LIII Simpósio Brasileiro de Pesquisa Operacional
João Pessoa - PB, 3 a 5 de novembro de 2021

A Hybrid Approach to Time-Dependent Vehicle Routing Problem with Time Windows in Green Logistic

Igor de Andrade Junqueira, Lorenza Leão Oliveira Moreno, Luciana Brugiolo Gonçalves, Stênio Sã Rosário F. Soares
Universidade Federal de Juiz de Fora
Rua José Lourenço Kelmer - Campus Universitário, Juiz de Fora - MG
{igor.andrade.junqueira, lorenza, lbrugiolo, ssoares}@ice.ufjf.br

RESUMO

O problema de roteamento verde de veículos com dependência de tempo, frota heterogênea e janela de tempo é um problema em que o objetivo é minimizar a emissão de CO_2 considerando as variações de velocidade em cada via ao longo do tempo. Este trabalho propõe um método híbrido eficiente que combina as metaheurísticas GRASP e ILS, que incluem também RVND e *Local Branching*. A abordagem proposta foi testada com um conjunto de instâncias e os resultados obtidos, se comparados aos da literatura, mostram sua competitividade com ganhos expressivos no tempo de processamento, mantendo a qualidade das soluções.

PALAVRAS CHAVE. Pollution Routing Problem, Green Logistic, GRASP-ILS-RVND-MIP.

L&T – Logística e Transportes, MH – Metaheuristicas, IC – Inteligência Computacional

ABSTRACT

The Green Vehicle Routing Problem with heterogeneous fleet and time windows approached in this work aims to reduce CO_2 emissions by considering time-varying traffic conditions to deliver goods to a set of customers. Our proposal combines GRASP and ILS metaheuristics, as well RVND and Local Branching matheuristic to solve the problem. Experiments showed that the algorithm is able to achieve high quality solutions three times faster than other literature results.

KEYWORDS. Pollution Routing Problem, Green Logistic, GRASP-ILS-RVND-MIP.

L&T – Logística e Transportes, MH – Metaheuristicas, IC – Inteligência Computacional

1. Introdução

Nos últimos anos tem sido cada vez mais frequente a busca por parte de entidades governamentais e do setor industrial por iniciativas que considerem aspectos ambientais e sustentabilidade Macrina et al. [2020]. O efeito estufa é um processo natural gerado por gases do efeito estufa (GEE), como dióxido de carbono (CO_2), metano (CH_4), óxido nitroso (N_2O) e Compostos perfluorados ($PFC's$) na atmosfera, que aquecem a superfície do planeta [Le Treut et al., 2007]. Por outro lado, o aquecimento global é o efeito estufa acelerado pela ação humana, como a queima de combustíveis fósseis, gerando CO_2, por exemplo.

A logística verde, ao contrário da logística tradicional, considera o impacto ambiental ao buscar soluções para problemas de roteamento de veículos, como minimizar: emissão de GEE em problemas de roteamento de veículos a combustão, otimizar rotas com uso de veículos elétricos considerando a autonomia de baterias, otimizar o uso de energia elétrica em processos de manufatura, minimizar o desperdício de recursos etc. Segundo [Paskannaya e Shaban, 2019], diminuir a emissão de CO_2 é um dos principais objetivos da logística verde.

Uma vez que a logística do transporte impacta significativamente no meio ambiente, pesquisas nesta área vêm se intensificando no mundo inteiro. O Problema de Roteamento Verde de Veículo (G-VRP) (do inglês *Green Vehicle Routing Problem*) é uma variação do Vehicle Routing Problem (VRP) que engloba as restrições da logística de transporte tradicional, como janela de tempo do cliente, coleta e entrega, único ou múltiplos depósitos etc além das restrições associadas à sustentabilidade.

Em [Macrina et al., 2020] os autores apresentam uma classificação do G-VRP bem como das abordagens propostas para as diversas variações do problema no período de 2011 a 2019. Por esta classificação, existem duas classes de G-VRP: os problemas que tratam de veículos à combustão, escopo deste trabalho, e os que tratam de veículos que usam outros combustíveis.

Bektas e Laporte, em [Bektas e Laporte, 2011] apresentaram uma formulação para o problema de roteamento em cuja função objetivo considera a emissão de poluentes e propuseram a denominação *Pollution Routing Problem* - PRP para problemas que tratam de roteamento de veículos à combustão. Em [Figliozzi, 2011] foi introduzido o *time-dependent VRP with time windows* (T-DVRPTW), onde se considera a quantidade de CO_2 emitida conforme as condições do tráfego ao longo de diferentes período do dia. [Küçükoğlu et al., 2015], utiliza um algoritmo para calcular o consumo de combustível para o G-VRPTW (problema de roteamento verde de veículos com janela de tempo) onde se utiliza o consumo de combustível como função objetivo. O consumo de combustível leva em consideração a aceleração e desaceleração de um veículo até atingir uma velocidade constante, a distância, o peso do veículo e as especificações do veículo. Uma abordagem *Simulated Annealing* (SA) é apresentada, que utiliza uma memória de rotas, para guardar o consumo das rotas já calculadas.

