Pickup e *Delivery* com Janelas de Tempo: um Programa Inteiro

Arthur Henrique Sousa Cruz

Universidade Federal de Lavras

Resumo O *Pickup and Delivery* com Janelas de Tempo é uma clássica variação do Problema do Roteamento de Veículos. Por ter um

1 Introdução

Uma das variantes mais clássicas do Problema de Roteamento de Veículos é o *Pickup* e *Delivery* com Janelas de Tempo (PDJT). Este problema consiste em criar uma rota para os veículos de forma a atender pedidos de coleta (*pickups*) e entrega (*deliveries*) sem repetir os clientes. Os atendimentos, tanto para os *pickups* quanto para os *deliveries*, deve ocorrer em um determinado intervalo de tempo, denominado *Janelas de Tempo*. Este problema é NP-difícil, já que contém o Problema do Caixeiro Viajante como um caso especial (Ropke and Pisinger, 2006).

Este problema pode ser visto, por exemplo, no contexto dos correios. Neste cenário pode ser ainda mais agravante devido a necessidade de reotimizações (Bender et al., 2020).

A fim de encontrar novas possibilidades para a solução do PDJT, neste trabalho propõe-se uma formulação de um Programa Inteiro adaptado de Ropke and Pisinger (2006). A formulação foi linearizada, visto que o foco dos autores foi a apresentação de uma meta-heurística.

Por se tratar de um problema NP-difícil, o modelo por si só é incapaz de resolver grandes instâncias. Devido a isso, foi proposto uma heurística, baseada em Fischetti et al. (2005), para tentar encontrar soluções factíveis para o problema. A heurística foi testada para instâncias obtidas a partir do trabalho de Sartori and Buriol (2020). As instâncias utilizadas estão disponíveis em (Cruz, 2020b), assim como o algoritmo implementado.

Na Seção 2, define-se formalmente o problema. Em seguida, na Seção 3, apresenta-se o Programa Inteiro e a heurística proposta. Na Seção 4, apresenta-se os experimentos práticos e, na Seção 5, apresentamos as conclusões obtidas e as possibilidades para continuação do projeto.

2 Definição Formal

Baseando-se em Ropke and Pisinger (2006), define-se o PDTJ como se segue. Dado um grafo $G = (V, A)$, tem-se os conjuntos de vertices *pickups* P e deliveries

D. Além disso, tem-se um terminal inicial τ e um terminal final σ. Define-se $N = P \cup D$ e $V = N \cup \{\tau\} \cup \{\sigma\}$. Assim, definimos as arestas como $A = V \times V$. Um pedido é formado por um par de vértices (p, d), tal que $p \in P$ e $d \in D$. A função $par(v)$ com $v \in N$ retorna o par *delivery* para um vértice *pickup* e vice-versa. Reforça-se que $par(par(v)), v \in N$ é o próprio v.

Para toda aresta $(i, j) \in A$, atribui-se uma distância c_{ij} e um tempo de viagem de t_{ij}. Assume-se que ambos c_{ij} e t_{ij} são não negativos e que o tempo satisfaz a desigualdade triangular, ou seja $c_{ij} >= 0$, $t_{ij} >= 0$ e $t_ij <= t_{ik} + t_{kj}$ para todo $i, j, k \in V$.

Para todo pedido (p, d) com $p \in P$ e $d \in D$ é tem-se que um veículo $k \in K$ o atende, sendo K um conjunto de veículos homogêneos com capacidade de carga C. Os veículos devem atender demandas de pedidos em tempos específicos tanto para *pickups* quanto para *deliveries*. A janela de atendimento, ou janela de tempo, de um vértice $v \in V$ é dada por $[a_v, b_v]$, sendo a_v o início da janela e b_v o final. Além disso, o tempo para a realização do serviço no vértice $v \in V$ é definido como s_v. Considera-se que $t_{v,par(v)} + s_v > 0$, o que torna mais simples para o modelo a eliminação de subciclos e a definição de que um *pickup* deve vir antes de seu par *delivery*.

