Seleção de Rotas Modelos para resolver o Last-Mile Incremental Capacitated Vehicle Routing Problem (ICVRP)

Ascânio Sávio de Araujo Neves
asan2@ic.ufal.br
Instituto de Computação, Universidade Federal de Alagoas
Maceió, Alagoas, BRA

Ruan Heleno Correa da Silva
rhcs@ic.ufal.br
Instituto de Computação, Universidade Federal de Alagoas
Maceió, Alagoas, BRA

Bruno Costa e Silva Nogueira
bruno@ic.ufal.br
Instituto de Computação, Universidade Federal de Alagoas
Maceió, Alagoas, BRA

Rian Gabriel Santos Pinheiro
rian@ic.ufal.br
Instituto de Computação, Universidade Federal de Alagoas
Maceió, Alagoas, BRA

RESUMO

O objetivo desse projeto é resolver o *Last-Mile Incremental Capacitated Vehicle Routing Problem* (Last-Mile ICVRP) utilizando um conjunto de pools contendo rotas modelos e um algoritmo de identificação de quadrantes para alocar as entregas em unidades de carregamento. Ao alocar as entregas, é calculado a menor distância entre a entrega atual, as entregas que já foram alocadas e as entregas presentes na rota modelo escolhida. Nos resultados parciais obtidos, o algoritmo desenvolvido conseguiu bons resultados ao ser comparado com os resultados de outros algoritmos.

KEYWORDS

Clusterização, Last Mile ICVRP

ACM Reference Format:
Ascânio Sávio de Araujo Neves, Ruan Heleno Correa da Silva, Bruno Costa e Silva Nogueira, and Rian Gabriel Santos Pinheiro. 2021. Seleção de Rotas Modelos para resolver o Last-Mile Incremental Capacitated Vehicle Routing Problem (ICVRP). In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

O *Last-Mile Incremental Capacitated Vehicle Routing Problem* (Last-Mile ICVRP) é um caso particular do problema SD-CVRP (Stochastic and Dynamic Capacitated Vehicle Routing Problems) [6]. O problema é denominado Last-Mile por caracterizar as entregas entre o centro de expedição e o destinatário final. Ele pode ser descrito como se segue. Dado um conjunto (histórico) de clientes atendidos anteriormente $H = \{v_1, \ldots, v_\ell\}$, um conjunto $\mathcal{U}$ de unidades de carregamento e um conjunto com n novos clientes $D = \{v_{\ell+1}, \ldots, v_{\ell+n}\}$, o Last-Mile ICVRP tem como objetivo alocar cada uma das entregas do conjunto D a uma unidade de carregamento. Cada demanda é alocada, uma por vez, em uma única unidade de carregamento de acordo com a ordem de chegada pré-definida. As unidades de carregamento tem sua capacidade máxima que, quando atingida, precisa ser despachada para entrega. O problema busca achar uma alocação das entregas às unidades de carregamento, de maneira que a distância percorrida pelos veículos de entrega seja minimizada.

De forma mais específica, a demanda do i-ésimo cliente ($v_{\ell+i}$) deverá ser alocada em uma unidade de carregamento $U_j \in \mathcal{U}$, de acordo com: (i) a distância entre o local de entrega de $v_{\ell+i}$ e a rota atual associada a U_j — tal rota é formada pelos elementos do conjunto $\{v_{\ell+1}, \ldots, v_{\ell+i-1}\}$ que foram alocados em U_j —; e (ii) disponibilidade de espaço na unidade U_j, *i.e.*, o volume das entregas na rota U_j deve respeitar a capacidade Q do veículo. Vale ressaltar que, cada alocação é realizada de acordo com informações extraídas do histórico de clientes atendidos anteriormente. Estas informações são obtidas do conjunto H.

