O problema do caixeiro viajante com vários passageiros, cota de bônus opcional e tempo

The Traveling Salesman Problem with Multiple Passengers Optional Bonus Quota and Time

DOI:10.34117/bjdv7n2-612

Recebimento dos originais: 26/01/2021
Aceitação para publicação: 26/02/2021

Allan Vilar de Carvalho
Mestre em Sistemas e Computação pela Universidade Federal do Rio Grande do Norte
Universidade Federal do Rio Grande do Norte
Endereço: Campus Universitário Lagoa Nova, CCET, DIMAp, Natal - RN, Brasil
E-mail: allanvilarcarvalho@gmail.com

Marco César Goldbarg
Doutor em Engenharia de Sistemas e Computação pela Universidade Federal do Rio de Janeiro
Universidade Federal do Rio Grande do Norte
Endereço: Campus Universitário Lagoa Nova, CCET, DIMAp, Natal - RN, Brasil
E-mail: gold@dimap.ufrn.br

Elizabeth Ferreira Gouvêa Goldbarg
Doutora em Engenharia de Produção pela Universidade Federal do Rio de Janeiro
Universidade Federal do Rio Grande do Norte
Endereço: Campus Universitário Lagoa Nova, CCET, DIMAp, Natal - RN, Brasil
E-mail: beth@dimap.ufrn.br

RESUMO
Este artigo apresenta o Problema do Caixeiro Viajante com Múltiplos Passageiros Bônus Optativos Quota e Tempo. O problema consiste em maximizar o lucro de um caixeiro viajante que realiza serviço de transporte de mercadorias, considerando a possibilidade de rateio das despesas da rota com eventuais passageiros embarcados em seu veículo. As mercadorias devem ser transportadas obrigatoriamente das suas origens até os seus destinos e devem contabilizar uma quota mínima definida a priori. Nesta variante, ao passar em uma cidade, o caixeiro decide se coleta ou não a mercadoria a ser transportada. A coleta da mercadoria requer tempo de carregamento e de descarregamento. É proposto um modelo de programação matemática que é resolvido por um solver. São propostas, também, três heurísticas, sendo duas desenvolvidas segundo as meta-heurísticas Colônia de Formigas e GRASP. Os resultados e análises de um experimento computacional são apresentados.

Palavras-chave: Problema do caixeiro viajante, programação matemática, ACO, GRASP.

ABSTRACT
This study presents the Traveling Salesman with Multiple Passengers Optional Bonus Quota and Time Problem. It consists in maximizing the profit of a travelling salesman who transports freights among several locations, considering the possibility of sharing travel expenses with eventual passengers in his vehicle. The transportation of goods from their origins to required destinations is mandatory. The salesman has to satisfy a minimum quota, defined a priori, of goods delivery. In this variant, when passing through a city, the salesman decides if or not to collect the goods to be transported. The goods collection requires loading and unloading time. This study presents a mathematical programming model which is solved. Besides, three heuristics are presented, two of them based on the Ant Colony and GRASP metaheuristics. Results and analyses of a computational experiment are presented.

Keywords: Traveling salesman problem, mathematical programming, ACO, GRASP.

1 INTRODUÇÃO

O problema alvo desta pesquisa está contextualizado na área dos problemas de roteamento enriquecido, e possui diversas aplicações práticas. As aplicações envolvem transporte de pessoas, de mercadorias, ou, ambos, normalmente compostos por diversas características restritivas. Existem diversos tipos de aplicativos que lidam com o transporte de mercadorias e pessoas, por exemplo, Uber, IFood, Uber Eats, 99 dentre outros. Estas modalidades de transporte são comumente investigadas na literatura, por serem capazes de deslocar de forma eficiente uma grande quantidade de mercadorias ou pessoas e, por proporcionarem fortes impactos no transporte mundial. Algumas vantagens destas modalidades de transporte impactam diretamente na economia, segurança e qualidade de vida dos usuários como, por exemplo, a minimização de engarrafamentos, diminuição dos tempos de viagens, minimização dos custos com transporte, redução da poluição e de áreas destinadas a estacionamento de veículos.

O problema investigado neste estudo, nomeado Problema do Caixeiro Viajante com Múltiplos Passageiros Bônus Optativos Quota e Tempo (PCV-MPBOQT) aborda o transporte de pessoas e mercadorias (também chamadas de bônus). Trata-se de uma aplicação importante e de difícil solução para a classe dos problemas de roteamento enriquecido. O PCV-MPBOQT consiste em um caixeiro transportador de mercadorias entre localidades de uma rede, que pode realizar embarques de passageiros em seu veículo para diminuir os custos de sua rota. Existem restrições que devem ser satisfeitas, algumas relacionadas ao tempo de rota, origem e destino de mercadorias e passageiros. O objetivo do caixeiro é encontrar uma configuração de transporte de mercadorias em uma rota que proporcione o maior lucro possível.

Quando o transporte de mercadorias é realizado, o caixeiro ganha bônus. Existe uma quantidade mínima de bônus que deve ser coletada. Caso uma mercadoria seja transportada o caixeiro deve obrigatoriamente carregá-la na sua origem e descarregá-la no seu destino. O serviço de coleta demanda uma quantidade de tempo para ser realizado. O mesmo ocorre na entrega. As quantidades de tempo gastas na coleta e entrega são somadas ao tempo de realização da rota. Existe uma restrição de tempo máximo para realizar o trajeto.

Similar ao transporte de mercadorias, os passageiros transportados devem ser obrigatoriamente embarcados na sua origem e desembarcados no seu destino. Quando o caixeiro embarca um ou mais passageiros os custos dos percursos desses passageiros são divididos com os demais ocupantes do veículo. Existe uma restrição de tarifa máxima para cada passageiro. Finalmente, existe uma restrição de capacidade do veículo quanto ao número de passageiros. Nesta variante, considera-se que a capacidade para transporte de mercadorias é ilimitada.

