forte presença da experiência humana, somadas a algoritmos de procura, podem melhorar a qualidade das soluções encontradas ao evitar ótimos locais.

As técnicas de metaheurísticas para resolver problemas de otimização combinatória, como o problema de roteamento de veículos, utilizam abordagens diversificadas e inspiradas em diferentes fenômenos naturais e sociais. Em geral, as metaheurísticas podem ser classificadas em métodos baseados em trajetória, que trabalham com uma única solução, e métodos baseados em população, que utilizam várias soluções simultaneamente. Entre os métodos baseados em trajetória, destacam-se o *Simulated Annealing (SA)* de Kirkpatrick et al. (1983), o *Tabu Search (TS)* de Glover e Laguna (1997), e o *Variable Neighborhood Search (VNS)* de Mladenović e Hansen (1997), que exploram o espaço de soluções movendo-se para soluções vizinhas até atingir um ótimo local. Já entre os métodos baseados em população, que simulam processos de seleção natural ou comportamento coletivo, estão os *Genetic Algorithms (GA)* de Holland (1992), o *Ant Colony Optimization (ACO)* de Dorigo e Stützle (2004), e o *Particle Swarm Optimization (PSO)* de Kennedy e Eberhart (1995). Além disso, a combinação dessas técnicas vem se tornando comum, gerando híbridos que tentam unir as forças de diferentes métodos para melhorar a qualidade das soluções encontradas.

A utilização de redes neurais para resolver problemas de otimização combinatória, como o CVRP, data da publicação de Hopfield e Tank (1985), a qual fez uso da chamada Hopfield-network para resolver pequenas instâncias do TSP. Desde então, redes neurais foram implementadas de variadas maneiras para resolver problemas relacionados, geralmente de maneira *online*, na qual a rede aprende uma solução para cada instância partindo-se do zero. Porém, em 2015, Vinyals introduziu o conceito de PN, a qual utiliza o mecanismo de atenção para retornar permutações do conjunto de entrada como os dados de saída de uma rede neural (Vinyals et al., 2017). As PN podem ser treinadas de maneira *offline* para resolver classes de problemas combinatórios, marcando uma guinada no uso de aprendizado profundo na generalização de soluções entre diferentes instâncias.

Bello et al. (2016) inovou ao empregar a arquitetura Ator-Crítico para treinar uma PN sem dados supervisionados, considerando cada instância como um exemplo de treinamento e o custo de cada rota como uma estimativa do gradiente da política utilizada, alcançando bons resultados para instâncias maiores dos problemas. A contribuição de Bello mostrou que o aprendizado por reforço poderia aprimorar o treinamento de redes neurais para resolver problemas

combinatórios. Nazari et al. (2018) alterou o encoder da PN ao introduzir projeções por elemento, facilitando a atualização do *embedding* após mudanças no estado durante o treinamento.

Os avanços recentes em sistemas de inteligência artificial levaram a um aumento do uso de redes neurais para tratar problemas de otimização. O trabalho de Bello et al. (2016) abriu caminho para o uso do aprendizado por reforço como principal paradigma utilizado para o treinamento dessas redes neurais. Dentro desse contexto, as GNN (Gori et al., 2005) despontaram como uma importante ferramenta na construção de sistemas de otimização, oferecendo grandes vantagens no processamento de problemas combinatórios, como o CVRP. As GNN são capazes de preservar a topografia da informação estruturada em grafos e de refletir a estrutura combinatória de problemas como o TSP, como demonstrado por Dai et al. (2017).

Em *Relational Inductive Biases, Deep Learning and Graph Networks* (Battaglia et al., 2018), os autores apresentam como as GNN podem ter um papel crucial no avanço da compreensão e do processamento de dados dotados de estrutura relacional. Eles argumentam que as GNN podem integrar vieses indutivos relacionais diretamente em sua arquitetura, possibilitando, dessa forma, modelar sistemas complexos - como redes sociais ou estruturas moleculares. Os autores também propuseram uma estrutura (*Graph Nets*) para processar essas informações. O ponto central dessa estrutura é a possibilidade de "empilhar" blocos de processamento de grafos para uma maior representatividade dos dados. Dando sequência, Velickovic et al. (2018), em *Graph Attention Networks*, avançou no domínio das GNN ao propor um mecanismo de atenção para atuar sobre dados estruturados em grafos. Essa abordagem resulta em uma melhora na agregação de informações provenientes dos vizinhos de um vértice ao dinamicamente atribuir diferentes valores acerca da importância relativa entre eles. Além disso, o mecanismo de atenção melhora a interpretabilidade dos dados, pois possibilita entender qual o nível de significância relativa das diferentes conexões dentro de um grafo.

Kool et al. (2019) inaugura o processo de agregação das diferentes estratégias até então utilizadas e propõe um modelo baseado em uma GAT conjugada com um PN para resolver problemas de roteamento, como o TSP e o VRP. Para treinar o modelo, os autores propõem uma arquitetura de RL baseada em um Ator-crítico simples, no qual o crítico é atualizado pelo método do gradiente descendente, utilizando o estimador do algoritmo REINFORCE (Williams, 1992) como sua *baseline*. Os autores reportam resultados quase ótimos para instâncias de até cem nodos para o TSP. O processo de refinamento das estratégias adotadas continua em Lei et al. (2021), com o seu artigo *Solve Routing Problems with a Residual Edge-Graph Attention Neural Network*. Os autores propõem uma nova abordagem ao concatenar os atributos dos arcos no cálculo do mecanismo de atenção da sua GAT ao mesmo tempo que utilizam uma conexão residual entre as suas camadas neurais para possibilitar redes mais profundas sem incorrer em problemas como o de desparecimento dos gradientes. Os resultados reportados sugerem uma diminuição da lacuna entre as soluções ótimas de *benchmarks* como os das bibliotecas *TSPLIB e CVRPLIB* e as soluções encontradas pelo modelo proposto.

## 3.1 CONSIDERAÇÕES

A partir das considerações destacadas na revisão da literatura, nosso modelo GAT elaborado para resolver o CVRP integra algumas das abordagens citadas. O trabalho de Vinyals et al. (2017) sobre as Pointer Networks provê o arcabouço necessário para lidar com a natureza variável do dicionário de saída do CVRP, mapeando diretamente os vértices de entrada às sequências de saída correspondentes às rotas. Essa técnica permite que o modelo possa processar instâncias de tamanhos diferentes, habilidade importante para problemas combinatórios nos quais o tamanho da solução depende do tamanho do conjunto de entrada.

Além disso, seguimos a linha de pesquisa de Bello et al. (2016) e Nazari et al. (2018), aplicando o aprendizado por reforço para treinar o nosso modelo sem a necessidade de dados previamente anotados, como o que ocorre no aprendizado supervisionado. Ao considerar cada instância como um exemplo de treinamento e utilizar o custo das rotas como sinal de recompensa, o nosso modelo aprende a otimizar diretamente a função objetivo do CVRP. O trabalho de Velickovic et al. (2018) sobre as redes de atenção em grafos (GAT) demonstra como aplicar o mecanismo de atenção para aprimorar a agregação de informações presentes nos atributos dos vértices, de modo a atribuir dinamicamente os coeficientes de atenção para capturar relações abstratas entre os elementos do grafo. Lei et al. (2021) aprimora essa estrutura ao considerar, também, os atributos dos arcos no cálculo do mecanismo de atenção. A arquitetura da nossa GAT foi baseada nas arquiteturas propostas por Kool et al. (2019) e Lei et al. (2021) que, influenciados pelo trabalho de Bello et al. (2016), também utilizaram o algoritmo REINFORCE com baseline para treinar a rede neural sob o paradigma do aprendizado por reforço.

# 4 MATERIAIS E MÉTODOS

Neste capítulo, apresentam-se os métodos utilizados para comparar dois modelos na solução do problema de Roteamento de Veículos Capacitado (CVRP). O primeiro modelo busca resolver o problema de forma exata, elaborado segundo a sintaxe do resolvedor SCIP (Bestuzheva et al., 2023) em Python[1]. O segundo modelo corresponde à elaboração de uma Rede de Atenção em Grafos (GAT) estendida com os atributos dos arcos, a qual funcionará como uma metaheurística para encontrar soluções quase-ótimas para o CVRP[2]. Detalhes da implementação de ambos os modelos, assim como os procedimentos para treinamento e avaliação serão também apresentados.

O principal objetivo desta comparação é o de avaliar qual o desempenho de um modelo baseado em uma rede neural de atenção em grafos relativamente a uma solução exata fornecida pelo SCIP em termos de sua qualidade (distância percorrida) e tempo para solução. Dados os desafios de escala enfrentados por métodos exatos em soluções maiores, estima-se que o modelo GAT providenciará respostas quase-ótimas em um tempo significativamente inferior quando comparado com método exato (SCIP).

## 4.1 MODELO SCIP

Apresenta-se o modelo utilizado para resolver o CVRP de forma exata, o qual busca otimizar as rotas usadas por uma frota de veículos para suprir a demanda de um conjunto de consumidores partindo de um único depósito. O objetivo é minimizar a distância total ao mesmo tempo em que as demandas dos consumidores são supridas e a capacidade de carga do veículo é respeitada.

O modelo apresentado segue a formulação básica do CVRP, apresentada na seção 2.2: dado um grafo **G = (N, A)**, no qual **N** = {**No**∪{0}} é um conjunto formado pela união do conjunto dos consumidores, **No = {1, 2, ..., n}**, com o depósito, **{0}**, e **A** = $\{(i, j), \forall i, j \in N, i \neq j\}$ o conjunto de todos os possíveis arcos que conectam os elementos de **N**. O conjunto **M** denota quantos veículos estão disponíveis para realizar entregas. A cada veículo $\boldsymbol{m}$ corresponde uma capacidade $\boldsymbol{Q_m}$ de carga e a cada consumidor $\boldsymbol{i}$ uma quantidade de mercadoria demandada, $\boldsymbol{q_i}$ .