De maneira geral, a resolução dos problemas relacionados ao SD-CVRP pode ser dividida em duas etapas. A primeira etapa, chamada de *planejamento*, ocorre de maneira *offline* e utiliza dados históricos de entrega para computar um conjunto de decisões pré-definidas. A segunda etapa, chamada de *execução*, ocorre em tempo real e utiliza as ações pré-definidas na etapa anterior assim como as informações sobre os eventos em tempo real (ex.: chegada de uma nova demanda) para gerar as rotas finais. A metodologia mais comum utilizada na literatura consiste em definir um *pool* com várias soluções possíveis (etapa de *planejamento*) e usar algoritmos de seleção de rotas na etapa de *execução*.

2 TRABALHOS RELACIONADOS

Tradicionalmente, os VRP dinâmicos e estocásticos são formulados como processos de Markov [5, 8, 9] ou como programação estocástica [1]. Ambas as metodologias modelam os dados do problema como variáveis aleatórias que seguem uma distribuição previamente conhecida. O objetivo é otimizar uma medida de risco (como o valor esperado, a variância ou o valor em risco condicional de alguma função de custo), sujeito à satisfação de algumas restrições [3]. Note que ambas as abordagens não estão alinhadas com o problema encontrado na Loggi.

A partir de apresentações institucionais da empresa Loggi, é possível identificar duas estratégias recentes para resolver o Last-Mile ICVRP. A primeira estratégia foi apresentada em [2] e usa uma combinação de agentes autônomos com mineração de dados. Nesta abordagem, os agentes autônomos modelam as rotas a serem construídas. Além disso, técnicas de mineração de dados são usadas

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

para extrair informações estocásticas sobre a distribuição de pacotes e assim otimizar a alocação de cada pacote em uma das rotas. Para determinar em qual rota r novo pacote p será alocado, a seguinte função é maximizada: $bet(p,r) = \alpha * bet_{DataMining}(p,r) + (1-\alpha) * bet_{Distance}(p,r)$, onde $bet_{DataMining}(p,r)$ é um valor extraído da mineração de dados, $bet_{Distance}(p,r)$ é uma função que mede a distância entre o pacote p e os pacotes já inseridos na rota r, e α é um valor no intervalo $[0,1]$. Os resultados mostram que o número de rotas que precisam ser geradas pela abordagem proposta é em média superior em 17% quando comparado com o número de rotas gerado por um solver para a versão estática do *CVRP*.

A segunda estratégia da empresa para resolver o Last-Mile IC-VRP foi apresentada em video [7]. A partir do vídeo, é possível depreender que a estratégia consiste em duas etapas: uma etapa de clusterização rotas boas geradas a partir do histórico de entregas (*offline*), seguida de uma etapa alocação dos pacotes em um dos clusters de rotas (*online*). Três métodos de clusterização (*bag of cells*, distância euclidiana e distância entre ruas) e dois métodos de alocação de rotas (guloso e clusters artificiais) foram experimentados. A combinação do método de clusterização baseado em distância entre ruas com o de alocação usando clusters artificiais apresentou os melhores resultados. Estes resultados mostram que a solução apresentada é apenas 16% pior que as encontradas por meio de um solver para a versão estática do *CVRP*.

3 MÉTODO PROPOSTO

Para explicar o funcionamento do algoritmo desenvolvido, as principais funcionalidades foram divididas em sete tópicos.

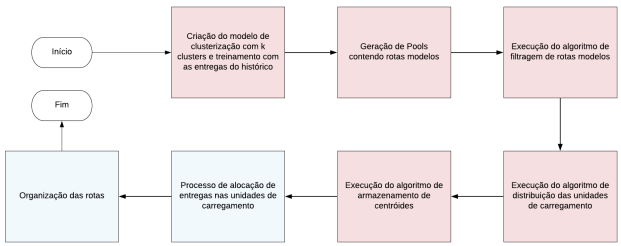

Figura 1: Fluxograma do método proposto.

