Profitable Tour Problem with Passengers and Time and Cost Restrictions

Vinícius A. Petch[1], Marco Cesar Goldbarg[1], Elizabeth F. G. Goldbarg[1], José G. L. Filho[1]

[1]Departamento de Informática e Matemática Aplicada – Universidade Federal do Rio Grande do Norte (UFRN)

{viniciusap,zefilho}@ppgsc.ufrn.br, {gold,beth}@dimap.ufrn.br

Abstract. *This paper aims to present a new problem in the context of the collaborative transport named Profitable Tour Problem with Passengers and Time and Cost Restrictions. The problem consists in finding the best tour for a courier who needs to perform services in several locations. The tour is made by car. To reduce costs may give rides and share travel cost with the riders. A mathematical model is proposed which is implemented in a solver and applied to instances of the problem. Heuristic algorithms are presented according to the Variable Neighborhood Search and Greedy Randomized Adaptive Search Procedure meta-heuristics. Results of a computational experiment with 36 instances with up to 150 vertices are reported.*

Resumo. Este artigo tem como objetivo apresentar um novo problema no contexto do transporte colaborativo denominado *Problema do Passeio Lucrativo com Passageiros e Restrições de Tempo e Custo*. O problema consiste em determinar a melhor rota de um courier que precisa realizar serviços em diversos locais. O courier realiza seu percurso em um veículo. Para reduzir custos, o courier pode levar passageiros que compartilham com ele os custos de viagem. Um modelo matemático é proposto o qual é implementado em um *solver* e aplicado a instâncias do problema. São apresentados algoritmos heurísticos segundo as meta-heurísticas *Variable Neighborhood Search* e *Greedy Randomized Adaptive Search Procedure*. São relatados resultados de um experimento computacional com 36 instâncias com até 150 vértices.

1. Introdução

O problema de circulação de veículos nas grandes metrópoles é um dos mais complexos da atualidade, e envolve dentre outros fatores a qualidade de vida dos usuários, impacto ambiental, segurança física e aspectos econômicos. Iniciativas que minimizam o volume de veículos particulares em circulação são cada vez mais bem vindas, e várias alternativas já são aplicadas e estudadas com esse fim.

O transporte público coletivo é capaz de deslocar uma grande quantidade de passageiros ocupando um espaço reduzido nas vias. Já o transporte público individual, como táxi e *ridesharing* (Uber, Lyft), reduz a quantidade de veículos em circulação e pode ter um custo inferior a um veículo particular. Transportes não-motorizados como bicicletas são viáveis em algumas cidades com um custo reduzido, porém o seu uso é limitado a curtas distâncias e necessita de infraestrutura adequada.

Uma alternativa de transporte cada vez mais utilizada é o *carpooling*, onde um usuário (motorista) transporta um ou mais usuários (passageiros) com trajetos similares

ao do motorista. Os usuários transportados compartilham os custos do trajeto com o motorista. Dessa forma, os usuários podem se beneficiar do conforto de um veículo particular com um custo reduzido.

Dentro do contexto do transporte compartilhado, está o problema apresentado neste trabalho. O problema proposto, nomeado Problema do Passeio Lucrativo com Passageiros e Restrições de Tempo e Custo (PPL-PRTC), é baseado em um *courier* que realiza serviços pela cidade e se aproveita do compartilhamento de assentos do seu veículo para diminuir os custos de sua rota. O problema consiste em um *courier* que, em sua jornada de trabalho, realiza serviços de entrega e coleta de correspondências em diferentes locais utilizando um veículo como método de locomoção. Em cada local visitado é possível realizar uma coleta ou entrega, que resultará em um pagamento (bônus) ao *courier* e um gasto de tempo na entrega. O trajeto é escolhido pelo *courier*, que leva em consideração o custo e tempo gastos para seu deslocamento. O *courier* possui como objetivo realizar os serviços de entrega e coleta em um trajeto que lhe propicie o maior lucro possível. O *courier* é livre para escolher quantos e quais locais ele passará, inclusive optando por não cumprir qualquer rota. Todavia o *courier* só entrega ou recolhe correspondências em uma localidade uma vez.

Durante seu trajeto, o *courier* poderá compartilhar assentos do seu veículo com passageiros que estiverem disponíveis para embarque nos locais a serem visitados. As condições do embarque devem respeitar restrições de tarifa e tempo definidas pelos passageiros. Caso o *courier* embarque um ou mais passageiros, o custo da viagem entre os locais será repartido igualmente entre os passageiros ocupantes do carro e o *courier*, em rateio uniforme. O *courier* só poderá embarcar o passageiro no seu local de origem caso possa desembarca-lo em seu local de destino. O passageiro estará disponível em um intervalo de tempo determinado. Cada passageiro possui um valor máximo que aceita pagar pela viagem e só é embarcado caso a viagem programada atenda essa restrição. O veículo possui um número máximo de assentos disponíveis. A restrição de capacidade do veículo (número máximo de assentos disponíveis) deve ser respeitada qualquer que seja o trecho da rota.