As variáveis de decisão são $\boldsymbol{x_{ijm}}$ e $\boldsymbol{l_{ijm}}$. A variável $\boldsymbol{x_{ijm}}$ é uma variável binária responsável por indicar se o arco $(i, j)$ foi utilizado na rota pelo veículo $m$. Já a variável $\boldsymbol{l_{ijm}}$ indica a carga do veículo $m$ ao deixar o nodo $i$ e antes de chegar ao nodo $j$. O uso destas variáveis permite que o modelo especifique qual a rota tomada por determinado veículo e a quantidade de carga transportada.

Em nosso modelo, baseado no padrão básico do CVRP apresentado por, entre outros, Toth et al. (2014), Borcinova (2017) e Munari et al. (2017), teremos apenas um depósito do qual partem e ao qual voltam os veículos; os veículos partem simultaneamente e a cada um será atribuída uma única rota; o número de consumidores sempre será maior que o número de veículos; e o custo a ser minimizado será a distância, dada pelo somatório de todas as rotas percorridas pela frota.

[1] https://github.com/DanielSacy/CVRP_SCIP
[2] https://github.com/DanielSacy/GAT_RL

### 4.1.1 Função Objetivo

A função objetivo busca minimizar o total da distância percorrida por todos os veículos, e pode ser expressa da seguinte forma:

$$\textbf{Minimizar} \quad Z = \sum_{m \in M} \sum_{(i,j) \in \mathrm{A}} d_{ij} x_{ijm} \tag{4.1}$$

aqui, $d_{ij}$ indica a distância entre os vértices $ie\,j$.

### 4.1.2 Restrição de Visita Única

Essa restrição garante que cada consumidor seja visitado uma única vez:

$$\sum_{m \in M} \sum_{j \in N, j \neq i} x_{ijm} = 1, \quad \forall i \in No \tag{4.2}$$

### 4.1.3 Rotas dos Veículos

A restrição das rotas dos veículos garante que o número correto de veículos deixe o depósito para suprir a demanda dos consumidores. Na forma elaborada, ela assegura que todos os veículos serão usados na solução, partindo do depósito.

$$\sum_{j \in No} x_{0jm} = 1, \quad \forall m \in M \tag{4.3}$$

### 4.1.4 Conservação do Fluxo

Essa restrição garante que o fluxo seja contínuo entre os vértices do grafo, evitando que ocorra acúmulo em algum vértice que não o depósito. Em outras palavras, o número de veículos que entra em um nodo deve ser igual ao número de veículos que sai deste:

$$\sum_{i \in N, i \neq j} x_{ijm} - \sum_{k \in N, k \neq j} x_{jkm} = 0, \quad \forall m \in M, \forall j \in N \tag{4.4}$$

### 4.1.5 Inicialização da Carga ao Partir

Garante que a carga inicial do veículo ao partir do depósito seja corretamente atribuída com base na demanda dos consumidores que aquele servirá. A restrição assegura que o veículo não sairá do depósito com um montante de carga incorreto.

$$l_{0km} = \sum_{(i,j) \in Arcs} q_j \cdot x_{ijm}, \quad \forall m \in M, \forall k \in No \tag{4.5}$$

### 4.1.6 Controle da Demanda

A restrição do controle da demanda garante que o montante total de carga que chega a um consumidor menos o montante que deste sai deve ser igual à demanda desse consumidor.

$$\sum_{j \in N} \sum_{m \in M} l_{jim} \cdot x_{jim} - \sum_{k \in N} \sum_{m \in M} l_{ikm} \cdot x_{ikm} = q_i, \quad \forall i \in No \tag{4.6}$$

### 4.1.7 Restrição de Capacidade de Carga

Essa restrição garante que a carga ($l_{ijm}$) transportada pelo veículo $m$ após partir do vértice $i$ e antes de chegar ao vértice $j$ está entre os limites aceitáveis. Mais precisamente, a carga transportada deve ser no mínimo a demanda do vértice $j$, aqui definida como $q_j$, se a rota $(i, j)$ fizer parte da solução, e no máximo a capacidade de carga do veículo após realizar a entrega em $i$, $(Q_m - q_i)x_{ijm}$. Caso a rota $(i, j)$ não seja utilizada, a variável $l_{ijm}$ será igual a zero.

$$q_j x_{ijm} \leq l_{ijm} \leq (Q_m - q_i)x_{ijm}, \quad \forall i, j \in N, \forall m \in M \tag{4.7}$$

### 4.1.8 Restrição de não-negatividade e variável binária

$$x_{ijm} \in \{0, 1\} \tag{4.8}$$

$$l_{ijm} \geq 0 \tag{4.9}$$

A figura 4.1 apresenta um exemplo de solução encontrado utilizando-se o modelo acima proposto com um conjunto de sete consumidores e três veículos, detalhando-se a carga de cada veículo durante o trajeto e a ordem em que cada veículo visitou os consumidores.

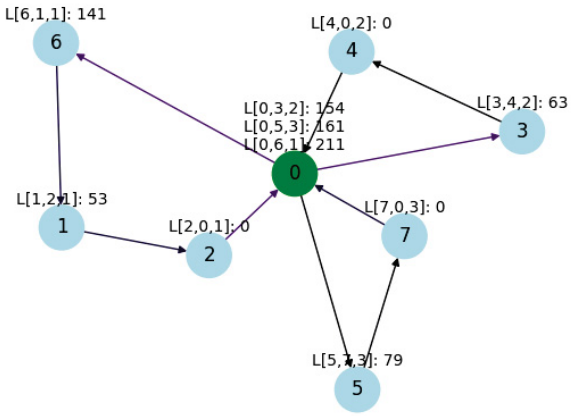



Figura 4.1: Exemplo de solução do CVRP dado pelo modelo apresentado utilizando um conjunto de sete consumidores e três veículos. A variável $l_{ijm}$ acima dos vértices apresenta a informação do vértice visitado, o próximo vértice na rota, o veículo utilizado e a carga restante.

## 4.2 MODELO GAT ESTENDIDO

A arquitetura da nossa GAT foi inspirada pelos trabalhos apresentados na subseção 2.5.1 e na seção 3.1. Dado um grafo de entrada **G = (N, A)**, no qual **N** = {**No** ∪ {**0**}} é um conjunto formado pela união do conjunto dos consumidores, **No = {1, 2, ..., n}**, com o depósito, **{0}**, e **A = {(*i*, *j*), ∀*i*, *j* ∈ *N*, *i* ≠ *j*}** o conjunto de todos os arcos que conectam os elementos de **N**, elabora-se um modelo inspirado nos avanços recentes em GAT, especialmente nos conceitos apresentados por Lei et al. (2021) para superar os obstáculos presentes na solução do CVRP, utilizando tanto dos atributos dos vértices como os dos arcos para encontrar soluções eficientes.

Cada vértice do grafo referencia um consumidor ou o depósito e será representado por suas coordenadas $(x_i, y_i)$ e pela sua demanda $q_i$, ou seja, $n_i \in \mathbb{R}^3 \mid \forall n_i \in N$, enquanto que

os arcos serão representados por um único atributo, a distância euclidiana entre os vértices, $e_{ij} \in \mathbb{R} \mid \forall e_{ij} \in A$. A estrutura escolhida espelha as representações comumente encontradas nos trabalhos precedentes relacionados ao uso de GNN na otimização de problemas combinatórios, nos quais os vértices são representados por seus atributos, como em Bello et al. (2016).

O objetivo do modelo implementado é, dado um grafo de entrada $g$, encontrar uma permutação dos vértices, denominada rota ($\pi$), na qual cada vértice é visitado uma única vez, com a exceção do depósito, $n_0$, de modo que a distância total percorrida seja a menor possível, respeitando a restrição de capacidade de cada veículo:

$$D(\pi \mid g) = \sum_{m \in M} \sum_{(i,j) \in \pi} d_{ij} x_{ijm} \tag{4.10}$$

Para tanto, a rede neural é treinada de modo a aprender uma política estocástica, $p(\pi|g)$, que priorize as rotas que apresentem distâncias pequenas em detrimento daquelas que resultem em distâncias maiores, utilizando-se da regra da cadeia para processamento sequencial (Sutskever et al., 2014):

$$p(\pi|g) = \prod_{t=1}^{n} p(\pi_t|\pi_{1:t-1}, g), \tag{4.11}$$

na qual dada uma rota $\pi$, a probabilidade do vértice $\pi_t$ ser escolhido no passo $t$ é função condicional da instância $g$ e dos vértices previamente selecionados, $\pi_{1:t-1}$.

O modelo seguirá uma arquitetura baseada na divisão entre *encoder* e *decoder* e será treinado seguindo o paradigma do aprendizado por reforço. Assim como em Kool et al. (2019), o *encoder* será o responsável por realizar a vetorização (*embedding*) do grafo de entrada, concatenando os atributos dos vértices aos atributos dos arcos. Ao *decoder* caberá a tarefa de produzir sequencialmente a rota $\pi$ utilizando como entrada o *embedding* produzido pelo *encoder* e uma máscara destinada a evitar que um vértice previamente escolhido seja novamente selecionado.