A Figura 1 mostra o fluxograma do algoritmo desenvolvido para resolver o *Last-Mile Incremental Capacitated Vehicle Routing Problem(ICVRP)*. Em vermelho, estão relacionados os processos referente à primeira etapa, chamada de *planejamento*, que ocorre de maneira *offline*. Em azul, estão relacionados os processos referente à segunda etapa, chamada de *execução*, que ocorre em tempo real.

3.1 Modelo de Clusterização

No algoritmo desenvolvido, é executado o algoritmo *K-Means* sobre as entregas com o valor pré-definido de k. O modelo é treinado a partir dos pontos das entregas presentes no histórico.

3.2 Geração do conjunto de Pools

Cada pool é composto pelas rotas modelos das entregas de um cluster do modelo de clusterização inicialmente criado com k clusters. Cada rota modelo é uma rota encontrada a partir do algoritmo que busca solucionar o CVRP.

Com isso, é feito a predição para identificar o cluster de cada entrega e assim separar as entregas do histórico no array do cluster identificado. Após a separação, cada array é repartido em lotes com uma quantidade de entregas pré-definida B. Para cada repartição, o algoritmo referente ao *CVRP* é chamado para buscar soluções nas entregas presentes daquela repartição. Com isso, as rotas modelos encontradas são armazenadas nos pools referente ao seu cluster.

3.3 Filtragem das Rotas Modelos

O algoritmo de filtragem das rotas modelos busca remover as rotas modelos que contêm entregas em quadrantes poucos frequentes.

No algoritmo desenvolvido, utilizamos a biblioteca *s2phere* que contém uma parte da biblioteca *geometry S2* implementada em *Python* para o mapeamento das células s2, com nível 12. Somamos a quantidade de entregas que estão situadas em cada quadrante e, de forma percentual e decrescente, ordenamos tais quadrantes. Depois disso, buscamos definir um percentual de corte, isto é, um valor que se o percentual do quadrante for menor o mesmo será considerado um quadrante pouco frequente. Com isso, há análise dos quadrantes em que cada rota modelo passa, caso identifique um quadrante considerado pouco frequente, a rota modelo será adicionada na lista de rotas a serem removidas daquele conjunto de pools. Por fim, antes de haver a remoção, analisamos se a quantidade de rotas modelos a serem removidas do pool é igual a quantidade de rotas presente naquele pool, caso seja igual, o valor do percentual de corte é diminuído e o processo de análise de quadrantes para selecionar as rotas modelos é reiniciado, caso não seja igual, as rotas modelos selecionadas são removidas.

3.4 Distribuição das Unidades de Carregamento

A distribuição das Unidades de Carregamento tem como objetivo fornecer a quantidade de unidades de carregamentos para cada cluster. Ao utilizar as entregas do histórico e separar tais entregas em seus determinados clusters, a distribuição é feita de forma proporcional à quantidade de entregas presentes nesses clusters em relação a quantidade total de entregas. Com isso, multiplicamos cada percentual pelo número de unidades de carregamento. Por ser um número inteiro é necessário depois fazer o balanceamento para que a quantidade de unidades de carregamento distribuídas seja igual ao número de unidades de carregamento inicial, além de buscar garantir que cada cluster tenha, no mínimo, uma unidade de carregamento. O balanceamento é feito removendo uma unidade de carregamento, por iteração, do primeiro cluster que tem mais de uma unidade de carregamento.

3.5 Armazenamento de Centróides

O algoritmo de armazenamento de centróides busca armazenar centróides a partir de modelos de clusterização tendo como objetivo designar cada centróide em uma unidade de carregamento. A partir dela, poderemos tratar as unidades de carregamento de forma diferente caso as mesmas ainda não tenham entregas alocadas. A partir do modelo com k clusters, é criado k modelos de clusterização, onde cada modelo terá a quantidade de clusters definida pelo algoritmo de distribuição de unidades de carregamento. Cada modelo com M_i

clusters terá M_i centróides. Com isso, armazenamos cada centróide em sua determinada unidade de carregamento.

