UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Avaliação do uso de *Neural Large Neighborhood Search* em problemas de roteamento de veículo com coletas e entregas

Camila Stenico dos Santos

Monografia - MBA em Inteligência Artificial e Big Data

ICMC USP SÃO CARLOS

Camila Stenico dos Santos

Avaliação do uso de *Neural Large Neighborhood Search* em problemas de roteamento de veículo com coletas e entregas

Monografia apresentada ao Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, como parte dos requisitos para obtenção do título de Especialista em Inteligência Artificial e Big Data.

Área de concentração: Inteligência Artificial

Orientador: Prof. Dr. Jó Ueyama

Versão original

São Carlos

2023

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi, ICMC/USP, com os dados fornecidos pelo(a) autor(a)

S856m	Santos, Camila Stenico dos Avaliação do uso de *Neural Large Neighborhood Search* em problemas de roteamento de veículo com coletas e entregas / Camila Stenico dos Santos ; orientador Jó Ueyama. – São Carlos, 2023. 97 p. : il. (algumas color.) ; 30 cm. Monografia (MBA em Inteligência Artificial e Big Data) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2023. 1. LaTeX. 2. abnTeX. 3. Classe USPSC. 4. Editoração de texto. 5. Normalização da documentação. 6. Tese. 7. Dissertação. 8. Documentos (elaboração). 9. Documentos eletrônicos. I. Ueyama, Jó, orient. II. Título.

Camila Stenico dos Santos

Evaluation of Neural Large Neighborhood Search for Vehicle Routing Problem with Pickup and Delivery

Monograph presented to the Departamento de Ciências de Computação do Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo - ICMC/USP, as part of the requirements for obtaining the title of Specialist in Artificial Intelligence and Big Data.

Concentration area: Artificial Intelligence

Advisor: Prof. Dr. Jó Ueyama

Original version

São Carlos

2023

Este trabalho é dedicado aos meus pais,
que desde cedo me ensinaram que a maior riqueza de uma pessoa é o conhecimento

AGRADECIMENTOS

Primeiramente agradeço ao Prof. Dr. Jó Ueyama pela orientação no trabalho, e a Prof. Dra. Solange Rezende e Prof. Dra. Roseli Romero pela coordenação do curso de MBA. O MBA foi uma ótima oportunidade de resgatar conceitos da graduação e adquirir novos conhecimentos e habilidades. Também agradeço a empresa Loggi e em especial aos meus gestores Nicolas e Janaina pela oportunidade de estudar no MBA através da bolsa de incentivo ao estudo cedida pela empresa.

Gostaria de agradecer também aos meus pais por todas as oportunidades de estudo que me deram desde criança. Cheguei onde estou hoje graças ao esforço deles em garantir que eu sempre tivesse acesso ao estudo. Agradeço também a minha namorada, Veridiana, minha parceira de vida e meu grande apoio nessa jornada.

RESUMO

Santos, C. S. **Avaliação do uso de *Neural Large Neighborhood Search* em problemas de roteamento de veículo com coletas e entregas**. 2023. 97p. Monografia (MBA em Inteligência Artificial e Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2023.

O aumento das vendas por e-*commerce* por pequenas e médias empresas afeta diretamente a busca por demandas de *first-mile* na cadeia logística, ou seja, há mais itens para serem coletados dentro da malha urbana, causando maior custo de operação e aumento na complexidade do planejamento logístico. Este impacto abre espaço para soluções que buscam aumentar a eficácia do processo. O problema de roteamento de veículos, estudado desde 1960, traz como uma de suas variantes o problema com demandas mistas, onde pontos de coleta e entrega são combinados na mesma rota em busca de eficiência operacional. Nos últimos anos, o aprendizado de máquina, em especial, o aprendizado por reforço, tem sido estudado como uma metaheurística para problema de roteamento de veículo, combinado a heurísticas existentes. Em 2019, foi proposto o uso de aprendizado por reforço junto a heurística de busca em grandes vizinhanças, resultando em uma meta heurística chamada *Neural Large Neighborhood Search* (NLNS), com grande potencial de performance e resultados. Neste cenário, o presente trabalho de pesquisa tem como objetivo analisar a eficácia do NLNS para gerar rotas com demandas de coletas e entregas combinadas em três cidades do território brasileiro: Rio de Janeiro, Brasília e Belém. Os resultados indicam que, embora o NLNS tenha demonstrado potencial, seus resultados não superaram o *benchmark* utilizado, que integra técnicas de agrupamento com a ferramenta Google OR-Tools.

Palavras-chave: Aprendizado por reforço. Problema de roteamento de veículos. Aprendizado de máquina. Problema de roteamento de veículos com entrega e coleta

ABSTRACT

Santos, C. S. **Evaluation of Neural Large Neighborhood Search for Vehicle Routing Problem with Pickup and Delivery**. 2023. 97p. Monograph (MBA in Artificial Intelligence and Big Data) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2023.

The rise of e-commerce sales by small businesses and vendors led to an increase in first-mile demands, meaning that there are more items to be collected in the urban region, leading to bigger costs in operation and more complexity in logistics planning. This impact leaves space for solutions looking to enhance the effectiveness of the logistic process. The Vehicle Routing Problem, which has been studied since the 1960s, has a variant to work with mixed demands problems, where first-mile pickups and last-mile deliveries are planned together in the same route for efficiency. In recent years, machine learning, especially reinforcement learning, has been studied as a metaheuristic for vehicle routing problems, combined with existing heuristics. In 2019, the use of reinforcement learning along with large neighborhood search heuristics was proposed, resulting in a metaheuristic called Neural Large Neighborhood Search (NLNS), with great potential for performance and results. In this scenario, the present research aims to analyze the effectiveness of NLNS in generating routes with combined collection and delivery demands in three Brazilian cities: Rio de Janeiro, Brasília, and Belém. The results indicate that, although the NLNS showed potential, its outcomes did not surpass the utilized benchmark, which integrates clustering techniques with the Google OR-Tools.

Keywords: Machine Learning. Vehicle Routing Problem. Vehicle routing Problem with Pickup and Delivery. Reinforcement Learning. Neural Large Neighborhood Search

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A2C	*Advantage Actor-Critic*
A3C	*Asynchronous Advantage Actor-Critic*
CNN	*Convolutional Neural Network*
CVRP	*Capacitated Vehicle Routing Problem*
DNN	*Deep Neural Network*
DPG	*Deterministic Policy Gradients*
gb	*GigaByte*
HFVRP	*Heterogeneous Fleet Vehicle Routing Problem*
IBGE	Instituto Brasileiro de Geografia e Estatística
IPEA	Instituto de Pesquisa Econômica Aplicada
KL	*Kullback-Leibler Divergence*
km	Quilômetro
LNS	*Large Neighborhood Search*
MFVRP	*Mixed Fleet Vehicle Routing Problem*
MDP	*Markov Decision Process*
NLNS	*Neural Large Neighborhood Search*
NP	*Nondeterministic Polynomial time*
OSRM	*Open Source Routing Machine*
PME	Pequenas e médias empresas
PPO	*Proximal Policy Optimization*
RAM	*Random-access memory*
RL	*Reinforcement Learning*
SARSA	*State-Action-Reward-State-Action*
TRPO	*Trust Region Policy Optimization*

USP	Universidade de São Paulo
USPSC	Campus USP de São Carlos
VRP	*Vehicle Routing Problem*
VRPB	*Vehicle Routing Problem with Backhaul*
VRPPD	*Vehicle Routing Problem with Pickup and Delivery*
VRPTW	*Vehicle Routing Problem with Time Windows*

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização, Motivação e Lacunas

No Brasil, o comércio eletrônico apresenta um crescimento acelerado desde 2020. Entre os motivos para tal crescimento estão: população jovem com conhecimento de internet, rápida expansão ao acesso da internet e aumento do uso de *smartphone* (GONÇALVEZ, 2022). Entre os países da América Latina, o Brasil tem o maior crescimento do segmento, e até 2026, o e-*commerce* brasileiro deve dobrar de tamanho (STACHEWSKI, 2022).

Analisando o perfil de empresas que atuam com comércio eletrônico, há um incremento de microempreendedores individuais e micro e pequenas empresas (PMEs) que utilizam a internet para venda de produtos. De 2020 para 2021, houve um aumento de 15% no número de pequenos negócios online de acordo com (FERNANDES, 2021), saindo de 59% para 74%. Ainda, em 2022, as PMEs movimentaram um total de 2,7 bilhões de reais com a venda de produtos online como publicado por (REDAÇÃO, 2023).

Em adição, a Amazon Brasil apresentou seus dados sobre pequenas e médias empresas brasileiras durante o Amazon Conecta. Conforme a publicação do dia 25 de maio de 2023, disponível em (AMAZON, 2023), dos 50 mil vendedores parceiros na plataforma, 99% deles são PMEs, e 77% das vendas são destinadas a clientes em estados diferentes de onde o vendedor está localizado.

A adesão de PMEs ao comércio eletrônico traz novos desafios para a área de logística. Em 2020, (BERGMANN; WAGNER; WINKENBACH, 2020) destacou o impacto do aumento de pequenas empresas no *e-commerce*, afirmando que este cenário aumenta a demanda por serviços logísticos, e com isto, há o aumento da fragmentação nas demandas de *last-mile delivery* (etapa de entrega ao consumidor final) e *first-mile pickup* (etapa inicial, onde o pacote é coletado em sua origem). Em especial, o aumento das vendas no *e-commerce* por PMEs aumenta a fragmentação de demandas de coleta, dado que há um maior número de pequenos itens a serem coletados em vendedores locais.

Bergmann ainda afirma em seu estudo, que, devido a essa fragmentação na demanda, há um aumento nos custos da cadeia logística e aumento de complexidade no planejamento, abrindo espaço para soluções que buscam aumentar a eficácia do processo. O autor então propõe tratar entregas e coletas como parte da mesma operação, o que ao final, é concluído que há ganhos de eficiência em operações de entrega para e-*commerce* ao integrar-se *first-mile* e *last-mile*.

O planejamento de rotas com demanda mista pode ser modelado como um problema de roteamento de veículo (*Vehicle Routing Problem*), onde, dado um conjunto finito de pontos, uma solução ótima, seja ela local ou global, deve ser encontrada. O

VRP é um problema NP-*hard*, o que torna a escalabilidade da solução muito custosa computacionalmente. Ao longo dos anos, soluções exatas, heurísticas e meta heurísticas foram propostas para resolver essa classe de problemas, e com o avanço do aprendizado de máquina, o aprendizado por reforço começou a ser estudado como uma alternativa de solução, dado que aborda o problema de forma a propor uma heurística com o modelo, e não aplicar uma heurística especifica ao problema, podendo então ser uma solução que atende duas necessidades do VRP: generalista e escalável (ARDON, 2022).

Assim, o presente trabalho busca avaliar a aplicação de aprendizado por reforço (MOUSAVI; SCHUKAT; HOWLEY, 2017) para programação de rotas com demandas mista (*first-mile pickup* e *last-mile delivery*) em tempo real, respeitando-se as restrições do *Vehicle Routing Problem* (VRP) utilizado e avaliando sua eficácia em três cidades brasileiras: Rio de Janeiro, Brasília e Belém.

1.2 Objetivos geral e específicos

O objetivo geral é analisar a eficiência de rotas de demandas mistas propostas geradas a partir *Neural Large Neighborhood Search*, avaliando a eficiência de rota e redução da pegada de tráfego das rotas.

Objetivos Específicos:

- Construir um *dataset* com pontos de coleta e entrega nas cidades de Rio de Janeiro, Brasília e Belém com o *LoggiBud*.
- Modelar o problema de roteamento definindo as restrições a serem aplicadas;
- Estudar a aplicação de aprendizado por reforço para solucionar VRPs com demandas mistas.
- Implementar o aprendizado por reforço para solucionar VRPDP.
- Comparar a performance da solução proposta com o *LoggiBud*.
- Analisar a performance da solução nos quesitos de eficiência de rota e redução da pegada de tráfego.

1.3 Organização do texto

Sobre a organização desse trabalho, o Capítulo 2 contém o referencial teórico introduzindo conceitos gerais sobre *Vehicle Routing Problem*, aprendizado por reforço e *Neural Large Neighborhood Search* (NLNS). No Capítulo 3, será apresentado a metodologia proposta, abordando a modificação no *LoggiBud* e o NLNS. O Capítulo 4 abrange análise dos resultados para cada experimento realizado. O Capítulo 5 compreende a conclusão dos resultados alcançados e possibilidades futuras.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 *Vehicle Routing Problem*

Vehicle Routing Problem é o nome dado a classe de problemas de otimização discreta que buscam definir rotas para uma determinada frota de veículos, de acordo com objetivos e restrições impostas. O VRP tem sido estudado a mais de 50 anos, e novas variações do problema são apresentadas na Literatura de acordo com novas restrições baseadas em cenários encontrados no mundo real.

O VRP clássico foi apresentado em 1959 no *paper "The Truck Dispatching Problem"* (DANTZIG; RAMSER, 1959), onde é definido como:

Sendo $G = (V, E)$ um grafo ponderado, sendo seu peso definido por: $d : E \to \mathbb{R}^{\geq 0}$. O grafo é composto por um conjunto de arestas E e conjunto de vértices $V = 1, ..., n$, onde o vértice 1 representa o depósito, e os outros vértices representam cidades ou clientes.

Uma matriz $D = (d_{ij})$ é associada ao grafo G, onde d_{ij} é igual ao peso do aresta (i, j) e pode ser lido como custo de viagem.

Uma frota de veículos, estacionada no depósito, está disponível para atender os clientes e cidades, e cada veículo tem a mesma característica. Cada vértice $i > 1$ é associado a uma demanda $q_i \geq 0$, e a soma das demandas não deve exceder a capacidade do veículo. O objetivo é determinar o conjunto de rotas de menor custo que satisfaz:

- cada vértice $v \in V \setminus 1$ é visitado apenas uma vez por exatamente um veículo,
- cada rota começa e termina no depósito, isto é, em $v = 1$,
- a capacidade do veículo não é excedida.

2.1.1 Variantes

Desde a Proposta de Dantzig e Ramser em 1959, novos modelos de VRP foram propostos buscando incorporar complexidades do mundo real. Características como frota heterogênea, janela de tempo, trânsito e múltiplos depósitos foram aplicadas como restrições do VRP ou características do VRP. Também, o avanço de velocidade de processamento e capacidade de memória de computadores, tem possibilitado processar VRPs mais complexos, já que este por ser um problema *NP-hard* exige um alto custo computacional para cálculo de rota.

O VRP mais básico é o *Capacitated Vehicle Routing Problem*, onde clientes espalhados geograficamente tem demanda de um mesmo produto, e são atendidos por uma

frota de veículos iguais, que partem de um mesmo depósito. É modelado diretamente a partir do VRP clássico.

A primeira variante a ser apresentada foi a *Heterogeneous Fleet* VRP (HFVRP), também conhecida como *Mixed Fleet* VRP (MFVRP), que estendeu o VRP clássico ao assumir uma frota de veículos com capacidades diferentes.

A variante mais popular (WIDUCH, 2020) é a *Vehicle Routing Problem with Time Windows* (VRPTW), onde é assumido que a entrega deve ocorrer dentro de uma janela de tempo, que varia de cliente a cliente. O primeiro trabalho propondo uma heurística para essa variante é (BAKER; SCHAFFER, 1986). Quase 40 anos depois, o trabalho de (TAN; YEH, 2021) mostra que no período de 2019 a 2021, o VRPTW é o modelo com maior número de artigos publicados.

Uma outra variante, de interesse do trabalho, é a VRP *with Pickup and Delivery* (VRPPD), proposto em (ARCHETTI; CHRISTIANSEN; SPERANZA, 2018). No VRPPD, um mesmo veículo é responsável por entregar e coletar demandas na mesma rota, deixando as demandas coletadas no ponto de origem. No mesmo contexto, foi apresentado o VRP *with Backhaul* (VRPB), onde as demandas de entregas devem ser feitas antes das demandas de coleta.

Ainda, é possível combinar as variantes apresentadas de forma a descrever de forma mais precisa o problema encontrado no mundo real. Entre os citados acima, (WIDUCH, 2020) lista como presentes na literatura *Dynamic* VRP com *Time Windows*, *Open* VRP com Time Windows, VRPPD com *Time Windows*.

Por fim, (LAHYANI; KHEMAKHEM; SEMET, 2015) consolida o termo *Rich* VRP, agrupando sob essa taxonomia problemas que apresentam um mínimo conjunto de características físicas, estratégicas e táticas, englobando restrições de tempo, veículos, número de depósitos, tipo de operação, objetivo proposto e outras características sumarizadas em (SENATOR, 2021)

Outras variantes são descritas no trabalho de (WIDUCH, 2020), mas não são de relevância para o escopo proposto.

2.1.2 *Vehicle Routing Problem with Delivery and Pickup*

O foco deste trabalho é avaliar a aplicação de aprendizado por reforço a categoria do VRP que engloba tanto demandas de coletas quanto entregas. Essa classe é nomeada *Vehicle Routing Problem with Pickup and Delivery* (VRPPD). Como já mencionando anteriormente, essa variante foi apresentada no trabalho de (ARCHETTI; CHRISTIANSEN; SPERANZA, 2018).

Formalmente, conforme apresentado no trabalho de (GORA *et al.*, 2020), um VRPPD é definido matematicamente por um grafo direcionado $G = (V, E)$. O grafo é

formado por um conjunto de nós definido por $V = V^* \cup M$, onde V^* representa os locais de coleta e entrega, enquanto que M é o conjunto de todos os pontos de origem e destino da frota com k veículos.

Cada veículo k da frota tem capacidade dada por $Qk > 0$. Se Qk é constante para todo veículo em k, então a frota é definida como homogênea.

Os vértices E do grafo representam o caminho percorrido entre os pontos de demanda. Cada vértice $(i, j), i \neq j$ tem um custo $c_{ij}, c_{ij} \geqslant 0$, distância $d_{ij}, d_{ij} > 0$ e tempo $t_{ij}, t_{ij} > 0$ para todo i, j .

O número total de demandas é denotada por N, sendo N^+ o conjunto de todos os pontos de origem e N^- o conjunto dos pontos de destino.

Finalmente, a rota atribuída a um veículo k, que passa por $Vk \subset V$, é denominada de R^k, e é dita bem definida se:

- Rk começa em k^+ e termina em k^- ;
- A carga do veículo k não excede a capacidade máxima definida por Qk ;
- Se um veículo k visita um ponto de coleta N^+, ele deve obrigatoriamente visitar um ponto de entrega N^-, onde N^+ ocorre antes de N^-. Formalmente, se existe $i \in \mathbb{N}$ tal que $(N_i^+ \cup N_i^-) \cap V_k = N_i^+$ ou se existe $i \in \mathbb{N}$ tal que $(N_i^+ \cup N_i^-) \cap V_k = N_i^-$, então $(N_i^+ \cup N_i^-) \cap V_k = (N_i^+ \cup N_i^-)$, e a localização N_i^+ é visitada antes de N_i^-.

O conjunto $R = \{R_k : k \in K\}$ é dito uma solução para o VRPPD caso:

- Para todo k, Rk é bem definido;
- $\bigcup_{i=1}^{k} V_k = V$

O principal problema para o PDP é minimizar o tamanho de M e a soma das distâncias percorridas para um dado R, o que pode ser expresso como:

$$\min_{f(R)} = \sum_{i=1}^{K_R} x_i$$

onde K_R é o tamanho da frota e x_i é o custo da rota para o veículo i, que é, na verdade, uma função dependente do custo c_{ij}, da distância d_{ij} e do tempo de viagem t_{ij}.

A definição do PDP pode ser estendida para considerar a restrição de janelas de tempo, como no VRPTW, onde a definição de R_k também considera:

- Para cada $i \in V$, o veículo k deve chegar ao ponto i dentro de uma janela de tempo $[e_i, l_i]$.

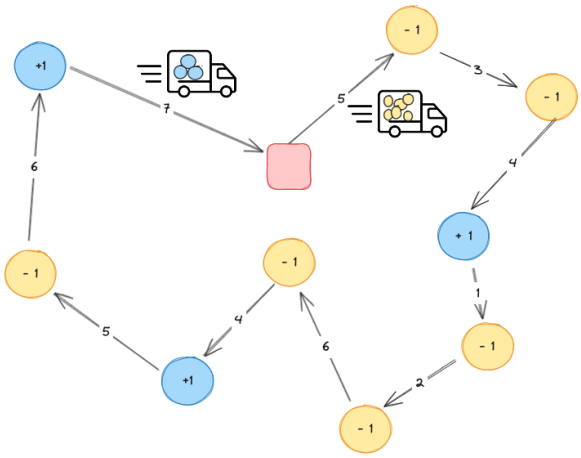


Figura 1 – Representação do problema de roteamento de veículos com coletas e entregas. Veículo sai carregado do depósito (quadrado vermelho), e entrega demandas nos pontos amarelos. Pelo caminho, coleta demandas nos pontos em azul e volta ao depósito carregado.

Esse caso é conhecido como *Vehicle Routing Problem with Pickup and Delivery and Time Windows* (VRPPDTW)

A Figura 1 ilustra o VRPPD, onde a distância entre as demandas é dada pelo peso das arestas, os nós em amarelo representam as demandas de entrega, e os nós em azul representam as demandas de coleta. O veículo, representado pelo desenho do caminhão, sai do depósito (marcado em vermelho) carregado de demandas de entrega, e retorna ao depósito carregado de demandas coletadas na rota.

2.1.3 Solucionando o *Vehicle Routing Problem*

Desde 1960, inúmeros estudos foram publicados com uma gama de possíveis soluções para o VRP. Conforme sumarizado no trabalho de (FOA *et al.*, 2022), as principais soluções para VRP são divididas em algoritmos exatos, heurísticas e meta heurísticas. Os detalhes de cada tipo de solução são apresentados a seguir:

2.1.3.1 Algoritmos exatos

Algoritmos ingênuos, onde para todo problema de tamanho finito, será possível determinar uma solução ótima. Essa classe de algoritmos é considerada ingênua por comparar todas as possíveis rotas para selecionar a com menor distância, o que torna a complexidade próxima a O(n!), ou seja, os métodos exatos são inviáveis para casos onde a instância do VRP é grande.

Detalhados por (GORA *et al.*, 2020), os algoritmos exatos para VRP podem ser classificados como: (1) Busca em árvore, (2) Programação dinâmica e (3) Programação Linear Inteira.

Apesar de introduzidos como solução do *Capacited Vehicle Routing Problem* (CVRP), também podem ser utilizados para solucionar o VRPPD.

2.1.3.2 Heurísticas

Métodos heurísticos são definidos por (BLOCHO, 2020) como uma abordagem para resolver um problema que não garante a melhor solução, porém, garante uma solução factível de acordo com o proposto pelo problema.

Ainda segundo esse mesmo trabalho, o uso de heurísticas é vantajoso por encontrar boas soluções em pouco tempo. Também, provê flexibilidade para lidar com mais restrições do VRP e supre a lacuna de não existir um método exato para um dado problema.

Ao longo do tempo, as heurísticas publicadas foram classificadas de diferentes formas por diferentes autores. Em (GORA *et al.*, 2020), dois agrupamentos acabam prevalecendo para diferentes tipos de VRP: construtivos e aperfeiçoadores.