### 4.2.1 Encoder

O encoder proposto receberá como entrada um grafo **G = (N, A)**. Cada vértice do grafo representará as coordenadas e a demanda de um consumidor, $x_i, y_i, q_i$ - sendo a demanda do depósito igual a zero. Os arcos serão representados por $d_{ij}$, a distância euclidiana entre os vértices $(i, j)$. Esses atributos serão transformados nas dimensões, $d_x$ e $d_e$, por meio de uma camada inteiramente conectada (*fully connected layer (FC)*). O último passo de pré-processamento antes da entrada dos dados no encoder é o da normalização em lote (*batch normalization*) (BN). A BN busca normalizar as saídas das camadas FC, ajustando as ativações para manter a média igual a zero e o desvio-padrão igual a um. Utiliza-se a BN para diminuir a covariância interna, evitando problemas de desequilíbrio entre os gradientes, aumentando a velocidade de treinamento e de convergência e a estabilidade durante o treinamento.

$$x_i^{(0)} = BN(\mathbf{A}_0 x_i + b_0), \forall i \in N, \tag{4.12}$$

$$\hat{e}_{ij} = BN(\mathbf{A}_1 e_{ij} + b_1), \forall (i, j) \in A, \tag{4.13}$$

sendo $x_i$ o *embedding* do vértice $i$ e $\hat{e}_{ij}$ o *embedding* do arco $(i, j)$. $\mathbf{A}_0$ e $\mathbf{A}_1$ são matrizes cujos parâmetros serão atualizados durante o treinamento. As camadas serão indexadas como $\ell \in \{1, ..., L\}$, para indicar qual os atributos dos vértices em determinada camada $\ell$. A entrada da primeira camada da GAT serão os atributos dos vértices $x_i^{(0)} \in \mathbb{R}^{d_x}$, $\forall i \in N$, e os atributos

dos arcos $e_{ij} \in \mathbb{R}^{d_e}$, $\forall (i,j) \in A$. Cada camada da GAT irá atualizar os atributos dos vértices por meio do mecanismo de atenção. Os atributos dos arcos não serão atualizados. O coeficiente de atenção, $\alpha_{ij}$, indicará qual a importância do vértice $j$ para o vértice $i$. O cálculo do mecanismo de atenção será realizado por meio da seguinte função:

$$\alpha_{ij}^{(\ell)} = \frac{\exp\left(\sigma\left(g^{\ell^T}[W^{\ell}(x_i^{(\ell-1)} \| x_j^{(\ell-1)} \| \hat{e}_{ij})]\right)\right)}{\sum_{z=1}^{n} \exp\left(\sigma\left(g^{\ell^T}[W^{\ell}(x_i^{(\ell-1)} \| x_z^{(\ell-1)} \| \hat{e}_{iz})]\right)\right)} \tag{4.14}$$

aqui, $g^{\ell}$ e $\mathbf{W}^{\ell}$ são vetores e matrizes parametrizáveis, respectivamente, $\sigma(\cdot)$ indica a função de ativação, *LeakyReLU*, e $\cdot||\cdot$ uma operação de concatenação.

A função de ativação *LeakyReLU* atribui um pequeno gradiente às entradas negativas, garantindo que a GAT possa aprender mesmo quando as ativações sejam inferiores a zero. Evita-se, dessa forma, que os neurônios apresentem saídas nulas em seu processamento e, por conseguinte, que o seu gradiente seja zero. Caso o gradiente de um neurônio torne-se zero, ele não contribuirá mais para o aprendizado da rede. Além disso, a *LeakyReLU* é computacionalmente mais simples, aumentando a eficiência de redes neurais maiores.

A saída da camada $\ell$, se $\ell \neq L$, será dada pela concatenação entre os atributos dos vértices obtidos pelo processamento da camada anterior e os atributos atualizados pelo mecanismo de atenção da camada atual:

$$x_i^{(\ell)} = x_i^{(\ell-1)} + \sum_{j=1}^{n} \alpha_{ij}^{(\ell)} W_1^{\ell} x_j^{(\ell-1)}, \tag{4.15}$$

Lei et al. (2021) a chama de conexão residual. As conexões residuais operam de forma a adicionar o seu *input* ao *output* decorrente do processamento da camada atual, aumentando a estabilidade do treinamento e evitando que os gradientes tornem-se tão baixos que acabem por não contribuir ao aprendizado - *"vanishing gradients"*.

A última camada do encoder, a camada $L$, apresentará como saída os atributos de cada vértice, $x_i^{(L)}$, os quais serão usados para elaborar a representação do grafo, um vetor que representa toda a informação contida no grafo, por meio da função:

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} (x_i^{(L)})_j, \; j \in \{1, \ldots, d_x\}, \tag{4.16}$$

a representação do grafo será dada, portanto, pela média dos atributos dos vértices apresentados pela última camada sobre todos os vértices, $\bar{x} = \{\bar{x}_1, \ldots, \bar{x}_{d_x}\}$, $\bar{x}_j \in \mathbb{R}$.

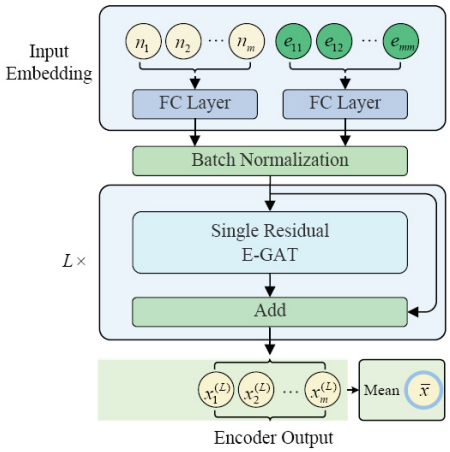



Figura 4.2: Arquitetura do encoder. Fonte: (Lei et al., 2021)

### 4.2.2 Decoder

O *decoder* implementado baseia-se no modelo proposto por Vaswani et al. (2023) e utiliza um mecanismo de atenção múltiplo - *multi-head attention mechanism (MHA)*. Conjuntamente ao mecanismo de atenção, será implementada uma PN (Vinyals et al., 2017), pois esta possibilita utilizar um modelo transformer para processar problemas combinatórios, afastando o problema do dicionário fixo no processamento do conjunto de saída. A PN atribui a cada vértice uma determinada probabilidade que será utilizada, a cada passo de processamento, para apontar qual a posição ocupada, no conjunto de entrada, pelo vértice escolhido para compor a rota. Dessa forma, o conjunto de entrada define o tamanho do conjunto de saída da GAT. Seguindo o modelo de Lei et al. (2021), essa implementação não faz uso de normalização de *batch*, conexões residuais e FC no *decoder*.

O *decoder* é composto por duas camadas de atenção. A decodificação acontece sequencialmente e a cada passo, $t \in \{1, \ldots, n\}$, o decoder apresenta um vértice para compor a rota, $\hat{\pi}_t$, baseado na representação vetorial recebida do encoder e nas decodificações prévias, $\hat{\pi}_{t'}$, geradas em $t' < t$.

A primeira camada do decoder é composta por um MHA (com $H$ cabeças) que recebe como entrada um vetor de contexto, $c_t^{(0)}$, e produz um novo vetor de contexto, $c_t^{(1)}$. O $c_t^{(0)}$ é construído pela concatenação da representação do grafo recebida do encoder, $\bar{x}$, do último vértice escolhido pelo decoder, $\hat{\pi}_{t-1}$, e do primeiro vértice selecionado, $\hat{\pi}_1$. Caso seja o primeiro passo no processamento, ou seja, $t = 1$, $c_t^{(0)}$ será obtido pela conjunção entre $\bar{x}$ e um parâmetro vetorial $\vec{v}$ de dimensão $d_x$:

$$c_t^{(0)} = \begin{cases} \bar{x} + W_x(x_{\hat{\pi}_1}^{(L)} \,||\, x_{\hat{\pi}_{t-1}}^{(L)}), & \text{if } t > 1 \\ \bar{x} + \vec{v}, & \text{if } t = 1 \end{cases} \tag{4.17}$$

O MHA utilizado envolve três componentes essenciais: vetores de consulta, $\vec{q}$, vetores chave, $\vec{k}$, e vetores de valor, $\vec{v}$. O vetor de consulta é obtido pela transformação do vetor de contexto, $c_t^{(0)}$, por uma matriz parametrizável, $W^Q$. Os vetores chave e de valor são obtidos pela transformação da representação vetorial dos vértices, $x_i^{(L)}$, pelas matrizes parametrizáveis $W^K$ e $W^V$, respectivamente. A dimensão dos vetores será dada por $d_v = (d_x/H)$:

$$q = W^Q c_t^{(0)}, \quad W^Q \in \mathbb{R}^{d_v \times d_x}, \tag{4.18}$$

$$v_i = W^V x_i^{(L)}, \quad W^V \in \mathbb{R}^{d_v \times d_x}, \quad i \in \{1, 2, \ldots, n\}, \tag{4.19}$$

$$k_i = W^K x_i^{(L)}, \quad W^K \in \mathbb{R}^{d_v \times d_x}, \quad i \in \{1, 2, \ldots, n\}. \tag{4.20}$$

Os vetores de consulta e chave são utilizados para calcular o coeficiente de atenção, $u_{i,t}^{(1)}$, por meio da sua combinação, onde $u_{i,t}^{(1)} = \frac{q^T k_i}{\sqrt{d_v}}$, se o vértice $i$ ainda não foi selecionado, e $-\infty$ caso contrário. Ao indicar que o coeficiente de atenção de determinado vértice $i$ é igual a $-\infty$, o modelo acaba por mascarar $i$ para que esse não seja selecionado novamente durante o processamento. Para garantir que os coeficientes de atenção estejam normalizados, sobre eles aplica-se a função softmax, transformando-os em uma distribuição de probabilidades que indica a importância relativa de cada vértice dentro do contexto de atenção:

$$\hat{u}_{i,t}^{(1)} = softmax(u_{i,t}^{(1)}) \tag{4.21}$$

O processo será repetido $H$ vezes, cada qual com um conjunto diferente de parâmetros, compondo o MHA. O resultado de cada processo será então concatenado em série e utilizado para calcular o vetor de contexto resultante da primeira camada do decoder, $c_t^{(1)}$, por meio de uma FC:

$$c_t^{(1)} = W_f \cdot \left( \Big\|_{h=1}^{H} \sum_{i=1}^{n} (\hat{u}_{i,t}^{(1)})^h \, (v_i^h) \right) \tag{4.22}$$

A utilização do MHA permite que o modelo aumente o seu poder representacional, capturando padrões mais complexos presentes nos dados de entrada ao considerá-los sob diferentes perspectivas. Além disso, o MHA torna o aprendizado decorrente do mecanismo de atenção mais estável.