3.6 Alocação das Entregas nas Unidades de Carregamento

O processo de alocação das entregas é feito a partir da análise das entregas em relação às rotas modelos e aos quadrantes presentes, e é organizado em três etapas:

(1) Análise do ponto da entrega atual em relação à rota modelo atual de cada unidade de carregamento.
(2) Análise do quadrante que o ponto da entrega atual está presente em relação aos pontos das outras entregas já alocadas.
(3) Cálculo do centróide ou cálculo do ponto da entrega atual em relação aos pontos da rota modelo escolhida e das entregas já alocadas em cada unidade de carregamento.

No Algoritmo 1, é mostrado o pseudocódigo referente ao cálculo da menor distância entre uma entrega e um conjunto de entregas.

Algorithm 1 Pseudocódigo referente ao cálculo da distância entre uma entrega e um conjunto de entregas

```
Require: delivery : Delivery, deliveries : [Delivery]
Ensure: dist_min ← inf
  for d in deliveries do
    dist ← distance(delivery.point, d.point)
    if dist < dist_min then
      dist_min ← dist
    end if
  end for
```

3.6.1 Análise das rotas modelos atuais: É verificado se o ponto da entrega atual está presente na rota modelo atual de alguma unidade de carregamento. Caso esteja, já adicionamos a entrega naquela unidade de carregamento.

No Algoritmo 2, é mostrado o pseudocódigo referente à análise das rotas modelos atuais. A entrada consiste da entrega atual, um array referente as unidades de carregamento do cluster identificado a partir da predição do ponto da entrega no modelo de clusterização.

Algorithm 2 Pseudocódigo referente ao análise das rotas modelos atuais

```
Require: delivery : Delivery, UCS_c : [UC]
Ensure: phi ← False, dist_min ← inf
  for UC in UCS_c do
    if delivery in UC.model_route then
      uc ← UC
      phi ← True
      model_route ← UC.model_route
      break
    end if
  end for
```

3.6.2 Análise dos quadrantes: É verificado se a entrega atual está no mesmo quadrante de alguma entrega já alocada. Caso esteja, alocamos na unidade de carregamento de menor distância. Esse algoritmo só será executado caso não esteja presente em nenhuma rota modelo atual.

No Algoritmo 3, é mostrado o pseudocódigo referente à análise dos quadrantes presentes nas entregas já alocadas e o quadrante da entrega atual. A entrada consiste da entrega atual, um array referente as unidades de carregamento do cluster, identificado a partir da predição do ponto da entrega no modelo de clusterização, e das células S2. A função *equalQuadrant* busca identificar a presença de quadrantes em comum entre o ponto da entrega atual e os pontos das entregas já alocadas daquela unidade de carregamento.

Algorithm 3 Pseudocódigo referente à análise dos quadrantes

```
Require: delivery : Delivery, UCS_c : [UC], Cells : [S2Cells]
Ensure: s2cells ← False, dist_min ← inf
  for UC in UCS_c do
    for d_uc in UC.deliveries do
      if equalQuadrant(delivery, d_uc) then
        dist ← distance(delivery.point, d_uc.point)
        if dist < dist_min then
          dist_min ← dist
          uc ← UC
          s2cells ← True
          model_route ← UC.model_route
        end if
      end if
    end for
  end for
```

3.6.3 Cálculo do centróide e da rota modelo: No Algoritmo 4, é mostrado o pseudocódigo referente ao cálculo do centróide e da rota modelo. A entrada consiste da entrega atual, um array referente as unidades de carregamento do cluster, identificado a partir da predição do ponto da entrega no modelo de clusterização, e do Pool referente ao cluster.