Os problemas Caixeiro Viajante com Passageiros (PCV-P) apresentado por Calheiros (2017), Caixeiro Viajante com Passageiros e Lotação (PCV-PL) apresentado por Bastos (2017), Caixeiro Viajante com Múltiplos Passageiros e Quota (PCV-MPQ) apresentado por Carvalho (2019), e Passeio Lucrativo com Passageiros e Restrições de Tempo e Custo (PPL-PRTC) apresentado por Petch (2018), são contextualizados na área dos problemas de roteamento enriquecido, similarmente ao problema alvo da pesquisa. O PCV-P aborda o transporte de passageiros para reduzir os custos de viagens de motoristas e passageiros, e foi solucionado com os algoritmos Genético, Memético, GRASP, e ACO, utilizando um conjunto de 95 instâncias de até 150 vértices. O PCV-PL também aborda o transporte de passageiros similarmente ao PCV-P, e diferentemente ao PCV-MPBOQT o caixeiro pode obter descontos no valor de pedágio por utilização de faixas de trânsito, também chamadas de *high-occupancy vehicle lanes*. O PCV-PL foi solucionado com os algoritmos *Simulated Annealing*, *Variable Neighborhood Search*, Colônia de Abelhas, e Genético, utilizando um conjunto de 95 instâncias de até 150 vértices. O PCV-MPQ aborda o carregamento de bônus e o transporte de passageiros, e foi solucionado com os algoritmos GRASP, ACO, e Colônia de Abelhas, utilizando um conjunto de 288 instâncias de até 100 vértices. E o PPL-PRTC aborda a realização de serviços de coleta e entrega de correspondências em uma cidade por um *courier* que pode transportar passageiros e, similarmente ao PCV-PL obter descontos no valor de pedágio. O PPL-PRTC foi solucionado com os algoritmos *Variable Neighborhood Search*, GRASP,

Genético, Memético, e Transgenético, utilizando um conjunto de 180 instâncias de até 300 vértices. Diferentemente do PCV-P e PCV-PL, o PCV-MPBOQT considera múltiplos passageiros nas localidades da rede demandando transporte. Também, diferentemente do PCV-MPQ, o PCV-MPBOQT considera: tempo de rota, múltiplos bônus nas localidades demandando transporte e, não obrigatório coletar o bônus da localidade visitada. O PCV-MPBOQT aborda um problema de roteamento enriquecido para transporte de passageiros, carregamento de mercadorias e, restritivo a tempo, diferente de qualquer outro abordado na literatura, mais próximo da realidade do transporte de passageiros que o problema PCV-P e, mais próximo da realidade do carregamento de mercadorias que o problema PCV-MPQ.

A Seção 2 apresenta a formulação matemática do PCV-MPBOQT. Métodos que são utilizados nas propostas heurísticas, são descritos na Seção 3. A Seção 4 apresenta uma heurística *naíve*. Heurísticas baseadas nas abordagens de Colônia de Formigas (ACO) e GRASP são apresentadas, respectivamente, nas Seções 5 e 6. A Seção 7 apresenta os resultados e uma análise de um experimento computacional. As conclusões finais são apresentadas na Seção 8.

2 DEFINIÇÃO DO PROBLEMA

O PCV-MPBOQT é definido por um grafo $G = (N, M, B)$, onde $N = \{1, ..., \#n\}$ é um conjunto de vértices ou localidades de uma rede, $M = \{1, ..., \#m\}$ um conjunto de arestas ou estradas que ligam as localidades da rede e $B = \{1, ..., \#b\}$ um conjunto de bônus associados às localidades. Também, considera-se um conjunto de passageiros $P = \{1, ..., \#p\}$. A coleta de uma mercadoria, e consequente recebimento do bônus, é opcional. À cada mercadoria está associado um bônus, vb_b, que representa o ganho pelo transporte da mercadoria da sua origem para o seu destino, uma origem o_b, destino, d_b, $o_b \neq d_b$, um tempo de carregamento, tc_b, e um tempo de descarregamento, td_b. A cada $p \in P$, estão associados uma origem, o_p, um destino, d_p, $o_p \neq d_p$, e um recurso máximo, w_p, que p está disposto a pagar para realizar sua viagem no veículo do caixeiro. A cada aresta $(i, j) \in M$ está associado um custo, c_{ij}, que é dividido igualmente com todos os ocupantes do veículo e representa o custo do deslocamento da localidade i para j e um tempo, tp_{ij}, que representa o tempo de deslocamento da localidade i para a j.

As variáveis e parâmetros a seguir são usados na formulação matemática.

- x_{ij}, variável binária que representa a utilização da aresta (i,j) pelo caixeiro e terá valor 1 se a aresta (i,j) é utilizada e 0, caso contrário;
- v_{ij}^{p}, variável binária que representa o transporte do passageiro *p* pela aresta (i,j) e terá valor 1 se o passageiro *p* é transportado pela aresta (i,j) e 0, caso contrário;
- u_i, variável inteira que representa a ordem do vértice *i* na rota do caixeiro;
- l_b, variável binária que representa o transporte da mercadoria *b*;
- m_{ij}^{b}, variável binária que representa o carregamento da mercadoria *b* pela aresta (i,j) e terá valor 1 se a mercadoria *b* é carregada pela aresta (i,j) e 0, caso contrário.
- Q, parâmetro real que representa a quota mínima de bônus que o caixeiro precisa transportar;
- K, parâmetro inteiro que representa a capacidade máxima de passageiros no veículo do caixeiro;
- vb_b, parâmetro real que representa o valor do bônus da mercadoria *b*;
- tc_b, parâmetro inteiro que representa o tempo de carregamento da mercadoria *b* em segundos;
- td_b, parâmetro inteiro que representa o tempo de descarregamento da mercadoria *b* em segundos;
- tp_{ij}, parâmetro inteiro que representa o tempo de percurso da aresta (i,j) em segundos;
- T, parâmetro inteiro que representa o tempo máximo da rota em segundos;