Uma variação do problema em que são considerados múltiplos depósitos foi tratada em [Shen et al., 2018], onde a função objetivo é minimizar os custos como salario dos motoristas, penalidade da janela de tempo (adiantar ou atrasar a entrega), custo do combustível, além do custo associado à emissão CO_2. Os autores apresentam Um *PSO (Particle Swarm Optimization)* e uma Busca Tabu (*TS*) para resolver o problema. Ainda sobre a variação com múltiplos depósitos, o trabalho de [Wang et al., 2019] considera frota homogênea e janela de tempo com penalização por atrasos e antecipações. A função objetivo busca minimizar a poluição e o custo das penalidades. Os autores propõem um *multi-objective particle swarm optimization* (MOPSO).

Já em [Rezaei et al., 2019], os autores descrevem um G-VRP multiobjetivo que busca minimizar os custos de transporte e de emissão de CO_2 e consideram frota heterogênea quanto à

capacidade, custo e taxa de consumo de combustível, além de janela de tempo. No problema proposto, os veículos podem ser abastecidos ao longo da rota e devem atender um dado valor máximo de tempo de viagem. O cálculo do CO_2 depende da distância percorrida e da carga do veículo. os autores Utilizaram um Algoritmo genético e um algoritmo *Simulated Annealing* baseado em população.

Sobre o *time-dependent VRP with time windows* (T-DVRPTW), [Xiao e Konak, 2016] aborda a versão do problema com frota heterogênea. Sobre variação de velocidade ao longo dos períodos de tempo, eles utilizam valores tabelados de velocidades para cada período de cada arco que liga dois clientes. O problema é modelado sobre um multigrafo direcionado $D = (V, A)$ onde são permitidas paradas do veículo ao longo do trajeto sobre os arcos (i, j), de forma que se admite a mudança de velocidade em um mesmo arco sempre que há uma mudança de um período para outro ao longo do trajeto. Para resolver o problema, os autores propuseram um método que utiliza *Variable Neighbour Source* (VNS) e Programação Linear Inteira. Em [Rylan et al., 2017] os autores estendem o problema descrito em [Xiao e Konak, 2016] para permitir que um cliente possa ser atendido por múltiplos veículos (*Split Delivery*) e apresentam um ILS *mult start* com a busca local baseada em R-VND. O presente trabalho trata do T-DVRPTW sem *Split Delivery*.

Neste trabalho é tratado o problema de roteamento verde de veículos com frota heterogênea abordado em [Xiao e Konak, 2016]. As principais contribuições do trabalho são:i) Uma heurística de verificação de viabilidade e inserção de tempo de parada nos clientes que permite a construção de rotas mais otimizadas quanto ao consumo de combustível, implicando em menor emissão de CO_2 é apresentada; ii) Uma heurística baseada no MIP que permite a inserção de tempo de parada em qualquer ponto da rota (gerando rotas melhores que a heurística) e permite pequenas mudanças na ordem de visitação dos clientes;iii) Uma hash de rotas é apresentada que permite a utilização de rotas geradas pelo MIP durante a busca local.

O restante do trabalho está assim organizado: a Seção 2 apresenta a descrição formal do problema; Os métodos computacionais são apresentados na Seção 3, enquanto os resultados e a comparação com a literatura são apresentados na Seção 4. Por último, as conclusões do trabalho são apresentados na Seção 5.

2. Descrição do Problema

O problema T-DVRPTW é descrito em detalhes em [Xiao e Konak, 2016] e é definido sobre um grafo orientado $G = (V, A)$ onde $V = \{0\} \cup C$, sendo $C = \{1, \dots n\}$ é o conjunto clientes e 0 o depósito; A é o conjunto de arcos $(i, j) \mid i \neq j$ e $i, j \in V$. A cada arco $(i, j) \in A$ está associada uma distância $d_{i,j}$ relativa ao caminho que liga i e j e a cada nó $i \in C$ está associada uma demanda r_i, uma janela de tempo $[s_i, e_i]$ e um tempo de atendimento ta_i. Seja $P = \{[i_1, f_1], [i_2, f_2], \dots [i_p, f_p]\}$ o conjunto que define o horizonte de planejamento, onde i_k e f_k indicam, respectivamente, o início e fim do k-ésimo período. Seja s_{ij}^k a velocidade de qualquer veículo sobre cada arco $(i, j) \in A$ no período $k \in P$. Considere ainda $U = \{1, \dots u\}$ o conjunto de tipos de veículos da frota. Para todo tipo de veículo $t \in U$, conhece-se a capacidade de carga C_t, e o limite de capacidade do tanque de combustível L_t. Além disso, para cada tipo t de veículo, a taxa c_t de conversão de combustível para CO_2 é conhecida e permite, a partir de uma função que calcula do consumo de combustível conforme a velocidade e o peso total da carga, que seja obtido o total de emissão de CO_2 em qualquer arco (i, j) em qualquer período. A cada veículo t da frota, está associada uma janela de tempo $[v_{ini}, v_{fim}]$, que define o intervalo de tempo em que o veículo está disponível para uso.

O problema consiste em definir uma sequência de atendimento dos clientes pelos veículos e os possíveis instantes de paradas ao longo da rota de forma a atender às restrições de janela de

tempo e demanda dos clientes, do período de disponibilidade, da capacidade de carga e da autonomia de combustível dos veículos que minimize o total de emissão de CO_2 pelos veículos usados.