Em métodos construtivos, a solução apresentada começa por pequenas rotas, e então pontos são adicionados ou rotas pequenas são combinadas para formar uma rota factível final, com foco em custo, sem se preocupar em aperfeiçoar as rotas sendo construídas. Aqui, destaca-se a heurística de inserção, onde a inserção de um ponto tem um custo calculado, e a busca local, onde o nó vizinho com menor custo é selecionado. Em métodos aperfeiçoadores, o foco é gerar novas rotas melhores a partir de um conjunto inicial de rotas factíveis.

Em (ZONG *et al.*, 2022), a desvantagem de heurísticas é ressaltada através da visão de que, apesar de métodos baseados em heurísticas gerarem soluções próximas de ótimas em um tempo razoável, estes ainda dependem de regras definidas manualmente, o que é limitado pelas experiências humanas. Ainda, o tempo de resposta não é satisfatório em situações de alta dinâmica.

2.1.3.3 Meta heurísticas

Enquanto que as heurísticas clássicas, descritas na Seção 2.1.3.2, tem como objetivo encontrar uma solução factível rapidamente e depois otimizá-la, as meta heurísticas, foco de pesquisa dos últimos 40 anos (GORA *et al.*, 2020), trazem métodos de alto nível, com capacidade de selecionar, achar ou gerar uma heurística, combinando diferentes algoritmos para explorar o conjunto de pontos de maneira eficiente e efetiva.

Em adição, as meta heurísticas tem maior potencial (FOA *et al.*, 2022) que heurísticas por considerarem soluções intermediárias não aprimoradoras ou até mesmo inviáveis, o que permite uma exploração maior do universo de possibilidades de rotas para a solução global ótima (GORA *et al.*, 2020).

Em (FOA *et al.*, 2022), as meta heurísticas foram classificadas em busca local,

busca populacional e mecanismos de aprendizado. Dentre os algoritmos mais utilizados, cita-se algoritmos genéticos e algoritmos naturais (*Particle Swarm* e *Ant Colony*). O uso de *Reinforcement Learning* é classificado como a última categoria apresentada.

2.2 *Reinforcement Learning*

A área de aprendizado de máquina é classificada em três grandes paradigmas: o aprendizado supervisionado, o aprendizado não supervisionado e o aprendizado por reforço. O aprendizado por reforço, ou *Reinforcement Learning* (RL) é caracterizado por não requerer dados rotulados, pois seu aprendizado é construído através de interação com ambiente. Chega a ser descrito em (FOA *et al.*, 2022) como o paradigma de aprendizado de máquina que mais se compara ao aprendizado humano, devido ao fato que o conhecimento, nesse caso, é adquirido através de tentativa e erro.

O paradigma foi apresentado pela primeira vez em 2015 no trabalho de (LECUN; BENGIO; HINTON, 2015). Nas palavras de (MOUSAVI; SCHUKAT; HOWLEY, 2018), métodos de aprendizados baseados em RL se mostraram promissores em áreas como robótica e alocação de recursos, e devido a característica de aprender de forma interativa, RL se tornou um dos principais candidatos de atingir os objetivos de inteligência artificial, onde agentes autônomos aprendem ambientes complexos e incertos.

A interação com o ambiente é protagonizada por um agente. A posição do agente é determinada por um estado. Dado um estado, o agente pode performar uma série de ações que causam mudança no estado. Por fim, a combinação do estado e de ação resultam em uma recompensa ao agente. Se a ação tomada foi boa, o agente terá uma recompensa positiva, enquanto que se a ação for ruim, o agente terá uma recompensa negativa. Assim, no aprendizado por reforço, o objetivo é encontrar um conjunto de ações a serem executadas pelo agente de forma a maximizar sua recompensa. Ou, como descrito por (MOUSAVI; SCHUKAT; HOWLEY, 2018), o agente tenta aprender uma sequência ótima de ações para realizar observando o resultado das ações que ele está executando no ambiente.

Como demonstrado por (MOUSAVI; SCHUKAT; HOWLEY, 2018) e (FOA *et al.*, 2022), o aprendizado por reforço pode ser modelado como *Markov Decision Process* (MDP). De acordo com (LITTMAN, 2001), MDPs modelam a tomada de decisão em contextos discretos, estocásticos e sequenciais. O MDP, de forma análoga ao apresentado do RL, é composto por um agente tomador de decisão incorporado ao ambiente. O estado do ambiente é alterado aleatoriamente de acordo com uma ação tomada pelo agente, e a mudança de estado afeta a recompensa. Para solucionar um MDP, deve-se encontrar uma política π^* que maximiza o total esperado da recompensa.

Para a formalização de um problema de RL como MDP, o modelo para o ambiente deve estar presente, ou seja, as regras que definem a dinâmica dos estados e as combinações

entre estado, ação e recompensa são conhecidas. Ainda, se os estados e os espaços de ações são finitos, o problema é um MDP finito. Mousavi *et al.,* 2018 destaca que MDPs finitos são amplamente usados na literatura sobre aplicações desse paradigma. Neste trabalho, o problema será modelado como um MDP finito.

2.2.1 Definição formal

Um MDP finito é definido por:

- Um período de tempo por t, dado por T+1 passos necessários para resolver uma instância, onde $t = 0, 1, ..., T$.
- Um conjunto finito de ações $A = 1, ..., k$, onde uma ação em t é definida por a_t.
- Um conjunto de estados finito S, onde um estado em um passo t é descrito por s_t.

Tem-se que, a cada passo t, o agente seleciona uma ação a_t no estado s_t. Cada ação tomada resulta em uma recompensa escalar r_t, com $r_t \in \mathbb{R}$. Então após cada ação a_t, o agente recebe uma recompensa r_t e observa o estado s_{t+1}.

A probabilidade de cada próximo estado possível s_{t+1} é dada por uma distribuição de transição dada por:

$$P(s_{t+1}|s_t, a_t), s_{t+1}, s_t \in S, a_t, \in A(s_t)$$

E a probabilidade de cada possível recompensa r_t é dada por:

$$P(r_t|s_t, a_t), s_t \in S, a_t, \in A(s_t)$$

Então, a recompensa esperada r_t, a ser recebida ao executar a em s é dada por:

$$E_{p(}r_t|s_t,a\,t)(r_t|s_t = s, a_t = a)$$

A probabilidade do agente tomar uma ação a_t em um estado s_t é denominada como uma política $\pi(a_t|s_t)$. O objetivo do agente pode ser dito como aprender uma política π ótima de tal forma que a recompensa final seja maximizada. A recompensa esperada ao final de t é dada por:

$$R_t = E_\pi \left[\sum_{t=1}^{T} \gamma^t r_t \right] \tag{2.1}$$

E_π é a expectativa da recompensa descontada de acordo com π, e $\gamma \in (0, 1)$ é o fator desconto para recompensas futuras. O objetivo é maximizar essa função, maximizando-se a recompensa final.

Em adição a função objetiva, há funções auxiliares que modelam as probabilidades de transições e a recompensa imediata, ou seja, quantificam as consequência das ações tomadas. No trabalho de (FOA *et al.*, 2022), as funções são definidas por função estado-valor $V\pi(s_t)$, função ação-valor $Q\pi(s_t, a_t)$e função vantagem $A(s_t, a_t)$.

A função estado-valor representa a bondade do estado s_t, de acordo com a recompensa esperada e de acordo com a política π. É dada por:

$$V_\pi(s_t) = E_\pi \left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} | S_t = s_t \right] \tag{2.2}$$

A função ação-valor representa a qualidade de tomar uma ação a_t em um estado s_t, onde a qualidade é representada pelo retorno esperado atingível a partir do estado s_t de acordo com a política π (também denominado *Q-value*). É dada por:

$$Q\pi(s_t, a_t) = E_\pi \left[\sum_{k=0}^{T-t} \gamma^k r_{t+k} | S_t = s_t, A_t = a_t \right] \tag{2.3}$$

Finalmente, a função vantagem pode metrificar se uma ação a_t pode ser boa ou ruim em um estado s_t a partir das funções acima. É dada por:

$$A(s_t, a_t) = Q\pi(s_t, a_t) - V_\pi(s_t) \tag{2.4}$$

2.2.2 Algoritmos para RL

Inúmeros algoritmos foram propostos na literatura para resolver o aprendizado por reforço. Para escolher o melhor algoritmo a ser usado, é necessário compreender com quais características do RL aquele algoritmo se relaciona. No trabalho de (ALMAHAMID; GROLINGER, 2021), são apresentados os principais algoritmos dado os diferentes tipos de ambientes do RL sendo utilizado. Nas palavras do autor, há diferentes algoritmos para diferentes ambientes.

Os algoritmos podem ser classificados em três grupos dado o ambiente: 1) Número limitado de estados e ações discretas, 2) Número ilimitado de estados e ações discretas e 3) Número ilimitado de estados e ações contínuas. Outras classificações são usadas, e alguma delas serão introduzidas em conjunto aos algoritmos como forma de destacar a diferença entre eles.

No primeiro caso, os ambientes são simples, com número limitado de estados e ações discretas, e todos os estados e ações são conhecidos, como por exemplo, em um jogo de tabuleiro. Para esse contexto, o algoritmo de *Q-learning* é comumente usado (ALMAHAMID; GROLINGER, 2021). Outra solução conhecida é *State-Action-Reward-State-Action* (SARSA). Ambas soluções são formuladas a partir da do Aprendizado por Diferença Temporal (*Temporal-Difference Learning*), uma técnica que combina elementos

do método de Monte Carlo e do método de *Bellman* para atualizar estimativas de valores em um ambiente dinâmico (WINDER, 2021).

No *Q-learning*, uma tabela nomeada de *Q-table*, que contém todo par de valor ação-estado é usada para manter o valor esperado de cada par de ação-estado. Esse valor é a função *Q-value*.

Através de um número de episódios, que são uma sequência de estados realizados até o sucesso ou falha do agente, a tabela *Q-table* é atualizada com o retorno esperado da ação a tomada no estado s. Após esse treinamento da *Q-table*, o *Q-learning* atualiza a política *pi* de acordo com a ação a ser tomada que maximiza a recompensa esperada do estado de acordo com a *Q-table*. Pela forma que o algoritmo é implementado, e pelas bases que possui no TD, o agente é possibilitado de criar políticas ótimas locais e ótimas no futuro (WINDER, 2021). A Equação 2.5 ilustra a atualização do *Q-value* usada a cada iteração.

$$Q\pi(s_t, a_t) \leftarrow Q\pi(s_t, a_t) + \alpha(r + \gamma maxQ(s'_t, a_t) - Q(s_t, a_t)) \tag{2.5}$$

O algoritmo de SARSA, de forma similar ao *Q-learning*, busca potencializar a função ação-valor, mas de forma mais generalizada ao deixar de olhar para o máximo de cada estado de uma forma gulosa (ALMAHAMID; GROLINGER, 2021), e passa a olhar para o retorno esperado a partir da aplicação da ação escolhida pela política *pi*, comumente *epsilon-greedy*, usada para equilibrar a exploração e o aproveitamento do conhecimento atual (*exploitation versus exploration*. O fator de escolher a ação a_{n+1} usando a política *pi* antes de atualizar a função ação-valor torna o SARSA um algoritmo *on-policy*, enquanto que o *Q-learning* é caracterizado por ser *off-policy* por atualizar *pi* de acordo com a ação tomada (WINDER, 2021). A diferença da atualização do *Q-value* se torna evidente ao comparar a Equação 2.5 com a equação que caracteriza o SARSA, dada pela Equação (2.6).

$$Q\pi(s_t, a_t) \leftarrow Q\pi(s_t, a_t) + \alpha(r + \gamma Q(s'_t, a_t) - Q(s_t, a_t)) \tag{2.6}$$

Em ambas equações, o α representa a taxa de aprendizado do algoritmo, com valor entre (0, 1).

Nem todo problema pode ser modelado com um ambiente contendo um número limitado de estados, e em muitos cenários complexos, o número de estados é virtualmente infinito, como por exemplo, em jogos complexos ou tarefas de robótica. Nesses ambientes, os algoritmos apresentados não escalam bem devido ao grande número de estados, pois a cada iteração, cada par de estado-ação teria seu *Q-value* calculado em memória. Para esses ambientes com número ilimitado de estados e ações discretas, propõe-se o uso dos algoritmos *Deep Q-Networks* e *Deep SARSA*.

A proposta desses algoritmos é que, ao invés de ter uma tabela com os valores do par ação-valor, uma função de aproximação é usada para calcular o valor esperado de retorno. A cada passo, o agente observa o par atual de ação-estado e prediz o valor esperado, tornando o aprendizado um problema de regressão (WINDER, 2021).

Problemas complexos extrapolam uma modelagem linear, se valendo de Aprendizado Profundo (*Deep Learning*) (DL) para o agente. Segundo Winder, o modelo mais usado para esse problema é uma rede neural artificial profunda (*deep artificial neural network*) ou DNN, e AlMahamid reforça o uso de DNN em seu artigo, com o complemento de que o agente comumente é uma *Convolutional Neural Network* (CNN). Winder ainda acrescenta que o uso de *Deep Learning* permite traduzir observações brutas, de qualquer tipo, em ações, potencializando o aprendizado por reforço.

O uso de DL para RL tem certa desvantagem com relação ao aprendizado de máquina. No aprendizado por reforço, o agente demora a receber *feedback*, o que se propaga para demora em sua otimização. Isso motivou a proposta de novos algoritmos para lidar com o aprendizado do DL em tempo razoável, como o *Deep Q-Learning* (WINDER, 2021).

Deep Q-learning ou DQN surgiu da evolução de duas ideias formuladas para o problema de convergência de DL em RL: replay de experiências e clone de *Q-network* (WINDER, 2021). No *replay* de experiência é proposto um *buffer* para manter observações, ações recompensas e novas observações, o que permite usar dados antigos para treinar o modelo. Como evolução dessa proposta, surge a ideia de manter duas camadas de *Q-network* na rede neural: uma camada seleciona ações, e outra camada, *target network*, continua aprendendo. Após certo número de iterações, a camada de ações copia os pesos da camada de aprendizado, melhorando o modelo de forma rápida. Essas ideias se juntam na DQN, de forma que a combinação das camadas da rede neural produzam *Q-values* para cada ação no estado atual. Finalmente, a ação com o melhor *Q-value* é escolhida. Uma desvantagem da DQN é superestimar o *Q-value*, o que pode levar a escolher de uma ação que na verdade não é tão boa.

Variações da DQN foram propostas: *Double* DQN e *Dueling* DQN. *Double* DQN utiliza a mesma ideia de clone de *Q-network*, onde mantém uma camada de política e uma cópia, a camada *target*, que produz o *Q-value* estimado. Após um tempo, a camada de política copia os pesos da camada de *Q-value*, também com o objetivo de ter um aprendizado mais rápido, e também como forma de contornar o problema de superestimar o *Q-value*. Em *Dueling* DQN, a função de *Q-value* é quebrada em duas funções: estado-valor $V(s)$ e vantagem-valor $A(s, a)$, que medem o quão bom é para o agente estar no estado s, e quão bom é aquela ação comparada a outras ações do estado respectivamente (ALMAHAMID; GROLINGER, 2021)

Deep SARSA, espelhando as diferenças entre *Q-learning* e SARSA, tem uma rede

neural muito semelhante ao *Deep Q-learning*, porém, computa o *Q-value* performando a próxima ação a'.

A classificação dos algoritmos apresentados quanto a política é a mesma que seus similares para estados finitos. São também considerados algoritmos *critic-only*, pois a função estado-valor é aproximada e a política não tem uma função explicita para sua aproximação (FOA *et al.*, 2022).

Apesar de cenários complexos poderem ser representados com ações discretas e estados ilimitados, algumas ações como virar o volante de um carro ou movimentar o personagem de um *videogame* em mundo aberto não são mapeadas por ações discretas como movimento direcional para direita, e sim, por ações contínuas, que representam a quantidade de movimento em diferentes direções para execução da ação, como definido por (ALMAHAMID; GROLINGER, 2021).

Neste contexto, calcular a função ação-valor para cada ação em cada estado é considerado impossível, o que exige que o agente aprenda uma política parametrizada π_θ para maximizar a soma esperada das recompensas (denominada G). Isso implica que, ao contrário dos métodos em que a política π é definida de maneira indireta, cenários com ações contínuas requerem a busca direta pela política ótima.

A utilização do aprendizado por política direta é destacado como vantajoso para o dilema de *exploitation versus exploration* do agente do RL (ALMAHAMID; GROLINGER, 2021). O dilema em questão envolve duas abordagens que o agente pode ter ao escolher sua ação: *exploration* define o comportamento de escolher uma ação que leva a novas descobertas sobre o ambiente, enquanto *exploitation* define a aplicação do conhecimento já obtido. Uma política determinística sempre irá escolher a ação já conhecida como a que maximiza a recompensa esperada, contudo, essa política não leva em consideração observações estocásticas do ambiente, reduzindo a exploração do agente. Já políticas estocásticas dão liberdade ao agente escolher baseado em probabilidades arbitrárias, removendo a necessidade de estratégias que forçam a exploração, como o *e-greedy*. Por fim, políticas estocásticas podem resultar em políticas mais robustas, pois permitem ações probabilísticas, enquanto políticas definidas pela função estado-valor podem ser sensíveis a pequenas alterações no *Q-value*.

Voltando para a política parametrizada, é necessário estabelecer um caminho para achar a política ótima, que maximiza o retorno esperado G (Equação 2.1). Para isso, é utilizado-se modelos parametrizáveis. Para quantificar o desempenho de um modelo, tem-se:

$$J(\theta) \doteq \mathbb{E}_\pi G \mid s, a] \tag{2.7}$$

Segundo (WINDER, 2021), a Equação 2.7 representa a função objetivo, ou a

expectativa da recompensa total dado uma política π, onde $J(\theta)$ é a função objetivo, θ é o conjunto de parâmetros da política, π é a política e G é a recompensa total esperada dado um estado e uma ação.

Dado a função objetivo, o próximo passo é encontrar a parametrização de θ através de algoritmos de gradiente descendente. O valor de θ é atualizado conforme:

$$\theta = \theta + \alpha \nabla J(\theta) \tag{2.8}$$

onde α é a taxa de aprendizado, θ são os parâmetros da política, e $\nabla J(\theta)$ é o gradiente da função objetivo (ALMAHAMID; GROLINGER, 2021).

(WINDER, 2021) demonstra a derivação do gradiente considerando as Equações 2.7 e 2.8. O objetivo da derivação é encontrar o gradiente da função objetiva. Como resultado da derivação, é apresentado:

$$\nabla J(\theta) \doteq \propto \mathbb{E}_{\pi} \quad G \nabla \ln \pi(a \mid s)] \tag{2.9}$$

O autor sumariza a equação apresentada como o gradiente do retorno em relação à política é igual ao valor esperado do retorno multiplicado pelo gradiente da política. Em outras palavras, o retorno esperado amplifica o gradiente da política.

O teorema apresentado pela Equação 2.10 é considerado a base fundamental de diferente algoritmos de Política de Gradiente (*Policy Gradient* - PG) como *REINFORCE*, *Actor-Critic*, *Trust Region Policy Optimization* e *Proximal Policy Optimization* (ALMAHAMID; GROLINGER, 2021).

REINFORCE é um algoritmo de gradiente de política de Monte Carlo. Seu nome vem do acrônimo *REward Increment = Nonnegative Factor × Offset Reinforcement × Characteristic Eligibility* (ALMAHAMID; GROLINGER, 2021). O algoritmo depende de coletar uma trajetória (uma sequência de estados, ações e recompensas que um agente experimenta enquanto interage com um ambiente) por um episódio completo para então poder ter atualizar θ através da Equação 2.8.

O algoritmo, por usar Monte Carlo, tem alta variância e por consequência, um aprendizado mais lento. Ainda, (WINDER, 2021) complementa que REINFORCE vai continuar a aumentar a probabilidade de escolher uma ação proporcional ao retorno calculado, mesmo quando o agente fez a escolha correta. A longo prazo, pode fazer com que uma ação específica seja favorecida em relação às outras pois mesmo que todas as ações tenham um retorno esperado igual, o REINFORCE continuará a aumentar a probabilidade da última ação escolhida, criando um desequilíbrio nas probabilidades de escolha de ação e afetando o desenpenho do aprendizado. A adição de uma linha de base ao REINFORCE

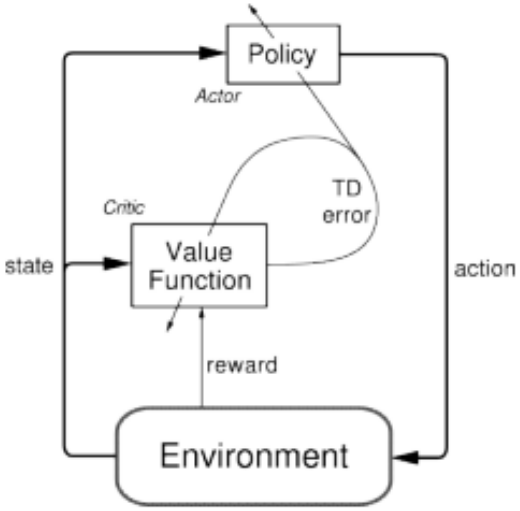


Figura 2 – Representação do Ator Crítico. Fonte: (FOA *et al.*, 2022)

reduz a variância e acelera o aprendizado, mantendo o viés inalterado por subtrair o valor da linha de base do retorno esperado G_t (ALMAHAMID; GROLINGER, 2021).

Após um período de aprendizado, a variação de gradiente apresentada pelo REINFORCE ainda é alta, enquanto que algoritmos com uma linha de base tem uma estabilização na variação. Essa estabilização é um objetivo estudado na literatura, onde busca-se aprimorar a redução, ou em outras palavras, melhorar a estabilidade dos gradiente sem prejudicar a performance. Nessa direção, (WINDER, 2021) introduz os algoritmos *Actor-Critic*.

Essa classe de algoritmos é constituída de dois componentes: Um ator responsável por ajustar o parâmetro θ da política π_θ, e um crítico, que utiliza um vetor parametrizado w para estimar a função valor $Q^w(s_t, a_t) \approx Q^\pi(s_t, a_t)$ através de um algoritmo de avaliação de política como Aprendizado por Diferença Temporal, citado anteriormente (ALMAHAMID; GROLINGER, 2021). Entre os principais algortimos dessa classe, tem-se *Deterministic Policy Gradients* (DPG), *Advantage Actor-Critic* (A2C e A3C).

O algoritmo A2C - *Advantage Actor Critic* tem o crítico treinado para prever $V(s)$, de forma que a função-valor possa ser usada para estimar a função de vantagem $A(s, a) = Q(s, a) - V(s)$ para sua inicialização. O Ator é treinado usando a função de vantagem como guia para atualizar sua política. Sua evolução, A3C - Asynchronous Advantage Actor Critic, propõe uma implementação paralela e assíncrona com várias threads. Múltiplos agentes são treinados em paralelo em seus próprios ambientes, explorando diferentes partes dos espaços de estado simultaneamente. Os agentes calculam gradientes de política e periodicamente enviam atualizações para uma rede global ou quando um estado terminal é alcançado. A rede global, então, propaga novos pesos para os agentes em cada atualização para garantir que eles compartilhem uma política comum.