Caso o ponto da entrega atual não esteja nem nas rotas modelos atuais e nem no mesmo quadrante de alguma entrega já alocada, então a análise e escolha da unidade de carregamento é feita de duas maneiras:

(1) Se a unidade de carregamento estiver vazia, calculamos a distância da entrega atual para o centróide dela que definimos antes da alocação.
(2) Se houver entregas na unidade de carregamento, escolhemos a rota modelo, a partir do somatório das distâncias mínimas entre as entregas alocadas e as entregas das rotas modelos, e depois fazemos uma cópia das entregas junto das entregas da unidade de carregamento. Por fim, escolhemos a unidade de carregamento com a menor distância encontrada entre a entrega atual e as entregas da cópia feita.

A partir dos Algoritmos 1, 2, 3 e 4, podemos organizar o pseudocódigo do algoritmo de alocação das entregas nas unidades de carregamento, mostrado no Algoritmo 5. A entrada consiste de um array com as entregas a serem alocadas, um array referente

Algorithm 4 Pseudocódigo referente ao cálculo do centróide e da rota modelo:

Require: $delivery : Delivery, UCS\_c : [UC], pool : Pool$
Ensure: $dist\_min \leftarrow inf$
  for UC in UCS_c **do**
    if $UC.size = 0$ **then**
      $dist \leftarrow distance(delivery.point, UC.centroide.point)$
    end if
    if $UC.size > 0$ **then**
      $model\_route\_esc \leftarrow modelRoute(pool, UC, delivery)$
      $copy\_deliveries \leftarrow model\_route.deliveries$
      $copy\_deliveries.push(UC.deliveries)$
      $dist \leftarrow distDelivery(delivery, d\_copy.deliveries)$
    end if
    if $dist < dist\_min$ **then**
      $dist\_min \leftarrow dist$
      $uc \leftarrow UC$
      $model\_route \leftarrow model\_route\_esc$
    end if
  end for

à distribuição das unidades de carregamento, um array referente as unidades de carregamento, o modelo K-Means, um array referente aos pools, contendo as rotas modelos, e as células S2, para identificar os quadrantes.

Algorithm 5 Pseudocódigo referente à alocação das entregas nas unidades de carregamento

Require: $deliveries : [Delivery], D : [int], UCS : [UC], Model : KMeans, P : [Pool], Cells : [S2Cells]$
Ensure: $routes \leftarrow []$
  for $delivery$ in $deliveries$ **do**
    $UCS\_c \leftarrow []$
    $cluster \leftarrow Model.predict(delivery.point)$
    for i in range(0,$len(D)$) **do**
      if $D[i] = cluster$ **then**
        $UCS\_c.push(UCS[i])$
      end if
    end for
    $phi, uc, model\_route \leftarrow ARMA(delivery, UCS\_c)$
    if phi not $True$ **then**
      $s2cells, uc, model\_route \leftarrow AQ(delivery, UCS\_c, Cells)$
    end if
    if phi not $True$ and $s2cells$ not $True$ **then**
      $uc, model\_route \leftarrow CRM(delivery, UCS\_c, P[cluster])$
    end if
    if $(UCS[uc].size + delivery.size) > capacity\_vehicles$ **then**
      $routes.push(UCS[uc].deliveries)$
      $UCS[uc].pop(UCS[uc].deliveries)$
    end if
    $UCS[uc].push(delivery)$
    $UCS[uc].model\_route \leftarrow model\_route$
  end for

3.7 Organização das rotas

O despacho de uma unidade de carregamento acontece quando a capacidade máxima da mesma é atingido. Com isso, é executado o algoritmo que busca uma solução para o *TSP(Traveling Salesman Problem)* em cada conjunto de entregas. No fim, o mesmo processo de despacho acontece nas unidades de carregamento que não ultrapassaram a capacidade, executando o determinado algoritmo para organizar as rotas.