Para evitar congestionamentos e consequentemente aumento na emissão de CO_2, o problema tratado permite paradas tanto nos clientes como entre dois vértices i e j quaisquer. Ao percorrer um arco (i, j) a mudança de período tem que ser considerada, porque pode haver mudança de velocidade entre períodos consecutivos. Assim, se (i, j) pertence à solução, seja t_{ijk} o tempo total de viagem sobre o arco (i, j) no período k. a partir da velocidade s_{ijk} e da taxa de conversão do consumo do combustível em emissão de CO_2, é possível calcular exatamente o total de emissão de cada veículo ao longo de toda a sua rota.

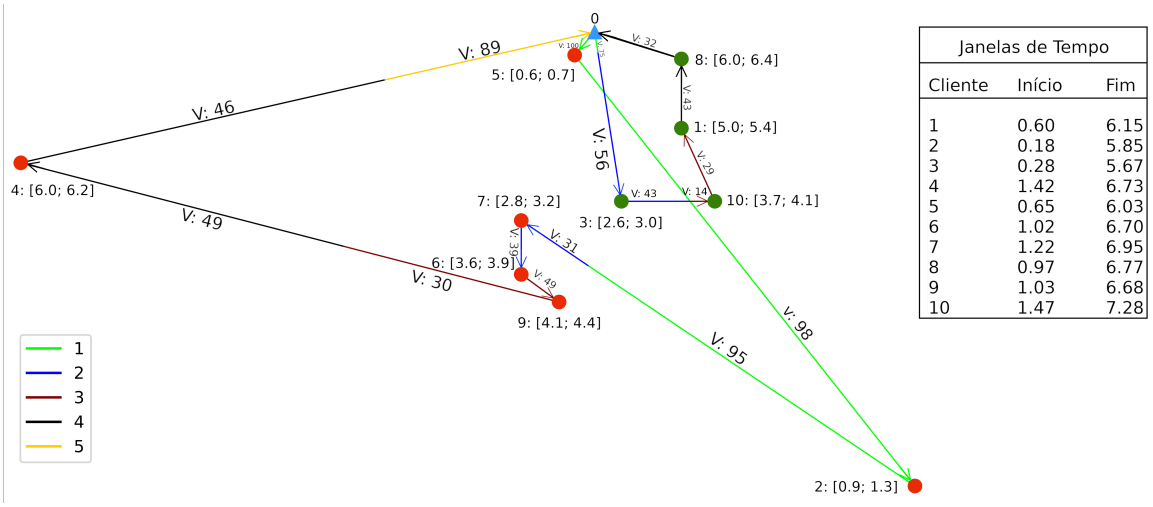


Figura 1: Exemplo de uma solução para UK_10x5_1

A Figura 1 apresenta uma solução para a instância UK_10x5_1 com 10 clientes. A velocidade é apresentada, em km/h, para cada arco e períodos utilizados. Na rota $\langle 0-3-10-1-8-0\rangle$ somete uma parte do período 1 é utilizado no arco $(0-3)$, já na rota $\langle 0-5-2-7-6-9-4-0\rangle$ o período 1 é utilizado nos arcos: $(0-5)$, $(5-2)$ e parcialmente, $(2-7)$. Essa diferença se deve ao fato de que a distância do arco $(0-3)$ representa 70% da distância de $\langle 0-5-2-7\rangle$, porque são utilizadas distâncias reais e com isso o grafo é assimétrico.

3. Métodos Computacionais

Nesta seção, são descritos os algoritmos propostos para o T-DVRPTW com frota heterogênea. Em linhas gerais, a estratégia utiliza duas metaheurísticas de forma sequencial: inicialmente, um GRASP (*Greedy Randomized Adaptive Search Procedure*) é utilizado para gerar diferentes soluções viáveis para o problema e, com a melhor solução encontrada, é inicializado um algoritmo ILS (*Iterated Local Search*).

Para o GRASP proposto foi utilizada uma heurística de construção e um algoritmo baseado no RVND (*Randomized Variable Neighborhood Descent*) para refinamento. Os dois métodos utilizam uma heurística de verificação de viabilidade e inserção de tempo de parada nos clientes.

Para o ILS, um RVND simplificado foi adotado, além de um algoritmo tipo *Local Branching*, baseada em modelos de programação linear inteira mista (MIPs), para otimização das rotas e inserção de tempo de parada nos arcos.

3.1. Heurística construtiva

O algoritmo construtivo é baseado em uma heurística de inserção mais barata. Inicialmente, todos os veículos têm rotas vazias e os clientes são inseridos em uma lista de candidatos, LC. A cada iteração, um cliente de LC é selecionado, removido da lista e inserido em uma rota, conforme descrição a seguir.

Para cada cliente em LC, verifica-se a possibilidade de este ser inserido em cada posição da rota atual de cada veículo da solução corrente. A viabilidade das inserções é determinada através do algoritmo descrito na Seção 3.1.1. Caso não exista uma rota viável para determinado cliente, considera-se que este pode ser inserido na rota de um veículo fictício. Entre as inserções viáveis de um mesmo cliente, apenas a opção que apresenta a menor distância percorrida para atendê-lo é armazenada.

Os clientes em LC são ordenados de acordo com um dos seguintes critérios: poluição gerada ou distância percorrida pelas respectivas rotas armazenadas (o critério é um dos parâmetros do algoritmo). Os $\alpha \cdot |LC|$ clientes melhor avaliados são identificados e, aleatoriamente, um deles é selecionado. O cliente selecionado é removido da lista e inserido na rota previamente armazenada.