Comparando-se os algoritmos de REINFORCE, *Actor-Critic* e *Advantage Actor-*

Critic, (WINDER, 2021) traz que a diferenciação entre eles é como o retorno é quantificado. Esta interpretação sumariza a distinção sutil entre eles. As equações apresentadas abaixo reproduzem a comparação feita pelo autor (considerando algoritmos apresentados e outras variações mencionadas no livro), ilustrando a ideia principal de que o gradiente da política é igual à expectativa do gradiente dos parâmetros multiplicado pelo retorno, e esse retorno é quantificado de diferente maneiras.

$$
\begin{aligned}
\nabla_\theta J(\theta) &\doteq \mathbb{E}_{\pi_\theta} \nabla_\theta \ln \pi_\theta(a \mid s) G && \text{(REINFORCE)} \\
&\doteq \mathbb{E}_{\pi_\theta} \nabla_\theta \ln \pi_\theta(a \mid s) \left(G - V_w(s)\right] && \text{(REINFORCE with Baseline)} \\
&\doteq \mathbb{E}_{\pi_\theta} \nabla_\theta \ln \pi_\theta(a \mid s) Q_w(s, a) && \text{(Actor-Critic)} \\
&\doteq \mathbb{E}_{\pi_\theta} \nabla_\theta \ln \pi_\theta(a \mid s) A_w(s, a) && \text{(Advantage Actor-Critic)} \\
&\doteq \mathbb{E}_{\pi_\theta} \nabla_\theta \ln \pi_\theta(a \mid s) \delta && \text{(Temporal-Difference Actor-Critic)} \\
&\doteq \mathbb{E}_{\pi_\theta} \nabla_\theta \ln \pi_\theta(a \mid s) \delta(\lambda) && \text{(Eligibility Actor-Critic)}
\end{aligned}
$$

Seguindo pelos algoritmos da categoria *Actor-Critic*, destaca-se gradientes de política determinísticos (DPG). Os algoritmos apresentados anteriormente sempre atuam com uma política π estocástica, porém, um grande problema em todos os algoritmos de gradiente de política é que o agente deve amostrar tanto o espaço de estados quanto o espaço de ações para então ter certeza de que encontrou uma política ótima. Para escapar desse cenário, é proposto ter a política como uma distribuição de probabilidade que seleciona estocasticamente uma ação em um estado (WINDER, 2021).

Em suma, a proposta do DPG se baseia em mover a política de comportamento na direção da maior estimativa de valor da ação, em vez de maximizá-la diretamente. Portanto, a ação ótima se torna um cálculo simples usando a política de comportamento atual. Dado a mesma parametrização, ela é determinística. Destaca-se que essa classe de algoritmos não possue mecanismos de exploração natural. O DPG é representado por:

$$
\begin{aligned}
\nabla_\theta J_\beta\left(\mu_\theta\right) &= \mathbb{E}_{s \sim \rho_\theta} \nabla_\theta Q_\mu\left(s, \mu_\theta(s)\right)] \\
&= \mathbb{E}_{s \sim \rho_\beta} \nabla_\theta \mu_\theta(s) \nabla_a Q_\mu(s, a)\Big]
\end{aligned}
\tag{2.10}
$$

Onde μ representa a política determinística.

Por fim, tem-se como algoritmo de Política de Gradiente (PG) o *Trust Region Policy Optimization* (TRPO). Este algoritmo foi proposto considerando um problema do PG: Dado a Equação 2.8, qual o α a ser escolhido? Em outras palavras, qual o tamanho do passo a ser tomado? Um passo muito grande pode levar a uma política ruim, e um passo pequeno pode levar a dados insuficientes para atualizar a política (WANG, 2020). Pra lidar com esse problema, os métodos de região de confiança (*trust region methods*) especificam uma região no espaço de parâmetros onde os passos de gradiente podem ser

confiáveis, permitindo ajustar dinamicamente o tamanho do passo com base na confiança nos gradientes, usando medidas como a Divergência de Kullback-Leibler para medir a diferença entre duas políticas e determinar quanto a política pode ser atualizada.

A Divergência Kullbak-Leibler (KL) mede o quanto uma distribuição diverge de outra distribuição. Se as distribuições são iguais, a divergência é zero, mas se as distribuições são muito diferentes, a divergência é alta. O cálculo da divergência é dado pela Equação 2.11.

$$D_{KL}(P\|Q) \doteq \sum_{x\in X} P(x)\log\frac{P(x)}{Q(x)} \tag{2.11}$$

A partir do TRPO, extende-se o algoritmo *Proximal Policy Optimization* (PPO), considerado o estado da arte em Aprendizado por Reforço. O algoritmo, introduzido pela OpenAI em 2017, parece encontrar o equilíbrio certo entre desempenho e compreensão (HEESWIJK, 2023). Enquanto no TRPO é colocado-se uma limitação (região de confiança), no PPO a restrição é colocada como uma penalidade ao algoritmo, e o objetivo é limitado para garantir que a otimização seja realizada dentro da faixa predefinida (ALMAHAMID; GROLINGER, 2021).

Antes de detalhar o PPO, é necessário aprofundar a teoria do TRPO. Em (HEESWIJK, 2023), é feito o desenvolvimento matemático do TRPO até o PPO, começando com a Equação 2.11, apresentada na forma genérica da divergência. Aplicada a política, a função é tida como:

$$\mathcal{D}_{\mathrm{KL}}\left(\pi_\theta\|\pi_{\theta+\Delta\theta}\right) = \sum_{x\in\mathcal{X}} \pi_\theta(x)\log\left(\frac{\pi_\theta(x)}{\pi_{\theta+\Delta\theta}(x)}\right) \tag{2.12}$$

A Equação 2.12 quantifica a distância entre uma política antes e depois de uma atualização do parâmetro. Heeswijk et al. também traz a formalização da restrição da divergência da atualização não ser maior que ϵ, que mostra a seguinte equação para atualizar θ:

$$\Delta\theta^* = \underset{\mathcal{D}_{\mathrm{KL}}(\pi_\theta\|\pi_{\theta+\Delta\theta})\leq\epsilon}{\arg\max} J(\theta+\Delta\theta) \tag{2.13}$$

Então, o autor introduz a vantagem substituta $\mathcal{L}$, a qual reflete a vantagem esperada de uma atualização de parâmetro. A Equação 2.14 define a vantagem esperada da política atualizada $\pi_\theta + \nabla\theta$, usando amostras da política π_θ.

$$J\left(\pi_{\theta+\Delta\theta}\right) - J\left(\pi_\theta\right) \approx \mathbb{E}_{s\sim\rho_{\pi_\theta}} \frac{\pi_{\theta+\Delta\theta}(a\mid s)}{\pi_\theta(a\mid s)} A^{\pi_\theta}(s,a) = \mathcal{L}_{\pi_\theta}\left(\pi_{\theta+\Delta\theta}\right) \tag{2.14}$$

E também introduz a Equação 2.15, que demonstra a desigualade que descreve a mudança mínima no valor do objetivo após a atualização. O lado direito denota o limite inferior, considerando a vantagem substituta corrigida pelo erro de divergência.

$$J\left(\pi_{\theta+\Delta\theta}\right) - J\left(\pi_{\theta}\right) \geq \mathcal{L}_{\pi_{\theta}}\left(\pi_{\theta+\Delta\theta}\right) - C\mathcal{D}_{KL}^{\max}\left(\pi_{\theta} \| \pi_{\theta+\Delta\theta}\right) \tag{2.15}$$

Com este contexto, a penalidade do PPO, assim como a limitação do objetivo podem ser melhor descritos. Conforme apresentado, a divergência KL é usada como restrição a atualização de α, mas não é usada como penalidade. O algoritmo de PPO se vale da penalização para prevenir grandes mudanças na política. Winder et. al. apresenta a ideia da penalização utilizando a Equação 2.13. É proposto a adição de um termo negativo dentro da função de perda que diminui arbitrariamente o valor da vantagem sempre que as políticas mudarem muito. Esse termo é denonimado de β e a ideia apresentada é formalizada na seguinte equação:

$$\Delta\theta^{*} = \arg\max_{\Delta\theta} \mathcal{L}_{\theta+\Delta\theta}(\theta + \Delta\theta) - \beta\left(\mathcal{D}_{\mathrm{KL}}\left(\pi_{\theta} \| \pi_{\theta+\Delta\theta}\right)\right) \tag{2.16}$$

O problema da restrição é que é muito difícil determinar um único valor para β que funcione para várias configurações de problemas, ou até mesmo para o mesmo problema. O PPO acaba propondo uma solução pragmática ao definir uma 'divergência alvo' σ. A divergência alvo deve ser grande o suficiente para alterar substancialmente a política, mas pequena o suficiente para que as atualizações sejam estáveis (HEESWIJK, 2023).

A variação de PPO apresentada acima é chamada de *Penalty* PPO. A variante atualmente em uso e que é geralmente referenciada quando 'PPO' é citado de forma abstrata é a *Clipped* PPO (HEESWIJK, 2023).

O objetivo desta variante é restringir espaço no qual a política pode ser alterada, de forma a superar o problema de atualizar a penalidade com o avançar do tempo. A função objetivo ainda depende da vantagem substituta $\mathcal{L}$, porém, passa a ser formalizada por:

$$\mathcal{L}_{\pi_{\theta}}^{CLIP}\left(\pi_{\theta_k}\right) = \mathbb{E}_{\tau\sim\pi_{\theta}}\left[\sum_{t=0}^{T}\left[\min\left(\rho_t\left(\pi_{\theta}, \pi_{\theta_k}\right) A_t^{\pi_{\theta_k}}, \operatorname{clip}\left(\rho_t\left(\pi_{\theta}, \pi_{\theta_k}\right), 1-\epsilon, 1+\epsilon\right) A_t^{\pi_{\theta_k}}\right)\right]\right] \tag{2.17}$$

A limitação pode vir da razão de amostragem de importância como uma estimativa do tamanho da mudança na política. Valores que não estão próximos de 1 indicam que o agente deseja fazer uma grande mudança na política. O corte desse valor é o demonstrado pela Equação 2.17 (WINDER, 2021). A taxa de amostragem de importância é dada por 2.18.

$$p_t(\theta) = \frac{\pi\theta(a_t|st)}{\pi\theta_k(at|st)} \tag{2.18}$$

Ainda aprofundando no que a Equação 2.17 traz, destaca-se que os termos $(1-\epsilon)\dot{A}$ e$(1+\epsilon)\dot{A}$ não dependem de θ, resultando em um gradiente igual a 0. Como resultado, as amostras fora da região de confiança são efetivamente descartadas, desencorajando atualizações excessivamente grandes (HEESWIJK, 2023).

2.3 Aprendizado não Supervisionado - *K-Means*

Como dito anteriormente, a área de aprendizado de máquina é classificada em três grandes paradigmas, entre eles, há o aprendizado não supervisionado. No aprendizado não supervisionado, é dado ao modelo um conjunto de dados não rotulado, e o modelo irá aprender sem supervisão humana, ou seja, tem permissão de encontrar padrões e ter percepções sem instruções ou orientação explicita (WHAT...,).

Algoritmos de aprendizado não supervisionado são adequados para organizar dados não rotulados em *clusters* por identificarem padrões úteis a categorização dos dados. Porém, *clustering* é só um dos tipos deste aprendizado. Ele ainda pode ser tipificado como regra de associação ou redução da dimensionalidade (WHAT...,).

O aprendizado não supervisionado do tipo *clustering* pode ser realizado de diferentes maneiras, sendo comumente dividido em *Partitional Clustering*, *Hierarchical Clustering* e *Density-based Clustering* (KEVIN,).

No *Partitional Clustering*, os dados são divididos em grupos, sem possibilidade de um dado pertencer a mais de um grupo, e todo grupo deve conter um dado. Para esta técnica, é necessário especificar o número de clusters, sendo este número denominado k.

No *Hierarchical Clustering*, os dados são divididos através de uma hierarquia, construída de forma *bottom-up*, onde os dados são agrupados com o dado mais similar a eles até resultar em m único *cluster*, ou *top-down*, onde os dados são separados em *clusters* até restar um dado por *cluster*. Estes métodos resultam em uma hierarquia do tipo árvore chamada de dendrograma, e um número pré definido k determina o total de *clusters* final.

Por fim, no *Density-Based Clustering* os *clusters* são determinados pela densidade dos dados em uma região. Ao contrários dos outros métodos, um número de *cluesters* k não é determinado, mas o valor do limiar de proximidade dos pontos para determinar a fronteira do *cluster* deve ser pré estabelecido.

O algoritmo de *K-Means* (AHMED; SERAJ; ISLAM, 2020) é uma técnica de aprendizado de máquina não supervisionado do tipo *Partitional Clustering*. É um algoritmo simples, que requer poucos passos para sua aplicação.

O primeiro passo para executar o *K-Means* é determinar o número k de *clusters*. A partir de k, serão inicializados de forma aleatória k centroides, que representam o centro do *cluster*. Depois dessa inicialização, aplica-se um processo de duas etapas chamado de *expectation-maximization*, onde a etapa de expectativa atribui cada dado ao centroide mais próximo, e na etapa de maximização, novos centroides são calculados utilizando-se a média de cada cluster. Os passos são repetidos até que os centroides não sejam mais alterados.

A inicialização aleatória faz com que o *K-Means* seja um algoritmo não determinístico, onde a atribuição dos dados ao clusters varia a cada execução, mesmo mantendo-se o mesmo k (KEVIN,).

Algorithm 1 Algoritmo *k-means*

1: Escolher o número k de *clusters*.
2: Inicializar de forma aleatória k centroides.
3: **repeat**
4: **expectation**: Atribuir cada ponto ao centroide mais próximo.
5: **maximization**: Calcular o novo centroide de cada cluster pela média das distâncias.
6: **until** A posição dos centroides não alterar.

2.4 *Large Neighborhood Search*

O algoritmo *Large Neighborhood Search* (LNS), proposto por (SHAW, 1998), é considerado uma meta heurística pertencente a classe de heurísticas conhecida como *Very Large Scale Neighborhood search* (VLSN), que se baseia na ideia de que a exploração de grandes vizinhanças pode levar a soluções locais ótimas (PISINGER; ROPKE, 2019).

Antes de aprofundar na LNS, é necessário compreender o que é a busca na vizinhança. Dado uma instância I de um problema de otimização combinatória, onde X é o conjunto de soluções factíveis para a instância e $c : X \to \mathbb{R}$ é uma função que mapeia uma solução para o seu custo, e assumindo que o problema de otimização combinatória seja um problema de minimização, ou seja, o objetivo é encontrar uma solução x^* tal que $c(x^*) \leq c(x)$ para todo $x \in X$, tem-se que uma vizinhança de uma solução $x \in X$ é definida como $N(x) \subseteq X$. Ou seja, N é uma função que mapeia uma solução para um conjunto de soluções. Uma solução x é dita localmente ótima ou um ótimo local com respeito a uma vizinhança N se $c(x) \leq c(\bar{x})$ para todo $\bar{x} \in N(x)$ (PISINGER; ROPKE, 2019).

Um algoritmo de busca em vizinhança recebe uma solução inicial x como entrada e computa $\bar{x} = \underset{\bar{\bar{x}} \in N(x)}{\operatorname{argmin}}\{c(\bar{\bar{x}})\}$, ou seja, ele encontra a melhor solução $\bar{x}$ na vizinhança de x. Se $c(\bar{x}) < c(x)$, então o algoritmo realiza a atualização $x = \bar{x}$. A vizinhança da nova solução x é pesquisada por uma solução de melhoria e isso é repetido até que um ótimo local x seja alcançado, caso em que o algoritmo para. O algoritmo é denotado como algoritmo de melhor melhoria devido a sempre escolher a melhor solução na vizinhança (PISINGER; ROPKE, 2019).

No LNS, a vizinhança é definida de forma implícita pelos métodos de destruição, o qual destrói uma parte da solução atual, e reparo, que conserta a parte destruída da solução. O método de destruição tipicamente contém um elemento de estocasticidade, de modo que diferentes partes da solução são destruídas cada vez que o método é invocado. A vizinhança $N(x)$ de uma solução x é então definida como o conjunto de soluções que podem ser alcançadas primeiro aplicando o método de destruição e depois o método de reparo.

Formalizando o algoritmo, dado x^b como a melhor solução obtida na busca, x como a solução atual, e x^t como a solução temporária que pode ser descartada ou atribuída como melhor solução; Dado $d(.)$ como o método de destruição e $r(.)$ como método de reparo, tem-se que para uma solução factível x^b inicial, enquanto a condição de parada não for atingida, será aplicado o método de destruição em x, e o método de reparo é aplicado no resultado da destruição, gerando x^t. É verificado se x^t é uma solução que pode ser aceita (os critérios de aceite serão detalhados a frente). Se x^t é aceito, verifica-se se seu custo é menor que o menor custo atual. Se sim, x^t passa a ser x^b. O pseudoalgoritmo completo é dado pelo Algoritmo 2.

Algorithm 2 Algoritmo LNS

```
1: Entrada: solução inicial factível x
2: x^b ← x
3: repeat
4:     x^t ← r(d(x))
5:     if accept(x^t, x) then
6:         x ← x^t
7:     end if
8:     if c(x^t) < c(x^b) then
9:         x^b ← x^t
10:    end if
11: until Condição de parada          ▷ tipicamente número de iterações ou tempo
12: return x^b
```

No artigo original sobre LNS de (SHAW, 1998), o método de aceite permitia apenas soluções melhoradas. Trabalhos posteriores, como (ROPKE; PISINGER, 2006), propõem um critério de aceite inspirado no *simulated annealing*. Este critério dita que a solução temporária x_t é sempre aceita se $c(x_t) \leq c(x)$, e aceita com probabilidade $\exp\left(-\frac{c(x_t)-c(x)}{T}\right)$ se $c(x) < c(x_t)$. Aqui $T > 0$ é a temperatura atual. A temperatura é inicializada em $T_0 > 0$ e diminuída gradualmente, por exemplo, realizando a atualização $T_{\text{new}} = \alpha T_{\text{old}}$ a cada iteração, onde $0 < \alpha < 1$ é um parâmetro. T é inicialmente definido para um valor relativamente alto, permitindo que soluções deteriorantes sejam aceitas. Conforme a busca progride, T diminui e poucas ou nenhuma solução deteriorante é aceita em direção ao final da busca. Quando este critério de aceite é aplicado, a heurística LNS pode ser vista como uma heurística de recozimento simulado com uma definição de vizinhança complexa

(PISINGER; ROPKE, 2019).

Passando pelo método de destruição, destaca-se o grau de destruição a ser configurado no método. O grau de destruição determina o quando de x será destruído na aplicação de $d(.)$. Se apenas uma pequena parte da solução for destruída, então a heurística pode ter dificuldade em explorar o espaço de busca, pois o efeito de uma grande vizinhança é perdido. Se uma parte muito grande da solução for destruída, então o LNS quase se degrada em uma re-otimização repetida, o que consome tempo do algoritmo, e/ou resulta em soluções de baixa qualidade (PISINGER; ROPKE, 2019).

Por fim, a escolha do método de reparo é livre, ficando a cargo da implementação decidir se o reparo deve buscar a melhor solução completa, ou se o reparo aplica uma heurística buscando uma boa solução construída a partir da parcial. Em (PISINGER; ROPKE, 2019), as vantagens e desvantagens de cada método são destacados por uma operação de reparo ótima será mais lenta do que uma heurística, mas pode potencialmente levar a soluções de alta qualidade em algumas iterações. No entanto, do ponto de vista da diversificação, uma operação de reparo ótima pode não ser atraente: apenas soluções de melhoria ou de custo idêntico serão produzidas e pode ser difícil deixar vales no espaço de busca, a menos que uma grande parte da solução seja destruída em cada iteração.

Por fim, a escolha do método de reparo é livre, podendo variar entre buscar a melhor solução geral, ou apenas uma solução ótima. Em (PISINGER; ROPKE, 2019), é discutido o impacto que cada solução tem no LNS. Uma operação de reparo ótima será mais lenta do que uma heurística, mas pode potencialmente levar a soluções de alta qualidade em algumas iterações. No entanto, do ponto de vista da diversificação, uma operação de reparo ótima pode não ser atraente: apenas soluções de melhoria ou de custo idêntico serão produzidas e pode ser difícil deixar vales no espaço de busca, a menos que uma grande parte da solução seja destruída em cada iteração.

2.5 *Neural Large Neighborhood Search*

Em (HOTTUNG; TIERNEY, 2019), foi proposto uma extensão do LNS, denominada de *Neural Large Neighborhood Search* (NLNS), com o objetivo de automatizar a tarefa de desenvolver operadores de reparo usando aprendizado por reforço. Em contraste com o LNS, o NLNS utiliza múltiplos operadores de destruição e reparo (como na busca de grande vizinhança adaptativa (ALNS) (PISINGER; ROPKE, 2019)).

Os operadores de destruição o_D determinam a quantidade de nós a serem destruídos de x a partir do grau de destruição atribuído a cada um. A abordagem de destruição proposta no trabalho pode assumir duas linhas: *point-based destroy*, onde os nós próximos a um ponto aleatório são removidos de todas as rotas de x, ou *tour-based destroy*, que remove todas as rotas próximas a um ponto selecionado aleatoriamente. O aprendizado

de máquina fica reservado aos operadores de reparo o_R, onde cada operador de reparo o_R corresponde a uma parametrização aprendida θ pela rede neural, que repara uma solução em um processo de reparo sequencial (HOTTUNG; TIERNEY, 2019).

Durante o treinamento de um operador de reparo, o modelo é repetidamente apresentado a soluções incompletas (de instâncias amostradas de uma certa distribuição) que foram destruídas com um operador de destruição o_D específico. O objetivo do treinamento é ajustar os parâmetros do modelo θ de forma que o modelo construa soluções de alta qualidade. Nota-se que o treinamento é impactado pelas características das instâncias usadas, dado que o modelo é capaz de aprender operadores de reparo adaptados a essas características, sendo esta capacidade importante para problemas onde instâncias com características semelhantes são frequentemente resolvidas na prática, como é o caso das variantes de *VRP*. Isso também significa que o *NLNS* pode ser re-treinado no caso de as características das instâncias encontradas mudarem, evitando significativos recursos humanos necessários para projetar novos operadores (HOTTUNG; TIERNEY, 2019).

De forma detalhada, o aprendizado de reparo segue o seguinte processo: Dado uma solução incompleta $d(x)$, primeiramente é gerado a entrada para o modelo. Para cada tour incompleto consistindo de mais de um cliente, uma entrada é criada para cada extremidade que não seja o depósito (Passo 1). Em seguida, para cada tour incompleto com apenas um nó, uma única entrada é gerada (Passo 2). Finalmente, é criada uma entrada para o depósito (Passo 3). Cada entrada x é um vetor de características 4-dimensionais representado pela tupla $\langle x_X, x_Y, x_D, x_S \rangle$. Os valores são calculados da seguinte forma. No primeiro caso correspondente ao Passo 1, em que x representa o fim de um tour incompleto $\{v_i, \ldots, v_j\}$, x_X e x_Y são as coordenadas x e y do nó no fim do tour em consideração e x_D é a soma das demandas atendidas pelo tour $\{v_i, \ldots, v_j\}$. O valor x_S é definido como 3 se o tour $\{v_i, \ldots, v_j\}$ contém o depósito e 2 caso contrário. Correspondente ao Passo 2, se x representa um tour com um único nó $\{v_i\}$, x_X e x_Y são as coordenadas x e y do nó v_i e x_D é definido como a demanda atendida do tour $\{v_i\}^2$. O valor x_S é definido como 1. Para um tour incompleto no Passo 3, x representa o depósito v_0 e x_X e x_Y são a coordenada x e a coordenada y do depósito[2] e x_D e x_S são definidos como -1. Antes de ser alimentado na rede, os valores x_X, x_Y, e x_D são reescalados para o intervalo [0, 1] (exceto pelo valor x_D do depósito, que é sempre -1) (HOTTUNG; TIERNEY, 2019).