O vetor de contexto $c_t^{(1)}$ será a entrada para a segunda camada do decoder. Essa camada será composta por um mecanismo de atenção simples, que calculará novos coeficientes de atenção, $\hat{u}_{i,t}^{(2)} \in \mathbb{R}$, $\forall i \in \{1, \ldots, n\}$:

$$u_{i,t}^{(2)} = \begin{cases} C \cdot \tanh\left(\frac{c_t^{(1)} k_i}{\sqrt{d_v}}\right) & \text{, se } i \neq \hat{\pi}_{t'} \quad \forall t' < t. \\ -\infty & \text{, de outra forma.} \end{cases} \tag{4.23}$$

Assim como proposto por Bello et al. (2016), Lei et al. (2021) também optou por limitar a amplitude dos valores potencialmente assumidos pelos coeficientes de atenção $\hat{u}_{i,t}^{(2)}$ em $[-C, C]$. Essa operação visa aumentar a estabilidade dos resultados e o controle na saída de dados, capacidades importantes em ambientes de aprendizado por reforço. Os novos coeficientes de atenção serão então utilizados para calcular a distribuição de probabilidade para cada vértice em determinado momento $t$:

$$p_{i,t} = p_\theta(\hat{\pi}_t | s, \hat{\pi}_{t'}, \forall t' < t) = \text{softmax}(u_{i,t}^{(2)}), \tag{4.24}$$

essa distribuição de probabilidade será usada para escolher qual vértice, $\hat{\pi}_t$, incluir na rota por meio de uma estratégia de decodificação amostral - caso o modelo esteja em treinamento - ou

gulosa, escolhendo-se o vértice que apresentar a maior probabilidade $p_{i,t}$, - caso o modelo esteja na fase de testes.

### 4.2.3 Treinamento

O treinamento foi realizado utilizando-se duas estratégias diferentes para estimar os gradientes: a primeira empregou o algoritmo REINFORCE com uma *Greedy Rollout Baseline*, inspirando-se no trabalho de Kool et al. (2019); a segunda utilizou o estimador DiCE (Foerster et al., 2018). O algoritmo REINFORCE é um método Monte Carlo que estima os gradientes da política $\pi_\theta(A_t|S_t)$ por meio de amostragem e a baseline atua de modo a minimizar a variância produzida pelo método. A função *Loss*, $L(\theta|s)$, é definida como a recompensa esperada pela política $\pi_\theta$ dada uma instância s:

$$L(\theta|s) = \mathbb{E}_{\pi_\theta}\left[R(\hat{\pi}|s)\right] \tag{4.25}$$

na qual $R(\hat{\pi}|S)$ representa a recompensa para uma determinada rota $\hat{\pi}$. O gradiente da função *Loss* será então estimado usando o algoritmo REINFORCE com uma baseline $b(s)$ para minimizar a variância:

$$\nabla_\theta L(\theta|s) = \mathbb{E}_{\pi_\theta(\hat{\pi}|s)}\left[(R(\hat{\pi}|s) - b(s))\nabla_\theta \log \pi_\theta(\hat{\pi}|s)\right] \tag{4.26}$$

Em nosso modelo, a estratégia *Greedy Rollout* atua como uma baseline na qual a política escolhe gulosamente os vértices que irão compor a rota baseado na maior probabilidade a cada ponto de decisão, gerando uma solução determinística. Essa política é executada utilizando-se uma segunda execução do ator (GAT), mas em modo determinístico, em uma estrutura chamada de duplo-ator.

Ao final de cada época, o ator (GAT) é avaliado por um conjunto de instâncias de validação e o resultado comparado com os resultados obtidos por meio do *Greedy Rollout*. Se a política aprendida pelo ator for significativamente superior ao resultado obtido deterministicamente - avaliado por meio de um t-test com um nível de significância $\alpha = 5\%$ -, os parâmetros da baseline serão atualizados de acordo com os parâmetros do ator. Isso garante que o modelo será sempre testado contra uma baseline significativa, estimulando o ganho de desempenho. Nas palavras de Wouter Kool (tradução nossa):

> "Usando *Greedy Rollout* como uma baseline $b(s)$, a função $L(\pi) - b(s)$ será negativa se a amostra da solução da política $\pi$ for melhor que aquela dada pelo *Greedy Rollout*, implicando em um reforço de tais ações, e *vice-versa* (Kool et al., 2019)[3]"

Após a utilização do algoritmo REINFORCE, o modelo foi treinado usando o método DiCE (estimador Monte Carlo diferenciável), que oferece uma maneira mais estável para calcular os gradientes sob o contexto do aprendizado por reforço em problemas de otimização combinatória. O DiCE visa enfrentar os problemas ocasionados pela alta variância e pelo cálculo incorreto dos gradientes de alta ordem, comuns em estimadores como o REINFORCE. Ao utilizar a estrutura dos grafos de computação estocástica (SCG, na sigla em inglês), a técnica DiCE assegura que correta relação de dependência entre os vértices estocásticos, como as escolhas efetuadas pela política (ator), e os vértices de custo, a função objetivo, como a distância total percorrida, seja mantida ao longo do grafo de computação estocástica. Ao evitar a quebra da relação de dependência entre os vértices do SCG, o DiCE é capaz de calcular gradientes de

[3] "With the greedy rollout as baseline b(s), the function $L(\pi) - b(s)$ is negative if the sampled solution $\pi$ is better than the greedy rollout, causing actions to be reinforced, and vice versa."

ordens mais altas, melhorando a convergência do modelo ao aumentar a precisão das atualizações nos parâmetros da política (Foerster et al., 2018).

Ao treinar o modelo utilizando o estimador DiCE, duas arquiteturas foram testadas. A primeira corresponde à arquitetura padrão empregada no treinamento por meio da estratégia *Greedy Rollout*. A segunda arquitetura, visando ampliar as capacidades do estimador, foi alterada para permitir uma melhor propagação dos gradientes pela estrutura da rede neural na retro propagação para ajustes nos parâmetros. Dessa forma, as funções de ativação não-lineares *LeakyReLU* e *Tanh* foram alteradas para a ***Mish***, uma função de ativação mais moderna baseada na *tanh* (Misra, 2020), e as camadas de normalização em lote (*batch normalization*) foram retiradas.

### 4.2.4 Métricas

O trabalho apresenta quatro métricas para a avaliar o desempenho dos métodos aplicados para a solução do CVRP: distância média, distância média relativa ao método exato, tempo médio para solução e tempo total. A distância média corresponde à distância total percorrida ao longo de todas as instâncias de teste relativamente ao total de instâncias testadas. Essa métrica busca comparar a qualidade das soluções encontradas por cada modelo. A distância média relativa da solução ótima visa apresentar a distância percentual de cada modelo baseado em rede neural testado em relação ao resultado obtido pelo método exato utilizando o SCIP, de modo a demonstrar a proximidade das soluções "heurísticas" às soluções exatas. Já o tempo médio para solução atua como uma *proxy* para comparar a eficiência computacional das abordagens baseadas na GAT. O tempo total mostra a soma do tempo dispendido pelas técnicas utilizadas na solução de cada instância. Essa métrica exprime a grande diferença temporal existente entre o método exato e o método heurístico baseado em GAT em relação ao tempo necessário para resolver o problema.

## 4.3 CONSIDERAÇÕES

Neste capítulo, apresentamos a implementação de dois métodos distintos para resolver o problema de roteamento de veículos capacitado (CVRP). O primeiro método emprega programação linear inteira (ILP) para formular matematicamente o problema, definindo a função custo e as restrições de forma linear. Dessa forma, podemos utilizar a interface em python do resolvedor SCIP para elaborarmos um modelo que encontre soluções ótimas para as instâncias ao mesmo tempo em que respeita as restrições elaboradas, como a capacidade dos veículos e a demanda dos consumidores.

O segundo método utiliza conceitos de aprendizado profundo para implementar uma rede de atenção em grafos (GAT) treinada sob o paradigma do aprendizado por reforço. As tecnologias de GNN e GAT são importantes para garantir que uma boa representação da informação contida no grafo seja obtida para cada instância do problema. A integração das *Pointer Networks* supera a questão do dicionário de saída variável presente em problemas combinatórios, permitindo que o modelo GAT produza sequencialmente as rotas correspondentes às permutações dos vértices de entrada. O treinamento por meio do algoritmo REINFORCE e do estimador DiCE visam aprimorar a política da rede neural (ator), estabilizando o treinamento ao reduzir a variância presente na estimativa dos gradientes. Ambos os modelos elaborados serão utilizados nos experimentos detalhados no próximo capítulo, no qual terão sua eficiência e desempenho avaliados.

# 5 EXPERIMENTOS

O foco do trabalho é o de realizar uma comparação entre os resultados obtidos entre um modelo de otimização exata e um modelo de aprendizado profundo, utilizando um mecanismo de atenção em grafos, na resolução de instâncias do problema de roteamento de veículos capacitado. Entretanto, a arquitetura utilizada na elaboração da GAT segue os preceitos dos trabalhos apresentados por Kool et al. (2019) e Lei et al. (2021) e pode, portanto, sob poucas alterações na máscara empregada, nas entradas de dados e no processamento do decoder, ser utilizada para solucionar outros problemas combinatórios, como variações dos problemas de roteamento.

Ambos os modelos, exato e de aprendizado profundo, foram elaborados, treinados e testados em ambiente doméstico sob a seguinte configuração: processador Intel de 13ª geração com 10 núcleos de processamento (2500MHz) e 32GB de memória RAM; Placa gráfica apta ao processamento paralelo Nvidia GTX 1660 Super com 6GB de memória.