3.1.1. Algoritmo de verificação de viabilidade e inserção de tempo de parada nos clientes

O algoritmo descrito nesta seção verifica a viabilidade de uma rota e, ao mesmo tempo, adiciona tempo de parada nos clientes.

O primeiro passo é analisar a viabilidade no que se refere às janelas de tempo dos clientes. Para isso, considerando uma rota r sem paradas, verifica-se a possibilidade de atender os clientes dentro de suas janelas de tempo.

Considerando o tempo de retorno ao depósito após a visitação dos clientes, calcula-se o tempo de saída mais tarde para cada um destes clientes. Esta informação será importante para definição dos tempos de espera.

Para cada trecho da rota, partindo do depósito, são testadas diferentes opções de percurso do trecho com variação nos tempos de saída (de k em k instantes). Cada opção terá um tempo de chegada distinto no próximo cliente, sendo possível descartar as opções que não respeitam sua janela de atendimento. Também é possível descartar opções consecutivas que tem o mesmo custo, isto é, causam a mesma poluição e gasto de combustível, sendo que, neste caso, apenas o percurso com saída mais cedo é mantido. Ao fim desta iteração, temos, portanto, algumas opções de percurso para a mesma aresta da rota original, com tempos e custos distintos, todas respeitando a janela de atendimento do cliente.

Na iteração seguinte, o próximo trecho do percurso também será replicado em diferentes instantes de tempo. Entretanto, para cada possível opção de percurso, são verificados os percursos do trecho anterior finalizados até o respectivo instante de saída. O percurso anterior de menor custo é selecionado e a poluição e o gasto de combustível são acumulados ao longo do caminho. O algoritmo continua da mesma forma, criando as opções de percurso para o trecho atual, mas sempre com os dados de custo acumulados. As opções com tempo de chegada fora da janela de tempo do cliente e as opções dominadas (posteriores e de custo maior) são descartadas e o próximo trecho passa a ser analisado, até o fim da rota. Ao final, entre os percursos que retornam ao depósito, é selecionado aquele cujo custo acumulado é menor.

3.2. *Local Branching*

Local branching (LB) é um algoritmo de busca local proposto por [Fischetti e Lodi, 2003], que utiliza um modelo de programação linear inteira (MIP) para encontrar novas soluções viáveis para determinado problema. Uma restrição é inserida no MIP para restringir o espaço de soluções do problema a uma vizinhança em torno de uma solução de referência, usualmente, limitando o número

de modificações que podem ser realizadas nesta solução. A cada otimização em que uma nova solução é retornada, a solução de referência é substituída e a restrição é atualizada. Eventualmente, quando a otimização não é capaz de encontrar uma nova solução, o algoritmo atingiu um ótimo local.

Neste trabalho, um algoritmo de LB é implementado, utilizando o MIP definido por [Xiao e Konak, 2016] para o T-DVRPTW com frota heterogênea. O modelo, embora extenso e complexo, é reduzido para otimizar uma ou duas rotas por vez. Para isso, são fixadas em zero demandas e variáveis relacionadas a clientes que não são atendidos pelas rotas selecionadas, bem como variáveis relacionadas a outros veículos.

Para a criar a restrição de LB, a cada arco (i, j) das rotas selecionadas, a respectiva variável binária X_{ij} do modelo (que tem valor 1, se o arco (i, j) fizer parte da solução) é incluída em um conjunto S. A restrição (1) permite a modificação de até 3 arcos por iteração da busca local, ao obrigar que as soluções viáveis do modelo apresentem ao menos $|S| - 3$ variáveis do conjunto S.

$$\sum_{(i,j)\in S} X_{ij} \geq |S| - 3 \tag{1}$$

Vale ressaltar que o modelo matemático permite que os veículos realizem paradas em qualquer ponto da rota, inclusive durante o percurso de um arco - o que não é explorado em outros pontos do algoritmo.

Considerando o custo computacional de se otimizar as rotas pelo modelo matemático e que tais rotas podem ser geradas novamente no algoritmo, uma tabela hash é utilizada para armazenar as rotas, baseada na *hash djb2* criada por Dan Bernstein [Bernstein, 1991] para cadeias de caracteres. A busca local na Subseção 3.3 verifica se as rotas de uma solução encontram-se na hash, na tentativa de substitui-las por rotas similares já otimizadas, com menor poluição. A subseção 3.5 explica como o ILS utiliza o LB.

3.3. RVND

O VND (*Variable Neighborhood Descent*) é uma abordagem que permite a combinação de diferentes movimentos de busca local. Enquanto no VND é necessário indicar a ordem de exploração em cada uma das vizinahnças, no RVND (*Randomized Variable Neighborhood Descent*) isso não é necessário. Quando o RVND é chamado, define-se aleatoriamente a ordem que que as vizinhanças são analisadas.

No abordagem proposta foram utilizados movimentos clássicos normalmente aplicados em problemas de roteamento. Os movimentos foram aplicados considerandos clientes de uma mesma rota (intra) são shift, swap e 2-opt e envolvendo clientes de rotas distintas (inter) são shift, swap e cross. Também são utilizados os movimentos que realizam a inversão de rotas e troca do tipo de veículo de duas rotas.