Cada vértice *pickup* $p \in P$ tem uma demanda associada a seu pedido de forma que a demanda $l_p >= 0$. Nota-se que o par *delivery* de p assume o seu oposto, ou seja $l_{par(p)} = -l_p$.

3 Metodologia

Nesta Seção apresenta-se o Programa Inteiro (Seção 3.1) e uma heurística (Seção 3.2) baseada no método *Feasibility Pump* (Fischetti et al., 2005).

3.1 Programa Inteiro

Para a formulação de um Programa Inteiro para solucionar o PDTJ foi necessário a criação das seguintes variáveis:

$x_{ijk} \in \{0, 1\}$	igual a 1 se a aresta $(i, j) \in A$ está na rota do veículo k;
$z_i \in \{0, 1\}$	igual a 1 se o pedido referente ao *pickup* i não é atendido;
$S_{ik} \in \mathbb{N}$	indica quando o veículo k iniciou o serviço no vértice i;
$L_{ik} \in \mathbb{N}$	*upper bound* para a carga do veículo k após o servir no vértice i;

Assim sendo, define-se o seguinte Programa Inteiro para o PDTJ:

$$\min \ \alpha \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} + \beta \sum_{k \in K} S_{\sigma k} - a_\tau + \gamma \sum_{i \in P} z_i \tag{1}$$

Sujeito à:

$$\sum_{k \in K} \sum_{j \in N} x_{ijk} + z_i = 1, \; \forall i \in P \tag{2}$$

$$\sum_{j \in V} x_{ijk} - \sum_{j \in V} x_{jmk} = 0, \; \forall k \in K, \; \forall i \in P, \; m = par(i) \tag{3}$$

$$\sum_{j \in P \cup \{\sigma\}} x_{\tau jk} = 1, \; \forall k \in K \tag{4}$$

$$\sum_{i \in D \cup \{\tau\}} x_{i\sigma k} = 1, \; \forall k \in K \tag{5}$$

$$\sum_{i \in V} x_{ijk} - \sum_{i \in V} x_{jik} = 0, \; \forall k \in K, \; \forall j \in N \tag{6}$$

$$S_{ik} + s_i + t_{ij} <= (1 - x_{ijk})M + S_{jk}, \; \forall k \in K, \; \forall (i,j) \in A \tag{7}$$

$$a_i <= S_{ik} <= b_i, \; \forall k \in K, \; \forall i \in V \tag{8}$$

$$S_{ik} <= S_{jk}, \; \forall k \in K, \; \forall i \in P, \; j = par(i) \tag{9}$$

$$L_{ik} + l_i <= (1 - x_{ijk})M + L_{jk}, \; \forall k \in K, \; \forall (i,j) \in A \tag{10}$$

$$L_{ik} <= C, \; \forall k \in K, \; \forall i \in V \tag{11}$$

$$L_{\tau k} = L_{\sigma k} = 0, \; \forall k \in K \tag{12}$$

$$x_{ijk} \in \{0,1\}, \; \forall k \in K, \; \forall (i,j) \in A \tag{13}$$

$$z_i \in \{0,1\}, \; \forall i \in P \tag{14}$$

$$S_{ik} >= 0, \; \forall k \in K, \; \forall i \in V \tag{15}$$

$$L_{ik} >= 0, \; \forall k \in K, \; \forall i \in V \tag{16}$$

A função objetivo (1) minimiza soma da distância total percorrida pelos veículos, do tempo gasto pelos veículos e o número de pedidos não atendidos. Os fatores α, β e γ são utilizados como pesos de cada um dos elementos minimizados. Geralmente, tem-se um valor mais alto para γ, para tentar forçar o atendimento do maior número de pedidos possível.