Além do custo de deslocamento, o *courier* também terá gastos com pedágios em determinadas ligações que são vinculados à lotação do carro. O valor do pedágio depende do número de passageiros embarcados no carro no trecho com pedágio. A despesa de pedágio do *courier* não é rateada com os passageiros. O benefício em relação ao pedágio vem da diminuição da tarifa uma vez que existem passageiros no carro.

Este trabalho introduz o PPL-PRTC na Seção 2 onde é apresentado um modelo matemático. Foram desenvolvidas duas meta-heurísticas, VNS e GRASP, aplicadas ao PPL-PRTC. Na Seção 3, são apresentados operadores utilizados nas meta-heurísticas. O VNS e o GRASP são apresentados, respectivamente, nas Seções 4 e 5. Resultados de um experimento computacional contendo os algoritmos heurísticos e a implementação do modelo em um *solver* são relatados na Seção 6. Conclusões finais são apresentadas na Seção 7.

2. Definição do problema

É possível definir dois aspectos notáveis do PPL-PRTC: a escolha da rota e a escolha do carregamento de passageiros. O subproblema da rota tem como base o *Profitable Tour Problem* (*PTP*) (Dell'Amico et al., 1995), que por sua vez pertence a uma classe de

problemas denominada *Traveling Salesman Problems with Profits* (*TSPs with Profits*) (Feillet et al., 2001). A classe de problemas *TSPs with Profits* reúne problemas que buscam encontrar um ciclo ótimo em um grafo conexo ponderado em arestas e vértices. Os pesos nas arestas representam custos de viagem e os pesos nos vértices representam lucros (ou bônus). O PTP visa minimizar os custos de viagem subtraídos do lucro coletado nos vértices. O subproblema de carregamento de passageiros é uma atividade de decisão com várias restrições, limitada pela capacidade do veículo, origem e destino de cada passageiro e recursos dos mesmos. A decisão de carregamento está relacionada ao *Pickup and Delivery Problem* (PDP) (Savelsbergh, 1995), ainda que no PPL-PRTC o carregamento não atenda uma demanda, sendo realizado apenas para maximizar a função objetivo.

O PPL-PRTC é construído utilizando o PTP como base e adicionando a possibilidade de embarque de passageiros para a redução do custo das arestas. Ao problema modificado se soma o custo dos pedágios, estes que dependem da lotação do veículo. Dado grafo $G = (V, E)$, é associado um bônus d_v e um tempo de serviço s_v a cada vértice $v \in V$, e cada aresta $(i, j) \in E$ possui um custo de travessia c_{ij} e um tempo de travessia $t_{ij} > 0$. A capacidade do carro é de, no máximo, K passageiros. O conjunto P é formado por potenciais passageiros $p \in P$, distribuídos pelos vértices $v \in V$, onde cada passageiro possui uma origem u_p e um destino e_p, $u_p \neq e_p$, uma tarifa máxima r_p a ser paga pelo passageiro p e uma janela de tempo $[l_p, n_p]$ que representa o intervalo de tempo em que o passageiro p pode embarcar. O conjunto O é formado por pedágios $o_{i,j,k} \in O$, representando o valor do pedágio na aresta (i, j) caso esta seja atravessada com k passageiros no veículo.

Considere $x_{i,j}$ uma variável binária, sendo 1 se a aresta (i, j) pertence à solução, e 0 caso contrário. A variável y_i controla se o vértice i está na rota, sendo 1 se presente e 0 caso contrário. A variável z_i controla se o bônus do vértice i é coletado ou não, sendo 1 se coletado e 0 caso contrário. A variável $v_{i,j,k}$ assume valor 1 se o passageiro i trafegou pela aresta (j, k), e 0 caso contrário. A variável $f_{i,j}$ representa a quantidade de tempo gasta no trecho entre o vértice inicial e o vértice j, o qual foi alcançado a partir da aresta (i, j). A variável b_i indica o instante de início do atendimento do vértice i. A variável binária $q_{i,j,k}$ é utilizada para controlar o valor do pedágio utilizado na aresta (i, j), sendo 1 se o veículo contém k passageiros na aresta (i, j), e 0 caso contrário. C é uma constante grande o suficiente, que deve ser maior do que o maior tempo gasto para atravessar qualquer aresta do grafo.

O modelo do PPL-PRTC é apresentado nas expressões 2.1 a 2.28. A função objetivo do PPL-PRTC, equação 2.1, maximiza o lucro total dos bônus coletados pela execução do serviço de entrega ou coleta menos o custo dos trechos percorridos afetados pelo rateio da divisão de custos entre os passageiros e os custos relativos aos pedágios.

$$(2.1)\quad maximizar\ \mathrm{Z} = \sum_{i \in V} d_i z_i - \sum_{i \in V} \sum_{j \in V \setminus \{i\}} \frac{x_{i,j} c_{i,j}}{1 + \sum_{k \in P} v_{k,i,j}} - \sum_{i \in V} \sum_{j \in V \setminus \{i\}} \sum_{k \in K} q_{i,j,k} x_{i,j} o_{i,j,k}$$

Sujeita a:

$$(2.2)\quad \sum_{j \in V \setminus \{i\}} x_{i,j} = y_i, \qquad \forall i \in V$$