A heurística descrita na subSeção 3.1 pode gerar soluções inviáveis. Considerando $\overline{C}$ o conjunto dos clientes que estão associados ao veículo fictício e uma taxa de penalização ρ, a função objetivo considerada inclui uma parcela $|\overline{C}| \times \rho$ para penalização da inviabilidade na função objetivo. A primeira parte da equação, $Tx^{CO_2} \times q$, consiste no computo do CO_2 emitido pelos veículos da solução.

$$\text{Minimizar} \qquad F = Tx^{CO_2} \times q + |\overline{C}| \times \rho \tag{2}$$

Na tentativa de reduzir a inviabilidade da solução, quando esta ocorre, o RVND fixa os movimentos *shift* e *swap* inter rotas como movimentos iniciais, visando a remoção dos clientes

da rota do veículo fictício. A ordem de execução dos demais movimentos é definida de maneira aleatória. Se não houver nenhum cliente no veículo fictício, a ordenação é totalmente aleatória para todas as vizinhanças consideradas.

Nos movimentos, a cada rota gerada, o algoritmo verifica que esta ocorre na tabela hash descrita na Subseção 3.2, ou seja, verifica-se se a rota passou pela otimização via *local branching*. Caso encontre, a rota otimizada substitui a rota corrente mesmo se a sequencia de visitação dos clientes for diferente.

3.4. GRASP Reativo

O GRASP (*Greedy Randomized Adaptive Search Procedure*) é uma metaheurística que combina construção e busca local num processo iterativo [Talbi, 2009]. A versão utilizada neste trabalho é baseada no trabalho [Prais e Ribeiro, 2000], onde o grau de aleatoriedade da heurística de construção é ajustada durante a execução do algoritmo, denominada GRASP Reativo.

Nesta abordagem um conjunto de valores de α são considerados durante a execução do algoritmo. Inicialmente, todos os valores tem a mesma probabilidade de serem escolhidos. Mas, a cada bloco de iterações, estas probabilidades são ajustadas de forma a favorecer a escolha dos valores de α responsáveis pelos melhores resultados.

Em cada iteração, uma solução é construída utilizando a estratégia apresentada na Seção 3.1. Inicialmente o critério de ordenação da Lista de Candidatos, parâmetro da Heurística Construtiva, é a poluição resultante da inserção do cliente da rota. Mas, quando o número de iterações sem melhora supera 150 iterações inicia-se uma fase de alternância no critério de ordenação da lista de candidatos. A cada bloco de 150 iterações o critério começa uma alternância entre o critério apresentado inicialmente e a distância percorrida.

A fim de refinar a solução obtida pela estratégia de construção, esta é refinada usando o RVND descrito na Seção 3.3.

3.5. ILS

A estratégia adotada neste trabalho combina as metaheurísticas ILS e GRASP, duas estratégias iterativas baseadas em busca local.

O ILS(*Iterated local search*), a partir de uma solução inicial, alterna fases de perturbação e busca local. Para obter uma solução de partida, o algoritmo GRASP, descrito na Seção 3.4, é utilizado. Para etapa de perturbação foi adotado o movimento 2 opt-swap inter rotas, apresentado na Seção 3.3 (a hash de rotas não é utilizada na pertubação). Para busca local foi usado o RVND descrito na Seção 3.3, mas retirando-se o movimento 2 opt-swap inter rotas, para evitar ciclos.

Os modelos descritos na Seção 3.2 são utilizados a cada k iterações do ILS. Uma solução é escolhida de forma aleatória a partir de uma lista de soluções e, caso suas rotas ainda não tenham sido otimizadas, são submetidas ao modelo utilizando uma rota. Para o modelo que utiliza duas rotas, um limite de chamadas é definido e as rotas são escolhidas de forma aleatória. Só há repetição de rotas se uma das duas for atualizada.

O Algoritmo 1 recebe como parâmetros: instância, número de iterações, número de iterações sem melhora, primeira chamada mip, número de iterações Grasp, intervalo mip (bloco de iterações para chamada do mip) e ϵ (o gap de 0.01 para solução corrente ser considerada igual a best). A linha 3 a solução inicial é gerada pelo Grasp da subseção 3.4, a linha 10 o mip é chamado para a solução best quando o número de interações chega na primeira chamada do mip.

A linha 13 a solução corrente é perturbada pelo movimento 2 opt-swap seguido do RVND' (não utiliza o movimento de perturbação). A linha 16 verifica a iteração atual é maior que a iteração para começar o mip. A função *atualizaLista* utiliza o *gap* para incluir a solução corrente se a mesma estiver suficientemente próxima da melhor solução. O fluxo de execução atinge o bloco

do *if* da linha 18 se a iteração atual for múltiplo de *intervaloMip*, a linha 19 escolhe uma solução aleatoriamente da lista de soluções ou retorna a solução corrente se *gap* for menor que ϵ. O *if* da linha 21 atualiza a solução ótima se a solução resultante do mip for melhor que *best*. A linha retorna a melhor solução se o número de interações sem melhora é maior do que o parâmetro número de iterações sem melhora, caso esse ponto não seja alcançado a linha 31 retorna a melhor solução.