4 RESULTADOS PARCIAIS/PRELIMINARES

O algoritmo do projeto foi desenvolvido na linguagem de programação *Python*. Foi utilizado o método *K-Means*, da biblioteca *Sklearn*, como método de clusterização, além da utilização da biblioteca *Numpy*. Houve a instalação do servidor *OSRM* para utilizar as distâncias do *OpenStreetMap* e a aplicação do algoritmo implementado com a ferramenta *OR-Tools*, do *Google*, que busca resolver o *Traveling Salesman Problem (TSP)* e *Capacitated Vehicle Routing Problems (CVRP)* disponibilizado pelo repositório *loggibud* [4], da *Loggi*, presente na plataforma *GitHub*.

Como forma de avaliação do algoritmo desenvolvido foi utilizado as instâncias cvrp, */train* e */dev*, dos diretórios *df-0*, *pa-0* e *rj-0* presentes no banco de dados disponibilizado pela *Loggi*. A pasta */train*, contém os dados que utilizamos como o histórico, distribuídos em 90 instâncias. A pasta */dev* é utilizada para gerar os resultados, de modo que ela contém 30 instâncias de testes.

O seguinte formato foi usado para encontrar um resultado a ser considerado geral de cada diretório:

(1) Inicialmente, separamos as entregas das instâncias de treino em batches (ou lotes) para que o algoritmo referente ao CVRP consiga encontrar uma solução. O tamanho do batch escolhido para os resultados foi 300, pois foi o tamanho que o algoritmo do arquivo */loggibud/v1/baselines/shared/ortools.py*, do repositório *loggibud*, consegue resolver em tempos razoáveis.
(2) Buscamos a distância percorrida das soluções encontradas para cada instância. As distâncias percorridas encontradas denominamos em resultados *off-line*.
(3) Executamos os algoritmos, de modo que a pegamos a distância percorrida das soluções geradas, utilizando a função *evaluate_solution*, presente no repositório *loggibud*, no arquivo */loggibud/v1/eval/task1.py*, e fazemos o cálculo da diferença em relação ao resultado obtido de cada instância do passo 1.
(4) Fazemos o cálculo da média dos percentuais, somando o cálculo da diferença encontrada entre o determinado algoritmo e o algoritmo que busca resolver o CVRP de cada instância (off-line) e dividimos pela quantidade de instâncias presente no diretório. O resultado do cálculo da média será considerado o resultado geral do diretório.

Como forma de comparação, utilizamos os resultados encontrados após a execução de dois algoritmos presentes no repositório *loggibud*, na pasta */loggibud/v1/baselines/task2*: *kmeans_greedy.py* e *qrp_sweep.py* com os valores de k no intervalo de 4 a 10.

4.1 Diretório Pará (pa-0)

O diretório pa-0 contém mais de 25000 entregas em seu histórico. Por ter uma média próxima de 300 entregas por instância, consideramos esse diretório como um diretório com pequenas instâncias, em comparação com os outros dois diretórios utilizados. Para os testes, foram 14 unidades de carregamento no algoritmo referente ao algoritmo desenvolvido.

Tabela 1: Resultados encontrados do diretório pa-0.

N_clusters	Algoritmo	KMeans_greedy	QRP_sweep
4	**17.25%**	**16.34%**	41.30%
5	18.62%	17.22%	24.61%
6	24.75%	23.89%	33.68%
7	25.47%	23.17%	34.86%
8	24.02%	22.44%	30.91%
9	47.92%	25.34%	25.99%
10	38.85%	26.72%	**20.79%**

A Tabela 1 apresenta os resultados referentes a instância pa-0. Nesta tabela, as colunas 2, 3, 4. representam os resultados, para cada quantidade de cluster, dos métodos: *algoritmo desenvolvido*, *kmeans_greedy.py* e *qrp_sweep.py*. Em negrito, destacamos os melhores resultados de cada método.