(2.3) $\sum_{j \in V \setminus \{i\}} x_{j,i} = y_i, \qquad \forall i \in V$

(2.4) $\sum_{j \in V \setminus \{1\}} f_{1,j} = \sum_{j \in V \setminus \{1\}} x_{1,j} t_{1,j}$

(2.5) $\sum_{j \in V \setminus \{1\}} f_{j,1} = \sum_{i \in V} \sum_{j \in V \setminus \{i\}} x_{i,j} t_{i,j} + \sum_{i \in V \setminus \{1\}} z_i s_i$

(2.6) $\sum_{j \in V \setminus \{i\}} f_{i,j} = \sum_{j \in V \setminus \{i\}} f_{j,i} + \sum_{j \in V \setminus \{i\}} x_{i,j} t_{i,j} + z_i s_i, \qquad \forall i \in V \setminus \{1\}$

(2.7) $C x_{i,j} \geq f_{i,j}, \qquad \forall i, j \in V$

(2.8) $\sum_{j \in V \setminus \{s_p\}} v_{p,j,u_p} + \sum_{j \in V \setminus \{e_p\}} v_{p,e_p,j} = 0, \qquad \forall p \in P$

(2.9) $\sum_{j \in V \setminus \{i\}} v_{p,j,i} - \sum_{j \in V \{i\}} v_{p,i,j} = 0, \qquad \forall p \in P, i \in V \setminus \{u_p, e_p\}$

(2.10) $\sum_{p \in P} v_{p,i,j} \leq K x_{i,j}, \qquad \forall (i,j) \in E$

(2.11) $\sum_{j \in V \setminus \{1\}} v_{p,1,i} + \sum_{j \in V \setminus \{1\}} v_{p,j,1} = 0, \qquad \forall p \in P, u_p \neq 1, e_p \neq 1$

(2.12) $\sum_{i \in V} \sum_{j \in V \setminus \{i\}} \frac{c_{i,j} v_{p,i,j}}{1 + \sum_{k \in P} v_{k,i,j}} \leq r_p, \qquad \forall p \in P$

(2.13) $b_1 = 0$

(2.14) $\sum_{i \in V \setminus \{j\}} f_{ij} = b_j, \qquad \forall j \in V \setminus \{1\}$

(2.15) $b_{s_k} \geq l_k \sum_{j \in V \{s_p\}} v_{p,u_p,j}, \qquad \forall p \in P$

(2.16) $b_{s_k} \leq n_k + C(1 - \sum_{j \in V \{s_p\}} v_{p,u_p,j}), \qquad \forall p \in P$

(2.17) $z_1 = 0$

(2.18) $z_i \leq y_i, \qquad \forall i \in V \setminus \{1\}$

(2.19) $\sum_{k \in K = 0} q_{i,j,k} = 1, \qquad \forall i, j \in V$

(2.20) $\sum_{k \in K} k q_{i,j,k} = \sum_{k \in P} v_{i,j,k}, \qquad \forall i, j \in V$

(2.21) $x_{i,j} \in \{0,1\}, \qquad \forall i, j \in V$

(2.22) $y_i \in \{0,1\}, \qquad \forall i \in V$

(2.23) $z_i \in \{0,1\}, \qquad \forall i \in V$

(2.24) $w_{i,j} \in \{0,1\}, \qquad \forall i \in P, j \in V$

(2.25) $v_{i,j,k} \in \{0,1\}, \qquad \forall i \in P, \qquad \forall j, k \in V$

(2.26) $q_{i,j,k} \in \{0,1\}, \qquad \forall i, j \in V, \qquad \forall k \in K$

(2.27) $b_i \geq 0, \qquad \forall i \in V$

(2.28) $f_{i,j} \geq 0, \qquad \forall i, j \in V$

As restrições 2.2 e 2.3 garantem que todo vértice que estiver na rota possui apenas uma aresta saindo e uma aresta entrando nele e que não há arestas entrando ou saindo de vértices que não estão na rota. As restrições 2.4 a 2.6 garantem que a solução seja apenas um ciclo e realizam o cálculo do tempo gasto. A restrição 2.4 restringe a aresta que parte do vértice base, a 2.5 restringe a aresta que chega no vértice base, e a 2.6 restringe arestas que chegam e partem dos demais vértices. A restrição 2.7 garante que só haverá valor associado à variável $f_{i,j}$ se a aresta (i,j) estiver na solução. A restrição 2.8 inviabiliza o retorno do passageiro a sua cidade de origem. A restrição 2.9 garante que todo passageiro

que embarcar será desembarcado. A restrição 2.10 certifica que a quantidade de passageiros não ultrapasse a capacidade *K*. A restrição 2.11 garante que as arestas trafegadas pelos passageiros não sejam diferentes das arestas do motorista. A restrição 2.12 garante que o valor pago por um passageiro não exceda a sua tarifa máxima. A restrição 2.13 o tempo de início de atendimento do vértice inicial é 0. A restrição 2.14 garante que o serviço no vértice *j* inicia quando o vértice é alcançado na rota. As restrições 2.15 e 2.16 impedem que um passageiro seja embarcado fora da sua janela de tempo. A restrição 2.17 garante que nenhum serviço será realizado no vértice v_1. A restrição 2.18 garante que o serviço só será realizado em um vértice caso esse vértice seja visitado. A restrição 2.19 garante que somente um valor de pedágio será pago em cada aresta. A restrição 2.20 ativa a variável $q_{i,j,k}$ em função da quantidade de passageiros na aresta. As restrições 2.21 a 2.28 definem o escopo das variáveis.