Algoritmo 1: ILS

```
1   input: instancia, numIteracoes, itSemMelhora, itMip, itGRASP, intervaloMip, ϵ
2   output: solucao
3   best ← GRASP(instancia, itGRASP);
4   i ← 0; ultimaAtualizacao ← 0;
5   solCorrente ← best;
6   memoriaRotas ←∅; listaSolucoes ←∅;
7   while(i < numIteracoe)
8       if(i == itMip)
9           poluicao ← getPoluicao(best);
10          MIP(best, memoriaRotas);
11          atualizaSolucaoCorrente(best, poluicao, solCorrente, i, ultimaAtualizacao);
12      end if
13      solCorrente ← pertubacao(instancia, solCorrente);
14      rvnd'(instancia, solCorrente, memoriaRotas);
15      gap ← calculaGap(solCorrente, best);
16      if(i > itMip)
17          atualizaLista(listaSolucoes, solucaoCorrente, gap);
18          if((i%intervaloMip) = 0)
19              sol ← escolheSolucao(listaSolucoes, solCorrente, gap, ϵ);
20              MIP(sol, memoriaRotas);
21              if(atualizaSolucao(sol, best))
22                  best ← sol; solucaoCorrente ← sol; ultimaIt ← i;
23          end if
24      end if
25      if(atualizaSolucao(solucaoCorrente, best))
26          best ← solucaoCorrente; ultimaIt ← i;
27      if(verificaItSemMelhora(i, itMip, ultimaIt, itSemMelhora))
28          return best;
29      i++;
30  end while
31  return best;
```

4. Experimentos Computacionais

Nessa seção os testes computacionais são descritos. Os experimentos foram realizados sobre um conjunto o instâncias de [Demir et al., 2012], modificadas e disponibilizadas por [Xiao e Konak, 2016]. As instâncias são divididas em sete grupos com 10, 15, 20, 25, 50, 75 e 100 clientes, onde cada grupo possui 20 instâncias. O horizonte de planejamento integra nove horas e foi dividido em cinco períodos de 1.8 horas. A frota apresenta dois tipos de veículos, que diferem quanto às capacidade máxima de cargas e de combustível e quanto ao início e fim da janela de tempo. Pelo fato de se utilizar clientes como cidades e distâncias reais, a matriz com as distâncias é assimétrica e pode não existir um caminho direto entre duas cidades, onde considerou-se distância igual a infinito.

Para ajustar os parâmetros das abordagens propostas, utilizou-se o pacote *Irace* [López-Ibáñez et al., 2016]. Para cada parâmetro foi definida o conjunto de possíveis valores, a saber: *itGRASP*: $\{200, 500, 800\}$; para o ILS: *numIteracoes*: $\{2000, 2500, 3000, 3500\}$; *itSemMelhora*: $\{300, 500, 700, 900\}$; *itMip*: $\{500, 1000, 1400\}$; *intervaloMip*: $\{50, 100, 200, 400\}$; e número limite de chamadas para gerar duas rotas: $\{2, 4, 7, 10\}$. O número de execuções do código no *Irace* foi limitado a 1000. A melhor configuração encontrada pelo calibrador de parâmetro foi: número de iterações GRASP: 500; número de iterações ILS: 2500; número de iterações sem melhora ILS:

Tabela 1: Comparação entre os processadores com o *benchmark PassMark*

Métricas (MOps/S)	i5-8250U	i5-2400S	Razão
Operações de números inteiros	21.364	11.126	1,9
Operações de ponto flutuante	13.066	5.745	2,3
Single Thread	1.948	1.468	1,3

500; primeira chamada MIP: 500; intervalo para chamada MIP: 50; Número de chamadas para gerar duas rotas: 7.

Os parâmetros lista de alfas e número de iterações para atualizar a probabilidade referentes ao GRASP não foram calibrados pelo Irace. Para os valores de alfa, usou-se o conjunto $\{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$ e 150 para o número de iterações a cada atualização das probabilidades associadas aos valores de alfa candidatos.

4.1. Resultados

Os algoritmos foram desenvolvidos na linguagem *C++* e o modelo matemático implementado usando o resolvedor Gurobi [Gurobi Optimization, 2021]. Os testes foram executados em uma máquina Intel i5-8250U com frequência de 1,6 Ghz (*turbo* 3,4 Ghz), 8GB de memória, Sistema Operacional Debian 11 com kernel linux na versão 5.10.0-2amd64. Além disso, utilizou-se o compilador g++ na versão 10.2.1 e Gurobi na versão 9. [Xiao e Konak, 2016] utilizam uma maquina Intel i5-2400 com frequência base de 2,5 Ghz (*turbo* 3,3 Ghz), resolvedor CPLEX (versão 12.4.0.1) e linguagem de programação AMPL. As informações sobre sistema operacional, memória RAR e compilador não foram fornecidos pelos autores.

Na Tabela 1 são apresentadas métricas do *benchmark PassMark* para comparar as velocidades dos processadores. Utilizando o pior caso, operações de ponto flutuante, o processador utilizado nos testes desse trabalho é 2,3 vezes mais rápido do que o processador utilizado por [Xiao e Konak, 2016]. Essa informação permite uma comparação mais precisa dos tempos de processamento.

Cada instância foi executada 30 vezes. Para obter o resultado por grupo, o seguinte procedimento foi modificado de [Xiao e Konak, 2016]: cada instância é executada um número λ de vezes e calculam-se as seguintes métricas: médias de tempo e de poluição, o melhor tempo e o melhor valor de poluição para cada instância. Os resultados por grupos são obtidos pela média de cada uma destas métricas considerando todas as 20 instâncias do grupo. A diferença do experimento apresentado em [Xiao e Konak, 2016] é que os autores realizaram apenas 5 (cinco) execuções por instâncias nos grupos de 10 a 25 clientes, uma única execução para os grupos de 50 a 100 clientes e só o tempo dá média das soluções do grupo é apresentado, faltando o tempo médio da melhor solução obtida das 20 instâncias do grupo. A instância UK_15x5_11 é inviável, [Xiao e Konak, 2016] mostra isso pelo modelo matemático, todas as demais instâncias são viáveis. Os tempos de processamento da literatura foram convertidos usando a razão de 2,3 da tabela 1. Ou seja, para se obter uma estimativa de tempo de execução do P-MIP-INS no processador utilizado nesse trabalho, o tempo do mesmo foi dividido por 2,3.