A formulação apresentada em 2.1 a 2.30 mostra o modelo matemático proposto para o PCV-MPBOQT. A função objetivo expressa na equação 2.1 realiza o cálculo do lucro líquido do percurso, diminuindo do lucro obtido o custo dividido igualmente com todos os ocupantes do veículo, visando maximizar o lucro líquido do caixeiro.

$$(2.1)\quad maximizar\ L = \sum_{b\in B} l_b vb_b - \sum_{(i,j)\in M} \frac{x_{ij}c_{ij}}{\sum_{p\in P} v_{ij}^{p} + 1}$$

Sujeito a:

$$(2.2)\quad \sum_{i\in N\setminus\{j\}} x_{ij} \leq 1\,, \qquad \forall j \in N\setminus\{s\}$$

$$(2.3)\quad \sum_{i\in N\setminus\{j\}} x_{ji} \leq 1\,, \qquad \forall j \in N\setminus\{s\}$$

$$(2.4)\quad \sum_{i\in N\setminus\{s\}} x_{si} = 1$$

$$(2.5)\quad \sum_{i\in N\setminus\{s\}} x_{is} = 1$$

$$(2.6)\quad \sum_{i\in N\setminus\{j\}} x_{ij} - \sum_{i\in N\setminus\{j\}} x_{ji} = 0, \qquad \forall j \in N\setminus\{s\}$$

$$(2.7)\quad u_i - u_j + (n-1)x_{ij} \leq n-2, \qquad 2 \leq i,j \leq n, \qquad i \neq j$$

$$(2.8)\quad m_{ij}^b \leq x_{ij}, \qquad \forall b \in B, \qquad \forall (i,\, j) \in M, \qquad i \neq j$$

$$(2.9)\quad \sum_{j\in N\setminus\{i\}} m_{ji}^b - \sum_{j\in N\setminus\{i\}} m_{ij}^b = 0, \qquad \forall b \in B, \qquad i \in N\setminus\{o_b,\, d_b\}$$

$$(2.10)\quad \sum_{i\in N\setminus\{o_b\}} m_{io_b}^b + \sum_{i\in N\setminus\{d_b\}} m_{d_bi}^b = 0, \qquad \forall b \in B$$

$$(2.11)\quad \sum_{i\in N\setminus\{s\}} m_{si}^b = 0, \qquad \forall b \in B, \qquad o_b \neq s$$

$$(2.12)\quad \sum_{i\in N\setminus\{o_b\}} m_{o_bi}^b = l_b\,, \qquad \forall b \in B$$

$$(2.13)\quad \sum_{i\in N\setminus\{d_b\}} m_{id_b}^b = l_b, \qquad \forall b \in B$$

$$(2.14)\quad \sum_{b\in B} l_b vb_b \geq Q$$

$$(2.15)\quad \sum_{b\in B} tc_b l_b + \sum_{b\in B} td_b l_b + \sum_{\substack{(i,j)\,\in\, M \\ i\neq j}} tp_{ij}x_{ij} \leq T$$

$$(2.16)\quad \sum_{p\in P} v_{ij}^p \leq K x_{ij}, \qquad \forall (i,\, j) \in M, \qquad i \neq j$$

$$(2.17)\quad \sum_{(i,j)\in M} \frac{v_{ij}^p c_{ij}}{\sum_{p\in P} v_{ij}^p + 1} - w_p \leq 0\,, \qquad \forall p \in P$$

$$(2.18)\quad \sum_{j\in N\setminus\{i\}} v_{ji}^p - \sum_{j\in N\setminus\{i\}} v_{ij}^p = 0, \qquad \forall p \in P, \qquad i \in N\setminus\{o_p,\, d_p\}$$

$$(2.19)\quad \sum_{i\in N\setminus\{o_p\}} v_{io_p}^p + \sum_{i\in N\setminus\{d_p\}} v_{d_pi}^p = 0, \qquad \forall p \in P$$

$$(2.20)\quad \sum_{i\in N\setminus\{s\}} v_{si}^p = 0, \qquad \forall p \in P, \qquad o_p \neq s$$

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

$$(2.22)\quad v_{ij}^p \in \{0,1\}, \qquad \forall (i,j) \in M, \qquad \forall p \in P$$

$$(2.23)\quad l_b \in \{0,1\}, \qquad \forall b \in B$$

$$(2.24)\quad m_{ij}^b \in \{0,1\}, \qquad \forall (i,j) \in M, \qquad \forall b \in B$$