Em cada passo de tempo de reparo t, é dado ao modelo a solução incompleta $\dot{(x)}_t$ na forma da tupla (X_t, f_t), onde X_t contém todas as entradas geradas usando o procedimento descrito acima no passo de tempo t, e f_t descreve a entrada de referência. O modelo então gera uma distribuição de probabilidade sobre todas as ações. Ou seja, o modelo define uma política estocástica parametrizada $p_\theta(a_t|\pi_t)$ com θ representando os parâmetros do modelo. Cada ação a_t corresponde a conectar o fim de um tour representado por f_t a um dos elementos em X_t e leva à solução (incompleta) π_{t+1}. Esse processo é repetido até que

uma solução $r(d(x))_T$ seja alcançada que não tenha mais tours incompletos.

A entrada de referência f_t é um elemento de $X_t \setminus \{x_0\}$ e é selecionada aleatoriamente no passo de tempo $t = 0$. Nos passos de tempo seguintes, f_t é definida como a entrada representando o fim do tour que foi conectado a f_{t-1} se o tour associado a f_{t-1} ainda estiver incompleto. Caso contrário, f_t é selecionada aleatoriamente como na primeira iteração.

Com o modelo treinado, dado uma instância de entrada a ser resolvida, parâmetros T para aplicação do recozimento simulado e o grau de destruição, é realizado a busca pelas rotas através do Algoritmo 3.

Algorithm 3 Algoritmo Busca - NLNS

Entrada: solução inicial factível x, temperatura inicial T_s, temperatura de reaquecimento T_r, temperatura mínima T_m, cronograma de resfriamento δ e porcentagem Z do lote que é reiniciado para a solução atual.
$\leftarrow x$
$T \leftarrow T_s$
repeat
    $B \leftarrow x^b, ..., x^b$ ▷ Criar um batch de cópias de x^b
    repeat
        $B \leftarrow r(d(B))$
        $x^t \leftarrow \arg\min_{x \in B} c(x)$
        if accept(x^t, x, T) **then**
            $x \leftarrow x^t$
        end if
        if $c(x^t) < c(x^b)$ **then**
            $x^b \leftarrow x^t$
        end if
        Define os primeiros $Z\%$ dos elementos em B para x.
        $T \leftarrow$ atualiza(T, δ)
    until $T > T_m$
until Condição de parada ▷ tipicamente número de iterações ou tempo
return x^b

3 TRABALHOS RELACIONADOS

A literatura que aborda o problema de roteamento de veículo é extensa. Este capítulo revisa o atual estado da arte com relação a VRPPD, aplicação de aprendizado por reforço ao problema de roteamento de veículos e estudos de caso de VRP no Brasil.

Começando por (BERGMANN; WAGNER; WINKENBACH, 2020), a integração de coletas e entregas em uma mesma rota é avaliada a partir de dados reais da Índia. Já em (FOA *et al.*, 2022), a variação do VRP estudada é o CVRP, porém aplicou-se o uso de RL para resolver o problema. Finalmente, em (LI *et al.*, 2022), o aprendizado por reforço profundo é usado para o problema de roteamento de veículos com capacidade heterogênea. Por fim, será revisado o atual cenário de estudos de caso de VRP no Brasil.

Em 2020, foi publicado *Integrating first-mile pickup and last-mile delivery on shared vehicle routes for efficient urban e-commerce distribution* por Felix M. Bergmann, Stephan M. Wagner e Matthias Winkenbach. O objetivo do estudo é avaliar as vantagens e desvantagens de combinar coletas (*first-mile pickup*) e entregas (*last-mile delivery*) de forma teórica, e em seguida, apresentar a aplicação da teoria desenvolvida através de um estudo de caso feito com dados da cidade de Bengaluru, fornecidos por uma plataforma de *e-commerce* indiana.

Em relação ao artigo de Bergmann et al. (2020), um ponto muito forte do estudo é o desenvolvimento da parte teórica. Uma parte da metodologia é focada em apresentar notações e definir os fatores de proporcionalidade que guiam a avaliação de eficiência, como o k^{PD}, que denomina a proporção entre a distância de uma rota com demandas combinadas e a soma das distâncias caso a rota otimizada por demandas (duas rotas distintas otimizadas), e é a base para o desenvolvimento das análises de eficiência. Ainda destaca-se a condução do experimento teórico, que foi executado para uma variação de combinação de fatores englobando capacidade do veículo e proporção de demandas. Foram usados os *benchmark* de Ropke and Pisinger (2006) e Li and Lim (2003) para validar qualquer viés no modelo teórico. Por fim, no estudo de caso, o modelo desenvolvido sugere um ganho de até 30% na eficiência de rota, porém esse ganho é sensível a capacidade do veículo.

Em resumo, (BERGMANN; WAGNER; WINKENBACH, 2020) traz um modelo teórico robusto que, aplicado ao mundo real, valida uma parte da pergunta de pesquisa do presente trabalho, porém, ao usar a heurística de busca local com busca em grandes vizinhanças (LS-LNS) para resolver o VRP, acaba por não explorar novos meios de resolução do problema, se valendo de um método consolidado e presente no OR-TOOLS do Google.

Entrando na aplicação do aprendizado por reforço para o VRP, em 2022 (LI *et*

al., 2022) publicou *Deep Reinforcement Learning for Solving the Heterogeneous Capacitated Vehicle Routing Problem*, onde a variação do problema de roteamento de veículos considerando uma capacidade heterogênea foi resolvida através do aprendizado por reforço.

Deste artigo, destaca-se a modelagem do VRP como MDP, dada pela tupla $M = \{S, A, \tau, r\}$, onde cada estado é definido por $s_t = (V_t, X_t) \in S$, sendo V_t a representação do veículo v_t^n t no estado t e X_t a representação da demanda x_t^n em t. Para a arquitetura do aprendizado, (LI *et al.*, 2022) escolheu aplicar uma arquitetura de *encoder-transcoder*, onde um *decoder* é responsável por escolher o nó a ser atribuído, enquanto que o outro *decoder* escolhe o veículo que recebe a atribuição. A arquitetura é baseada na revisão trazida por (XIN *et al.*, 2021), que aborda a aplicação de aprendizado por reforço a problema de roteamentos de veículos a partir de dois modelos: *Transformer* e *pointer network*. A escolha de (LI *et al.*, 2022) parte da vantagem do uso do mecanismo de *self-attention*, que permite que o modelo aprenda as relações entre quaisquer dois elementos em uma sequência, facilitando a paralelização eficiente e uma extração de características mais robusta, sem as limitações de recorrência alinhada à sequência.

A conclusão de (LI *et al.*, 2022) é de que o modelo proposto obteve resultados melhores do que algumas heurísticas convencionais, e quanto ao tempo computacional, supera outros métodos de aprendizado, porém é necessário destacar a limitação do modelo: o número de demandas por instância de problema é limitado a um conjunto de números pré-definidos, e o modelo tem um número de veículos pré definido (modelo para três veículos, e modelo para cinco veículos), sendo impraticável sua aplicação no mundo real.

Em *Solving the vehicle routing problem with deep reinforcement learning*, de Simone Foa, Corrado Coppola, Giorgio Grani e Laura Palagia, o aprendizado por reforço também é aplicado ao VRP, porém com a restrição apenas da capacidade, com frota homogênea. Em comparação a (LI *et al.*, 2022), ressalta-se que os artigos assumem abordagens diferentes quanto ao MDP, sendo que (FOA *et al.*, 2022) propõe que o estado seja caracterizado por uma tupla de quatro matrizes, porém dessas quatro matrizes, apenas uma varia a cada passo de $t \in T$, fazendo com que o estado carregue informações estáticas.

Com relação a arquitetura do aprendizado, é proposto um modelo que parte da mesma ideia de ter uma escolha de nó (demanda) e *cluster* (veículo), com a diferença de ser usado uma arquitetura com dois Atores e um Crítico, otimizados por *Proximal Policy Optimization* (PPO). Outra diferença no modelo de aprendizado é a escolha da rede neural. Os autores optaram por *Convolutional Neural Network* (CNN) clássica com dimensões de preenchimento adaptáveis para a arquitetura dos Atores e do Crítico, dado que a entrada (matriz de adjacência do VRP) é densa e o seu tamanho varia conforme o tamanho problema de roteamento.

Com relação ao *benchmark* usado, (FOA *et al.*, 2022) escolhe usar o OR-TOOLs, e como concluído pelos autores, o algoritmo apresentado no trabalho não supera essa

ferramenta consolidada, mas como objetivo de trabalhos futuros é proposto-se a melhoria da performance do RL para problemas combinatórios.

Por fim, voltando a heurística de busca em grandes vizinhanças (LS-LNS) aplicada por (BERGMANN; WAGNER; WINKENBACH, 2020), (HOTTUNG; TIERNEY, 2019) já em 2019 propôs o uso de rede neural para melhorar heurísticas já existentes, com a justificativa de que métodos que dependem puramente de aprendizado de máquina ainda não superavam heurísticas tradicionais. Seu trabalho obteve resultados que performaram melhores que o LNS tradicional, com a vantagem de não ter restrição de tamanho e número de veículos, como visto nos outros trabalhos.

A proposta de (HOTTUNG; TIERNEY, 2019) é aprender uma política π que gerencia os passos de destruição e reparo de rotas, ou seja, a política conduz a aplicação do LNS. O modelo usa uma rede neural profunda com um mecanismo de atenção, especialmente projetado para ser integrado em um ambiente de busca LNS. A tarefa de reparar uma solução destruída é deixada para a rede neural, que é treinada através de aprendizado por reforço do tipo ator-crítico, com uso de REINFORCE para atualizar o gradiente.

Passando para os estudos de casos, foi pesquisado no *Google Scholar* a seguinte combinação de termos: "estudo de caso problema roteamento de veículos coletas". Os resultados da primeira página do buscador apontam uma tendência de trabalhos relacionados a coletas seletivas, como (BRASILEIRO; LACERDA, 2008), (FRANCA, 2017), (SIMONETTO; BORENSTEIN, 2006) e (MORO *et al.*, 2018), porém, é encontrado o trabalho de (MEIRA, 2013), que propõe o uso de Programação Linear Inteira Mista (PLIM) para resolver o problema de *(Vehicle Routing Problem with Mixed Pickup and Delivery and Time Windows* de uma transportadora de Curitiba. Como conclusão, é afirmado que o PLIM resolveu a dor da transportadora, contudo, conforme o problema crescia, era notado a degradação do desempenho computacional.

A Tabela 9 traz a comparação dos trabalhos citados anteriormente através dos critérios de foco do estudo, metodologia, resultados principais, dados e *benchmark*, contribuições específicas, limitações e futuros trabalhos.

Tabela 1 – Comparação dos Trabalhos Relacionados

	Foco do estudo
Bergmann et al.	Integração de coletas (*first-mile*) e entregas (*last-mile*) em uma mesma rota para distribuição urbana de e-commerce

Continua na próxima página

Tabela 1 – continuação da página anterior

Foco do estudo	
Li et al.	Aplicação de aprendizado por reforço em problema de roteamento de veículos com capacidades heterogêneas (HCVRP)
Foa et al.	Aplicação de aprendizado por reforço em problema de roteamento de veículos com capacidade homogênea (CVRP)
Hottung et al.	Proposta de uso de aprendizado por reforço para aperfeiçoar LNS aplicado a CVRP e SCVRP
Meira et al.	Aplicação de PLIM para solucionar VRPMPDTW - Estudo de caso com empresa de transporte de Curitiba
Metodologia	
Bergmann et al.	Análise de eficiência de rota, considerando *trade-offs* entre operações de coleta e entrega integradas, baseado em aproximação contínua e ajustes fechados para estimativas de comprimento de rota e análise de regressão e experimentos numéricos para desenvolver fatores de ajuste para estimativas baseadas em aproximação contínua
Li et al.	Aprendizado por Reforço usando arquitetura encoder-decoder. Treinamento feito por *Policy Gradient* com *baseline* (Actor-Critic)
Foa et al.	Método DRL baseado em Proximal Policy Optimization (PPO) com arquiteturas de redes neurais convolucionais para Actor e Critic
Hottung et al.	Uso de aprendizado por reforço para treinar parâmetros de reparo do LNS. Modelo baseado em FFN e treinamento do modelo usando PPO com REINFORCE
Meira et al.	Programação Linear Inteira Mista (PLIM) com algoritmos de pré-processamento e pós-processamento
Resultados Principais	

Continua na próxima página

Tabela 1 – continuação da página anterior

Foco do estudo	
Bergmann et al.	Sugestão de ganhos de eficiência de até 30% e redução do impacto de tráfego e emissões em até 16% pela integração de operações de coleta e entrega
Li et al.	Supera a maioria das heurísticas convencionais e oferece desempenho competitivo contra a heurística de ponta
Foa et al.	Desempenho inferior ao OR-TOOLS em termos de solução, mas com capacidade de aprendizagem e generalização
Hottung et al.	NLSN superou as abordagens tradicionais e de aprendizado de máquina para CVRP e SCVRP. Em comparação com métodos de otimização de ponta, NLNS demonstra desempenho similar ou superior, indicando eficácia na integração de aprendizado de máquina em heurísticas de alto nível
Meira et al.	Resultados válidos em tempo computacional não proibitivo para cenários típicos da empresa estudada
Dados e *benchmark*	
Bergmann et al.	Estudo de caso com dados reais de uma plataforma de e-commerce em Bengaluru, Índia
Li et al.	Testes baseados em instâncias selecionadas aleatoriamente do *benchmark* CVRPlib
Foa et al.	Testes com instâncias variadas, incluindo instâncias com diferentes características de distribuição e comparação com OR-TOOLS
Hottung et al.	Testes com instâncias sintéticas geradas com distribuição aleatória. Comparação com métodos de ML attention model(AM) e beam search (RL-BS). Comparação também com benchmark LKH3 e LNS
Meira et al.	Cenários reais de uma empresa de transporte na região metropolitana de Curitiba
Contribuições especificas	

Continua na próxima página

Tabela 1 – continuação da página anterior

Foco do estudo	
Bergmann et al.	Introduziu fatores de ajuste em forma fechada para CA-based RLE (Estimação de Comprimento de Rota Baseada em Aproximação Contínua), focando no impacto da integração de coleta no início e entrega no final na eficiência de rota. Analisou parâmetros-chave que afetam esses fatores de ajuste, como a integração de coleta e entrega, capacidade do veículo e restrições de precedência. Explorou as implicações mais amplas dessa integração para a pegada operacional e ambiental da distribuição urbana, utilizando dados reais
Li et al.	Abordagem inovadora de DRL para frotas heterogêneas e problemas com diferentes tamanhos de clientes
Foa et al.	Propõem uma arquitetura de aprendizado por reforço para resolver CVRP utilizando redes neurais convolucionais (CNNs) para o modelo Actor e o modelo Critic no contexto de aprendizado por reforço.
Hottung et al.	Introduziu uma evolução do método LNS utilizando RL para otimizar o processo de reparar. Implementou uma arquitetura de aprendizado para a proposta, com resultados equiparáveis a abordagens tradicionais.
Meira et al.	Desenvolvimento de uma interface humano-computador para manipulação de cenários e análise dos resultados
Limitações e Futuros trabalhos	
Bergmann et al.	Sensibilidade dos ganhos de eficiência às restrições de capacidade do veículo e complexidade na sequência de paradas
Li et al.	Foco em capacidade heterogênea, limitando generalização para frotas homogênea

Continua na próxima página

Tabela 1 – continuação da página anterior

Foco do estudo	
Foa et al.	Desempenho ainda inferior ao OR-TOOLS, com perspectivas de melhorias através de novas arquiteturas de rede neural
Hottung et al.	Desempenho ainda inferior ao OR-TOOLS, com perspectivas de melhorias através de novas arquiteturas de rede neural
Meira et al.	Potencial para redução do tempo computacional em cenários maiores com mais veículos e serviços

Em conclusão, este capítulo apresentou uma revisão de trabalhos recentes no campo do Problema de Roteamento de Veículos(VRP), abordando desde técnicas tradicionais de otimização até abordagens inovadoras de aprendizado por reforço. A análise destacou a variedade de métodos e contextos, incluindo estudos focados na eficiência operacional e outros explorando o potencial do aprendizado por reforço em diferentes variantes do VRP. A pesquisa no Brasil, embora limitada, oferece perspectivas importantes sobre desafios locais. Este panorama evidencia os avanços e lacunas no campo, estabelecendo uma base sólida para futuras investigações e desenvolvimentos em roteamento de veículos.

4 METODOLOGIA

4.1 Distância entre pontos

Para cálculo de todas as distâncias usadas tanto nos experimentos quantos na avaliação dos resultados, será usado a fórmula de Haversine, que fornece a distância aproximada entre dois pontos na superfície da Terra. Formalmente, é dada por:

$$d = 2 \cdot R \cdot \sin^{-1} \sqrt{\sin^2\left(\frac{\Phi_2 - \Phi_1}{2}\right) + \cos(\Phi_1) \cdot \cos(\Phi_2) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)} \tag{4.1}$$

Onde:

ϕ_1, ϕ_2 : Latitude do ponto 1 e do ponto 2 em radianos

λ_1, λ_2 : Longitude do ponto 1 e do ponto 2 em radianos

R : Raio médio da Terra, comumente aproximado para 6371 quilômetros

Durante a implementação do trabalho, foi avaliado o uso da ferramenta *Open Source Routing Machine* (OSRM). O OSRM é um roteirizador *open-source* construído com base no *OpenStreetMap*, um serviço colaborativo que provê dados geográficos abertos e gratuitos. O OSRM, a partir de *Multi-level Djkistra*, é capaz de processar o caminho mais rápido entre pontos de origem e destino (LUXEN; VETTER, 2011). Porém, o tempo de processamento utilizado pela ferramenta para processar as distâncias e as rotas foi uma grande desvantagem. O fórmula de Haversine calcula distâncias quase 1.5x menores que a distância real dada pelo OSRM, porém, o tempo de resposta do OSRM chega a ser 10x maior, de acordo com um teste feito para avaliação da ferramenta. Dado o tempo disponível para o trabalho e os recursos computacionais, o ganho da distância não justificou o tempo necessário.

4.2 Google *OR-Tools*

OR-Tools é uma ferramenta *open-source* de otimização do Google lançada em 2015. Destaca-se da ferramenta a disponibilização de uma API de alto nível para resolver diferentes tipos de VRP. Segundo (DIDIER *et al.*, 2023), o foco da ferramenta é a resolução de problemas de roteamento de veículos industriais em larga escala com complexas restrições: capacidade dos veículos com diversos depósitos de início e término, janelas de tempo dos clientes levando em consideração o tráfego rodoviário e pausas dos motoristas, regras de precedência de coleta e entrega, carregamentos incompatíveis no mesmo veículo, similaridade da solução com uma chamada anterior ao solucionador, entre outros. A

vantagem do *OR-Tools* para outros otimizadores é o foco na generalidade dos problemas que consegue resolver, assim como nas heurísticas que utiliza.

O artigo (DIDIER *et al.*, 2023) enumera o processo de construção das soluções: (i) heurísticas de primeira solução geram possíveis rotas de veículos; (ii) a busca local melhora as primeiras soluções, com meta-heurísticas para orientar a busca; (iii) um mecanismo de programação por restrições prova a otimalidade da melhor solução ou a aprimora.

Por fim, detalhando um pouco mais da interface voltada a problemas de roteamento de veículos, tem-se que a modelagem do problema começa com o fornecimento de um grafo direcional com pesos através da função de *callback* (RoutingModel:: RegisterTransit-Callback). Os nós do grafo representam as paradas a serem feitas, e o peso representa a distância entre as paradas. Os veículos, por sua vez, são representados por um conjunto de variáveis de otimização (RoutingModel::Start and RoutingModel::Next). O otimizador representa quantidades que se acumulam ao longo da rota, como distância, tempo, ou carga do veículo através de RoutingDimension e CumulVar. Algumas variantes do VRP possuem uma interface dedicada, como o VRPPD, onde o mesmo veículo deve primeiro coletar uma encomenda em sua rota antes de entregá-la (RoutingModel::AddPickupAndDelivery) (DIDIER *et al.*, 2023). Aqui, destaca-se que o problema de entregas e coletas modelado pelo *OR-Tools* é diferente do VRPPD sendo estudado neste trabalho. A variante da ferramenta destina-se a rotas onde obrigatoriamente o veículo coleta a demanda em rota (não no centro de distribuição), e entrega a demanda na mesma rota (não leva até o centro de distribuição), enquanto que o foco do VRPPD apresentado em outros capítulos é sair do centro de distribuição com demandas, entregar as demandas na rota, e pelo caminho coletar novas demandas que devem ser levadas até o centro de distribuição.

Na Subseção 4.3.2 será explicado o uso do Google *OR-Tools* no *benchmark* desenvolvido para comparação com resultados do algoritmo proposto na seção

4.3 *LoggiBud*

O *LoggiBud* é um *benchmark open-source* para *Capacited Vehicle Routing Problem - CVRP* desenvolvido pela empresa de logística *Loggi* em 2021 (LOGGI, 2021). A ferramenta gera dados de entregas sintéticos para três cidades brasileiras: Rio de Janeiro - RJ, Brasília - DF e Belém - PA partir de dados do Censo 2010, feito pelo Instituto Brasileiro de Geografia e Estatística (IBGE). O *benchmark* está disponível sob licença MIT no repositório <https://github.com/loggi/loggibud>.

Por ser limitado a CVRP, ou seja, desenhado considerando apenas demandas de entrega (*last-mile*), o *LoggiBud*, como é, não pode ser aplicado ao problema de roteamento estudado neste trabalho: o *Vehicle Routing Problem with Pickup and Delivery*. Para mensurar o algortimo a ser apresentado na próxima seção, o *LoggiBud* deve ser alterado de

forma a generalizar a criação de dados de suas demandas para suportar dados de entrega e coleta. Em conjunto a generalização, seus algoritmos de *benchmark* devem ser alterados para comparação com o VRP sendo estudado.

Para isso, a versão 1 (denominada V1) do LoggiBud foi adaptada para uma versão 2 (V2), viabilizando dados de entrega (*last-mile*) e coleta (*first-mile*). Na próxima subseção, será apresentado as adaptações feitas na V1 para a V2 do *LoggiBud*. Para contextualizar as alterações, será também apresentado a V1 do LoggiBud.

4.3.1 Geração de instâncias e classes

A primeira mudança aplicada na V1 foi a generalização das instâncias, para que estas contenham demandas de entrega e coleta. Uma instância no *LoggiBud* representa um dia de demandas para uma determinada cidade. Na V1, uma instância pode ser do tipo *DeliveryProblemInstance* ou *CVRPInstance*. Ambos modelos são compostos primariamente de uma lista de *Delivery*, classe que representa um ponto de entrega.