Neste trabalho, uma solução será representada por um vetor de vértices (representando o trajeto), um vetor para cada vértice (representando os passageiros que serão carregadas naquele vértice), e uma variável binária para cada vértice (representando se o serviço daquele vértice foi realizado).

3. Algoritmos e operadores

Foram implementados diversos algoritmos e operadores para as meta-heurísticas, onde alguns destes métodos foram baseados em algoritmos já apresentados para outros problemas. Outros foram criados especificamente para este problema. No total, foram implementados: um algoritmo para adicionar passageiros a uma rota, 4 algoritmos para construção de soluções, 7 operadores de busca local, 3 operadores de perturbação, e 2 procedimentos de *path-relinking*.

3.1. Carregamento de passageiros

A inserção dos passageiros em uma solução é realizada a partir de um algoritmo específico. Cada vez que uma rota é construída ou alterada, este algoritmo realiza o carregamento de passageiros na rota. O método verifica cada passageiro em *P* de forma sequencial. O algoritmo coloca no veículo todo passageiro que puder ser embarcado, i.e. aqueles que satisfazem todas as restrições.

3.2. Construção de soluções

Foram implementados 3 algoritmos para construção de soluções: um algoritmo construtivo aleatório, um algoritmo construtivo semi-guloso baseado em elite, e um algoritmo construtivo semi-guloso baseado em roleta.

O algoritmo construtivo aleatório escolhe o tamanho da solução de forma aleatória, e insere vértices aleatórios até que a solução alcance o tamanho sorteado. O algoritmo construtivo semi-guloso baseado em elite escolhe o tamanho da solução de forma aleatória. Os vértices que estarão na solução são selecionados de forma semi-gulosa. Para a escolha de cada vértice, é formado um conjunto com os 5 melhores vértices que não foram inseridos na solução. O critério de escolha dos vértices é o lucro gerado com a inserção do vértice específico. O lucro é calculado com a adição do bônus do vértice, menos o custo da aresta incluída na rota e o custo do pedágio na aresta considerando que o veículo passa sem passageiros na aresta. Desse conjunto é sorteado um vértice que será inserido na solução. O algoritmo construtivo semi-guloso baseado

em roleta escolhe o tamanho da solução de forma aleatória e seleciona os vértices de forma semi-gulosa, onde enquanto a solução não tiver o tamanho sorteado, os vértices que não foram inseridos na solução são ordenados com base na mesma avaliação do método anterior. É realizado um sorteio em roleta, utilizando a avaliação como pesos, para selecionar o vértice a ser inserido em cada iteração até formar uma solução.

3.3. Operadores de busca local e perturbação

Os operadores de busca local tem como objetivo melhorar uma dada solução, Foram implementadas 7 buscas locais: 2-*exchange*, 1-*shift*, 2-*opt*, 2-*interchange*, *add1, add2*, *remove* e *switchservice*.

O operador 2-*exchange* escolhe um vértice como pivô e gera soluções resultantes trocando o vértice pivô de posição com todos os outros vértices que pertencem à solução. O operador 1-*shift* escolhe um vértice como pivô e gera as soluções da vizinhança trocando o pivô pelos vértices vizinhos, até que o vértice pivô tenha ocupado todas as posições da solução. O operador 2-*opt*, criada por Croes (1958), remove duas arestas da solução e adiciona as arestas agora disponíveis. O 2-*opt* implementado escolhe uma aresta aleatoriamente e realiza a troca em relação a todas as outras arestas da solução.

O operador *2-interchange* escolhe um vértice pivô e gera soluções a partir da sua troca com todos vértices não pertencentes à solução. O operador *serviceswitch* escolhe um vértice aleatório da solução e caso o serviço de entrega ou coleta estivesse sendo realizado naquele vértice, ele deixa de ser realizado, caso o serviço passa a ser realizado.

O operador *add1* sorteia um vértice que não pertença à solução e tenta inseri-lo em todas as posições possíveis, comparando a melhor solução resultante à solução original. O operador *add2*, por sua vez, sorteia uma posição entre dois vértices sequenciais na solução e tenta inserir vértices que não pertencem à solução original neste local. Cada vértice que não está na solução é testado correspondendo a uma solução vizinha, e a melhor solução resultante é comparada à solução original e a substitui caso seja melhor. O operador *remove* gera as soluções da vizinhança removendo cada vértice da solução original, e comparando a melhor solução resultante à solução original. Em *add2* e em *remove*, a solução original difere de cada solução vizinha em apenas 1 vértice.