$$(2.25)\quad 1 \leq u_i \leq n-1, \qquad 2 \leq i \leq n$$

$$(2.26)\quad c_{ij} \in \mathbb{R}^+, \qquad \forall (i,j) \in M$$

$$(2.27)\quad vb_b \in \mathbb{R}^+, \qquad \forall b \in B$$

$$(2.28)\quad tc_b \in \mathbb{Z}^+, \qquad \forall b \in B$$

$$(2.39)\quad td_b \in \mathbb{Z}^+, \qquad \forall b \in B$$

(2.30) $tp_{ij} \in \mathbb{Z}^+, \qquad \forall (i,j) \in M$

As restrições de 2.2 a 2.7 estabelecem as condições necessárias de formação de uma única rota considerando um subconjunto de localidades de *N*. As restrições 2.4 e 2.5 garantem o início e o término da rota no vértice origem. A restrição 2.6 obriga que qualquer vértice visitado pelo caixeiro possua uma aresta de entrada e uma de saída. A restrição 2.7 assegura a formação de um único ciclo na solução (Miller et al., 1960). A restrição 2.8 garante que o bônus é transportado apenas em arestas que fazem parte da rota. A restrição 2.9 garante que os bônus em fluxo continuarão em fluxo exceto no destino e origem. A restrição 2.10 garante que os bônus não são carregados antes do vértice de origem do bônus na rota ou que permaneçam no veículo após o caixeiro visitar o vértice de destino do bônus. A restrição 2.11 proíbe o carregamento de qualquer bônus em *s*, se ele não tiver origem em *s*. As restrições 2.12 e 2.13 garantem que todo bônus transportado será carregado na sua origem e descarregado no seu destino. A restrição 2.14 obriga a coleta da quota mínima de bônus *Q*. A restrição 2.15 garante que o tempo máximo da rota não seja ultrapassado. A restrição 2.16 garante que a capacidade máxima de passageiros do veículo, *K*, não será ultrapassada. A restrição 2.17 assegura que a despesa total de cada passageiro $p \in P$ não ultrapasse w_p. A restrição 2.18 garante que os passageiros em fluxo continuarão em fluxo exceto no destino e origem. A restrição 2.19 garante que os passageiros não são embarcados antes do vértice de embarque e, que permaneçam no veículo após o caixeiro visitar o vértice de destino do passageiro. A restrição 2.20 proíbe o embarque de qualquer passageiro em *s*, se ele não tiver origem em *s*. As restrições de 2.21 a 2.30 definem o escopo das variáveis.

Uma solução do problema é um vetor de vértices, que representa a rota, um vetor para cada vértice, que representa os passageiros que serão embarcados no vértice e, um segundo vetor para cada vértice, que representa os bônus que serão carregados no vértice.

Segundo a formulação matemática proposta, é possível perceber que o PCV-MPBOQT compartilha características presentes em diversos problemas de roteamento enriquecido, todavia constitui uma aplicação com elementos próprios. O PCV-MPBOQT possui a característica do embarque de passageiros presente no PCV-P, PCV-PL, PCV-MPQ e PPL-PRTC. Também, possui a característica da coleta da quota mínima presente no Problema do Caixeiro Viajante com Cotas (PCV-Q) (Awerbuch et al., 1998) e PCV-MPQ.

3 MÉTODOS UTILIZADOS NAS HEURÍSTICAS

Esta seção apresenta métodos que são compartilhados pelas heurísticas apresentadas nas Seções 4, 5 e 6. Dentre tais métodos, está uma heurística para o carregamento de mercadorias na rota, chamada de heurística de carregamento de bônus, introduzida na Seção 3.1. A seção 3.2 apresenta operadores de busca local.

3.1 CARREGAMENTO DE BÔNUS

Uma vez construída uma rota, é aplicada uma heurística que define quais mercadorias são coletadas. A coleta da mercadoria tem como consequência a coleta do bônus a ela associado e a computação dos tempos gastos para carregamento e descarregamento.

O pseudocódigo do algoritmo para o carregamento de bônus é apresentado na Figura 1. O algoritmo possui três parâmetros de entrada: rota (*rs*), tempo gasto em *rs* (*ts*), e tempo máximo admitido em uma rota (*T*). Inicialmente, o algoritmo computa o conjunto de mercadorias que podem ser coletadas na rota (*cbp*). Os vértices de carregamento e descarregamento da mercadoria devem pertencer à *rs*, e a direção da rota deve ser compatível, isto é, o caixeiro passa no vértice de carregamento antes do vértice de descarregamento. A variável *quant_bp* guarda a cardinalidade do conjunto *cbp*. No laço principal são escolhidas as mercadorias que serão transportadas em *rs*. No passo 4, é sorteada uma mercadoria (aqui chamada de bônus) pelo método da roleta. Quanto maior o bônus associado, maior a probabilidade da mercadoria ser sorteada. As variáveis *tc* e *td* guardam o tempo necessário para o carregamento e descarregamento da mercadoria, respectivamente. No passo 7, verifica-se a restrição de tempo máximo da rota. A variável *cbs*, inicialmente vazia, guarda o conjunto de mercadorias, escolhidas para transporte. Caso o transporte da mercadoria sorteada não inviabilize o tempo da rota, a variável *cbs* é atualizada com a inclusão de *bonus*. O tempo da rota é atualizado no passo 9.

Figura 1: Pseudocódigo do algoritmo CarregamentoBonus

	CarregamentoBonus(*rs*, *ts*, *T*)
1	*cbp* = DefinirBonusPossiveis(*rs*);
2	*quant_bp* = Cardinalidade(*cbp*);
3	**PARA** *x* = 1 até *quant_bp* **FAÇA**:
4	*bonus = roleta(cbp);*
5	*tc* = tempo_carregamento(*bonus*);
6	*td* = tempo_descarregamento(*bonus*);
7	**SE** *tc* + *td* + *ts* ≤ *T* **ENTÃO**
8	*cbs* = AtualizarBonus(*bonus*);
9	*ts = ts + tc + td;*

3.2 OPERADORES DE BUSCA LOCAL

Foram desenvolvidos três operadores de busca local, chamados de InsereLKH, RemoveLKH e LKH-InsereRemove. Os três utilizam a heurística de Lin-Kernighan (LKH) conforme implementada por Helsgaun(2000).

InsereLKH insere um vértice não visitado pelo caixeiro no final de uma rota passada como parâmetro de entrada. Depois, aplica LKH com o objetivo de minimizar o custo da rota, sendo este custo calculado em função dos custos das arestas da rota. O procedimento testa a inserção de cada vértice fora da rota.

RemoveLKH remove um vértice da rota passada como parâmetro e aplica a heurística LKH. A remoção de cada vértice da rota é testada.

O operador LKH-InsereRemove gera soluções de três formas: primeira realizando na rota busca LKH com foco na minimização do seu custo, segunda inserindo um vértice não visitado pelo caixeiro no final da rota e, terceira removendo um vértice visitado pelo caixeiro. A segunda forma verifica a inserção de cada vértice não pertencente a rota e, a terceira forma verifica a remoção de cada vértice pertencente a rota. O LKH-InsereRemove recebe como parâmetro de entrada uma rota de solução (*rs*). Inicialmente faz um backup de *rs*, tempo de *rs* (*ts*) e bônus de *rs* (*bs*). Em seguida, realiza busca LKH na rota e, depois, gera soluções inserindo um vértice não visitado pelo caixeiro no final da rota e, logo após, gera soluções removendo um vértice visitado pelo caixeiro. Esse processo é realizado enquanto condição 1(conjunto de vértices da rota for modificado) e condição 2(melhor solução (*ms*) for atualizada) forem verdadeiras. O pseudocódigo do operador LKH-InsereRemove é apresentado na Figura 2.