Para a V2, parte-se da ideia que a classe *Delivery* não comporta a generalização de demandas, portanto, é alterado para uma nova classe *Demand*. Um objeto do tipo *Demand* possui um campo adicional *type* para indicar se representa uma entrega ou se representa uma coleta. Uma instância passa a assumir também uma forma mais genérica, onde *DeliveryProblemInstance* se torna *MixedProblemInstance* e *CVRPInstance* se torna *VRPPDInstance*, e passam a serem compostas por uma lista de *Demand*.

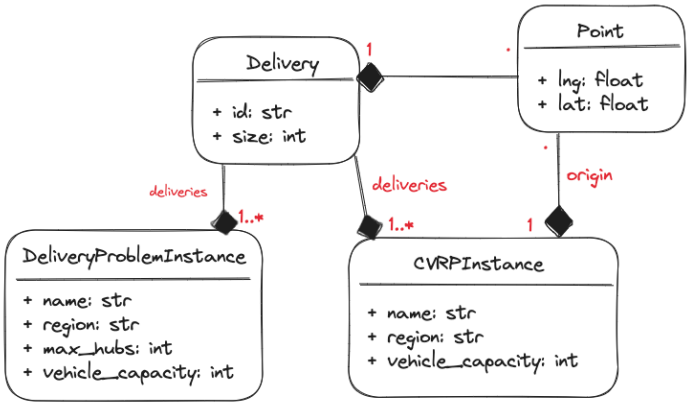
A Figura 3 ilustra as classes mencionadas como UML. As classes são definidas no arquivo types.py, disponível em <https://github.com/cstenico/loggibud/blob/master/loggibud/v2/types.py>.

Entrando mais no detalhe das instâncias, o que diferencia *DeliveryProblemInstance* de *CVRPInstance* são os campos *max_hubs* e *origin* respectivamente. Para o *DeliveryProblemInstance*, o número máximo de depósitos (de onde sai a entrega) é definido, porém, não há designação de quais são esses depósitos. Para o *CVRPInstance*, o depósito de origem é definido em *origin*, já marcando que o problema trabalha com uma origem, definida por longitude e latitude (tipo *Point*). Para a V2, essa diferença entre os novos modelos *MixedProblemInstance* e *CVRPInstance* se manteve, porém, *origin* foi alterado semânticamente para *depot*, representando além da origem da entrega, o destino da coleta.

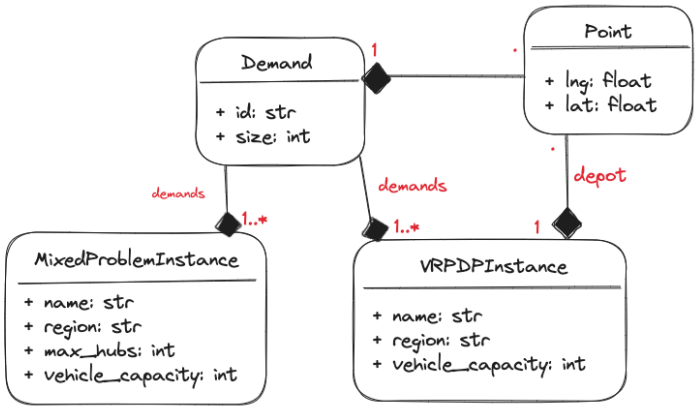
A mudança de classes se propaga para o gerador de instâncias do *LoggiBud*. As classes de configuração e resultado devem ser também alterados considerando-se as novos tipos apresentados e a geração de demandas de entregas e coletas.

O processo de geração de instâncias na V1 é definido por:

- Pré Processamento dos dados abertos do IBGE e IPEA;



(a) Classe *Delivery* e classes relacionadas



(b) Classe *Demand* e classes relacionadas

Figura 3 – Diagrama de classes para relações entre *Delivery* e *Demand*

- Definição das características das instâncias por meio das classes de configuração *DeliveryGenerationConfig* e *CVRPGenerationConfig*, definidas no arquivo do gerador;
- Geração das instâncias, considerando um conjunto de treinamento e um de teste, e para cada cidade, são geradas um conjunto de instâncias, como se representassem dias diferentes de demandas;
- Conjunto de dados gerados são salvos em arquivos no formato *json*. Para cada instância / cidade, é gerado um arquivo.

Para a V2, nada foi alterado no pré-processamento dos dados brutos, e nem no processo de distribuição dos pontos geográficos no polígono limitador de cada cidade. Já as classes *DeliveryGenerationConfig* e *CVRPGenerationConfig* foram evoluidas para *MixedGenerationConfig* e *VRPPDGenerationConfig*.

Das mudanças, destaca-se que a alteração da classe *CVRPGenerationConfig* para *VRPPDGenerationConfig* foi apenas semântica, mantendo-se todos os atributos, enquanto que a mudança entre *DeliveryGenerationConfig* e *MixedGenerationConfig* é composta da

adição de dois atributos, que definem a quantidade e a variação da quantidade de coletas por instância.

A adição destes novos atributos leva a alteração na geração do número de demandas por instância no gerador. Na V1, dado o total de instâncias a serem geradas somando-se *num_train_instances* e *num_dev_instances*, define-se uma *array* de tamanho igual ao total das instâncias. Cada elemento da *array* representa o número de demandas para cada instância, sendo esse número dado por definido de forma aleatória dado o *range* e a média das demandas.

```
# V1
sizes = (
    np.random.randint(
        -config.num_deliveries_range,
         config.num_deliveries_range,
         size=num_instances,
    )
    + config.num_deliveries_average
)
```

Então, para gerar as demandas de *first-mile* na V2, os campos *num_pickups_average* e *num_pickups_range* foram adicionados e o trecho de geração de número de demandas foi alterado para gerar duas *arrays*, cada uma gerenciando uma demanda.

```
# V2
delivery_sizes = (
    np.random.randint(
        -config.num_deliveries_range,
         config.num_deliveries_range,
         size=num_instances,
    )
    + config.num_deliveries_average
)

pickup_sizes = (
    np.random.randint(
        -config.num_pickups_range,
         config.num_pickups_range,
         size=num_instances,
    )
    + config.num_pickups_average
)
```

Por fim, há dois modelos auxiliares para resultados de etapas do gerador: *CensusGenerationResult*, que se manteve o mesmo nome, mas com tipagem diferente na lista de instâncias, e *CVRPGenerationResult*, que foi evoluido para *VRPPDGenerationResult*, também com tipagem atualizada das instâncias. As classes relacionadas a configuração das instâncias e resultados do gerador são declaradas em generators.py, disponível em <https://github.com/cstenico/loggibud/blob/master/loggibud/v2/instance_generation/generators.py>.

O diagrama completo das classes descritas pode ser encontrado em anexo 1.

4.3.2 *Benchmark*

Para estabelecer uma base de referência para os resultados obtidos pelo algoritmo descrito na Seção 4.4, foi desenvolvido um *benchmark* que utiliza o algoritmo *K-Means* e a ferramenta *OR-Tools* para solucionar o problema de roteamento, O *benchmark* foi baseado em um dos *benchmark* disponíveis na V1 do *LoggiBud*. O código usado para referência está disponível em <https://github.com/cstenico/loggibud/tree/master/loggibud/v1/baselines/task1>.

A combinação das técnicas surgiu no trabalho de (HE *et al.*, 2009), onde os autores propuseram que o problema de roteamento pode ser visto como um problema de particionamento, para então resolver as partições de forma independente, com o objetivo de especificamente resolver VRPs em larga escala com base em estratégias de decomposição.

Primeiramente, foi adaptada a interface do *LoggiBud* com o *Google OR-Tools*. O método *solve* de ortools.py passa a receber uma instância do tipo *VRPPDInstance* como entrada, e retorna uma solução do tipo *VRPPDSolution*. O cálculo da matriz de distância através do OSRM se manteve, assim como a atribuição do nó 0 do modelo do OR-Tools para o ponto do depósito da instância. Como dito na Seção 4.2, o problema de coletas e entregas disponibilizado pela interface (RoutingModel::AddPickupAndDelivery) não é a mesma variante sendo estudada, logo o uso deste modelo foi descartado. Para estabelecer pontos de entregas e coletas, alterou-se a declaração de capacidade dos pontos, sendo um ponto de entrega determinando por uma capacidade negativa (*size* do ponto é passado ao modelo como negativo) e um ponto de coleta determinando por capacidade positiva. Também, foi declarado no *callback* de capacidade que não necessariamente a capacidade acumulada deve começar por zero, permitindo a solução de rotas de apenas um tipo de demanda.

A classe *ORToolsParams* declarada no arquivo do *OR-Tools* permite ao usuário combinar diferentes parâmetros a serem passados para o solucionador da Google. Entre eles estão o máximo número de veículos que pode ser usado, o tempo limite para que a ferramenta retorne uma solução, e quais estratégias devem ser usadas para a primeira solução e a meta heurística de busca local. Por padrão, é estabelecido que o número máximo

de veículos é igual ao total de demandas, que o tempo limite é de 1 minuto, e que as estratégias de heurística são PATH_CHEAPEST_ARC e GUIDED_LOCAL_SEARCH para primeira estratégia e meta heurística de busca local, respectivamente.

Por fim, foi adaptado o método que aplica o *k-means* nas instâncias. Novamente os tipos de entrada e saída da função *solve* foram alterados para receber e retornar objetos do tipo *VRPPDInstance* e *VRPPDSolution* e ajustes semânticos relacionados aos novos atributos das classes foram feitos. O método particiona a instância em k regiões, podendo o k ser determinado de duas formas através da classe *KmeansPartitionORToolsParams*: um número fixo, que será o k do algoritmo, ou um número variável, determinado através da divisão do total de demandas pelo número variável determinado também na classe de configuração.

Cada *cluster* resultado da aplicação do *k-means* na instância de dados será resolvido de forma independente pelo *Google OR-Tools*. Por fim, cada sub-solução é combinada em uma solução, e retornada pelo método. Por cada *cluster* ser resolvido de forma independente, o método *solve* ganhou a possibilidade de ser executado de forma paralelizada caso o parâmetro *KmeansPartitionORToolsParams.multiprocessing* seja passado como verdadeiro. Com essa mudança, o tempo de execução passa a ser o maior tempo de resolução de uma sub instância, e não a soma dos tempos para resolver cada sub instância, tornando o *benchmark* mais rápido.

Para finalizar, a última adição feita ao método *k-means* da V2 foi a possibilidade de salvar a solução em um arquivo *json*. Esta adição foi feita com o objetivo de persistir os dados para análise dos resultados em outro momento.

O código do *benchmark* está disponível em <https://github.com/cstenico/loggibud/blob/master/loggibud/v2/baselines/task1/kmeans_partition_ortools.py>

4.4 NLNS

Foi escolhido o método de NLNS para estudar a aplicação de aprendizado por reforço para resolução de VRPPD. O modelo de NLNS usado foi adaptado do publicado por (HOTTUNG; TIERNEY, 2019), disponível em <https://github.com/ahottung/NLNS>.

4.4.1 Arquitetura do modelo e treinamento

Como introduzido na Seção 2.5, a ideia do algoritmo de NLNS é treinar os operadores de reparo o_R, sendo cada operador correspondente a uma parametrização θ obtida através da rede neural proposta. A partir disso, o problema de reparar uma solução após a aplicação do método de destruição é modelado como um problema de aprendizado por reforço. Para entrar nos detalhes da arquitetura proposta, é necessário relembrar os detalhes da entrada do modelo definidos na Seção 2.5. O modelo de aprendizado recebe como entrada o par

(X_t, f_t), que representa uma solução incompleta x_t em um passo de tempo t.

A partir dai, o modelo define que, para cada uma das entradas $x_i \in X_t$, um vetor de incorporação h_i é calculado usando a transformação *Embc*. *Embc* consiste em duas transformações lineares com uma ativação ReLU entre elas. Ela é aplicada a todas as entradas separadamente e de maneira idêntica. Ambas as camadas de *Embc* têm uma dimensionalidade de d_h (d_h é definido como 128 para todos os modelos treinados). Para a representação do fim do tour de referência f_t, uma incorporação h_f é gerada usando a transformação *Embf* que tem a mesma estrutura que *Embc*, mas usa diferentes parâmetros treináveis. Todas as incorporações são usadas pela camada de atenção *Att* para calcular um único vetor de contexto d_h-dimensional c que descreve todas as incorporações relevantes $h_0, \ldots, h_n$. Quais entradas são relevantes são determinadas com base em h_f. Por exemplo, o vetor c pode conter principalmente informações sobre as entradas representando pontos finais de tours que estão próximos ao ponto final do tour representado por f. Para calcular o vetor de contexto c, primeiro o vetor de alinhamento n-dimensional $\bar{a}$ é calculado que descreve a relevância de cada entrada: $\bar{a} = \text{softmax}(u_0^H, \ldots, u_n^H)$, onde $u_{H_i} = z_A \tanh(W_A[h_i; h_f])$, onde z_A é um vetor e W_A é uma matriz com parâmetros treináveis e ";" é usado para descrever a concatenação de dois vetores (HOTTUNG; TIERNEY, 2019).

Com base no vetor de alinhamento $\bar{a}$, o vetor de contexto c é gerado:

$$c = \sum_{i=0}^{n} \bar{a}_i h_i$$

O vetor de contexto c codifica informações sobre entradas relevantes, com a relevância de cada entrada $x_i \in X_t$ dada pelo vetor de alinhamento $\bar{a}$. A concatenação do vetor c e a incorporação da entrada de referência h_f são então fornecidas a uma rede *feed-forward* totalmente conectada com duas camadas (ambas usando uma ativação ReLU) que produz um vetor d_h-dimensional q. Este único vetor de saída q é usado junto com cada incorporação $h_0, \ldots, h_n$ para calcular a distribuição de saída sobre todas as ações:

$$p_\theta(a_t|\pi_t) = \text{softmax}(u_0, \ldots, u_n)$$

onde

$$u_i = z_B \tanh(h_i + q)$$

e o vetor z_B contém parâmetros treináveis.

Para garantir o cumprimento de restrições intrínsecas aos VRP, como capacidade máxima do veículo ou distância máxima viajada, ou para impedir casos onde um nó é escolhido para se conectar a ele mesmo, é aplicado uma máscara durante o processo de reparo. A máscara define as probabilidades das ações proibidas como zero.

Para treinar a rede neural, a abordagem *Actor-Critic* com REINFORCE foi escolhida para o cálculo do gradiente, conforme apresentado na Equação 2.10. O objetivo do

treinamento é ajustar os parâmetros θ de forma a minimizar a perda esperada durante a reparação de uma solução x ao longo dos passos T. Esta perda, $L(\pi_T|\pi_0)$, é calculada considerando uma solução inicial π_0 após a aplicação de um operador de destruição, e a solução final reparada π_T, resultante das ações $a_0, \ldots, a_{T-1}$ realizadas pelo modelo. A perda quantifica a diferença entre o comprimento total do tour da solução destruída π_0 e o da solução reparada π_T. Utilizando essa definição de perda, o gradiente é calculado pelo REINFORCE da seguinte maneira:

$$\nabla J(\theta|\pi_0) = \mathbb{E}_{\pi_T \sim p_\theta(\cdot|\pi_0)}[(L(\pi_0, \pi_T) - b(\pi_0))\nabla_\theta \log p_\theta(\pi_T|\pi_0)],$$

onde $b(\pi_0)$ representa a linha de base.

Em complemento à estratégia de REINFORCE, é definido o modelo do crítico, responsável por estimar o custo de reparar π_0. O crítico é treinado em alternância com o modelo de política para minimizar o erro quadrático médio entre sua previsão para os custos de reparar π_0 e os custos reais ao usar a política mais recentemente aprendida. O crítico recebe como entrada os valores X_0 gerados para π_0 conforme descrito acima. O crítico processa cada entrada usando uma rede f*eed-forward* posicional que produz um valor contínuo para cada entrada. A soma de todas as saídas é então a estimativa dos custos de reparo $b(\pi_0)$.

De forma prática, o treinamento é sintetizado pelo Algoritmo 4

Algorithm 4 Treinamento do NLNS

```
Entrada: actor, critic
Gerar ou carregar o conjunto de treinamento e validação
Inicializar otimizadores para actor e critic
Inicializar custo corrente como infinito
for cada lote de treinamento do
    Selecione um lote do conjunto de treinamento
    Destrói e repara instâncias do conjunto de treinamento
    Calcule os custos das soluções destruídas e reparadas
    Compute a recompensa e a vantagem
    Calcule a perda do actor e faça backpropagation
    Calcule a perda do critic e faça backpropagation
    Atualize o conjunto de treinamento com novas soluções
    Registre o desempenho atual
    if o lote atual é o último lote then
        Avalie o modelo com o conjunto de validação
        Salve o modelo se ele superar o desempenho atual
        Registre o custo de validação e o tempo de execução
    end if
end for
return caminho do modelo com melhor desempenho
```

4.4.2 Busca

O Algoritmo 3 traz que uma das entradas da busca do NLNS é uma solução inicial factível. Na proposta inicial de (HOTTUNG; TIERNEY, 2019), a solução inicial de uma instância é construída de forma gulosa, onde os pontos mais próximos a um ponto inicial são agrupados juntos enquanto couber, respeitando apenas a capacidade do ponto de vista de um CVRP de entregas. No VRPPD, esta solução já não atende a complexidade de volume do problema, onde a atribuição de uma coleta depende do carro ter espaço, seja por ter espaço ao sair do depósito ou seja por ter espaço por entregar demandas. Portanto, a solução inicial factível da busca do VRPPD será montada respeitando as condições de volume de forma não tão restritiva, mas também de forma simples e gulosa. A solução inicial será pela mesma heurística de construção original, com a verificação sendo alterada de forma que a soma do volume de coletas não ultrapasse a capacidade, e a soma do volume de entregas não ultrapasse a capacidade.

4.4.3 Entrada de dados

No modelo proposto por (HOTTUNG; TIERNEY, 2019), as instâncias usadas para treinamento e testes são instâncias sintéticas, geradas aleatoriamente ou de acordo com distribuições configuráveis em um *grid* 1000x1000. Para o treinamento da rede neural, são geradas n instâncias a partir de modelos pré definidos, onde n é o número de *batches* para o treinamento multiplicado pelo tamanho do *batch* (valores configuráveis). Para o teste, uma instância selecionada é carregada de arquivos .vrp disponibilizados no repositório do projeto. As distâncias entre os pontos é calculada de forma simplificada, por distância de pontos euclidiana.

Um dos objetivos deste projeto é entender a aplicação do aprendizado de máquina no mundo real, então a entrada dos dados foi modificada para receber os arquivos de dados gerados pelo *LoggiBud*. Para o treinamento, o comando de entrada foi modificado para receber, de forma opcional, o caminho para o diretório com os dados a serem treinados. O arquivo de treinamento também foi modificado para que, caso seja notado que a variável com o caminho do diretório de treinamento tenha valor, as instâncias de treinamento sejam carregadas do diretório. Já o comando de entrada para executar a busca em uma instância se manteve. A função que carrega a instância foi alterada para identificar que o arquivo sendo executado possui extensão *json*, se sim, os dados são carregados e convertidos para uma instância da classe *VRPInstance*. Por fim, durante a conversão dos dados *json* para *VRPInstance*, um novo campo da classe, denominado *original_costs*, recebe a matriz de adjacência dos pontos da instância, calculados pela fórmula de Haversine dada pela Equação 4.1. Dessa forma, as distâncias utilizadas são as mesmas utilizadas na execução do *benchmark*.

5 AVALIAÇÃO EXPERIMENTAL

5.1 Avaliação da proposta

O agrupamento de demandas de entrega e coleta na mesma rota por meio de aprendizado por reforço será avaliada através da comparação das rotas geradas pelo NLNS com as rotas de demandas integradas geradas pelo *benchmark*, com o objetivo de entender a performance do NLNS comparado a uma ferramenta *open-source* largamente adotada pelo mercado.

A eficiência de rota será definida por duas métricas propostas por (BERGMANN; WAGNER; WINKENBACH, 2020):

- Densidade média da rota: Média de quilômetros percorridos por parada
- Pegada de tráfego: Soma do total de quilômetros percorridos por instância.

Todas as instâncias possuem uma solução no formato *VRPPDSolution* para cada abordagem de roteamento de rota. Para cálculo dos resultados, as métricas foram traduzidas para uma função em *Python* que recebe uma solução e anexa as métricas calculadas a um arquivo *csv* de resultados, onde cada linha contém o nome da instância, o total de demandas, o número de rotas, a soma da distância de todas as rotas e a média de quilômetros por parada da solução. A função está disponível no arquivo <https://github.com/cstenico/loggibud/blob/master/loggibud/v2/eval/task1.py>.

5.2 Conjuntos de Dados

Com as mudanças feitas no *LoggiBud*, foram geradas as instâncias do experimento através do gerador de instâncias da ferramenta. O gerador, dado uma configuração, gera 120 instâncias (valor configurável) para cada cidade, representando 120 dias distintos de demandas. Dessas 120 instâncias, uma parte é separada para treinamento, e outra para desenvolvimento. A proporção de instâncias de treinamento / desenvolvimento é de 3:1, resultando em 90 instâncias de treinamento e 30 de desenvolvimento para cada cidade.

Em sua primeira versão, o *LoggiBud* utiliza um número definido de demandas de entregas na configuração, dado por *num_deliveries_average*, variado conforme o número definido por *num_deliveries_range*. Este número foi estabelecido por dados reais fornecidos pela empresa *Loggi* (LOGGI, 2021), e portanto, não será alterado para este trabalho.

Por não ter acesso a dados reais de coleta das cidades sendo avaliadas, serão geradas conjuntos de instâncias com diferentes proporções de coleta / entrega, sendo essa proporção

Tabela 2 – Resumo da configuração do número de coletas e entregas dos oito conjuntos de instâncias por região

Nome da Instância	α	*Delivery*		*Pickup*	
		Average	*Range*	*Average*	*Range*
RJ	0,25	28531	4430	7133	1108
	0,5	28531	4430	14266	2215
	0,75	28531	4430	21398	3322
	1	28531	4430	28531	4430
	1,25	28531	4430	35664	5538
DF	0,25	9825	2161	2466	540
	0,5	9825	2161	4932	1080
	0,75	9825	2161	7399	1621
	1	9825	2161	9865	2161
	1,25	9825	2161	12331	2701
PA	0,25	4510	956	1128	239
	0,5	4510	956	2255	478
	0,75	4510	956	3382	717
	1	4510	956	4510	956
	1,25	4510	956	5638	1195

dada por:

$$\alpha = \frac{n_p}{n_d}$$

Onde α é denominada a proporção de coletas com relação ao total de demandas, n_p é número de coletas e n_d é o número de entregas.

Assim, serão gerados 10 conjuntos de instâncias para experimentação através da variação de α de 0.25 a 1.25, com passos de 0.25, com o objetivo de avaliar se α tem interferência nos resultados do uso de RL, e não assumir um número aleatório para coletas dado a falta de dados reais.