A Tabela 2 mostra uma comparação entre os resultados obtidos pelo algoritmo descrito na Seção 3.5, coluna ILS_GRAS_MIP, com aqueles apresentados em [Xiao e Konak, 2016] (P-MIP-INS) e em [Rylan et al., 2017] (Raylan 2017) em relação ao melhor resultado obtido para o valor da função objetivo (coluna Poluição), onde a coluna GAP apresenta a diferença percentual em relação ao melhor resultado dentre as três abordagens. Os melhores resultados são destacados em negrito. Pode-se observar que o algoritmo ILS descrito nesse trabalho e o método descrito por

[Xiao e Konak, 2016] apresentaram resultados muito próximos para todas as instâncias, diferindo na média em apenas $0,1\%$ na média dos GAPs apresentados para o conjunto de instâncias, ao passo que abordagem de [Rylan et al., 2017] não se mostrou eficaz, tendo os piores resultados para seis dos sete grupos de instâncias.

Tabela 2: Média dos melhores resultados

Grupo	Melhor	P-MIP-INS		Raylan 2017		ILS_GRASP_MIP	
		Poluição	GAP	Poluição	GAP	Poluição	GAP
10	**557,9**	**557,9**	**0,0%**	597,9	7,2%	561,0	0,5%
15	**705,0**	746,1	5,8%	**705,0**	**0,0%**	745,9	5,8%
20	**954,3**	958,7	0,5%	1078,1	13,0%	**954,3**	**0,0%**
25	**936,0**	**936,0**	**0,0%**	1054,3	12,6%	937,7	0,2%
50	**1698,9**	1710,2	0,7%	2066,0	21,6%	**1699,0**	**0,0%**
75	**2483,5**	**2483,5**	**0,0%**	2932,3	18,1%	2487,2	0,2%
100	**3149,8**	**3149,8**	**0,0%**	3748,5	19,0%	3182,2	1,0%
Média	-	-	**1,0%**	-	13,1%	-	1,1%

A principal contribuição das abordagens propostas neste trabalho é fato de combinar um método baseado no modelo matemático com metaheurísticas sem comprometer a eficiência de tempo do algoritmo proposto. A Tabela 3 apresenta o tempo de processamento em minutos para os algoritmos P-MIP-INS e do ILS, já que os resultados apresentados em [Rylan et al., 2017] não trazem os tempos de processamento. Os resultados para o algoritmo P-MIP-INS foram retirados de [Xiao e Konak, 2016], sendo que para os grupo de até 25 clientes apresentam a média dos melhores resultados para cinco execuções por instância. Já para os grupos de 50, 75 e 100 clientes, os resultados referem-se à média do grupo considerando uma execução por instância. pode-se observar que o algoritmo ILS proposto apresenta desempenho 2,5 vezes melhor que a melhor abordagem da literatura, isso com um GAP de apenas $0,1\%$ acima em relação à qualidade das soluções apresentadas pelos dois algoritmos.

Tabela 3: Média dos melhores resultados tempo

Grupos	P-MIP-INS(Min)	P-MIP-INS Conversão(Min)	ILS_GRASP_MIP(Min)	*Speedup*
10	7,7	3,3	**0,9**	**3,7**
15	12,9	5,6	**2,4**	**2,3**
20	18,0	7,8	**5,2**	**1,5**
25	31,1	13,5	**9,2**	**1,5**
50	59,0	25,6	**11,0**	**2,3**
75	89,0	38,7	**11,2**	**3,5**
100	98,0	42,6	**15,0**	**2,8**
Média	45,1	19,6	**7,8**	**2,5**

Ao se considerar o tempo total médio dos algoritmos (não apenas os tempos das melhores execuções), a Tabela 4 apresenta uma comparação das médias dos resultados do ILS e do P-MIP-INS quanto ao valor da função objetivo (poluição) e ao tempo em minutos. A coluna GAP apresenta a diferença percentual do resultado da abordagem proposta em relação apresentado pelo algoritmo P-MIP-INS. Convém destacar que, para os grupos com 50, 75 e 100 clientes, a comparação não é precisa, dado que P-MIP-INS só foi executado um vez para cada instância nesses grupos. Nota-se que mesmo considerando todas as execuções do experimento, não apenas aquelas que levaram aos

Tabela 4: Média dos resultados

	P-MIP-INS			ILS_GRASP_MIP			
Grupos	Poluição	Tempo(Min)	Tempo Conv.(Min)	Poluição	GAP	Tempo(Min)	*Speedup*
10	**558,6**	7,7	3,3	561,3	0,5%	**0,9**	**3,7**
15	770,9	12,9	5,6	**748,9**	**-2,8%**	**2,2**	**2,5**
20	971,2	18,0	7,8	**969,9**	**-0,1%**	**6,5**	**1,2**
25	**952,3**	31,1	13,5	959,2	0,7%	**6,8**	**2,0**
50	1710,2	59,0	25,6	1764,4	3,2%	**9,8**	**2,6**
75	2483,5	89,0	38,7	2598,3	4,6%	**9,3**	**4,2**
100	3149,8	98,0	42,6	3312,9	5,2%	**11,6**	**3,7**
Média	1513,8	45,1	19,6	1559,3	0,02	**6,7**	**3,0**

melhores resultados por instância, a abordagem proposta apresenta um comportamento que justifica a proposta, já que apresenta uma diferença percentual de apenas $0,02\%$ com um *Speedup* de 3 vezes.