Cada rota produzida nos procedimentos InsereLKH, RemoveLKH e LKH-InsereRemove é submetida ao modelo matemático exato de embarque de passageiros descrito no trabalho de Calheiros (2017), com o auxílio de um solver. Depois, a rota é submetida à heurística de carregamento de bônus, definida na Seção 3.1. A solução final é comparada com a melhor solução corrente. A melhor solução é mantida.

Figura 2: Pseudocódigo do algoritmo LKH-InsereRemove

	LKH-InsereRemove(*rs*)
1	**FAÇA:**
2	*rs'* = *rs*; *ts'* = *ts*; *bs'* = *bs*;
3	*vnv* = DefinirVerticesNaoVisitados(*rs'*);
4	*rs'* = buscaLKH (*rs'*);
5	EmbarqueOtimo (*rs'*);
6	CarregamentoBonus (*rs', ts', T*);
7	AtualizarMelhorSolucao(*ms*, *rs'*, *ts'*, *bs'*);
8	**PARA** *x* = 1 até Cardinalidade(*vnv*) **FAÇA**:
9	Insere vértice vnv_x no final de *rs'*;
10	EmbarqueOtimo (*rs'*);
11	CarregamentoBonus (*rs', ts', T*);
12	Remove vértice vnv_x de *rs'*;
13	**PARA** *x* = 2 até Cardinalidade(*rs*) **FAÇA**:
14	Remover vértice rs_x de *rs'*;
15	EmbarqueOtimo (*rs'*);
16	CarregamentoBonus (*rs', ts', T*);
17	Insere vértice rs_x em *rs'*;
18	**SE** encontrou uma melhor solução **ENTÃO**
19	*rs* = melhor solução *rs'*;
20	*ts* = tempo da melhor solução *rs'*;
21	*bs* = bônus da melhor solução *rs'*;
22	AtualizarMelhorSolucao(*ms*, *rs*, *ts*, *bs*);
23	**Enquanto** ((*rs* foi modificado) e (*ms* foi modificado));

4 HEURÍSTICA NAÍVE

Foi desenvolvida uma heurística Naíve com o objetivo de encontrar as primeiras soluções viáveis para o PCV-MPBOQT. Esta heurística busca melhorias em soluções utilizando a heurística LKH (Helsgaun, 2000). Define os passageiros embarcados nos vértices de forma exata, utilizando o modelo de programação matemática para embarque de passageiros apresentado em Calheiros (2017). E realiza a definição dos bônus carregados dos vértices com o algoritmo de carregamento de bônus descrito na seção 3.1.

Inicialmente, o algoritmo heurístico Naíve seleciona um conjunto de vértices (*cl*), utilizando uma roleta composta pelas médias de bônus dos vértices. A média de bônus de um vértice é calculada dividindo a soma dos valores dos bônus que solicitam transporte no vértice pela quantidade de bônus que possui origem no vértice. No passo 2, de forma sequencial, gera uma solução (*s*) utilizando o LKH com foco na minimização do custo da rota. Em seguida, realiza o carregamento de bônus com o auxílio do tempo da solução (*ts*) e o tempo máximo (*T*). Caso a solução gerada pelo LKH não satisfaça a quota mínima, insere o vértice com a maior média de bônus não pertencente à rota e, realiza busca LKH, enquanto a condição 1 (solução não atingiu a quota mínima) e condição 2 (solução não visitou todas as localidades) forem verdadeiras. Por fim, define o embarque ótimo de passageiros da solução. O pseudocódigo da heurística Naíve é apresentado na Figura 3.

Figura 3: Pseudocódigo do algoritmo heurístico Naíve

```
Naíve()
1  FAÇA:
2    s = buscaLKH (cl);
3    CarregamentoBonus (s, ts, T);
4    SE s não satisfaz a quota mínima:
5      cl ← localidade com maior média de bônus não pertencente a cl;
6  Enquanto ((s não atingiu a quota mínima) e (s não visitou todas as localidades));
7  EmbarqueOtimo (s);
8  RETORNAR s;
```

5 ACO

Ant Colony Optimization (ACO) é uma meta-heurística que utiliza um conjunto de formigas artificiais para construir soluções para um dado problema. O algoritmo ACO proposto e implementado foi desenvolvido inspirado na clássica versão ACO proposta por Dorigo e Di Caro (1999), com ajustes necessários e relacionados ao contexto do problema. Sua estrutura pode ser dividida em 3 partes: primeira parte, construção de soluções por formigas artificiais; segunda parte, aplicação de buscal local; terceira parte, atualização do feromônio. Os operadores InsereLKH e RemoveLKH são utilizados na busca local e, a atualização do feromônio é realizada da maneira Ant-cycle, como abordado por Dorigo e Di Caro (1999). Para definir de forma exata os passageiros embarcados nas soluções, foi utilizado o modelo de programação matemática para embarque de passageiros apresentado em Calheiros (2017).