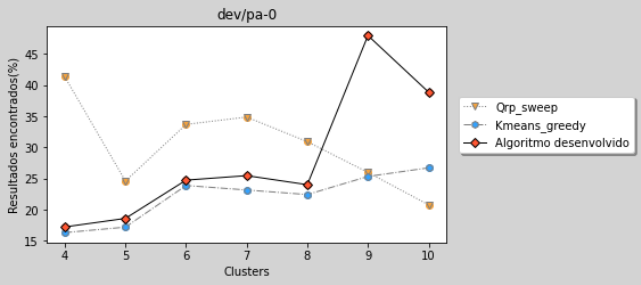

Figura 2: Resultados encontrados do diretório pa-0.

A Figura 2 busca mostrar o comportamento dos resultados encontrados de pa-0 dos três métodos no intervalo de número de clusters utilizado.

4.2 Diretório Distrito Federal (df-0)

O diretório df-0 contém mais de 88000 entregas em seu histórico. Por ter uma média próxima de 1000 entregas por instância, consideramos esse diretório como um diretório com instâncias médias, em comparação com os outros dois diretórios utilizados. Para os testes, foram 28 unidades de carregamento no algoritmo referente ao algoritmo desenvolvido.

De forma similar à Tabela 1 e à Figura 2, temos a Tabela 2 e a Figura 3, respectivamente. Eles apresentam os resultados os resultados encontrados e seu comportamento no intervalo de 4 a 10 clusters.

Tabela 2: Resultados encontrados do diretório df-0

clusters	Algoritmo	KMeans_greedy	QRP_sweep
4	16.97%	18.21%	25.56%
5	17.22%	17.22%	20.53%
6	17.74%	**12.04%**	23.30%
7	**16.27%**	15.41%	19.92%
8	19.42%	16.88%	18.19%
9	19.17%	16.92%	**18.09%**
10	19.65%	17.68%	21.56%

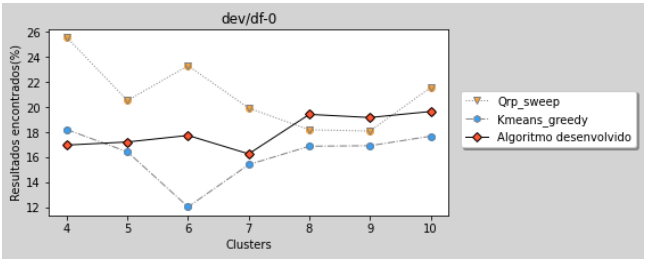

Figura 3: Resultados encontrados do diretório df-0.

4.3 Diretório Rio de Janeiro (rj-0)

O diretório rj-0 contém mais de 320000 entregas em seu histórico. Por ter uma média próxima de 3500 entregas por instância, consideramos esse diretório como um diretório com instâncias grandes, em comparação com os outros dois diretórios utilizados. Para os testes, foram 28 unidades de carregamento no algoritmo referente ao algoritmo desenvolvido.

Tabela 3: Resultados encontrados do diretório rj-0.

N_clusters	Algoritmo	KMeans_greedy	QRP_sweep
4	18.28%	25.67%	23.56%
5	18.78%	25.17%	23.87%
6	18.32%	24.93%	21.46%
7	18.60%	23.87%	21.44%
8	**18.17%**	**23.00%**	20.47%
9	18.99%	23.86%	**19.31%**
10	20.00%	23.56%	19.80%

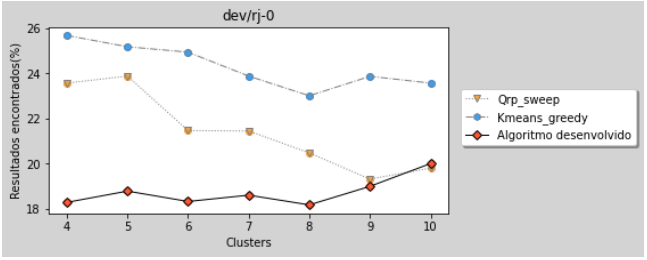

Figura 4: Resultados encontrados do diretório rj-0.