Os modelos de aprendizado profundo foram treinados sob duas estratégias de aprendizado por reforço: cálculo dos gradientes usando o algoritmo REINFORCE com uma baseline baseada em um *Greedy Rollout* do ator; e estimativa dos gradientes utilizando-se o estimador DiCE. Nesta segunda estratégia foram empregadas duas arquiteturas. A primeira utilizou a mesma arquitetura presente no treinamento sob o REINFORCE. A segunda sofreu pequenas alterações com o intuito de permitir a livre propagação dos gradientes pela rede neural, substituindo-se as funções de ativação não-lineares *LeakyReLU* e *Tanh* pela ***Mish*** e retirando-se a camada de normalização em lote da estrutura do encoder.

Os hiper-parâmetros empregados no treinamento das redes neurais foram mantidos constantes em todos os treinamentos efetuados, conforme os valores expressos na tabela 5.1. As instâncias de treinamento foram geradas aleatoriamente com 21 (consumidores mais depósito) vértices cada, com as coordenadas de cada vértice pertencendo ao intervalo dado por $[0, 1] \times [0, 1]$ e a distância dos arcos correspondendo à distância euclidiana entre os vértices. Ao todo, foram utilizadas 768.000 instâncias em lotes de 512 em um treinamento por cem épocas - 1500 iterações por época, totalizando 150.000 passos de treinamento.

Para teste, foram utilizados dois conjuntos distintos de dados. Primeiramente, foi elaborado um conjunto sintético de cem instâncias do problema com dez consumidores cada, seguindo o processo de geração de instâncias para os dados de treinamento. Esse dataset foi construído para possibilitar a comparação das soluções obtidas pela abordagem GAT com aquelas encontradas pelo modelo SCIP. Instâncias maiores (> 20 vértices) não puderam ser resolvidas em ambiente doméstico devido à restrição de tempo. As características do modelo SCIP foram apresentadas na seção 4.1. Um conjunto $M = 3$ foi empiricamente determinado, de modo a evitar instâncias inviáveis sem prejudicar a qualidade da solução, para a resolução desse primeiro conjunto de dados.

Um segundo conjunto sintético de dados de cem instâncias com vinte consumidores cada foi elaborado para comparar a qualidade das soluções e o tempo de resposta entre os modelos de aprendizado profundo. Esse segundo teste objetiva comparar as diferentes estratégias de treinamento empregadas: REINFORCE com *Greedy Rollout*, DiCE e $DiCE_{Mish}$.

[1] Valor definido com base no trabalho de Lei et al. (2022)

| Hiper-parâmetro | Valor |
|---|---|
| Dimensão dos vértices de entrada | 3 |
| Dimensão dos arcos de entrada | 1 |
| Dimensão da vetorização dos vértices | 128 |
| Dimensão da vetorização dos arcos | 16 |
| Camadas no Encoder | 4 |
| Inclinação negativa - LeakyReLU | 0.2 |
| Taxa de Dropout | 0.6 |
| Iterações do Decoder | 100 |
| Learning Rate (LR) | 1e-4 |
| Temperatura da SOFTMAX (T) | 2.5[1] |

Tabela 5.1: Hiper-parâmetros utilizados no treinamento das redes de atenção em grafos.

## 5.1 TREINAMENTO

Nesta seção, efetua-se uma análise das diferenças no desempenho do treinamento dos modelos GAT implementados. Os modelos diferem, em essência, na escolha do estimador dos gradientes empregado. Busca-se analisar as implementações em relação à convergência do aprendizado por época de treinamento e o tamanho da memória alocada para realizar a retro-propagação.

A figura 5.1 mostra a evolução do valor da recompensa (resultado da função objetivo) ao longo das épocas de treinamento dos três modelos de GAT implementados: *Greedy Rollout*, DiCE e $DiCE_{Mish}$. No eixo vertical temos o valor da função objetivo (recompensa) e no eixo horizontal o número de épocas. Percebe-se que todos os modelos convergem rapidamente até a vigésima época, quando o aprendizado torna-se mais estável. Cabe destacar que o método REINFORCE utilizando *Greedy Rollout* como baseline apresenta um desempenho ligeiramente superior em termos da velocidade de convergência

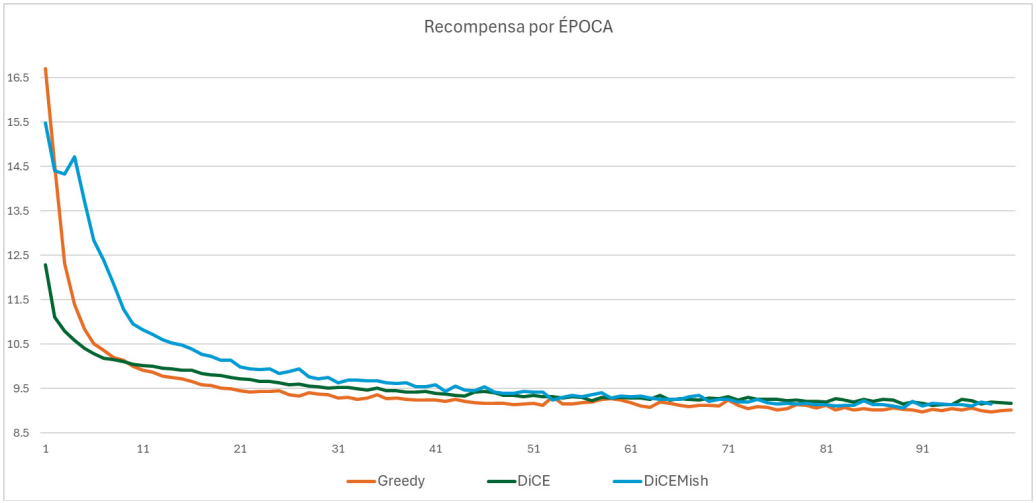



Figura 5.1: Recompensa por época

O gráfico da figura 5.2 mostra a quantidade de memória alocada por época por modelo em megabytes (MB). A memória foi registrada após o *forward pass* e antes da etapa de retro-propagação, durante o processo de treinamento. No início do treinamento, os modelos alocaram uma grande quantidade de memória, especialmente o modelo *Greedy Rollout*, com valores

próximos a 4GB. Ao longo do treinamento, a quantidade de memória alocada pelos modelos diminuiu, estabilizando-se após aproximadamente trinta épocas. Cabe notar que o modelo $DiCE_{Mish}$ utiliza, de maneira consistente, menos memória que os pares, o que sugere que esse seja um modelo mais eficiente em termos de memória.

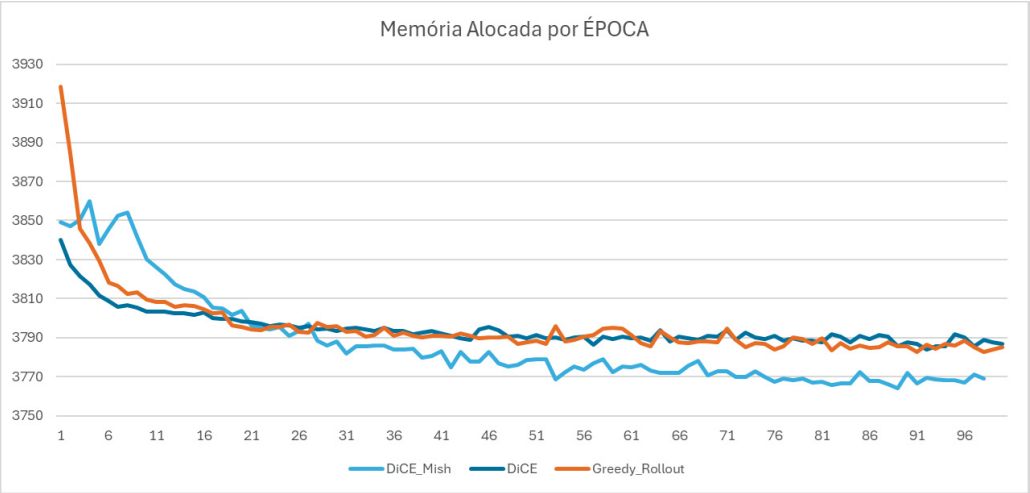



Figura 5.2: Memória alocada por época

Cabe ressaltar, ainda, uma diferença importante entre os modelos DiCE e o modelo *Greedy Rollout*. As implementações que utilizam o DiCE como estimador dos gradientes não fazem uso da arquitetura de duplo-ator, como acontece com métodos como o *Greedy Rollout*, na qual o segundo ator atua como um crítico. Ao evitar a necessidade de uma segunda rede neural para atuar como um crítico, a implementação DiCE acaba por reduzir significativamente a sobrecarga computacional, sendo mais eficiente em termos de tempo e recursos computacionais. Adicionalmente, a arquitetura DiCE apresenta menor complexidade computacional, pois a estrutura de duplo-ator do *Greedy Rollout* requer que tanto o ator quanto o crítico sejam otimizados constantemente, enquanto que o DiCE otimiza apenas um único ator.

Essa diferença torna-se evidente na comparação do tempo total gasto para o treinamento dos modelos citados. Para treinar por cem épocas, o modelo *Greedy Rollout* necessitou de 20.52 horas, aproximadamente 738 segundos por época. Já o modelo DiCE, levou apenas 14.24 horas, 512 segundos por época. Finalmente, a implementação $DiCE_{Mish}$ levou 14 horas para treinar por cem épocas ou 504 segundos por época. Percebe-se que as arquiteturas que empregam o estimador DiCE oferecem vantagens na redução do tempo total para treinamento sem, entretanto, reduzir a qualidade da solução apresentada.

## 5.2 RESULTADOS NA COMPARAÇÃO COM O MODELO SCIP

Nesta seção, o desempenho dos modelos implementados sob a arquitetura GAT na solução do CVRP será comparado às soluções apresentadas pelo método exato elaborado sob a arquitetura do resolvedor SCIP. As avaliações serão efetuadas com base em três métricas: **distância média percorrida**, **distância média relativa da solução ótima** e **tempo total dispendido**. As instâncias de teste foram geradas aleatoriamente, utilizando um conjunto de dez consumidores. No modelo SCIP, o conjunto de carros **M** foi previamente determinado e é composto por três veículos $\boldsymbol{m}$.