As formigas da versão clássica do ACO constroem soluções a partir de dois parâmetros definidos por feromônio τ e atratividade η. Como regra geral, recebem mais feromônios os percursos de rota mais utilizados. A equação (5.1) modela o processo de construção das soluções, onde o parâmetro α representa o peso referente ao feromônio e o parâmetro β representa o peso referente a atratividade. N_i^k representa as localidades vizinhas a localidade *i* e não visitadas pela formiga *k*. O *i* refere-se a última localidade visitada pela formiga *k*.

$$(5.1)\quad p_{ij}^k = \begin{cases} \dfrac{\tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta}}{\sum_{x \in N_i^k} \tau_{i,x}^{\alpha} \cdot \eta_{i,x}^{\beta}} & \forall j \in N_i^k \\ 0 & caso\ contrário \end{cases}$$

O processo de atualização de feromônio Ant-cycle é realizado com as equações (5.2) e (5.3), onde $\Delta_{\tau_{ij}^k}$ representa o valor do cálculo do novo deposito de feromônio e, p representa o processo de evaporação. No contexto do problema, L_k representa o lucro líquido da rota da formiga *k*, dado pela equação (2.1).

$$(5.2)\ \tau_{i,j} = (1-p)\cdot\tau_{i,j} + \sum_{k=1}^{m} \Delta_{\tau_{ij}^k}$$

$$(5.3)\ \Delta_{\tau_{ij}^k} = \frac{1}{L_k}$$

A parte construtiva do ACO proposto utilizou cinco equações, que o trabalho propõe para modelar o processo de construção das soluções, com o objetivo de englobar as características custo, bônus, passageiros e tempo do PCV-MPBOQT. A primeira equação é representada na equação (5.1), e modela o processo de construção com foco no custo, com $\eta_{i,j} = \frac{1}{c_{ij}}$. A segunda equação é representada na equação (5.4), e modela o processo de construção com foco na coleta de bônus, com λ_j representando a média de bônus da localidade *j* e, γ representando o peso de λ_j. A terceira equação é representada na equação (5.5), e modela o processo de construção com foco no embarque de passageiros, com ω_j representando a quantidade de passageiros solicitando embarque na localidade *j* e com destino não visitado pela formiga *k* e, δ representando o peso de ω_j. A quarta equação é representada na equação (5.6), e modela o processo de construção com foco no tempo de rota, com t_{ij} representando o tempo da aresta *i-j* e, ε representando o peso de t_{ij}. A quinta equação é representada na equação (5.7), e modela o processo de construção com foco no custo, bônus, passageiro e tempo. Cada formiga possui um tipo e utiliza uma equação de construção, tipo 1 utiliza a primeira equação (5.1), tipo 2 utiliza a segunda equação (5.4), tipo 3 utiliza a terceira equação (5.5), tipo 4 utiliza a quarta equação (5.6) e, tipo 5 utiliza a quinta equação (5.7). O tipo da formiga *k* (tif_k) é usado para definir qual equação de construção utilizar, para escolher a localidade e mover a formiga *k*. A cada inserção de vértice na rota da formiga *k* (rf_k), é realizado o carregamento de uma quantidade (definida aleatoriamente) de bônus ($quant\_bcf_k$), definidos utilizando uma roleta que recebe como parâmetro os bônus possíveis de carregar da formiga *k* ($bpcf_k$). O bônus (bf_k) e o tempo (tf_k) de uma formiga *k* são atualizados, respectivamente, pelas funções AtualizarBonusS e AtualizarTempoS. Logo após, se o tempo da formiga ultrapassou o tempo máximo (*T*), é realizado um reparo na solução, utilizando a função ReparoS, que retira da solução o vértice que menos embarca e desembarca passageiros e realiza carregamento de bônus utilizando uma roleta até atingir a quota mínima. Caso um reparo de solução não for possível, a solução é definida como inviável (não satisfaz todas

as restrições do modelo proposto) e não é utilizada nos próximos procedimentos da iteração. Uma quantidade de formigas (m) é criada. A Figura 4 mostra o pseudocódigo da parte construtiva do algoritmo ACO desenvolvido.

$$(5.4)\quad p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha} . [\lambda_j]^{\gamma}}{\sum_{j \in N_i^k} [\tau_{ij}]^{\alpha} . [\lambda_j]^{\gamma}} & \forall j \in N_i^k \\ 0 & caso\ contrário \end{cases}$$

$$(5.5)\quad p_{ij}^{k} = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha} . [\omega_j]^{\delta}}{\sum_{j \in N_i^k} [\tau_{ij}]^{\alpha} . [\omega_j]^{\delta}} & \forall j \in N_i^k \\ 0 & caso\ contrário \end{cases}$$

$$(5.6)\quad p_{ij}^{k} = \begin{cases} \dfrac{\dfrac{[\tau_{ij}]^{\alpha}}{[t_{ij}]^{\varepsilon}}}{\sum_{j \in N_i^k} \dfrac{[\tau_{ij}]^{\alpha}}{[t_{ij}]^{\varepsilon}}} & \forall j \in N_i^k \\ 0 & caso\ contrário \end{cases}$$

$$(5.7)\quad p_{ij}^{k} = \begin{cases} \dfrac{\dfrac{[\tau_{ij}]^{\alpha} . [\eta_{ij}]^{\beta} . [\lambda_j]^{\gamma} . [\omega_j]^{\delta}}{[t_{ij}]^{\varepsilon}}}{\sum_{j \in N_i^k} \dfrac{[\tau_{ij}]^{\alpha} . [\eta_{ij}]^{\beta} . [\lambda_j]^{\gamma} . [\omega_j]^{\delta}}{[t_{ij}]^{\varepsilon}}} & \forall j \in N_i^k \\ 0 & caso\ contrário \end{cases}$$

Figura 4: Pseudocódigo do algoritmo ConstrutorMultiploFormiga

ConstrutorMultiploFormiga()	
1	**FAÇA:**
2	**PARA** k = 1 até m **FAÇA:**
3	**SE** a formiga k não atingiu a quota mínima e não é inviável:
4	Escolher a localidade j para mover a formiga k do tipo tif_k com sua respectiva
5	equação ((5.1) ou (5.4) ou (5.5) ou (5.6) ou (5.7)) de construção;
6	Colocar a localidade j em rf_k;
7	$bpcf_k$ = DefinirBonusPossiveis(rf_k);
8	**PARA** x = 1 até $quant\_bcf_k$ **FAÇA**:
9	$bonus = roleta(bpcf_k);$
10	AtualizarBonusS(bf_k, $bonus$); //atualiza o bônus da formiga k
11	AtualizarTempoS(tf_k, rf_k, bf_k); //atualiza o tempo da formiga k
12	**SE** tf_k é maior que o tempo máximo de rota:
13	ReparoS(rf_k); //repara a solução da formiga k
14	**SE** o reparo da formiga k não foi possível:
15	Definir a formiga k como inviável;
16	**SE NÃO SE** ((bf_k é menor que a quota mínima) e (rf_k visitou todas as localidades)):
17	Carregar bônus na formiga k até atingir a quota mínima;
18	**SE** a formiga k não é viável:
19	Definir a formiga k como inviável;
20	**ATÉ** não existir formiga viável que não atingiu a quota mínima;