De forma similar à Tabela 1 e à Figura 2, temos a Tabela 3 e a Figura 4, respectivamente. Eles apresentam os resultados os resultados encontrados e seu comportamento no intervalo de 4 a 10 clusters.

5 CONCLUSÃO

Neste projeto trouxemos uma alternativa de seleção de rotas modelos, com uso da identificação de quadrantes iguais, com o objetivo de ser uma alternativa para resolver o *Last-Mile Incremental Capacitated Vehicle Routing (Last-Mile ICVRP)*. Os resultados parcialmente obtidos mostraram que ao ser comparado com os outros dois algoritmos executados, o algoritmo desenvolvido teve bons resultados nos diretórios que contém instâncias consideradas pequenas ou médias, $pa-0$ e $df-0$, e teve melhores resultados no diretório que contém instâncias grandes, $rj-0$.

6 TRABALHOS EM ANDAMENTO E FUTUROS

Na análise do desenvolvimento do algoritmo, entre os possíveis próximos trabalhos, pode-se relacionar:

(1) Nova forma de gerar o conjunto de pools contendo as rotas modelos.
(2) Desenvolvimento de um novo método de filtrar as melhores rotas modelos.
(3) Desenvolvimento de um novo método de distribuição das unidades de carregamento.
(4) Desenvolvimento de novo método de tratamento das rotas modelos.
(5) Desenvolvimento de novo método de tratamento de unidades de carregamento sem entregas alocadas.

Em relação aos trabalhos em andamento, está sendo estudado uma maneira de melhorar a escolha das rotas modelos no processo de alocação das unidades de carregamento.

REFERÊNCIAS

[1] Dimitris Bertsimas, Philippe Chervi, and Michael Peterson. 1995. Computational approaches to stochastic vehicle routing problems. *Transportation science* 29, 4 (1995), 342–352.

[2] Juan Camilo Fonseca-Galindo, Gabriela de Castro Surita, José Maia Neto, Cristiano Leite de Castro, and André Paim Lemos. 2020. A Multi-Agent System for Solving the Dynamic Capacitated Vehicle Routing Problem with Stochastic Customers using Trajectory Data Mining. *arXiv preprint arXiv:2009.12691* (2020).

[3] Chrysanthos E Gounaris, Panagiotis P Repoussis, Christos D Tarantilis, Wolfram Wiesemann, and Christodoulos A Floudas. 2016. An adaptive memory programming framework for the robust capacitated vehicle routing problem. *Transportation Science* 50, 4 (2016), 1239–1260.

[4] Loggi. 2021. loggiBUD: Loggi Benchmark for Urban Deliveries. https://github.com/loggi/loggibud.

[5] Warren B Powell. 1988. A Comparative review of alternative algorithms for the dynamic vehicle allocation problem. In *Vehicle routing: methods and studies. (Studies in Management Science and Systems, Volume 16)*, B. Golden and A. Assad (Eds.). Amsterdam ; New York : North-Holland, 249–291.

[6] Ulrike Ritzinger, Jakob Puchinger, and Richard F Hartl. 2016. A survey on dynamic and stochastic vehicle routing problems. *International Journal of Production Research* 54, 1 (2016), 215–231.

[7] Gabriela Surita and José C. Maldonado. 2020. Evolução das Soluções e Experiência de Roteirização na Loggi. https://www.youtube.com/watch?v=IDmaN7ima7k. [Online; acessado em 28 Março 2021].

[8] Barrett W Thomas and Chelsea C White Iii. 2004. Anticipatory route selection. *Transportation Science* 38, 4 (2004), 473–487.

[9] Marlin W Ulmer, Justin C Goodson, Dirk C Mattfeld, and Marco Hennig. 2019. Offline–online approximate dynamic programming for dynamic vehicle routing with stochastic requests. *Transportation Science* 53, 1 (2019), 185–202.