Os operadores de perturbação são responsáveis por perturbar uma solução, tentando removê-la de ótimos locais. Os operadores recebem como um parâmetro de entrada, uma porcentagem referente à quantidade de perturbações de acordo com o tamanho da solução alvo. Foram implementados 3 operadores de perturbação: *inc*, *red*, e *mut*. O operador *mut* realiza trocas aleatórias entre dois vértices da rota, o operador *inc* adiciona vértices aleatórios que não estão contidos na solução em posições aleatórias da rota, e o operador *red* remove vértices aleatórios da rota.

3.4. *Path-relinking*

Path-relinking, método proposto por Glover (1997), é uma estratégia de intensificação que explora o espaço entre duas soluções de elite, realizando as operações necessárias para transformar uma solução na outra e explorando essa transformação passo a passo. Foram implementadas uma versão aleatória e uma versão sequencial do algoritmo, em duas fases. A primeira fase consiste em escolher um vértice da solução destino, S_b, e verificar sua posição na solução origem, S_a. Caso a posição seja a mesma nas duas soluções, nada se faz. Caso contrário, suponha que o vértice está na *j*-ésima posição em

S_b, e na *i*-ésima posição em S_a. Os vértices nas posições *i* e *j* em S_a são trocados. O algoritmo aleatório seleciona os vértices aleatoriamente e o algoritmo sequencial os escolhe na ordem em que se encontram na solução.

A segunda fase, consiste em igualar o tamanho da solução S_b com o tamanho da solução S_a, se necessário. Caso $|S_a|>|S_b|$, os vértices são removidos um a um sequencialmente, iniciando do fim da solução, até que ambos cheguem ao mesmo tamanho. Caso $|S_a|<|S_b|$, os vértices cuja posição seja maior que o tamanho de S_a são adicionados um por um sequencialmente até que as soluções cheguem ao mesmo tamanho.

4. *Variable Neighborhood Search*

O Variable Neighborhood Search (VNS), proposto por Mladenovic e Hansen (1997), é uma meta-heurística que utiliza múltiplas buscas locais e métodos de reinício para assim diversificar a vizinhança de busca. Neste trabalho foi implementado o *Variable Neighborhood Descent* (VND), uma variação do VNS desenvolvida por Brimberg et al. (2000). No VND, caso uma busca local não consiga melhorar a solução, esta é passada para a busca local seguinte, e caso contrário, a solução é retornada à primeira busca local utilizada.

```
VND
s = solução inicial;
sb = melhor solução;
m = quantidade de operadores de busca local;
1   FAÇA:
2     s' = buscaLocal(s, n);
3     SE lucro(s') > lucro(s):
4       s = s';
5       n = 1;
6     SENÃO:
7       n = n + 1;
8     SE n > m:
9       SE lucro(s) > lucro(sb):
10        sb = s;
11      s = perturbação(s);
12    n = 1;
13  ATÉ Condição de parada;
14  RETORNAR sb
```

Figura 1. Pseudocódigo da implementação da meta-heurística VND

A Figura 1 mostra o VND e a Figura 2 mostra a função *buscaLocal()*. Foram implementadas duas versões do VND diferindo em como a função de perturbação foi implementada. A primeira versão (VND-Mut) utiliza múltiplos operadores de perturbação para perturbar a solução. Tais operadores são o *red*, *inc* e *mut*. Na segunda versão (VND-Rst), o procedimento de perturbação reconstrói a solução. Para que o reinício seja aplicado, é necessário que uma quantidade mínima de iterações tenha sido executada desde o último reinício, e que o lucro da solução resultante seja inferior a 0. A restrição de lucro mínimo faz com que soluções com lucro negativo ou 0 sejam descartadas buscando maior eficiência na busca de uma solução de lucro positivo.

Para o VND-Mut foi necessário escolher os parâmetros referentes às porcentagens utilizadas pelos operadores de perturbação. Para isso, foi utilizada a ferramenta *irace* descrita no trabalho de López-Ibánez et al. (2011). O *irace* foi executado em um conjunto de 12 instâncias de tamanho 100 criadas especificamente para a parametrização, com um limite de 5.000 execuções. A Tabela 1 mostra os limites definidos e as configurações retornadas pelo *irace* para os 3 métodos de perturbação. Como o *irace* retorna um conjunto de configurações, ordenado da melhor para a pior, a configuração #1 foi utilizada.

```
buscaLocal(si, n)
n = identificador da busca local;
si = solução inicial;
sr = solução resultante;
1   CASO n:
2     SEJA 1:
3       sr = remove(si); // remove
4     SEJA 2:
5       sr = add1(si); // add1
6     SEJA 3:
7       sr = add2(si); // add2
8     SEJA 4:
9       sr = shift(si); // 1-shift
10    SEJA 5:
11      sr = exchange(si); // 2-exchange
12    SEJA 6:
13      sr = interchange(si); // 2-interchange
14    SEJA 7:
15      sr = opt(si); // 2-opt
16    SEJA 8:
17      sr = service(si) // serviceswitch
18  RETORNAR sr;
```

Figura 2. Pseudocódigo da implementação da função de busca local utilizada pelo VND

Tabela 1. Limites definidos para os parâmetros do VND-Mut e configurações retornadas pelo *irace*, da melhor para a pior

Variável	Tipo	Mínimo	Máximo	#1	#2	#3
red	Real	0 (0%)	1 (100%)	0,1288	0,1030	0,1948
inc	Real	0 (0%)	1 (100%)	0,7679	0,7990	0,7928
mut	Real	0 (0%)	1 (100%)	0,8928	0,8621	0,8945

5. *GRASP com VND e Path-relinking*

O *Greedy Randomized Adaptive Search Procedure* (GRASP), criado por Feo e Resende (1989), busca unir métodos construtivos semi-gulosos com procedimentos de busca.