As restrições (2) garante que ou o *pickup* é visitado ou o pedido é considerado como não atendido. As restrições (3) garante que um *delivery* é visitado se, e

somente se, um *pickup* é visitado, assim como que as visitas sejam feitas pelo mesmo veículo. As restrições (4) e (5) obrigam que os veículos saiam do terminal inicial τ e terminem no terminal final σ. Com a adição das restrições (6), obriga-se a geração de um caminho consecutivo entre τ e σ.

Tem-se as restrições (7) e (8) para garantir que S_{ik} tenha um valor correto. Com elas garante-se que não haverá subciclos. As restrições (9), por sua vez, garantem que um *pickup* seja atendido antes de seu par *delivery*. L_{ik} tem um valor correto assegurado a partir das restrições (10), (11) e (12). Por fim, as restrições (13), (14), (15) e (16) definem o escopo para cada uma das variáveis.

3.2 Heurística

Como o problema é NP-difícil, foi proposta uma heurística baseado no *Feasibility Pump* (FP) (Fischetti et al., 2005). Abaixo segue o algoritmo proposto:

Algorithm 1 FP para o PDTJ

```
1: B* = resolve_relaxacao_linear()
2: B̄ = arredonda(B*)
3: while eh_otimo(B*) e nao_factivel e nao_estourou_tempo() do
4:    B* = resolve_nova_relaxacao(B̄)
5:    if arredonda(B*) ≠ B̄ then
6:       B̄ = arredonda(B*)
7:       if B* == B̄ then
8:          nao_factivel = Falso
9:       end if
10:   else
11:      pertubacao(B̄)
12:   end if
13: end while
14: if nao_factivel then
15:    return ∅
16: end if
17: return B̄
```

Inicialmente resolve-se a relaxação linear do modelo apresentado na Seção 3.1 e suas variáveis são arredondadas (linhas 1 e 2). Após isso, inicia-se a repetição (linhas 3 a 13), onde cria-se uma nova solução a partir da relaxação do modelo inteiro, com algumas modificações. Primeiro há uma mudança da função objetivo para:

$$\sum_{b \in \overline{B}_{bin} | x=0} b + \sum_{b \in \overline{B}_{bin} | b=1} 1 - b + \delta \sum_{b \in \overline{B}_S} b - \rho \sum_{b \in \overline{B}_L} b \tag{17}$$

Essa função minimiza primeiramente a distância do resultado linear para as variáveis binárias do modelo, ou seja, $B_{bin} = \{x_{ijk}, \forall (i,j) \in A, \forall k \in K\} \cup$

$\{z_i, \forall i \in P\}$. Por outro lado, minimiza-se os valores das variáveis que indicam o tempo, ou seja, $B = \{S_{ik}, \forall i \in V, \forall k \in K\}$. Por fim minimiza-se o valor das restrições de *upper bound* o negativo da capacidade, logo, $B = \{L_{ik}, \forall i \in V, \forall k \in K\}$. Ao se multiplicar a soma dos elementos de S por um valor alto $\delta >= 0$, garante-se que ele assumirá um valor inteiro sem a necessidade de gerar diversas novas variáveis, como seria no modelo padrão do FP. Isso é análogo para os elementos de L, porém multiplica-se por $-\rho$, sendo $rho >= 0$ um valor pequeno. A ideia da variação da função objetivo FP clássico foi acelerá-lo ao impedir a geração de diversas novas variáveis que poderiam tornar o modelo mais lento.

A execução do programa, nesse estado, pode acarretar no que foi chamado de *soluções triviais*, que ocorre quando $x_{\tau\sigma k} = 1, \forall k \in K$, sendo. Isso significa que os veículos não atenderam nenhum pedido, já que sua rota foi sair do terminal de início e ir diretamente para o terminal final. Assim, foi sorteado um veículo $k \in K$ e adicionada as seguintes restrições no modelo original:

$$x_{\tau\sigma k} = 0 \tag{18}$$

$$x_{\sigma\tau k} = 0 \tag{19}$$

Com isso garante-se que não serão geradas soluções triviais, visto que ao menos um dos veículos será obrigado a atender ao menos um pedido. Nota-se que, se esta restrição tornar o modelo infactível, isso significa que há somente soluções triviais para o modelo, logo não é necessário continuar a tentativa de resolver o problema.