5. Conclusão e Trabalhos Futuros

Esse trabalho propõe um método híbrido que combina as técnicas de Inteligência Computacional GRASP, ILS e RVND com um algoritmo de *local branching* para resolver o problema de roteamento verde de veículos com frota heterogênea e variação de velocidade ao longo do tempo, conforme as condições de tráfego, em que são permitidas paradas em qualquer ponto ao longo da rota dos veículos.

Para resolver o problema, três métodos foram propostos: uma heurística que considera somente paradas nos clientes, e dois métodos exatos que consideram as paradas em qualquer lugar de um arco, que permitem a troca de clientes. Para inicializar a solução do ILS um Grasp com RVND é utilizado. A geração da solução é realizada por uma heurística de inserção mais barata. O RVND utiliza movimentos de *shifit*, *swap*, 2-opt inter e intra rotas, além de movimentos que permitem inverter uma rota e trocar os tipos veículos entre duas rotas. Por outro lado, o ILS proposto utiliza o movimento 2-opt inter rotas como pertubação combinado com o RVND sem este movimento, alé de chamadas periódicas dos métodos exatos.

Os resultados obtidos mostraram que a abordagem proposta consegue resultados similares aos apresentados na literatura, porém com ganho expressivo em termos de tempo de processamento, chegando a tempos seis vezes inferiores. Para trabalhos futuros, propõe-se explorar outros movimentos como pertubação no ILS e priorizar a escolha de duas rotas diferentes no modelo para gerar duas rotas.

O modelo completo para uma rota e os resultados por instância estão disponíveis em: 'https://github.com/junqueira200/HGVRSP-SBPO-2021'.

Referências

Bektas, T. e Laporte, G. (2011). The Pollution-Routing Problem. *Transportation Research Part B: Methodological*, 45(8):1232–1250. URL `https://ideas.repec.org/a/eee/transb/v45y2011i8p1232-1250.html`.

Bernstein, D. (1991). O. yigit. hash functions. `http://www.cse.yorku.ca/~oz/hash.html`. Accessed: 2021-06-07.

Demir, E., Bektaş, T., e Laporte, G. (2012). An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, 223(2):346–359.

Figliozzi, M. (2011). The impacts of congestion on time-definitive urban freight distribution networks co2 emission levels: Results from a case study in portland, oregon. *Transportation Research Part C-emerging Technologies*, 19:766–778.

Fischetti, M. e Lodi, A. (2003). Local branching. *Mathematical programming*, 98(1):23–47.

Gurobi Optimization, L. (2021). Gurobi optimizer reference manual. URL `http://www.gurobi.com`.

Küçükoğlu, İ., Ene, S., Aksoy, A., e Öztürk, N. (2015). A memory structure adapted simulated annealing algorithm for a green vehicle routing problem. *Environmental Science and Pollution Research*, 22(5):3279–3297.

Le Treut, H., Somerville, R., Cubasch, U., Ding, Y., Mauritzen, C., Mokssit, A., Peterson, T., e Prather, M. (2007). Historical overview of climate change science chapter 1.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., e Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Macrina, G., Pugliese, L. D. P., e Guerriero, F. (2020). The green-vehicle routing problem: A survey. *Modeling and Optimization in Green Logistics*.

Paskannaya, T. e Shaban, G. (2019). Innovations in green logistics in smart cities: USA and EU experience. , (1):173–181.

Prais, M. e Ribeiro, C. C. (2000). Parameter variation in grasp procedures. *Investigación Operativa*, 9(1):1–20.

Rezaei, N., Ebrahimnejad, S., Moosavi, A., e Nikfarjam, A. (2019). A green vehicle routing problem with time windows considering the heterogeneous fleet of vehicles: two metaheuristic algorithms. *European Journal of Industrial Engineering*, 13(4):507–535.

Rylan, M., Abitbol, Y., e Satoru, L. (2017). Problema de roteamento de veículos voltado para redução de emissões de carbono. *49 SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL*.

Shen, L., Tao, F., e Wang, S. (2018). Multi-depot open vehicle routing problem with time windows based on carbon trading. *International journal of environmental research and public health*, 15 (9):2025.

Talbi, E. (2009). *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. Wiley. ISBN 9780470496909. URL `https://books.google.com.br/books?id=SIsa6zi5XV8C`.

Wang, Y., Assogba, K., Fan, J., Xu, M., Liu, Y., e Wang, H. (2019). Multi-depot green vehicle routing problem with shared transportation resource: Integration of time-dependent speed and piecewise penalty cost. *Journal of Cleaner Production*, 232:12–29.

Xiao, Y. e Konak, A. (2016). The heterogeneous green vehicle routing and scheduling problem with time-varying traffic congestion. *Transportation Research Part E: Logistics and Transportation Review*, 88:146–166.