A variação usada para entregas por dia (*num_deliveries_range*) representa aproximadamente 15% da média de entregas por dia para RJ e 21% para DF e PA. Para gerar a variação de coletas por dia para cada região será usado essa mesma constante de variação.

A Tabela 2 sumariza os 8 conjuntos de instâncias por região a serem gerados para cada cidade de acordo com a configuração a ser usada.

O *dataset* gerado está disponível em <https://drive.google.com/drive/folders/13oRLaZWjxa2Pr9pOdUBdefxUcK4n3v8n?usp=drive_link>

A Figura 4 ilustra uma das instâncias geradas para Brasília/DF. A instância está disponível no caminho *data/025/vrppd-instances-1.0/train/df-1/cvrp-1-df-19.json*. O α da instância é 0.25, e fica evidente a predominância dos pontos em azul, que marcam as demandas de entrega geradas para a instância. Em verde está marcado o depósito, e por fim, em vermelho, as demandas de coleta. Destaca-se que o uso dos dados do IBGE e

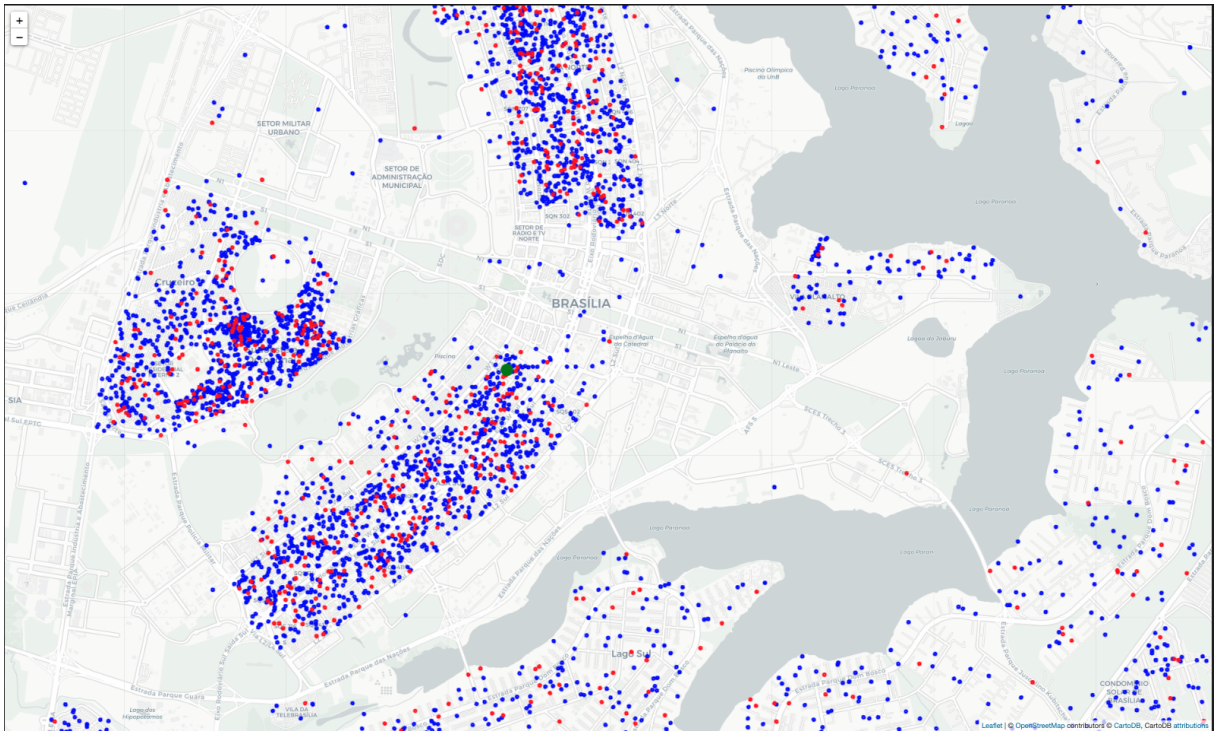


Figura 4 – *Plot* de uma das instâncias de Brasília/DF. O depósito é marcado pelo ponto verde, enquanto que as demandas de entrega são marcadas em azul, e as demandas de coleta são marcadas em vermelho.

INPE, detalhados na Seção 4.3 fazem com que a geração aleatória dos pontos não seja feita nas áreas do lago.

5.3 Restrições do VRP

Tanto para a roteirização pelo *benchmark* quanto para a roteirização do NLNS, duas restrições foram aplicadas:

- Restrição por volume: A capacidade da rota não pode ultrapassar a capacidade do veículo em nenhum momento. Para todos os experimentos, foi declarado que a capacidade máxima por veículo é de 180 (sem unidade). Em nenhum momento da rota, a soma das demandas presentes no veículo naquele instante deve ultrapassar 180.

- Restrição por distância: Considerando 8 horas de trabalho e uma velocidade média de 30km/h, foi adicionado uma restrição de distância máxima de 240 km percorrida por rota, para aproximar o experimento de um cenário encontrado na vida real, onde uma pessoa entregadora teria um dia de trabalho para cumprir a rota atribuída a ela.

5.4 *Benchmark* - Escolha do k

Os experimentos descritos a seguir foram executados em um computador MacOS 10.15.7, com 16gb de RAM e processador 2,3 GHz Intel Core i7 Quad-Core.

Para cada cidade sendo estudada, três instâncias foram escolhidas aleatoriamente para testar cinco de valores de k, com o objetivo de entender qual o valor de k a ser usado de forma a extrair o melhor resultado do *benchmark*.

As soluções obtidas para cada k foram analisadas conforme os critérios descritos na Seção 5.1. Para cada instância, o valor de k escolhido foi o que obteve a menor combinação das métricas de avaliação. Como não necessariamente a menor distância total tem a menor média por parada, os valores foram normalizados utilizando o *MinMaxScaler* da biblioteca *sklearn*, e combinados pela média de valores. O menor valor combinado foi o escolhido.

Os resultados completos para cada região estão compilados no Anexo B. Os melhores valores de k para cada instância por região foram:

Tabela 3 – Melhores valores k para a região de Brasília - DF

Instância	α	Nº Demandas	k	Nº Rotas	Distância total (km)	Média distância / parada
cvrp-0-df-0	0.25	642	3.00	13	509.38	0.79
cvrp-2-df-25	1.00	1072	1.00	3	458.02	0.37
cvrp-1-df-19	0.25	2620	9.00	58	1038.19	0.44
cvrp-0-df-1	0.75	4146	7.00	14	931.63	0.28
cvrp-2-df-42	1.25	4828	9.00	19	777.59	0.19

Tabela 4 – Melhores valores k para a região de Rio de Janeiro - RJ

Instância	α	Nº Demandas	k	Nº Rotas	Distância total (km)	Média distância / parada
cvrp-4-rj-16	0.75	2360	7.00	16	639.80	0.36
cvrp-2-rj-9	0.25	2520	7.00	48	1211.74	0.50
cvrp-2-rj-77	0.50	4457	9.00	60	1052.96	0.26
cvrp-3-rj-10	1.25	4599	5.00	9	1019.52	0.24
cvrp-5-rj-89	1.25	6018	9.00	38	1622.10	0.34

Tabela 5 – Melhores valores k para a região de Belém - PA

Instância	α	Nº Demandas	k	Nº Rotas	Distância total (km)	Média distância / parada
cvrp-0-pa-3	0.25	186	1.00	4	209.31	1.01
cvrp-1-pa-17	0.75	260	1.00	3	287.41	0.89
cvrp-0-pa-42	0.75	3650	7.00	16	874.61	0.42
cvrp-0-pa-10	1.00	4600	9.00	33	1161.68	0.37

A correlação entre o valor de k e o número total de demandas em Belém - PA é de 0.9994962235448926. Para Brasília - DF, o valor é de 0.8188644438677688. Porém, para o Rio de Janeiro - RJ, o valor é bem baixo, 0.3651490604647216. Removendo o *outlier* (instância cvrp-3-rj-10), o valor sobe para 0.9295524790307523. Destaca-se que o outlier pode indicar outros fatores dentro da distância que podem influenciar o valor de k. Dado a alta correlação, foi aplicado regressão linear simples nos resultadores encontrados. A partir da regressão linear, tem-se que o valor de k pode ser estimado através da parte inteira da relação

$$\frac{n_{demandas}}{498.24}$$

.

5.5 *Neural Large Neighborhood Search*

5.5.1 Treinamento

Para avaliar a melhor estratégia de treinamento com relação a *dataset* de treinamento, foi escolhido avaliar a instância *vrppd-instances-1.0/dev/pa-1*, com $\alpha = 0{,}5$, contra quatro configurações de treinamento: (i) Modelo treinado apenas com as instâncias de treinamento (disponíveis em vrppd-instances-1.0/train/pa-0) de PA-0, com 2800 *batches* com 25 instâncias em cada *batch*, (ii) Modelo treinado apenas com as instâncias de PA-0, com 5000 *batches* com 50 instâncias em cada *batch* (iii) Modelo treinado com todas as instâncias de PA (train/pa-0 e train/pa-1) com 2800 batches com 25 instâncias em cada *batch* e (iv) Modelo treinado com todas as instâncias de PA com 2800 *batches* com 25 instâncias em cada *batch*. Para o treinamento, foi usado o Google Colab, no ambiente de execução de CPU com alta RAM.

A configuração das instâncias de treinamento foi definida por base no tempo de execução e memória RAM consumida em primeiros experimentos. Ao carregar as instâncias de treinamento limitando as demandas a apenas 10% das demandas da instância, o consumo de RAM durante a execução chegou a 48gb para os conjuntos de PA e DF, e não foi

possível executar o treinamento do RJ (execução de 2500 *batches*. O melhor equilíbrio entre RAM e tempo de treinamento foi encontrado ao utilizar sub instâncias com 100 demandas escolhidas aleatoriamente. Com essa configuração, o tempo de treinamento diminui de 8 horas (exemplo de treinamento de todas as instâncias de PA com $\alpha = 50$) para em média 1,5 hora, com o consumo de memória RAM estável em 6gb durante o treinamento (chegando a 12gb em treinamentos de 5000 *batches* / 50 instâncias.

O treinamento pode ser avaliado por três métricas relacionadas ao aprendizado: a perda (*loss*), a perda do crítico (*critic-loss*) e o custo de reparo da solução. A perda é a medida da qualidade das ações executadas pelo modelo durante o processo de reparo de uma solução. O objetivo do treinamento é minimizar a perda. O crítico minimiza o erro quadrático médio entre sua previsão dos custos de reparação e os custos reais observados após o reparo pela política, assim, a perda do crítico mede quão precisas são as previsões do crítico em relação aos resultados reais. Por fim, o custo de reparo mede o a distância total das rotas reparadas.

O treinamento (i), com tempo de execução de 1:46:20.557579 horas, teve os valores da perda e perda do crítico estabilizados após o *batch* 1000 ser executado. Mesmo com a estabilização, destaca-se que o custo de reparo só foi para o valor mínimo após o *batch* 2700, e após a maior minimização, indicou uma tendência de subida. Essa tendência foi a motivadora para o treinamento com 5000 *batches*. A evolução dos valores pode ser visualizada pela Figura 5

O treinamento (ii), com tempo de execução de 5:16:01.107585 horas, também teve os valores da perda e perda do crítico estabilizados após o *batch* 1000 ser executado. Com relação a tendência de subida notada em (i), é perceptível a evolução dela após o *batch* 2800, mas essa subida não parece ter afetado os valores de perda. Os valores referentes ao treinamento (ii) estão disponíveis em Figura 6.

O treinamento (iii), com tempo de execução de 1:46:40.094256 horas, novamente apresenta o comportamento de estabilização das perdas após o *batch* 1000, com a diferença de agora apresentar um comportamento senoidal na perda após a estabilização. O mesmo comportamento é notado na recompensa. A hipótese aqui é que a mistura de duas regiões de Belém no treinamento (pa-0 e pa-1) cause a variação no custo, considerando as diferentes distribuições de demandas nas regiões (378 x 5112). Em contra partida, a perda do crítico se mantém estável, considerando apenas análise do gráfico. A Figura 7 traz os valores referentes a (iii).

Por fim, o treinamento (iv), com tempo de execução de 13:36:44.556051 horas, apresentou o mesmo comportamento de (iii), mesmo com o número de *batches* maior. O resultado está na figura 8.

Saindo das métricas de aprendizado e entrando nas soluções geradas pelos modelos,

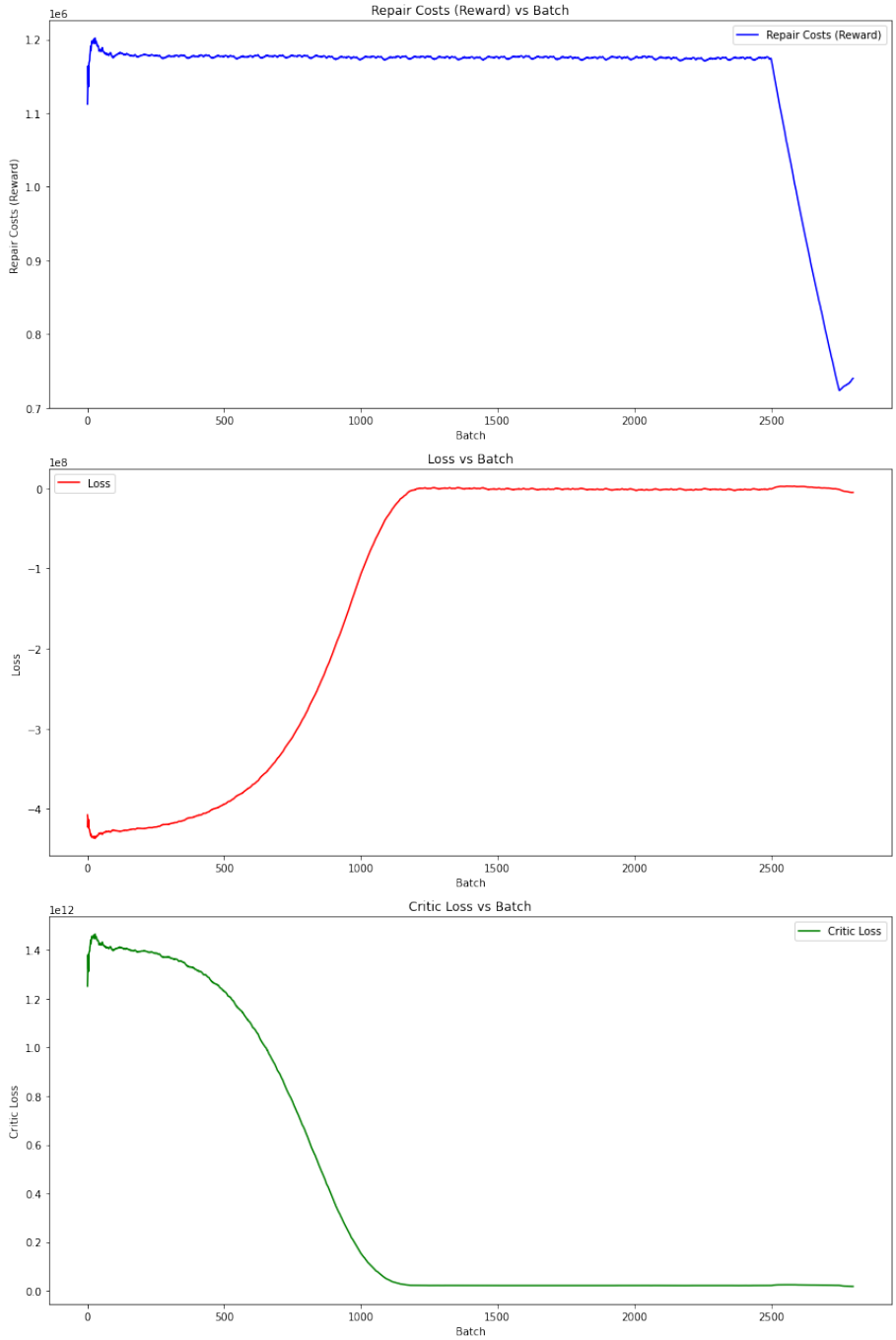


Figura 5 – Valores de recompensa, *critic-loss* e *loss* por *batch* do treinamento (i)

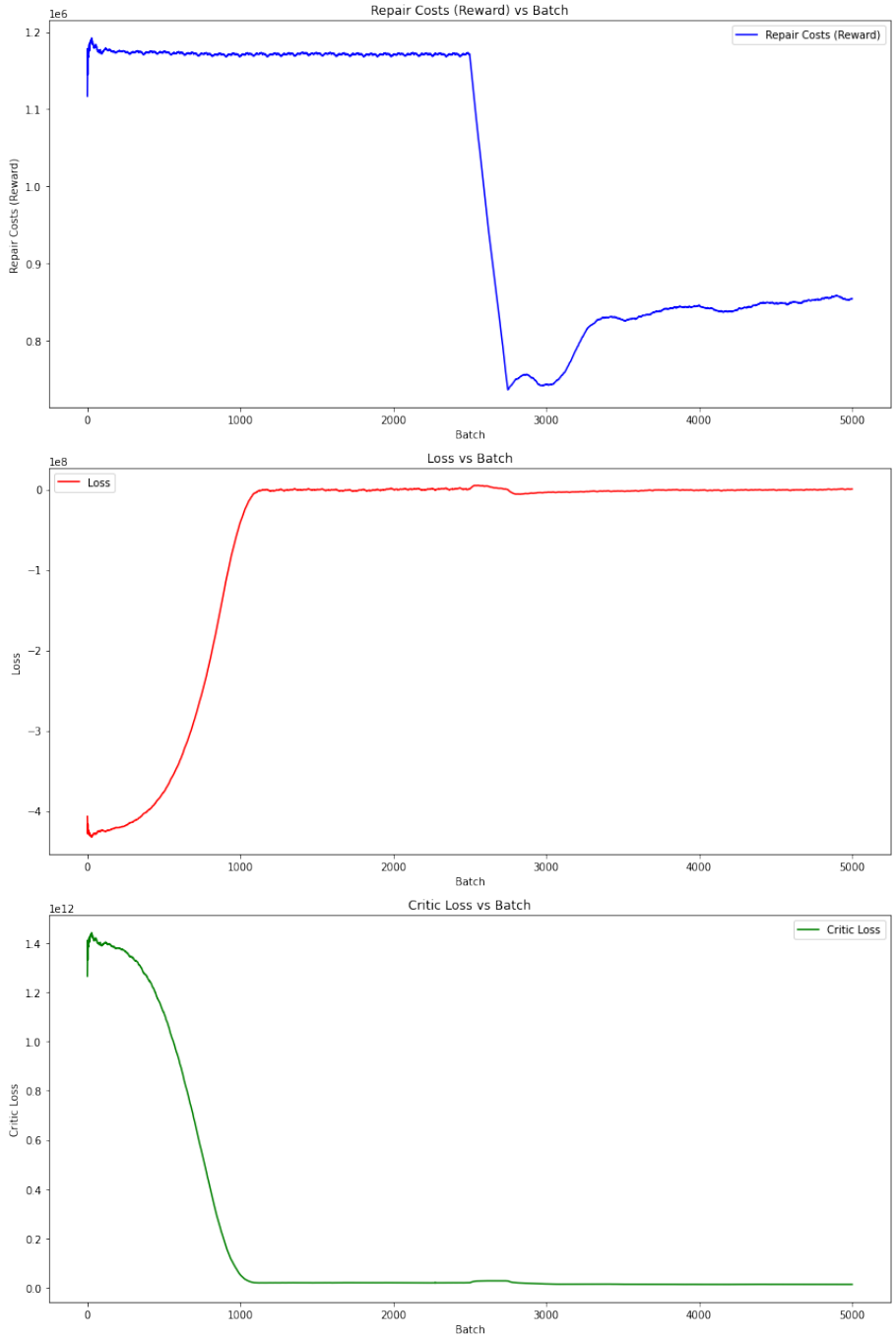


Figura 6 – Valores de recompensa, *critic-loss* e *loss* por *batch* do treinamento (ii)

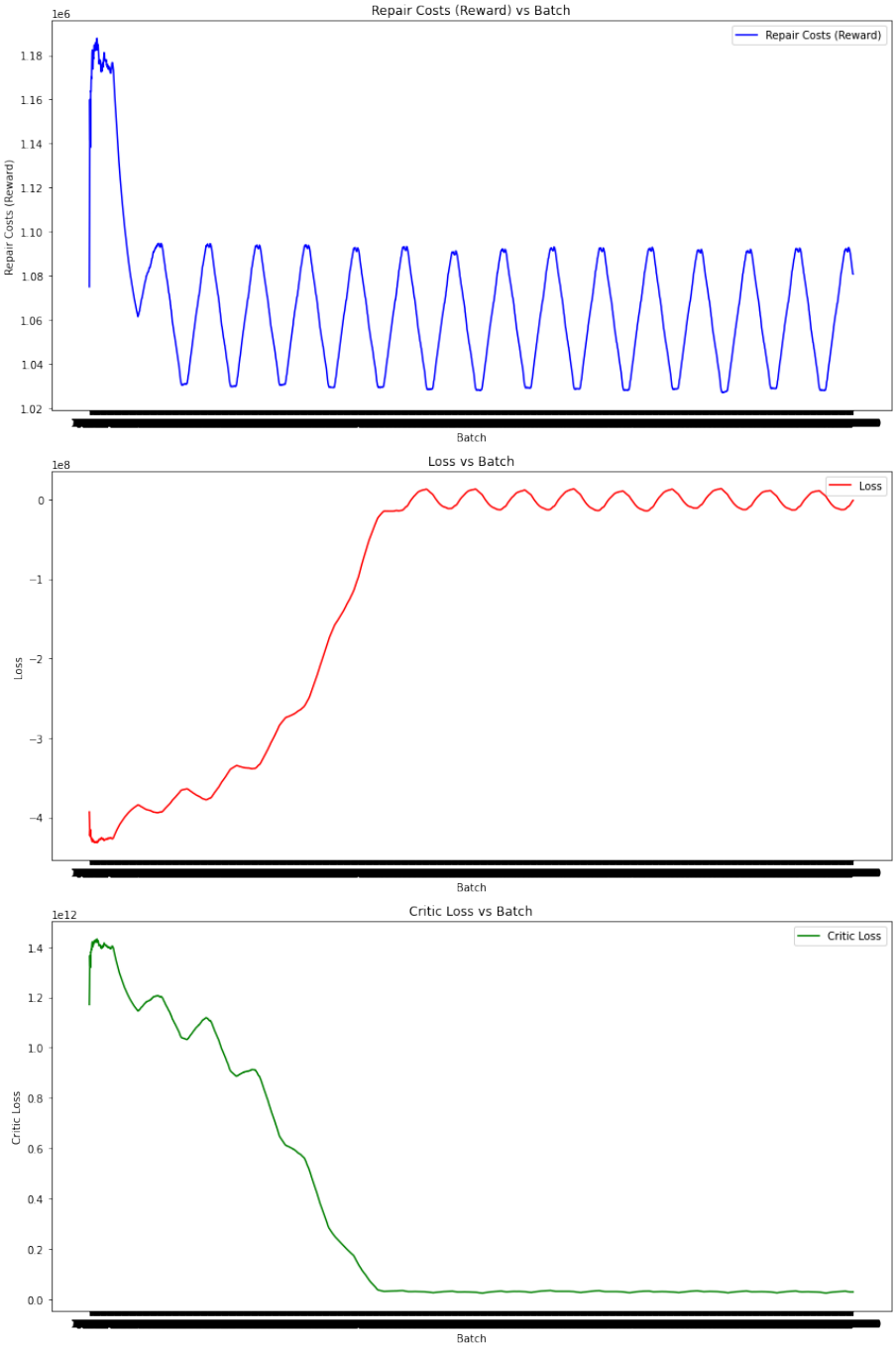


Figura 7 – Valores de recompensa, *critic-loss* e *loss* por *batch* do treinamento (iii)