A primeira métrica, distância média percorrida, exprime a distância média das rotas geradas por cada modelo em um conjunto de cem instâncias de teste. Como pode ser observado

na figura 5.3, que apresenta o desempenho médio de todos os modelos, o modelo SCIP apresenta a menor distância média, como era esperado. Com exceção do resultado apresentado pelo modelo previamente ao treinamento, "GAT_Base", os modelos gerados por aprendizado profundo não apresentam grandes variações entre si, apresentando, como um todo, um resultado muito próximo à solução ótima apresentada pelo SCIP.

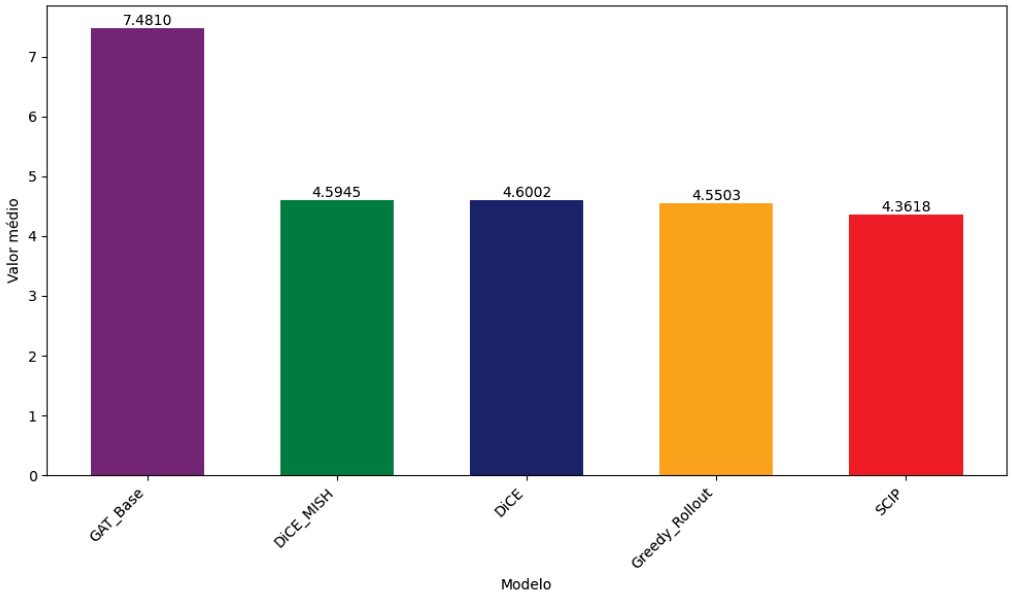



Figura 5.3: Distância média percorrida, por modelo, em 100 instâncias

A figura 5.4 nos permite avaliar precisamente a variação percentual entre as soluções apresentadas pelos modelos. A coluna **GAT_base** mostra o resultado obtido pelo modelo previamente ao treinamento. A estratégia *Greedy Rollout* apresenta um resultado apenas 4,3% superior ao apresentado pelo modelo SCIP. Cabe notar que as demais técnicas também apresentam resultados muito próximos, com o modelo $DiCE_{Mish}$ apenas 1% superior ao modelo de treinamento determinístico.

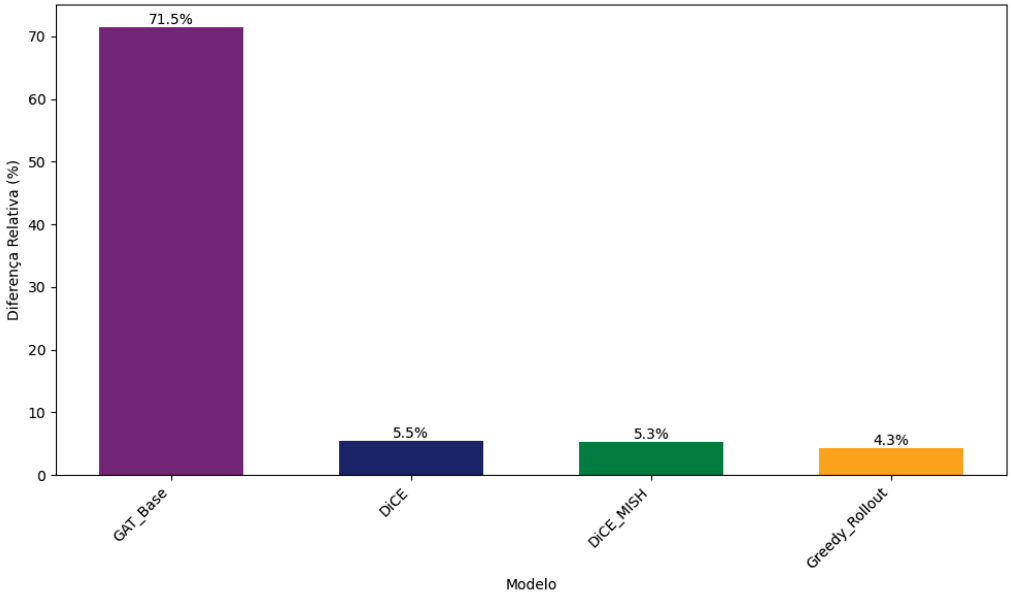



Figura 5.4: Distância média relativa da solução ótima

Finalmente, a figura 5.5 mostra o tempo total dispendido para a solução de cem instâncias de teste, cada uma com dez consumidores. A imagem confirma que a solução do problema CVRP por um método exato, de forma exaustiva, requer um tempo significativamente superior aos modelos de aprendizado profundo para ser encontrada.

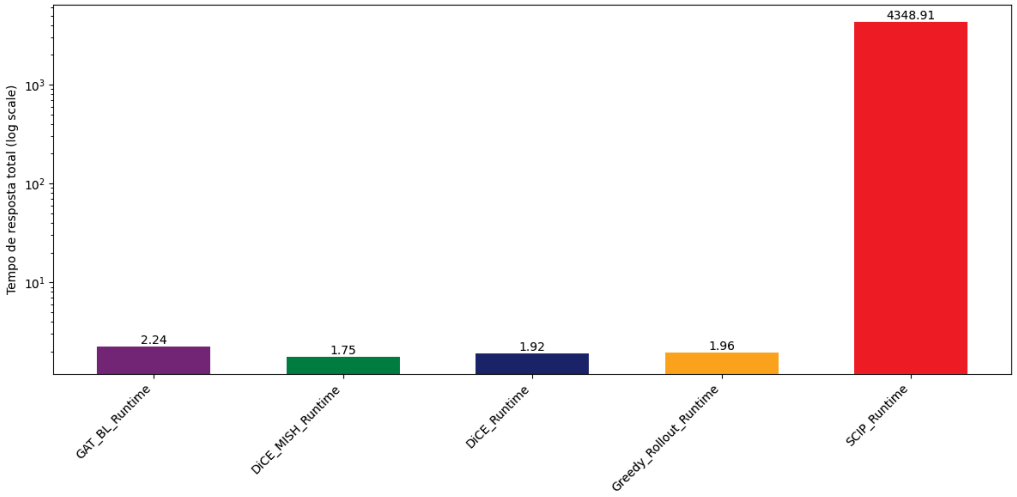



Figura 5.5: Tempo total para solução de 100 instâncias em escala Logarítmica

## 5.3 COMPARAÇÃO ENTRE MODELOS DE APRENDIZADO PROFUNDO

Nesta seção, comparamos o desempenho dos modelos implementados sob a arquitetura GAT - *Greedy Rollout*, DiCE e $DiCE_{Mish}$ - na solução do CVRP. As avaliações serão efetuadas com base em quatro métricas: **distância média percorrida, distância média relativa da solução do modelo com *Greedy Rollout*, tempo total dispendido** e **tempo relativo ao modelo não-treinado**. As instâncias de teste foram geradas aleatoriamente, utilizando um conjunto de vinte consumidores.

O gráfico apresentado na figura 5.6 apresenta a distância média percorrida, por modelo, na solução de cem instâncias com vinte consumidores cada. Percebe-se que os modelos treinados - *Greedy Rollout*, DiCE e $DiCE_{Mish}$ - apresentam valores muito próximos entre si. Entre eles, aquele que apresentou o menor valor médio foi o treinado sob o algoritmo REINFORCE com uma baseline do tipo *Greedy Rollout*, com um valor médio de 7,0951. A arquitetura $DiCE_{Mish}$ apresentou o segundo melhor resultado, 7,1279, com a arquitetura DICE em último com um valor de 7,1584.