Primeiramente o algoritmo ACO proposto inicia o feromônio e visibilidade dos trechos de rota, em seguida, define o tipo das formigas (*TIF*) com o procedimento DefinirTipoFormiga e, constrói soluções com o algoritmo ConstrutorMultiploFormiga. Após a parte construtiva, é realizada para cada formiga o cálculo do seu lucro (lf_k) e custo (cf_k). Em seguida, realiza a execução dos operadores InsereLKH e RemoveLKH que tentam melhorar a melhor solução da iteração (*msi*), para posteriormente, atualizar os parâmetros α, γ, δ e ε das formigas tipo 5 com o procedimento AtualizarFormigasTipo5 e, atualizar o tipo das formigas com o procedimento AtualizarTipoFormiga. Na atualização dos parâmetros se a melhor formiga da iteração for do tipo 1 aumenta o parâmetro α, se a melhor formiga da iteração for do tipo 2 aumenta o parâmetro γ, se a melhor formiga da iteração for do tipo 3 aumenta o parâmetro δ, se a melhor formiga da iteração for do tipo 4 aumenta o parâmetro ε, se a pior formiga da iteração (*psi*) for do tipo 1 diminui o parâmetro α, se a pior formiga da iteração for do tipo 2 diminui o parâmetro γ, se a pior formiga da iteração for do tipo 3 diminui o parâmetro δ e, se a pior formiga da iteração for do tipo 4 diminui o parâmetro ε. Na atualização do tipo das formigas, aumenta em uma unidade a quantidade de formigas do tipo da melhor formiga da iteração e diminui em uma unidade a quantidade de formigas do tipo da pior formiga da iteração. Por fim, realiza a atualização do feromônio com as equações (5.2) e (5.3) e, a visibilidade dos trechos de rota. A *Condição de parada* do algoritmo é quando realizar 10 iterações sem melhoria da melhor solução (*ms*). Na Figura 5 mostra o pseudocódigo do algoritmo ACO proposto.

Figura 5: Pseudocódigo do algoritmo ACO

```
ACO()
1  Iniciar feromônio e visibilidade dos trechos de rota;
2  DefinirTipoFormiga (TIF);
3  FAÇA:
4    ConstrutorMultiploFormiga(RF, BF, TF, TIF, m);
5    PARA k = 1 até m:
6      SE a formiga k é viável:
7        CarregamentoBonus(rf_k, tf_k, T);
8        cf_k = EmbarqueOtimo(rf_k);
9        lf_k = bf_k - cf_k;
10       Atualizar msi e psi;
11   InsereLKH(msi);
12   RemoveLKH(msi);
13   SE msi tem maior lucro líquido que ms:
14     ms = msi;
15   AtualizarFormigasTipo5(RF, TIF, msi, psi);
16   AtualizarTipoFormiga(TIF, msi, psi);
17   Atualizar o feromônio e a visibilidade dos trechos de rota;
18   Esvaziar as listas rf_k;
19 ATÉ Condição de parada;
20 RETORNAR ms;
```

6 GRASP

Greedy Randomized Adaptive Search Procedure (GRASP) é um algoritmo meta-heurístico bastante usado em problemas de otimização combinatória. Foi criado por Feo e Resende (1989). Sua estrutura clássica é composta basicamente por duas partes: primeira parte, criar uma solução com um procedimento construtivo semi-guloso; segunda parte, melhorar a solução utilizando um procedimento de busca local.

O GRASP desenvolvido utiliza a estrutura clássica, com adaptações indispensáveis para o contexto do problema. Também utiliza o modelo de programação matemática para embarque de passageiros apresentado em Calheiros (2017), para definir de forma exata os passageiros embarcados dos vértices. Na parte construtiva, cada vértice inserido na rota da solução (*s*) é selecionado por uma roleta composta por um conjunto de dados (*lpvs*), definidos pelo procedimento DefinirLocalidadesPossiveis e referentes às médias de bônus dos vértices não pertencentes a rota da solução. O DefinirLocalidadesPossiveis gera o conjunto de dados verificando quais vértices não pertencem a rota da solução. A cada inserção de vértice na rota da solução, é realizado o carregamento de todos os bônus possíveis utilizando uma roleta e o procedimento AtualizarBonusS que definem o bônus da solução (*bs*) e, é atualizado o tempo da solução (*ts*) com o procedimento AtualizarTempoS. A roleta utilizada no carregamento de bônus utiliza um conjunto de dados (*bpcs*), definidos pelo procedimento DefinirBonusPossiveis, e referentes aos valores dos bônus não transportados pelo caixeiro. O DefinirBonusPossiveis gera o conjunto de dados verificando quais bônus não foram transportados na rota da solução e, gera a quantidade desses dados (*quant_bpcs*). Logo após, se o tempo da rota da solução ultrapassou o tempo máximo de rota, é realizado um reparo na solução, com o procedimento ReparoS, que retira da rota da solução o vértice que menos embarca e desembarca passageiros e, realiza carregamento de bônus utilizando uma roleta até atingir a quota mínima. Após a execução do ReparoS, se o reparo da solução não foi possível, a solução é definida como inviável e não utilizada nas etapas seguintes da iteração. Posteriormente a definição da rota, calcula o seu custo (*cs*) e lucro (*ls*). Na parte busca local, aplica-se o operador LKH-InsereRemove apresentado na seção 3.2, enquanto a melhor solução (*ms*) atualizar para uma solução de maior lucro líquido. A condição de parada do algoritmo é 100 iterações sem melhoria da melhor solução. A Figura 6 mostra o pseudocódigo do algoritmo GRASP desenvolvido.