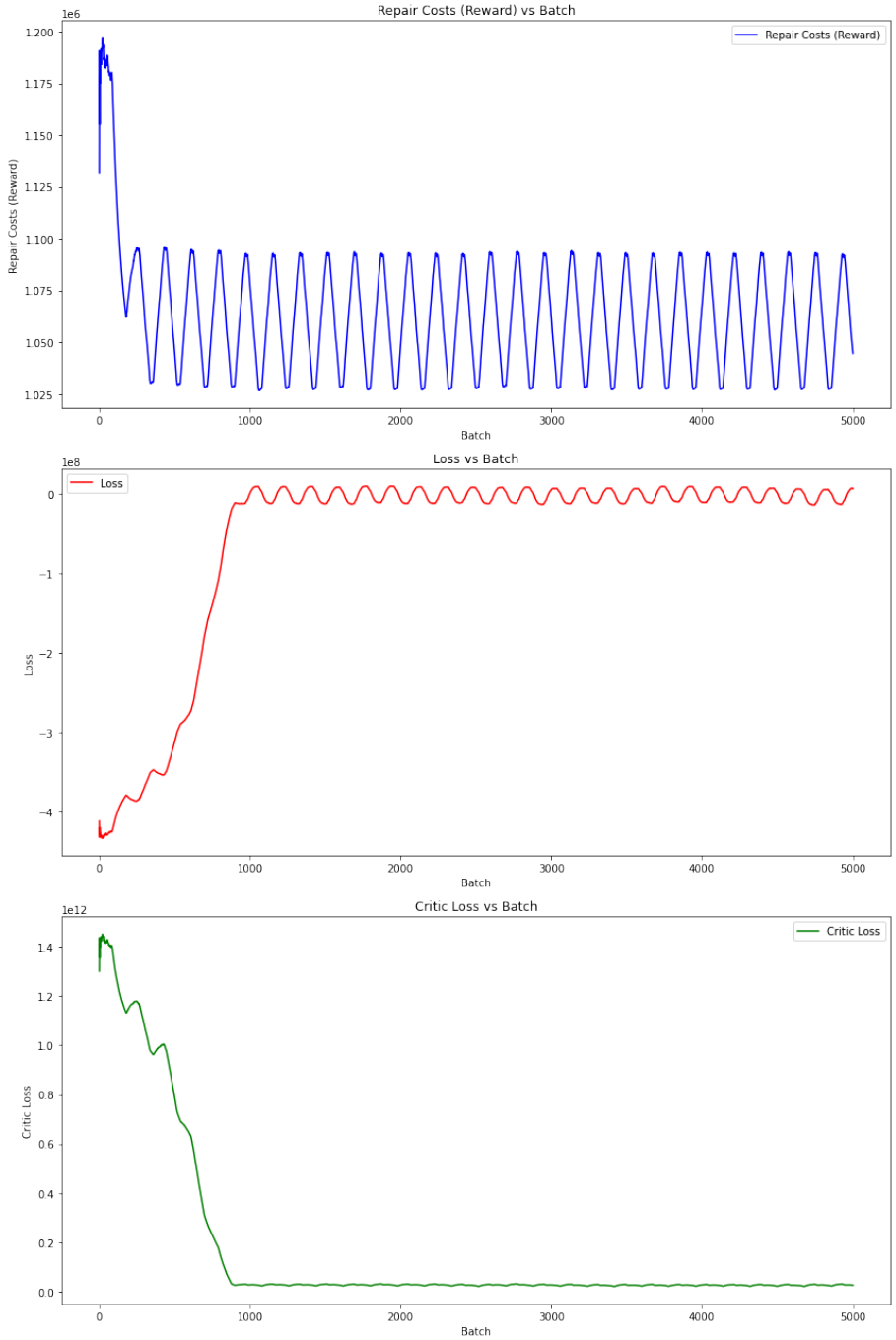


Figura 8 – Valores de recompensa, *critic-loss* e *loss* por *batch* do treinamento (iv)

tem-se que todas as buscas realizadas obtiveram o mesmo resultado para a instância *vrppd-instances-1.0/dev/pa-0* e para a instância *vrppd-instances-1.0/dev/pa-0*, ou seja, o desempenho não foi impactado pelas diferentes configurações de treinamento.

Portanto, dado os resultados apresentados acima, foi escolhido executar todos os treinamentos com 2800 *batches*, com 25 instâncias em cada *batch*, priorizando o menor tempo de execução. Com relação as instâncias de treinamento, foi escolhido selecionar todas as instâncias por cidade (como em (iii)), generalizando o modelo para a cidade. Os parâmetros de destruição foram escolhidos igualmente aos parâmetros de melhor resultado de (HOTTUNG; TIERNEY, 2019): os operadores de destruição seguem o método de *tour-based destroy* (parâmetro –lns_destruction T) e grau de destruição de 20% (parâmetro –lns_destruction_p 0.2). Outros parâmetros utilizados: taxa de aprendizado do ator: 1e-4, taxa de aprendizado do crítico: 5e-4, Z da busca: 80%. Os detalhes dos comandos de execução e parâmetros estão no Anexo C

Todos os treinamentos da rede neural fora executados no Google Colab, com ambiente de execução de CPU com alta RAM. Os modelos treinados estão disponíveis em <https://github.com/cstenico/NLNS/tree/master/runs>.

5.6 Resultados e discussão

Para o experimento final, foi escolhido roteirizar todas as instâncias relativas ao dia 90 de entregas e coletas de cada cidade. O objetivo é simular: dado n dias de demandas, a rede neural é treinada com os dados obtidos e planeja o dia $n + 1$. A Tabela 6 traz todos os resultados obtidos pela execução da busca nas instâncias selecionadas utilizando NLNS e o *benchmark*.

Analisando primeiramente o *benchmark*, a proposta de clusterizar de acordo com o número de demandas juntamente a paralelização da chamada ao OR-TOOLs criou um bom *baseline* para resolução do VRPPD. Utilizando o limite de tempo de até 10 minutos para a busca no OR-TOOLs, todas as rotas foram distribuídas em até 12 minutos de execução, independente do total de demandas e proporção de entregas e coletas. Dos resultados listados, destaca-se a densidade média da rota, que se manteve abaixo de 1 para grande parte das instâncias, tendo valor médio de 0,47 parada / km. Este valor indica rotas densas, com alta proximidade entre pontos, implicando em minimização da distância percorrida por rota. Portanto, como um dos resultados deste trabalho, tem-se a evolução de uma ferramenta de *benchmark* para abranger uma outra variante do VRP, disponível para uso pelo repositório no GitHub, com resultados eficientes de roteirização. Os resultados também incentivam o uso da estratégia de clusterização e agrupamento como uma estratégia eficiente de roteirização, podendo ser aplicado para além do *benchmark*.

Entrando no uso de NLNS, primeiramente voltando ao trabalho de (HOTTUNG;

TIERNEY, 2019), destaca-se a escolha do algoritmo pelo potencial dos resultados demonstrados com relação a dados sintéticos e CVRP. Também, destaca-se a facilidade de usar o NLNS com um *dataset* próprio, tendo liberdade de alterar como as distâncias são calculadas (na proposta original, a distância é dada por distância euclidiana, para este trabalho, foi aplicado haversine para melhor cálculo a partir de coordenadas geográficas), e alterar restrições relacionadas ao VRP de acordo com o exigido do problema.

Porém, analisando toda a parte prática, entre treinamento e busca, é necessário evidenciar o tempo investido e a exigência computacional do algoritmo. Apenas uma configuração de 5000 *batches* para treinamento exigiu 6 horas de execução, com altos níveis de consumo de RAM. O algoritmo permite o uso de GPU para treinamento e busca, contudo o uso de GPU depende de máquinas específicas e um custo financeiro maior que uma CPU para execução. É um ponto importante a ser levado em consideração no *trade-off* da escolha do algoritmo como ferramenta de roteirização.

O *trade-off* de custos ganha mais camadas com o resultado final. Nenhuma das rotas geradas obteve resultado melhor ou próximo ao *benchmark*. É importante registrar que o resultado obtido neste trabalho não invalidam o apresentado em (HOTTUNG; TIERNEY, 2019), visto que são aplicações diferentes, com treinamentos diferentes e buscas baseadas em treinamentos bem distintos. O resultado do NLNS primeiramente diz que o treinamento usado pode não ter sido bom, mesmo com as métricas mostradas na Sub Seção 5.5.1 terem sido minimizadas/maximizadas de acordo com o objetivo do ator-crítico. Aqui é importante registrar que após o fim das buscas, foi investigado o motivo de um resultado tão abaixo do esperado. Um dos motivos encontrados foi que a parametrização dos operadores de reparo não teve força o suficiente para encontrar rotas melhores para além das soluções iniciais factíveis, o que leva a conclusão que o treinamento deve ser refeito com mais *batches* e instâncias, além de outras variações de parâmetros, buscando melhorar as métricas encontradas. Porém, esse treinamento mais profundo irá exigir mais memória computacional e mais tempo para além das 6 horas encontradas, pesando ainda mais no *trade-off* de custos computacionais.

Em suma, o trabalho cumpriu o objetivo de propor e implementar o *benchmark* eficiente para o VRPPD, utilizando dados reais de três cidades brasileiras. Com relação ao NLNS, os resultados obtidos não foram satisfatórios, mas também não devem indicar que o aprendizado de máquina e o NLNS não são um caminho para resolver o problema de roteamento de veículos, pelo contrário, os resultados obtidos aqui mostram que há muito a ser explorado e entendido neste campo de pesquisa. Como trabalhos futuros, é proposto refazer o treinamento e busca visando melhor entendimento dos parâmetros e geração de rotas melhores, uso de OSRM para cálculo de distância no NLNS, buscando entender se o formato do território impacta no treinamento, e por fim, trazer o uso de *K-means* para melhorar a busca do NLNS, buscando combinar duas estratégias para melhores rotas,

como visto no *benchmark*.

						NLNS			*Benchmark*		
	α	Instância	nd	np	total	Rotas	Distância (KM)	Me parada/km	Rotas	Distância (km)	Me parada/km
pa	25	cvrp-0-pa-90.json	307	71	378	10	1187,3016	3,3128	8	411,6113	1,0573
		cvrp-1-pa-90.json	4155	957	5112	128	24960,7458	4,9805	102	2212,8546	0,4672
	50	cvrp-0-pa-90.json	293	138	431	10	1402,6811	3,6675	6	422,5147	0,9178
		cvrp-1-pa-90.json	4169	2100	6269	127	30672,6616	4,9594	69	1.925,4957	0,3846
	75	cvrp-0-pa-90.json	4187	3615	7802	127	37797,063	4,6931	26	1567,2504	0,4643
		cvrp-1-pa-90.json	275	226	501	9	1444,0787	3,6021	3	385,6026	0,9118
	100	cvrp-0-pa-90.json	4186	4746	8932	146	41544,6681	4,5571	30	1618,683	0,3526
		cvrp-1-pa-90.json	276	293	569	9	1483,9984	2,6784	3	412,9233	0,7438
	125	cvrp-0-pa-90.json	4162	6283	10445	193	50783,1258	4,759	75	2390,4123	0,3294
		cvrp-1-pa-90.json	300	442	742	14	2061,5255	3,0679	6	525,3546	0,7773
df	25	cvrp-0-df-90.json	978	189	1167	30	4286,2917	3,6697	25	914,5639	1,4323
		cvrp-1-df-90.json	4306	886	5192	131	19373,3809	3,5465	109	1910,5569	0,4175
		cvrp-2-df-90.json	4479	944	5423	137	34256,958	6,459	114	2905,4814	0,6208
	50	cvrp-0-df-90.json	974	436	1410	31	5631,1456	3,7624	19	979,8041	0,9166
		cvrp-1-df-90.json	4453	1912	6365	138	39793,1828	6,535	85	2402,4406	0,4362
		cvrp-2-df-90.json	4336	1927	6263	132	21594,7764	3,2854	80	1638,1792	0,3477
	75	cvrp-0-df-90.json	4322	2532	6854	133	43202,6538	6,4097	61	2034,4648	0,379
		cvrp-1-df-90.json	4529	2863	7392	140	26094,8131	3,3491	61	1707,399	0,4077
		cvrp-2-df-90.json	912	580	1492	28	5792,5255	3,7491	11	771,5139	0,5688
	100	cvrp-0-df-90.json	4448	5305	9753	163	62015,0343	6,2085	40	1936,319	0,2852
		cvrp-1-df-90.json	4373	5196	9569	160	33857,0709	3,4674	36	1465,09	0,365
		cvrp-2-df-90.json	942	1132	2074	36	7670,9261	3,6647	9	900,0142	1,2065
	125	cvrp-0-df-90.json	4556	4578	9134	141	55998,8294	6,1487	19	1655,0776	0,4446

Tabela 6 continuação da página anterior

						NLNS			*Benchmark*		
		cvrp-1-df-90.json	928	917	1845	29	6146,5253	3,3818	5	761,7777	1,793
		cvrp-2-df-90.json	4279	4281	8560	29	11337,1687	6,1023	21	1258,5643	1,6637
rj	25	cvrp-0-rj-90	4393	1050	5443	134	21148,0102	3,9748	107	1906,5217	0,3977
		cvrp-1-rj-90	4450	1104	5554	135	20453,8555	3,6607	107	2530,7806	0,5163
		cvrp-2-rj-90	4395	1003	5398	134	24913,0519	4,6511	109	2751,2276	0,5465
		cvrp-3-rj-90	2476	579	3055	77	13067,5578	4,5203	62	1618,3428	0,5424
		cvrp-4-rj-90	8361	2011	10372	258	28864,6489	2,8878	206	1964,5222	0,2093
		cvrp-5-rj-90	5557	1331	6888	170	24671,622	3,6952	135	2145,9878	0,3444
	50	cvrp-0-rj-90	4354	2453	6807	134	23778,0966	3,091	65	2072,9719	0,4105
		cvrp-1-rj-90	2246	1275	3521	70	13195,5989	3,9577	34	1168,5254	0,3577
		cvrp-2-rj-90	5659	3210	8869	173	33126,6079	3,6365	87	1701,6519	0,2168
		cvrp-3-rj-90	4414	2454	6868	136	27190,4545	3,9885	67	1452,6215	0,2667
		cvrp-4-rj-90	4538	2385	6923	138	34369,6332	5,1052	72	2378,2258	0,3815
		cvrp-5-rj-90	8421	4591	13012	258	34433,6358	2,8172	131	1466,9229	0,1359
	75	cvrp-0-rj-90	4799	3680	8479	148	41852,4534	4,9481	41	1955,1345	0,238
		cvrp-1-rj-90	7822	5995	13817	240	36068,2575	2,6672	72	1060,0857	0,111
		cvrp-2-rj-90	4506	3363	7869	137	27732,0546	3,221	42	1700,2997	0,2506
		cvrp-3-rj-90	4030	3059	7089	126	22815,7276	3,0453	41	1082,7017	0,2218
		cvrp-4-rj-90	2731	2086	4817	85	20461,7515	4,3697	25	1258,6276	0,3178
		cvrp-5-rj-90	5744	4418	10162	175	32322,7138	3,1014	49	1312,6978	0,1883
	100	cvrp-0-rj-90	8194	8404	16598	259	44809,1326	2,7087	49	1000,3362	0,0854
		cvrp-1-rj-90	5603	5984	11587	183	41220,6565	3,5152	32	1300,6136	0,147
		cvrp-2-rj-90	4136	4314	8450	133	26403,0196	3,0929	22	873,2471	0,1407

Tabela 6 continuação da página anterior

					NLNS			*Benchmark*		
		cvrp-3-rj-90	5105	5398	10503	165	58121,7164	5,4706	30	2093,9093	0,2473
		cvrp-4-rj-90	2219	2431	4650	74	18010,2705	3,8925	16	1040,9779	0,2404
		cvrp-5-rj-90	4375	4424	8799	135	31504,8259	3,5619	29	1686,7957	0,299
	125	cvrp-0-rj-90	2224	2675	4899	82	18688,9959	3,8314	17	1096,1814	0,2842
		cvrp-1-rj-90	4455	5386	9841	164	34009,8591	3,2725	38	1829,0839	0,2609
		cvrp-2-rj-90	3991	4928	8919	153	28136,7811	2,9846	40	1083,3497	0,1541
		cvrp-3-rj-90	7910	9877	17787	304	46847,5455	2,7055	82	1233,7544	0,1128
		cvrp-4-rj-90	5772	7006	12778	213	40635,9014	3,1055	54	1566,806	0,1855
		cvrp-5-rj-90	5280	6527	11807	202	65670,152	5,676	55	2616,2453	0,262

Tabela 6 – Resultados das buscas executadas nas instâncias listadas utilizando NLNS e o *benchmark*

6 CONCLUSÕES

Este trabalho buscou avaliar a eficácia do uso do aprendizado por reforço na programação de rotas com demandas mistas em cenários urbanos, com foco em três cidades brasileiras: Rio de Janeiro, Brasília e Belém. O desafio central estava em analisar como as estratégias de aprendizado por reforço poderiam otimizar a eficiência de rotas de entrega e coleta (*first-mile pickup* e *last-mile delivery*), em um contexto onde o crescimento acelerado do e-*commerce* no Brasil abre espaço para propostas de eficiência em operações logísticas.

Através da construção de um *dataset* específico, da modelagem do problema de roteamento com restrições apropriadas e da implementação do *Neural Large Neighborhood Search*, foi possível realizar uma comparação abrangente com o *benchmark* estabelecido pelo LoggiBud. Os resultados indicaram que, apesar do aprendizado por reforço ter um potencial significativo, ainda há muito a ser compreendido e avaliado para que esta abordagem supere as ferramentas heurísticas tradicionais em termos de eficiência e praticidade.

Como contribuição, este estudo explorou o uso do aprendizado por reforço em desafios de roteamento de veículos, com especial atenção aos contextos urbanos do Brasil. Destaca-se a criação de um *dataset* específico para Rio de Janeiro, Brasília e Belém, disponível para uso em pesquisas futuras e aplicações práticas. A implementação do NLNS para problemas de VRP com demandas mistas, embora não tenha superado as ferramentas tradicionais, representa um esforço inicial significativo no uso prático de tais algoritmos. Além disso, a análise comparativa realizada entre as abordagens convencionais e de aprendizado destacou pontos importantes a serem levados em consideração no *trade-off* entre as novas abordagens por aprendizado de máquina e heurísticas tradicionais, destacando os desafios encontrados.

Neste trabalho, a implementação do NLNS para problemas de VRP com demandas mistas enfrentou desafios significativos em termos de exigências computacionais. O algoritmo, embora promissor, mostrou-se bastante exigente em recursos e tempo, o que limitou a extensão dos experimentos e impediu um aprofundamento maior que poderia levar a resultados mais otimizados.

Para trabalhos futuros, sugere-se realizar novos treinamentos com o NLNS, considerando diferentes números de *batches*, como 10000, 50000 e 100000, a fim de confirmar se o modelo atingiu uma estabilização final ou se o aumento de 2800 para 5000 *batches* não foi suficiente para impactar os valores parametrizáveis do modelo. Adicionalmente, é recomendável explorar o aumento do número de instâncias por *batch* para avaliar seu impacto no treinamento. Devido às limitações de tempo e recursos computacionais, não foi possível realizar essa análise no presente trabalho. Além disso, propõe-se estender

as iterações do treinamento para aprimorar o aprendizado do modelo com o NLNS. Há também interesse em incorporar distâncias reais nas avaliações de rotas para examinar o impacto desses dados mais precisos no desempenho do modelo. A combinação de técnicas, como o *k-means* utilizado no *benchmark*, é considerada para melhorar a compreensão de como aplicar eficientemente o aprendizado por reforço em desafios complexos de roteamento urbano.

Em conclusão, o resultado final da aplicação do NLNS não foi como o esperado, porém este trabalho é apenas um dos primeiros passos em direção a aplicação do aprendizado de máquina em problemas de roteamento de veículos. Espera-se que as contribuições feitas ajudem futuros estudos que levem a consolidação dessa abordagem.

REFERÊNCIAS

AHMED, M.; SERAJ, R.; ISLAM, S. M. S. The k-means algorithm: A comprehensive survey and performance evaluation. **Electronics**, MDPI, v. 9, n. 8, p. 1295, 2020.

ALMAHAMID, F.; GROLINGER, K. Reinforcement learning algorithms: An overview and classification. *In*: IEEE. **2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)**. [*S.l.: s.n.*], 2021. p. 1–7.

AMAZON. **Amazon Brasil apresenta dados inéditos sobre seu impacto nas pequenas e médias empresas brasileiras durante o Amazon Conecta [Press Release]**. 2023. Press Release. (Accessed on 07/24/2023). Available at: <https://www.amazon.com.br/gp/browse.html?rw_useCurrentProtocol=1&node=115791995011&ref_=amb_link_Vg4VBdefQOOt5NjHMlUylQ_1>.

ARCHETTI, C.; CHRISTIANSEN, M.; SPERANZA, M. G. Inventory routing with pickups and deliveries. **European Journal of Operational Research**, Elsevier, v. 268, n. 1, p. 314–324, 2018.

ARDON, L. Reinforcement learning to solve np-hard problems: an application to the cvrp. **arXiv preprint arXiv:2201.05393**, 2022.

BAKER, E. K.; SCHAFFER, J. R. Solution improvement heuristics for the vehicle routing and scheduling problem with time window constraints. **American Journal of Mathematical and Management Sciences**, Taylor & Francis, v. 6, n. 3-4, p. 261–300, 1986.

BERGMANN, F. M.; WAGNER, S. M.; WINKENBACH, M. Integrating first-mile pickup and last-mile delivery on shared vehicle routes for efficient urban e-commerce distribution. **Transportation Research Part B: Methodological**, Elsevier, v. 131, p. 26–62, 2020.

BLOCHO, M. Heuristics, metaheuristics, and hyperheuristics for rich vehicle routing problems. *In*: **Smart Delivery Systems**. [*S.l.: s.n.*]: Elsevier, 2020. p. 101–156.

BRASILEIRO, L. A.; LACERDA, M. G. Análise do uso de sig no roteamento dos veículos de coleta de resíduos sólidos domiciliares. **Engenharia Sanitária e Ambiental**, SciELO Brasil, v. 13, p. 356–360, 2008.

DANTZIG, G. B.; RAMSER, J. H. The truck dispatching problem. **Management Science**, v. 6, p. 80–91, 10 1959.

DIDIER, F. *et al.* Or-tools' vehicle routing solver: a generic constraint-programming solver with heuristic search for routing problems. *In*: . [*S.l.: s.n.*], 2023.

FERNANDES, D. **Aumenta número de empresas que atuam com comércio eletrônico**. 2021. Available at: <https://www.ecommercebrasil.com.br/noticias/aumenta-numero-empresas-comercio-eletronico>.