A Figura 3 mostra o GRASP implementado para este estudo. Foram implementadas duas versões do GRASP. O GRASP-Elt utiliza o construtivo semi-guloso de elite (com o tamanho do conjunto de elite n=5), enquanto o GRASP-Rol utiliza o construtivo semi-guloso de roleta. Ao final de cada iteração, caso a solução resultante não seja a melhor solução já encontrada, são aplicadas 2 execuções de cada um dos algoritmos de *path-relinking* utilizando como origem a solução resultante da iteração e como destino a melhor solução já encontrada. Ao final das 4 execuções, a nova solução é comparada com a melhor solução encontrada.

```
GRASP
s = solução inicial;
sb = melhor solução;
m = quantidade de operadores de busca local;
1   FAÇA:
2     s' = buscaLocal(s, n);
3     SE lucro(s') > lucro(s):
4       s = s';
5       n = 1;
6     SENÃO:
7       n = n + 1;
8     SE n > m:
9       SE lucro(s) > lucro(sb):
10        sb = s;
11      SENÃO:
12        s = pathRelinking(s, sb);
13        SE lucro(s) > lucro(sb):
14          sb = s;
15      s = reinício(s);
16      n = 1;
17  ATÉ Condição de parada;
18  RETORNAR sb
```

Figura 3. Pseudocódigo do GRASP com VND e *path-relinking*

6. Experimentos

Foram realizados experimentos computacionais com o objetivo de comparar os resultados obtidos pelo *solver* Gurobi, das duas versões do VND e das duas versões do GRASP. Foi criado um banco de dados com 156 instâncias, das quais 36 foram utilizadas para os experimentos. As instâncias do experimento e a explicação de como foram geradas podem ser acessadas em https://github.com/viniciuspetch/PTPwPTCC_Lib. As 36 instâncias foram separadas em 3 grupos de 4 tipos cada, totalizando 12 configurações, com instâncias de tamanho 10, 100 e 150 vértices. Os testes foram executados em um PC com processador i5-2400 3.10 GHz, com memória de 4GB DDR3 1333 Mhz, rodando Ubuntu 16.04 64 bits.

O modelo descrito na Seção 2 foi implementado no *solver* Gurobi versão 7.51. Foram submetidas ao *solver* instâncias de tamanho 10, com um tempo limite de 80.000 segundos sendo executado em apenas 1 *thread*. Para instâncias maiores que 10, o *solver* não foi capaz de entrar na etapa de *branch-and-cut* dentro do tempo estipulado. As meta-heurísticas foram executadas 30 vezes para todas as instâncias, com cada execução sendo limitada a 2.000.000 de avaliações da função objetivo.

Na análise dos resultados foram utilizados testes estatísticos não-paramétricos: teste de Friedman e o teste *post-hoc* de Conover para o teste de Friedman. Como nível de significância foi utilizado 0,05. Os testes foram implementados no R versão 3.5.0, 64 bits.

As Tabela 2, 3 e 4 mostram os resultados em relação a avaliação da função objetivo e dos tempos de execução para instâncias de tamanho 10, 100 e 150, respectivamente. A coluna *Lucro* diz respeito à avalição da função objetivo e a coluna *Tempo* mostra o tempo em segundos. No caso dos valores relativos às colunas do Gurobi, é mostrado o valor da melhor solução encontrada (Lucro) e o tempo de processamento. No caso das colunas relativas às meta-heurísticas os valores mostrados nas duas colunas se referem às médias dos valores obtidos nas 30 execuções de cada algoritmo.