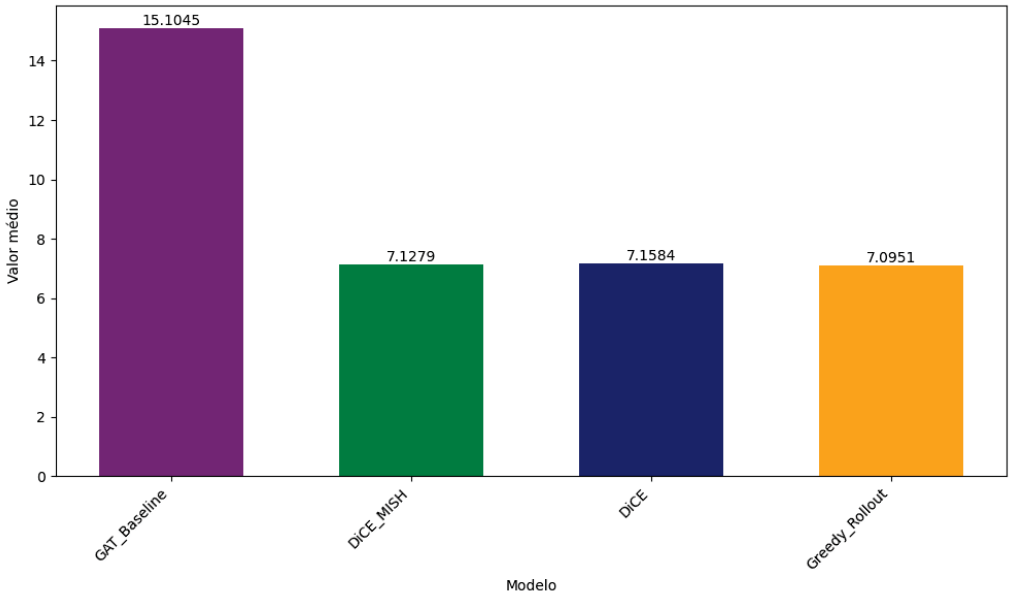



Figura 5.6: Distância média percorrida, por modelo, em 100 instâncias do CVRP com 20 consumidores

A figura 5.7 aprofunda a comparação entre os modelos ao mostrar a diferença relativa das soluções médias em comparação com aquela obtida pelo modelo com a baseline *Greedy Rollout*. A escala foi apresentada em base logarítmica para acentuar a diferença entre os valores. A significativa diferença observável entre o modelo não-treinado e os modelos treinados sugere que as GAT foram capazes de capturar os padrões necessários para a solução do CVRP. Cabe notar que a pequena diferença entre o valor da solução média encontrado pelo modelo $DiCE_{Mish}$ e o valor do modelo *Greedy Rollout* pode ser considerada não-significativa.

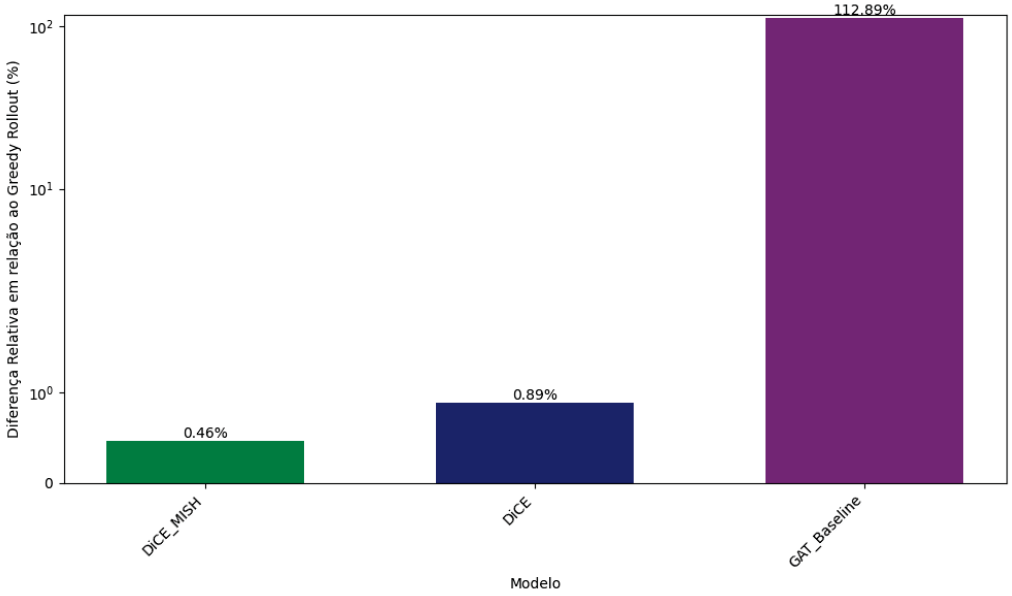



Figura 5.7: Distância média relativa do modelo Greedy Rollout em escala Logarítmica.

O gráfico da figura 5.8 mostra o tempo total necessário para resolver cem instâncias do CVRP para cada um dos modelos de aprendizado profundo. Dos modelos treinados, $DiCE_{Mish}$ foi aquele que apresentou o menor tempo total, 3,84 segundos. O modelo DiCE levou 4,05

segundos para resolver as cem instâncias, enquanto que o modelo com *Greedy Rollout* levou o maior tempo, 4,23 segundos.

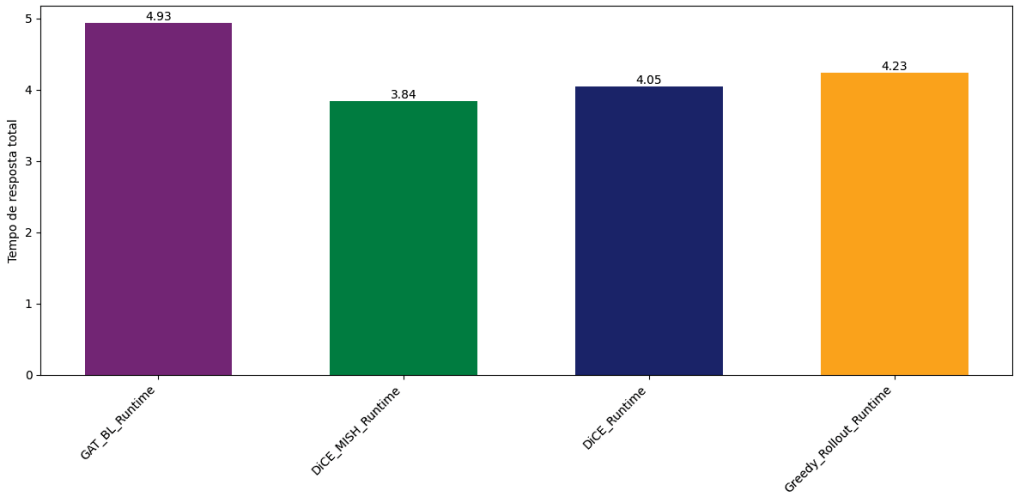



Figura 5.8: Tempo total para solução de 100 instâncias

Para ressaltar a diferença entre o tempo gasto pelos modelos para resolver o problema, a figura 5.9 apresenta o tempo de solução dos modelos treinados relativamente ao modelo não-treinado para as cem instâncias de teste. Observa-se que o modelo *Greedy Rollout*, apesar de ser mais rápido que a baseline, resolvendo as instâncias em um tempo **14,2%** menor, é mais lento que as arquiteturas DiCE. A diferença é significativa quando comparado com o modelo $DiCE_{Mish}$, que foi **22,1%** mais rápido que o modelo não-treinado para resolver o conjunto de teste. Percebe-se que o modelo $DiCE_{Mish}$, além de apresentar um bom desempenho na resolução do CVRP em termos de tempo, provê soluções quase-ótimas em termos de custo (**5,3%** do modelo SCIP), como mostrado na figura 5.4. Aliado a uma diferença de apenas **0,46%** (figura 5.7) em relação ao modelo *Greedy Rollout*, o modelo $DiCE_{Mish}$ desponta como uma alternativa eficiente para problemas de roteamento, equilibrando a qualidade da solução com eficiência computacional.

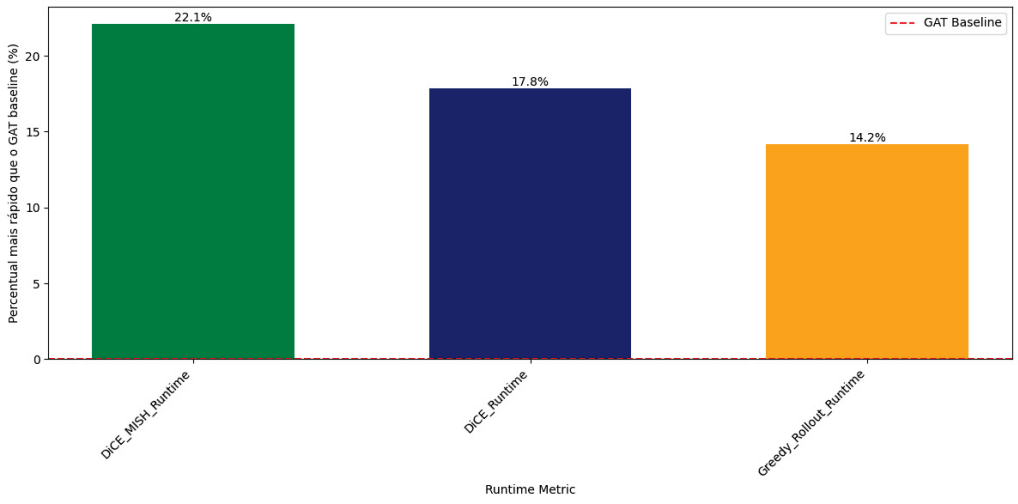



Figura 5.9: Tempo de resposta relativo ao GAT Baseline para diferentes arquiteturas (em %).

## 5.4 CONSIDERAÇÕES

Neste capítulo, apresentamos os experimentos realizados e os resultados obtidos ao comparar o desempenho do modelo SCIP com os modelos de aprendizado profundo baseados na arquitetura das redes de atenção em grafos (GAT) na resolução do CVRP. Cabe notar que os hiperparâmetros utilizados pelos modelos de redes neurais foram mantidos constantes, conforme tabela 5.1, com o intuito de proporcionar uma melhor comparação entre eles. Dessa forma, o efeito das técnicas de treinamento utilizadas, como o REINFORCE e o DiCE, pode ser isolado, avaliando-se o seu impacto no desempenho dos modelos.