FOA, S. *et al.* Solving the vehicle routing problem with deep reinforcement learning. **arXiv preprint arXiv:2208.00202**, 2022.

FRANCA, L. S. Implementação da coleta seletiva de estabelecimentos comerciais por meio de um aplicativo de roteamento de veículos: um estudo de caso no rio de janeiro. Universidade Federal do Rio de Janeiro, 2017.

GONÇALVEZ, G. **Em 2022, e-commerce brasileiro tem o maior crescimento da América Latina, mostra estudo**. 2022. Available at: <https://www.ecommercebrasil.com.br/noticias/e-commerce-brasileiro-tem-o-maior-crescimento-da-america-latina>.

GORA, P. *et al.* On a road to optimal fleet routing algorithms: a gentle introduction to the state-of-the-art. *In*: **Smart Delivery Systems**. [*S.l.: s.n.*]: Elsevier, 2020. p. 37–92.

HE, R. *et al.* Balanced k-means algorithm for partitioning areas in large-scale vehicle routing problem. *In*: IEEE. **2009 Third International Symposium on Intelligent Information Technology Application**. [*S.l.: s.n.*], 2009. v. 3, p. 87–90.

HEESWIJK, W. v. **Proximal Policy Optimization (PPO) Explained**. 2023. Available at: <https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b>.

HOTTUNG, A.; TIERNEY, K. Neural large neighborhood search for the capacitated vehicle routing problem. **arXiv preprint arXiv:1911.09539**, 2019.

KEVIN, A. **K-Means Clustering in Python: A Practical Guide – Real Python**. Available at: <https://realpython.com/k-means-clustering-python/>.

LAHYANI, R.; KHEMAKHEM, M.; SEMET, F. Rich vehicle routing problems: From a taxonomy to a definition. **European Journal of Operational Research**, Elsevier, v. 241, n. 1, p. 1–14, 2015.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group UK London, v. 521, n. 7553, p. 436–444, 2015.

LI, J. *et al.* Deep reinforcement learning for solving the heterogeneous capacitated vehicle routing problem. **IEEE Transactions on Cybernetics**, v. 52, n. 12, p. 13572–13585, 2022.

LITTMAN, M. Markov decision processes. *In*: SMELSER, N. J.; BALTES, P. B. (ed.). **International Encyclopedia of the Social Behavioral Sciences**. Oxford: Pergamon, 2001. p. 9240–9242. ISBN 978-0-08-043076-8. Available at: <https://www.sciencedirect.com/science/article/pii/B0080430767006148>.

LOGGI. **loggiBUD: Loggi Benchmark for Urban Deliveries**. [*S.l.: s.n.*]: GitHub, 2021. <https://github.com/loggi/loggibud>.

LUXEN, D.; VETTER, C. Real-time routing with openstreetmap data. *In*: **Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems**. New York, NY, USA: ACM, 2011. (GIS '11), p. 513–516. ISBN 978-1-4503-1031-4. Available at: <http://doi.acm.org/10.1145/2093973.2094062>.

MEIRA, W. H. T. **Problema de roteamento de veículos com entregas e coletas mistas e janelas de tempo: aplicação em uma empresa da região metropolitana de Curitiba**. 2013. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2013.

MORO, M. F. *et al.* O problema do carteiro chinês aplicado na otimização das rotas de coleta de resíduos recicláveis: Um estudo de caso. **Tecno-Lógica**, v. 22, n. 2, 2018.

MOUSAVI, S. S.; SCHUKAT, M.; HOWLEY, E. Deep reinforcement learning: An overview. *In*: **Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016**. Springer International Publishing, 2017. p. 426–440. Available at: <https://doi.org/10.1007%2F978-3-319-56991-8_32>.

MOUSAVI, S. S.; SCHUKAT, M.; HOWLEY, E. Deep reinforcement learning: an overview. *In*: SPRINGER. **Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016: Volume 2**. [*S.l.: s.n.*], 2018. p. 426–440.

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

REDAÇÃO. **Pequenas e médias empresas movimentaram R$ 2,7 bilhões com o e-commerce em 2022**. 2023. Available at: <https://mercadoeconsumo.com.br/13/01/2023/ecommerce/pequenas-e-medias-empresas-movimentaram-r-27-bilhoes-com-o-e-commerce-em-2022/>.

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

SENATOR. 2021. <https://www.senatorproject.eu/wp-content/uploads/2021/10/D2.1-SotA-in-optimization-and-machine-leaning-algorithms-applied-to-last-mile-logistics-Public.pdf>. (Accessed on 03/19/2023).

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

SIMONETTO, E. d. O.; BORENSTEIN, D. Gestão operacional da coleta seletiva de resíduos sólidos urbanos: abordagem utilizando um sistema de apoio à decisão. **Gestão & Produção**, SciELO Brasil, v. 13, p. 449–461, 2006.

STACHEWSKI, A. L. **E-commerce deve dobrar de tamanho no Brasil até 2026, diz estudo**. 2022. Available at: <https://epocanegocios.globo.com/empresas/noticia/2022/10/e-commerce-deve-dobrar-de-tamanho-no-brasil-ate-2026-diz-estudo.ghtml>.

TAN, S.-Y.; YEH, W.-C. The vehicle routing problem: State-of-the-art classification and review. **Applied Sciences**, v. 11, n. 21, 2021. ISSN 2076-3417. Available at: <https://www.mdpi.com/2076-3417/11/21/10295>.

WANG, J. **Trust Region Policy Optimization (TRPO)**. 2020. Available at: <https://www.cs.toronto.edu/~wangjk/slides/CS2621_TRPO_PPO.pdf>.

WHAT is unsupervised learning? Available at: <https://cloud.google.com/discover/what-is-unsupervised-learning>.

WIDUCH, J. Current and emerging formulations and models of real-life rich vehicle routing problems. *In*: **Smart Delivery Systems**. [*S.l.: s.n.*]: Elsevier, 2020. p. 1–35.

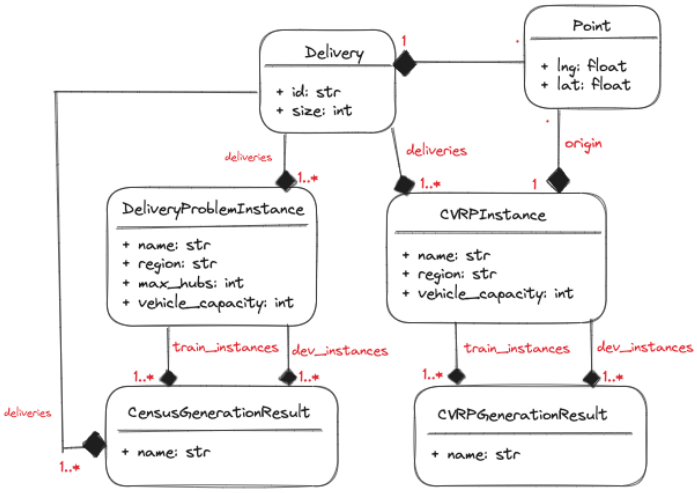
WINDER, P. **Reinforcement learning: industrial applications of intelligent agents**. First edition. Beijing Boston Farnham Sebastopol Tokyo: O'Reilly, 2021. ISBN 9781098114831.

XIN, L. *et al.* Step-wise deep learning models for solving routing problems. **IEEE Transactions on Industrial Informatics**, v. 17, n. 7, p. 4861–4871, 2021.

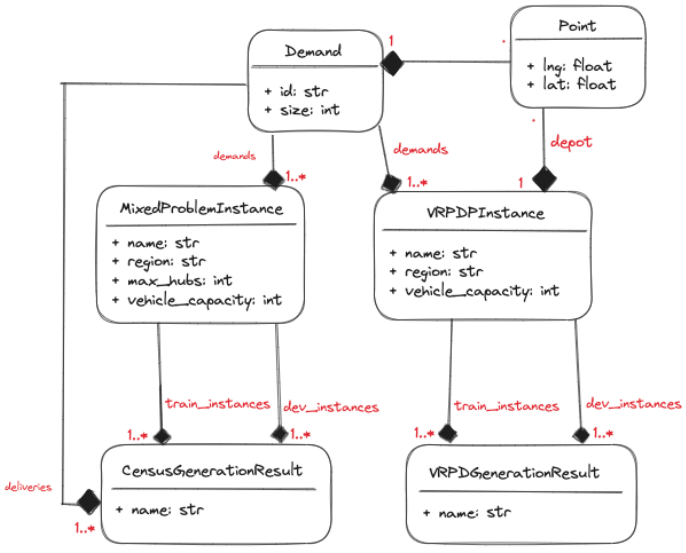
ZONG, Z. *et al.* Mapdp: Cooperative multi-agent reinforcement learning to solve pickup and delivery problems. *In*: **Proceedings of the AAAI Conference on Artificial Intelligence**. [*S.l.: s.n.*], 2022. v. 36, n. 9, p. 9980–9988.

ANEXOS

ANEXO A – DIAGRAMA DE CLASSES DO *LOGGIBUD*

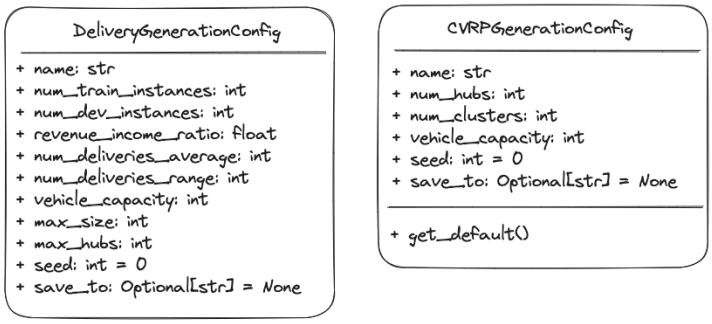


(a) Diagrama completo para instâncias - V1

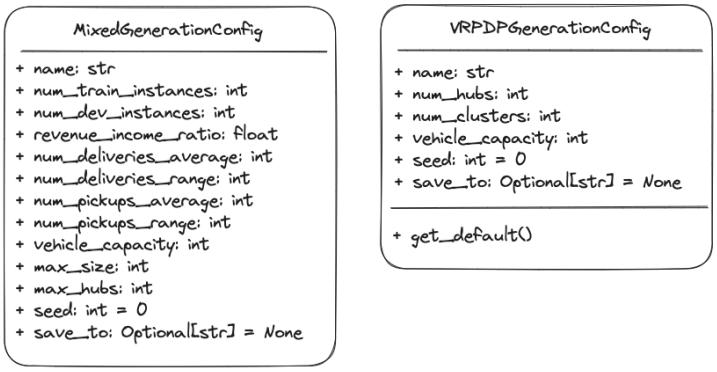


(b) Diagrama completo para instâncias - V2

Figura 9 – Diagrama completo com modelos base e modelos de instância



(a) Modelos para configuração da geração de instâncias (V1)



(b) Modelos para configuração da geração de instâncias (V2)

Figura 10 – Diagrama de classes para modelos de configuração do gerador de instâncias

ANEXO B – RESULTADOS COMPLETOS DA AVALIAÇÃO DE k

Tabela 7 – Resultados da avaliação de k para instâncias do Pará

Instância	α	Nº Demandas	k	Nº Rotas	Distância total (km)	Média distância / parada	Métricas combinadas
cvrp-0-pa-3	0.25	186	1.00	4	209.31	1.01	0.00
cvrp-0-pa-3	0.25	186	3.00	5	244.82	7.78	0.47
cvrp-0-pa-3	0.25	186	5.00	6	237.88	6.38	0.38
cvrp-0-pa-3	0.25	186	7.00	8	312.21	10.57	0.88
cvrp-0-pa-3	0.25	186	9.00	10	333.76	11.26	1.00
cvrp-1-pa-17	0.75	260	1.00	3	287.41	0.89	0.00
cvrp-1-pa-17	0.75	260	3.00	5	309.65	2.93	0.17
cvrp-1-pa-17	0.75	260	5.00	7	348.04	9.79	0.62
cvrp-1-pa-17	0.75	260	7.00	8	390.14	10.95	0.82
cvrp-1-pa-17	0.75	260	9.00	9	427.33	12.05	1.00
cvrp-0-pa-42	0.75	3650	1.00	14	1172.22	0.92	1.00
cvrp-0-pa-42	0.75	3650	3.00	14	982.28	0.44	0.20
cvrp-0-pa-42	0.75	3650	5.00	15	901.20	0.42	0.04
cvrp-0-pa-42	0.75	3650	7.00	16	874.61	0.42	0.00
cvrp-0-pa-42	0.75	3650	9.00	17	933.39	0.66	0.34
cvrp-0-pa-10	1.00	4600	1.00	30	1729.75	1.25	1.00
cvrp-0-pa-10	1.00	4600	3.00	31	1251.73	0.48	0.14
cvrp-0-pa-10	1.00	4600	5.00	32	1179.53	0.38	0.02
cvrp-0-pa-10	1.00	4600	7.00	34	1211.71	0.45	0.09
cvrp-0-pa-10	1.00	4600	9.00	33	1161.68	0.37	0.00

Tabela 8 – Resultados da avaliação de k para instâncias do Rio de Janeiro

Instância	α	Nº Demandas	k	Nº Rotas	Distância total (km)	Média distância / parada	Métricas combinadas
cvrp-4-rj-16	0.75	2360	3.00	15	741.48	0.44	0.31
cvrp-4-rj-16	0.75	2360	5.00	16	674.46	0.38	0.10
cvrp-4-rj-16	0.75	2360	7.00	16	639.80	0.36	0.00
cvrp-4-rj-16	0.75	2360	9.00	17	680.02	0.40	0.12

Table Continued from previous page

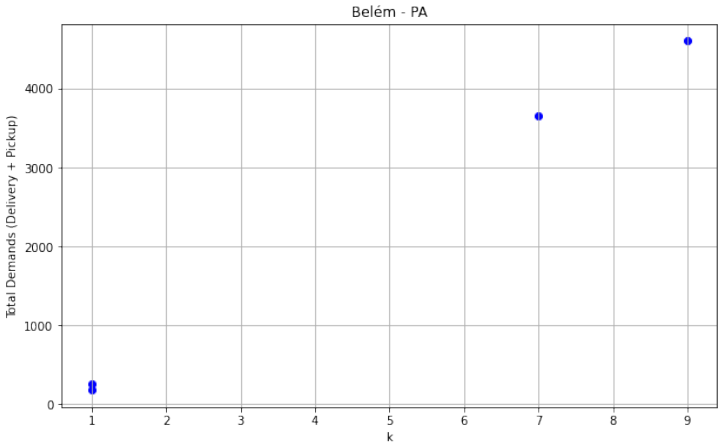
Instância	α	Nº Demandas	k	Nº Rotas	Distância total (km)	Média distância / parada	Métricas combinadas
cvrp-2-rj-9	0.25	2520	1.00	44	1597.35	0.89	0.85
cvrp-2-rj-9	0.25	2520	3.00	45	1356.26	0.59	0.27
cvrp-2-rj-9	0.25	2520	5.00	47	1239.46	0.52	0.05
cvrp-2-rj-9	0.25	2520	7.00	48	1211.74	0.50	0.00
cvrp-2-rj-9	0.25	2520	9.00	50	1250.70	1.05	0.55
cvrp-2-rj-77	0.50	4457	1.00	57	1380.42	0.59	1.00
cvrp-2-rj-77	0.50	4457	3.00	57	1226.39	0.47	0.58
cvrp-2-rj-77	0.50	4457	5.00	59	1119.14	0.32	0.18
cvrp-2-rj-77	0.50	4457	7.00	59	1055.22	0.28	0.02
cvrp-2-rj-77	0.50	4457	9.00	60	1052.96	0.26	0.00
cvrp-3-rj-10	1.25	4599	1.00	6	1199.55	0.34	1.00
cvrp-3-rj-10	1.25	4599	3.00	9	1133.73	0.32	0.73
cvrp-3-rj-10	1.25	4599	5.00	9	1019.52	0.24	0.00
cvrp-3-rj-10	1.25	4599	7.00	13	1072.21	0.31	0.49
cvrp-3-rj-10	1.25	4599	9.00	14	1095.46	0.31	0.57
cvrp-5-rj-89	1.25	6018	1.00	34	2439.89	1.40	1.00
cvrp-5-rj-89	1.25	6018	3.00	35	1998.22	0.76	0.43
cvrp-5-rj-89	1.25	6018	5.00	36	1789.95	0.50	0.19
cvrp-5-rj-89	1.25	6018	7.00	37	1643.31	0.32	0.01
cvrp-5-rj-89	1.25	6018	9.00	38	1622.10	0.34	0.01

Tabela 9 – Resultados da avaliação de k para instâncias de Brasília

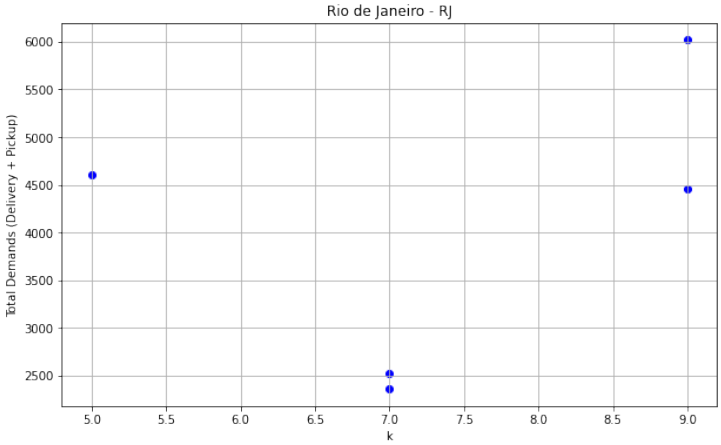
Instância	α	Nº Demandas	k	Nº Rotas	Distância total (km)	Média distância / parada	Métricas combinadas
cvrp-0-df-0	0.25	642	1.00	13	554.73	0.85	0.23
cvrp-0-df-0	0.25	642	3.00	13	509.38	0.79	0.00
cvrp-0-df-0	0.25	642	5.00	14	536.84	2.45	0.63
cvrp-0-df-0	0.25	642	7.00	16	575.83	2.37	0.79
cvrp-0-df-0	0.25	642	9.00	17	614.89	2.39	0.98
cvrp-2-df-25	1.00	1072	1.00	3	458.02	0.37	0.00
cvrp-2-df-25	1.00	1072	3.00	4	511.51	0.49	0.26
cvrp-2-df-25	1.00	1072	5.00	6	519.05	1.92	0.75

Continuação da tabela da página anterior

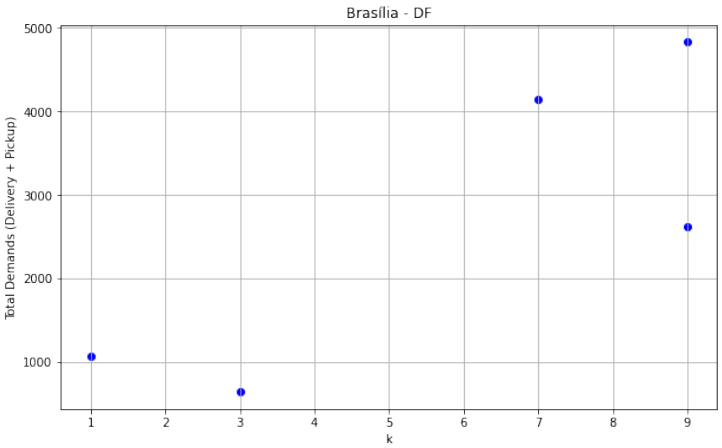
instance_-name	α	total_-demands	k	route_-count	total_distance_km	average_-distance_-stop	combined_-metric
cvrp-2-df-25	1.00	1072	7.00	8	534.51	1.71	0.75
cvrp-2-df-25	1.00	1072	9.00	10	580.40	1.50	0.87
cvrp-1-df-19	0.25	2620	1.00	54	1297.39	0.63	1.00
cvrp-1-df-19	0.25	2620	3.00	55	1088.26	0.45	0.13
cvrp-1-df-19	0.25	2620	5.00	56	1055.92	0.45	0.04
cvrp-1-df-19	0.25	2620	7.00	57	1037.64	0.45	0.01
cvrp-1-df-19	0.25	2620	9.00	58	1038.19	0.44	0.00
cvrp-0-df-1	0.75	4146	1.00	11	1310.90	1.36	1.00
cvrp-0-df-1	0.75	4146	3.00	12	1111.94	0.55	0.36
cvrp-0-df-1	0.75	4146	5.00	13	994.87	0.33	0.11
cvrp-0-df-1	0.75	4146	7.00	14	931.63	0.28	0.00
cvrp-0-df-1	0.75	4146	9.00	16	940.95	0.41	0.07
cvrp-2-df-42	1.25	4828	1.00	17	1184.04	1.20	1.00
cvrp-2-df-42	1.25	4828	3.00	18	955.07	0.44	0.34
cvrp-2-df-42	1.25	4828	5.00	20	889.35	0.26	0.17
cvrp-2-df-42	1.25	4828	7.00	19	815.03	0.20	0.05
cvrp-2-df-42	1.25	4828	9.00	19	777.59	0.19	0.00



(a) Relação entre k e número total de demandas para Belém - PA



(b) Relação entre k e número total de demandas para Rio de Janeiro - PA



(c) Relação entre k e número total de demandas para Brasília - DF

Figura 11 – Diagrama de classes para modelos de configuração do gerador de instâncias

ANEXO C – COMANDOS DE TREINAMENTO E BUSCA (NLNS)

Comando utilizado para treinar os modelos:

```
python3 main.py -mode train -lns_destruction T -lns_destruction_p 0.2
-nb_batches_training_set 2800 -nb_train_batches 2800 -batch_size 25
-lns_timelimit 300 -lns_batch_size 5 -test_size 10 -valid_size 10
-training_path 'path/to/data/025/vrppd-instances-1.0/train/pa'
-run_name '025_PA_TRAIN_FINAL' -load_partial_instance True -device cpu
```

Os caminhos de treinamento e nomes foram substituídos conforme características da região sendo treinada. Por exemplo, para o treinamento das instâncias do Rio de Janeiro foi utilizado:

```
python3 main.py -mode train -lns_destruction T -lns_destruction_p 0.2
-nb_batches_training_set 2800 -nb_train_batches 2800 -batch_size 25
-lns_timelimit 300 -lns_batch_size 5 -test_size 10 -valid_size 10
-training_path '/path/to/data/100/vrppd-instances-1.0/train/rj'
-run_name '100_RJ_TRAIN_FINAL' -load_partial_instance True -device cpu
```

Comando utilizado para busca em instância:

```
python3 main.py -mode eval_single
-model_path '/path/to/runs/075_DF_TRAIN_FINAL/models' -lns_batch_size 25
-instance_path '/path/to/data/075/vrppd-instances-1.0/dev/df-2/cvrp-2-df-90.json'
-lns_nb_cpus 10 -round_distances -lns_timelimit 600 -lns_reheating_nb 10
-device cpu -nb_runs 1 -run_name 'SEARCH_075_DF_2'
```

Nem todos os parâmetros são declarados explicitamente na linha de comando. A lista completa de parâmetros está disponível em <https://github.com/cstenico/NLNS/blob/master/config.py>