O restante do algoritmo FP segue a proposta de Fischetti et al. (2005). As pertubações também foram implementadas e foi assumida uma lista para a detecção de resultados repetidos com tamanho 5.

4 Testes e Resultados

Nesta seção são apresentados os detalhes relacionados aos experimentos práticos realizados.

4.1 Detalhes de implementação e Instâncias

O algoritmo proposto foi implementado em *Python3*, sendo utilizado o *solver* Gurobi para a solução dos modelos. O código fonte encontra-se em Cruz (2020a).

Foram executadas 8 instâncias obtidas a partir de Sartori and Buriol (2020) com 50 pedidos cada. Como não havia um terminal final nas instâncias, foi criada uma cópia do terminal inicial com distância 0 para todos os outros vértices. Assim tem-se que a rota começa e termina no terminal inicial.

As instâncias encontram-se em Cruz (2020b). Cada uma delas foi executada por 25 minutos com 5 veículos e os resultados obtidos estão em Cruz (2020c).

4.2 Resultados

Inesperadamente, o resultado obtido foi que nenhum dos experimentos obteve resultados factíveis. Apesar disso, houve um grande número de valores próximos a 1 em diversas das variáveis dentre as que representam as arestas. Isso provavelmente ocorreu pois a modificação feita na função objetivo do FP tentou impedir o aumento do valor das variáveis S_{ik}, já que esta possuía um grande peso para o resultado do objetivo. A tentativa de agilizar o modelo se tornou seu maior problema e impediu que fosse possível encontrar soluções factíveis.

5 Conclusão e Possíveis Melhorias

O PDTJ é uma variação clássica do Problema de Roteamento de Veículos e, assim como ele, um NP-difícil. Neste trabalho foi proposto um Programa Inteiro para PDTJ e um algoritmo baseado na heurística *Feasibility Pump* (Fischetti et al., 2005).

Foram aplicadas algumas variações em relação à heurística original. Experimentos práticos (utilizando instâncias obtidas a partir de Sartori and Buriol (2020)) apontam que, inesperadamente, as variações propostas acarretaram no impedimento da obtenção de factibilidade.

O frustrante resultado deixa, contudo, a possibilidade para a implementação do FP em seu formato clássico, que ainda não foi explorado. Como continuação (e melhoria) deste trabalho, propõe-se a utilização do método proposto por Fischetti et al. (2005) na integra e aplicando-se as restrições (18) e (19), para que seja feita uma análise quanto a factibilidade e se as restrições propostas para impedir soluções triviais é de fato necessária.

Referências Bibliográficas

Matthias Bender, Jörg Kalcsics, and Anne Meyer. Districting for parcel delivery services–a two-stage solution approach and a real-world case study. *Omega*, page 102283, 2020.

Arthur H. S. Cruz. Código fonte. `https://github.com/thuzax/PM-2020/tree/master/projeto-final`, 2020a. Acessado em agosto de 2020.

Arthur H. S. Cruz. Instâncias de teste. `https://github.com/thuzax/PM-2020/tree/master/projeto-final/instancias/testes_projetos`, 2020b. Acessado em agosto de 2020.

Arthur H. S. Cruz. Arquivos com resultados. `https://github.com/thuzax/PM-2020/tree/master/projeto-final/resultados/resultados_testes_projeto`, 2020c. Acessado em agosto de 2020.

Matteo Fischetti, Fred Glover, and Andrea Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation science*, 40(4):455–472, 2006.

Carlo S Sartori and Luciana S Buriol. A study on the pickup and delivery problem with time windows: Matheuristics and new instances. *Computers & Operations Research*, page 105065, 2020.