Os resultados mostram que, apesar das limitações encontradas para executar o modelo exato em instâncias maiores (tornando impraticável o uso do modelo SCIP em ambiente doméstico para conjuntos de dados com instâncias de vinte consumidores), os modelos GAT treinados apresentaram soluções próximas ao ótimo (**5,3%** do modelo SCIP), como mostrado na figura 5.4, com um tempo de processamento substancialmente inferior, demonstrado na figura 5.5. A utilização de dois datasets distintos - um com dez consumidores por instância, para comparação com o modelo SCIP, e outro com vinte consumidores por instância, para avaliar os modelos GAT entre si - é um indicativo da maior flexibilidade dos modelos de aprendizado profundo. Além disso, a evolução do resultado da função objetivo (recompensa) por época e o consumo de memória medido indicam que as arquiteturas que utilizaram o estimador DiCE apresentam vantagens quando comparadas ao modelo duplo-ator utilizado pelo *Greedy Rollout*. As arquiteturas DiCE alcançaram desempenho semelhante ao obtido pelo *Greedy Rollout* quanto à qualidade das soluções, necessitando, entretanto, de um menor tempo de treinamento e de menos recursos computacionais para serem treinadas. Desse modo, percebe-se que as técnicas de aprendizado profundo são alternativas viáveis e eficientes para resolver o CVRP, especialmente em cenários nos quais o uso de métodos exatos é inviável, por restrições de tempo ou de recursos computacionais.

# 6 CONCLUSÃO E TRABALHOS FUTUROS

O trabalho apresentou uma comparação entre métodos exatos e técnicas de aprendizado de máquina na resolução do problema de roteamento de veículos capacitado. A análise foi efetuada pela implementação de duas abordagens distintas: a do método exato, usando um modelo elaborado sob a estrutura do resolvedor SCIP; e a do aprendizado profundo, utilizando uma Rede de Atenção em Grafos (GAT) treinada sob o paradigma do aprendizado por reforço. Os resultados indicam que, enquanto o método exato garante uma solução ótima, sua alta utilização de recursos computacionais o torna inviável para grandes instâncias por problemas de escalabilidade. Por outro lado, os modelos de aprendizado de máquina, embora forneçam soluções sub-ótimas, são mais flexíveis e capazes de lidar com instâncias maiores em um tempo significativamente inferior.

Os experimentos conduzidos mostram que os modelos GAT, treinados por técnicas de aprendizado por reforço como o algoritmo REINFORCE com uma *Greedy Rollout* baseline e o estimador DiCE, apresentam um bom desempenho na solução de instâncias do CVRP. Esses modelos geraram resultados muito próximos aos obtidos pelo resolvedor SCIP. A análise dos resultados revela que, apesar dos modelos GAT apresentarem um pequeno aumento no custo da solução (distância percorrida) obtida, a sua vantagem em relação ao tempo necessário para obtê-la os tornam alternativas viáveis para aplicações práticas.

Uma importante contribuição do trabalho é a demonstração de que o estimador DiCE pode apresentar vantagens computacionais sobre o método do *Greedy Rollout*. Como o método DiCE não faz uso de uma estrutura de duplo-ator, o seu custo computacional pode ser significativamente inferior. Dessa forma, essa técnica leva a um menor tempo de treinamento e, possivelmente, a um menor uso de memória, sem comprometer a qualidade da solução. Nos experimentos realizados, as arquiteturas DiCE e $Dice_{Mish}$ levaram aproximadamente 14 horas para um treinamento de cem épocas, enquanto que o REINFORCE com *Greedy Rollout* levou 21 horas.

Essa linha de pesquisa abre importantes caminhos para serem explorados por trabalhos futuros. Um importante desenvolvimento seria a implementação de uma estratégia warm-start, com o potencial de reduzir o tempo de computação de modelos exatos ao prover valores de soluções sub-ótimas em sua inicialização. Essa estratégia é particularmente relevante em operações de otimização em larga-escala, nas quais uma boa solução inicial pode diminuir significativamente o espaço de busca do problema. Adicionalmente, combinar técnicas de otimização tradicionais, como a programação dinâmica e o *branch and bound*, com modelos de aprendizado de máquina pode levar à criação de soluções híbridas que empreguem os pontos fortes de cada abordagem.

Outra avenida de exploração potencial seria a de alterar a arquitetura de aprendizado profundo utilizada. Uma alternativa seria a de se utilizar redes convolucionais para grafos, ou alterar o mecanismo de atenção empregado. A ideia seria de melhorar o desempenho das GAT para, em conjunto com as estruturas de aprendizado por reforço, gerar ganhos de escalabilidade, no tempo de aprendizado ou da qualidade das soluções geradas. Por outro lado, técnicas como a de *Transfer Learning* poderiam ser utilizadas para empregar o aprendizado obtido pela resolução de instâncias menores ou menos complexas em instâncias maiores ou mais complexas sem a necessidade de realizar novo treinamento.

# REFERÊNCIAS


Applegate, D. L., Bixby, R. E., Chvátal, V. e Cook, W. J. (2006). The traveling salesman problem. `http://www.math.uwaterloo.ca/tsp/concorde.html`. Accessed: October, 2023.

Asghari, M. e Mirzapour Al-e hashem, S. M. J. (2021). Green vehicle routing problem: A state-of-the-art review. *International Journal of Production Economics*, 231(C).

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R. et al. (2018). Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*.

Bello, I., Pham, H., Le, Q. V., Norouzi, M. e Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940.

Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S. J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M. E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D. e Witzig, J. (2023). Enabling research through the scip optimization suite 8.0. *ACM Trans. Math. Softw.*, 49(2).

Borcinova, Z. (2017). Two models of the capacitated vehicle routing problem. *Croatian Operational Research Review*, 8:463–469.

Caceres Cruz, J., Arias, P., Guimarans, D., Riera, D. e Juan, A. (2014). Rich vehicle routing problem: Survey. *ACM Computing Surveys*, 47:1–28.

Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. (388).

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B. e Song, L. (2017). Learning combinatorial optimization algorithms over graphs. *CoRR*, abs/1704.01665.

Dantzig, G. B. e Ramser, J. H. (1959). The truck dispatching problem. *Management Science*, 6(1):80–91.

Domingos, P. (2012). A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87.

Dorigo, M. e Stützle, T. (2004). *Ant Colony Optimization*. Bradford Company, USA.

Foerster, J., Farquhar, G., Al-Shedivat, M., Rocktäschel, T., Xing, E. P. e Whiteson, S. (2018). Dice: The infinitely differentiable monte-carlo estimator.

Glover, F. e Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA.

Goodfellow, I., Bengio, Y. e Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Google (2023). Or-tools. `https://developers.google.com/optimization`. Accessed: October, 2023.

Gori, M., Monfardini, G. e Scarselli, F. (2005). A new model for learning in graph domains. 2:729–734 vol. 2.

Han, J., Kamber, M. e Pei, J. (2012). *Data mining concepts and techniques, third edition*. Morgan Kaufmann Publishers, Waltham, Mass.

Harrison, D. (2021). A brief introduction to automatic differentiation for machine arning. *CoRR*, abs/2110.06209.

Helsgaun, K. (2000). An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130.

Holland, J. H. (1992). Genetic algorithms. *Scientific American*.

Hopfield, J. e Tank, D. (1985). Neural computation of decisions in optimization problems. *Biological cybernetics*, 52:141–52.

João Pedro Pedroso, Abdur Rais, M. K. e Muramatsu, M. (2012). Mathematical optimization. `https://scipbook.readthedocs.io/en/latest/index.html`. Accessed: October, 2023.

Karimi-Mamaghan, M., Mohammadi, M., Meyer, P., Karimi-Mamaghan, A. M. e Talbi, E.-G. (2022). Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422.

Kennedy, J. e Eberhart, R. C. (1995). Particle swarm optimization. Em *Proceedings of the IEEE International Conference on Neural Networks*, páginas 1942–1948.

Kirkpatrick, S., Gelatt, C. D. e Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kool, W., van Hoof, H. e Welling, M. (2019). Attention, learn to solve routing problems!

LeCun, Y., Bengio, Y. e Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

Lei, K., Guo, P., Wang, Y., Wu, X. e Zhao, W. (2021). Solve routing problems with a residual edge-graph attention neural network. *CoRR*, abs/2105.02730.

Lei, K., Guo, P., Wang, Y., WU, X. e ZHAO, W. (2022). Solve routing problems with a residual edge-graph attention neural network. *Neurocomputing*, 508.

Matousek, J. e Gärtner, B. (2006). *Understanding and Using Linear Programming*. Universitext. Springer Berlin Heidelberg.

Misra, D. (2020). Mish: A self regularized non-monotonic activation function.

Mladenović, N. e Hansen, P. (1997). Variable neighborhood search. *Comput. Oper. Res.*, 24(11):1097–1100.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. e Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. Em Balcan, M. F. e Weinberger, K. Q., editores, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 de *Proceedings of Machine Learning Research*, páginas 1928–1937, New York, New York, USA. PMLR.

Munari, P., Dollevoet, T. e Spliet, R. (2017). A generalized formulation for vehicle routing problems.

Murphy, K. P. (2013). *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.].

Nazari, M., Oroojlooy, A., Snyder, L. V. e Takác, M. (2018). Deep reinforcement learning for solving the vehicle routing problem. *CoRR*, abs/1802.04240.

Russell, S. e Norvig, P. (2021). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 4 edition.

Sabet, S. e Farooq, B. (2022). Green vehicle routing problem: State of the art and future directions. *IEEE Access*, 10:101622–101642.

Schulman, J., Heess, N., Weber, T. e Abbeel, P. (2016). Gradient estimation using stochastic computation graphs.

Sutskever, I., Vinyals, O. e Le, Q. V. (2014). Sequence to sequence learning with neural networks.

Sutton, R. S. e Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation*. Wiley Publishing.

Talbi, E.-G. (2021). Machine learning into metaheuristics: A survey and taxonomy. *ACM Comput. Surv.*, 54(6).

Toth, P., Vigo, D., Toth, P. e Vigo, D. (2014). *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. Society for Industrial and Applied Mathematics, USA.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. e Polosukhin, I. (2023). Attention is all you need.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. e Bengio, Y. (2018). Graph attention networks. *ArXiv*, abs/1710.10903.

Vinyals, O., Fortunato, M. e Jaitly, N. (2017). Pointer networks.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256.