Tabela 2. Resultados para instâncias de tamanho 10

Ins-tância	Gurobi		VND-Mut		VND-Rst		GRASP-Elt		GRASP-Rlt	
	Lucro	Tempo	Lucro	Tempo	Lucro	Tempo	Lucro	Tempo	Lucro	Tempo
ga10	1412,75	80000,00	1678,06	9,87	1927,46	10,70	2013,67	9,20	2013,67	9,17
gb10	855,50	80000,00	1032,00	9,60	1195,51	10,81	1290,00	9,15	1290,00	9,09
gc10	1281,25	80000,00	1440,08	9,80	1639,00	10,97	1728,10	9,37	1728,10	9,39
gd10	850,25	80000,00	1049,49	9,53	1294,73	10,69	1311,87	9,22	1311,87	9,17
ha10	1666,67	80000,00	1820,20	9,66	2010,90	10,74	2275,25	9,15	2275,25	9,16
hb10	1297,92	80000,00	1355,31	9,32	1726,46	10,70	1767,80	9,34	1767,80	9,34
hc10	1465,00	80000,00	1485,57	9,46	1881,32	10,87	1937,70	9,32	1937,70	9,30
hd10	1099,50	80000,00	1197,20	9,81	1423,52	10,77	1496,50	9,32	1496,50	9,30
ia10	0,00	80000,00	461,76	6,84	437,87	7,87	532,80	6,89	532,80	6,85
ib10	721,42	80000,00	912,73	5,33	1316,83	7,65	1369,10	6,74	1369,10	6,74
ic10	396,92	80000,00	641,15	5,77	866,08	7,78	1012,35	6,76	1012,35	6,69
id10	195,58	80000,00	744,96	4,89	802,56	7,79	931,20	6,64	931,20	6,63

Tabela 3. Resultados para instâncias de tamanho 100

Instância	VND-Mut		VND-Rst		GRASP-Elt		GRASP-Rlt	
	Lucro	Tempo	Lucro	Tempo	Lucro	Tempo	Lucro	Tempo
ga100	13919,21	85,48	17891,90	92,73	15434,94	87,41	15828,55	88,46
gb100	12306,56	90,07	16024,24	85,31	14120,17	83,45	13612,29	88,37
gc100	12459,63	79,92	14878,46	81,91	12569,32	83,82	13237,39	87,23
gd100	9624,92	90,30	12803,06	83,14	9754,48	92,78	10110,79	88,00
ha100	15401,52	98,42	19766,45	100,96	15847,30	95,58	16369,28	89,02
hb100	10925,58	78,48	14356,00	79,84	12781,33	94,69	12305,14	84,77
hc100	14508,29	98,28	18486,57	90,68	14662,65	86,61	15398,43	88,72
hd100	11642,65	97,35	15836,67	90,67	11370,84	88,35	12291,83	88,79
ia100	5098,15	38,26	9945,67	63,32	6972,98	59,00	7412,66	62,68
ib100	8358,10	59,38	12184,72	65,70	10138,57	66,51	10142,62	64,20
ic100	9170,17	61,55	11714,52	64,24	11738,90	65,80	10425,40	63,23
id100	3813,97	40,68	5434,02	54,26	5185,90	64,65	4468,77	62,91

Tabela 4. Resultados para instâncias de tamanho 150

Instância	VND-Mut		VND-Rst		GRASP-Elt		GRASP-Rlt	
	Lucro	Tempo	Lucro	Tempo	Lucro	Tempo	Lucro	Tempo
ga150	14429,60	116,92	20704,84	126,35	11918,78	147,63	12389,24	146,68
gb150	10645,53	100,02	17237,74	121,24	9957,13	146,88	10225,99	145,84
gc150	13375,66	113,68	20410,95	121,51	12097,39	144,28	12643,09	143,91
gd150	8232,68	117,50	15637,23	117,73	8940,83	151,18	7921,83	147,63
ha150	17856,95	145,13	31317,42	155,94	15424,89	159,87	16583,67	151,70
hb150	11194,83	98,26	18628,90	121,62	10538,78	146,60	11060,40	144,15
hc150	13236,42	126,01	24069,97	142,31	12803,61	148,36	13395,64	147,03
hd150	11897,30	128,00	18319,56	119,75	11605,38	155,69	11056,82	148,15
ia150	6048,56	65,66	10098,42	85,12	6668,54	102,56	5802,86	103,45
ib150	9485,26	90,57	13634,88	94,84	8081,44	112,63	8361,85	109,58
ic150	8011,46	72,69	15046,86	96,38	7784,15	107,01	8212,15	108,28
id150	4483,13	72,24	8393,54	83,92	5575,60	110,24	4552,16	108,62

Observa-se que o processamento do software Gurobi foi interrompido no tempo limite para todas as instâncias de tamanho 10, significando que não há garantia dos resultados serem ótimos. Esse fato também pode ser observado em comparação com os resultados das meta-heurísticas. Comparando os resultados, observa-se que para as instâncias de tamanho 10 ambas as versões do GRASP obtiveram os melhores resultados entre os algoritmos propostos, enquanto que o resultado da solução exata alcançada pelo Gurobi

foi de pior qualidade. O VND-Rst obteve o maior tempo computacional dentre os algoritmos heurísticos, seguido pelo VND-Mut, enquanto as versões do GRASP obtiveram tempos similares.

Comparando os resultados para as instâncias de tamanho 100, o VND-Rst obteve o melhor resultado em quase todas as instâncias (com exceção da ic100), enquanto o VND-Mut obteve os piores resultados. Em relação ao tempo de processamento, todos os algoritmos obtiveram tempos similares, com exceção das instâncias ia100 e id100 para as quais o tempo do algoritmo VND-Mut foi consideravelmente inferior aos demais.

Comparando os resultados para as instâncias de tamanho 150, o algoritmo VND-Rst obteve resultados qualitativos consideravelmente superiores aos outros algoritmos, enquanto os outros algoritmos obtiveram resultados similares entre si. Em questão de tempo, o VND-Mut obteve os menores tempos, seguido do VND-Rst, enquanto ambas a versões do GRASP obtiveram tempos similares.

O teste de Friedman apresentou como resultado um p-valor de 1,357e-07 para o grupo de tamanho 10, 1,937e-06 para o grupo de tamanho 100 e 1,784e-05 para o grupo de tamanho 150. Com esses valores, não há evidência estatística suficiente para aceitar a hipótese nula, e é possível considerar que os resultados são estatisticamente diferentes. As Tabela 5, 6 e 7 mostram os resultados do teste de Conover. Valores menores que o nível de significância adotado estão em negrito.

Tabela 5. Resultado do teste *post-hoc* para as instâncias de tamanho 10.

Algoritmo	VND-Mut	VND-Rst	GRASP-Elt
VND-Rst	0,3483	-	-
GRASP-Elt	***9,60E-06***	0,0091	-
GRASP-Rst	***9,60E-06***	0,0091	1,0000

Tabela 6. Resultado do teste *post-hoc* para as instâncias de tamanho 100.

Algoritmo	VND-Mut	VND-Rst	GRASP-Elt
VND-Rst	***6,40E-07***	-	-
GRASP-Elt	0,0880	***0,0160***	-
GRASP-Rst	***0,0160***	0,0880	0,9240

Tabela 7. Resultado do teste *post-hoc* para as instâncias de tamanho 150.

Algoritmo	VND-Mut	VND-Rst	GRASP-Elt
VND-Rst	***0,0252***	-	-
GRASP-Elt	0,3132	***3,40E-05***	-
GRASP-Rst	0,6929	***0,0005***	0,9239

O teste mostra que as versões GRASP apresentam resultados superiores às versões VND para as instâncias de tamanho 10. Para as instâncias de tamanho 150, o VND-Rst apresenta resultados estatisticamente diferentes dos demais indicando, portanto, seu desempenho superior às demais heurísticas. Em todos os 3 grupos, os *p*-valores entre as versões do GRASP foram superiores a 0,90 permitindo sustentar a similaridade dos resultados.

7. Conclusões

Este trabalho introduziu o Problema do Passeio Lucrativo com Passageiros e Restrições de Tempo e Custo. Um modelo matemático foi proposto, implementado em um solver e aplicado a instâncias do problema. Foram apresentados algoritmos heurísticos segundo as meta-heurísticas: *Variable Neighborhood Search* e *Greedy Randomized Adaptive Search Procedure*. Foram relatados resultados de um experimento computacional com 36 instâncias com até 150 vértices.

A partir da análise dos experimentos foi possível concluir que para instâncias de tamanho grande, o VND-Rst obteve os melhores resultados. A partir dos testes estatísticos, não foi possível rejeitar a hipótese nula entre as versões do GRASP e o VND-Rst para as instâncias de tamanho 10, e entre o VND-Rst e o GRASP-Rlt para as instâncias de tamanho 100. Enquanto que o VND-Rst obteve resultados superiores ao VND-Mut em todas as instâncias, o GRASP-Elt e GRASP-Rlt obtiveram resultados similares e p-valores altos, não sendo possível concluir por superioridade qualitativa.

O Gurobi não foi capaz de encontrar a solução ótima para nenhuma das instâncias e alcançou resultados qualitativos inferiores aos outros algoritmos. Isso mostra a dificuldade do *solver* em encontrar soluções ótimas para o problema.

Referências

FEILLET, Dominique; DEJAX, Pierre; GENDREAU, Michel. Traveling Salesman Problems with Profits: an Overview. 2001

DELL'AMICO, Mauro; MAFFIOLI, Francesco; VÄRBRAND, Peter. On prize-collecting tours and the asymmetric travelling salesman problem. International Transactions in Operational Research, v. 2, n. 3, p. 297-308, 1995.

CROES, Georges A. A method for solving traveling-salesman problems. Operations research, v. 6, n. 6, p. 791-812, 1958.

GLOVER, Fred. Tabu search and adaptive memory programming—advances, applications and challenges. In: Interfaces in computer science and operations research. Springer US, 1997. p. 1-75.

MLADENOVIĆ, Nenad; HANSEN, Pierre. Variable neighborhood search. Computers & operations research, v. 24, n. 11, p. 1097-1100, 1997.

BRIMBERG, Jack; HANSEN, Pierre; MLADENOVIć, Nedad; TAILLARD, Eric D. Improvements and comparison of heuristics for solving the uncapacitated multisource Weber problem. Operations Research, v. 48, n. 3, p. 444-460, 2000.

FEO, Thomas A.; RESENDE, Mauricio GC. A probabilistic heuristic for a computationally difficult set covering problem. Operations research letters, v. 8, n. 2, p. 67-71, 1989.

LÓPEZ-IBÁNEZ, Manuel; DUBOIS-LACOSTE, Jérémie, STÜTZLE, Thomas; BIRATTARI, Mauro. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

SAVELSBERGH, Martin WP; SOL, Marc. The general pickup and delivery problem. Transportation science, v. 29, n. 1, p. 17-29, 1995.