Figura 6: Pseudocódigo do algoritmo GRASP

```
GRASP()
1   FAÇA:
2     FAÇA:
3       lpvs = DefinirLocalidadesPossiveis(s);
4       x = roleta(lpvs);
5       Colocar a localidade x em s;
6       bpcs = DefinirBonusPossiveis(s);
7       PARA x = 1 até quant_bpcs FAÇA:
8         bonus = roleta(bpcs);
9         AtualizarBonusS(bs, bonus);
10      AtualizarTempoS(ts, s, bs);
11      SE ts é maior que o tempo máximo de rota:
12        ReparoS(s);
13        SE o reparo da solução s não foi possível:
14          Definir a solução s como inviável;
15      SE NÃO SE ((bs é menor que a quota mínima) e (s visitou todas as localidades)):
16          Definir a solução s como inviável;
17    ENQUANTO ((bs não atingiu a quota mínima) e (s é viável));
18    SE s é viável:
19      cs = EmbarqueOtimo(s);
20      ls = bs - cs;
21      SE s tem maior lucro líquido que ms:
22        ms = s;
23      FAÇA:
24        LKH-InsereRemove(s);
25        SE s tem maior lucro líquido que ms:
26          ms = s;
27      ENQUANTO (ms atualizou);
28  ATÉ Condição de parada;
29  RETORNAR ms;
```

7 EXPERIMENTOS

Foram realizados experimentos computacionais com o objetivo de comparar os resultados obtidos com o modelo matemático, com a Naíve e, com as duas meta-heurísticas ACO e GRASP. Os algoritmos foram desenvolvidos utilizando a linguagem de programação C++ e, foram executados em um computador com a configuração que se segue, processador Intel Core i7-2600K de 3.40 GHz, 4 GB de RAM e sistema operacional Linux Ubuntu 16.04 LTS de 64 bits.

A Naive, os operadores, o ACO e o GRASP propostos utilizaram o solver Gurobi versão 7.52 como auxilio para executar o modelo de programação matemática para embarque de passageiros. Os operadores e a Naive utilizam a heurística LKH versão 2.0.8, implementação da heurística Lin-Kernighan (Helsgaun, 2000).

Todos os algoritmos foram executados em um conjunto de instâncias proposto e composto de 12 instâncias. O conjunto possui instâncias com matriz de distância simétrica e assimétrica. Os custos das arestas, os tempos das arestas, os bônus dos vértices, os tempos de carregamento e descarregamento dos bônus, a quantidade de

passageiros nos vértices e, os destinos dos passageiros foram definidos aleatoriamente em sorteio uniforme. Nas instâncias simétricas o custo e o tempo das arestas são definidos aleatoriamente para a aresta e nas instâncias assimétricas são definidos para cada direção. A nomeação das instâncias foi realizada com dois atributos separados com um traço, a saber: quantidade de vértices – quantidade de passageiros. Outras características das instâncias propostas são apresentadas na Tabela 1. No processo de parametrização foi utilizado um outro conjunto de instâncias composto com 6 instâncias, também criadas seguindo as mesmas características do processo de formação de instância descrito neste parágrafo e na Tabela 1.

Tabela 1: Características das instâncias propostas

Característica	**Valor(es)**
Quantidade máxima de passageiros do veículo	4
Quantidade de vértices	10, 20, 30, 40, 50, 100
Custo das arestas	50 até 350
Tempo das arestas	250s até 1000s
Bônus dos vértices	150 até 450
Tempo de carregamento de bônus	25s até 100s
Tempo de descarregamento de bônus	25s até 100s
Quantidade de passageiros nos vértices	0 até 6

O modelo matemático proposto foi executado no mesmo ambiente de execução dos algoritmos, utilizando o *solver* Gurobi da mesma versão utilizada pelas meta-heurísticas. Foi executado em instâncias com quantidade de vértices 10 e 20, tempo limite de 80.000 segundos, utilizando apenas um *thread*. As meta-heurísticas foram executadas 30 vezes para cada instância.

O *IRACE - Iterated Race for Automatic Algorithm Configuration* (López-Ibánez et al., 2011), foi utilizado para definir os valores dos parâmetros do ACO desenvolvido que se segue, quantidade de formigas tipo 1 (*quant-formiga-tipo1*), quantidade de formigas tipo 2 (*quant-formiga-tipo2*), quantidade de formigas tipo 3 (*quant-formiga-tipo3*), quantidade de formigas tipo 4 (*quant-formiga-tipo4*), quantidade de formigas tipo 5 (*quant-formiga-tipo5*), α, β, γ, δ, ε, p, valor que aumenta o parâmetro das formigas tipo 5 (*p-a*) e, valor que diminui o parâmetro das formigas tipo 5 (*p-d*). A Tabela 2 exibe os limites definidos e as configurações retornadas pelo *IRACE*.

Tabela 2: Limites definidos para os parâmetros do ACO desenvolvido, melhor configuração retornada pelo *IRACE* para as instâncias simétricas e, melhor configuração retornada pelo *IRACE* para as instâncias assimétricas

Variável	Tipo	Mínimo	Máximo	Inst. simétrica	Inst. assimétrica
quant-formiga-tipo1	Inteiro	5	10	8	5
quant-formiga-tipo2	Inteiro	5	10	9	8
quant-formiga-tipo3	Inteiro	5	10	8	9
quant-formiga-tipo4	Inteiro	5	10	10	7
quant-formiga-tipo5	Inteiro	5	10	10	6
α	Real	1.0	2.0	1.55	1.47
β	Real	3.0	4.0	3.46	3.42
γ	Real	5.0	6.0	5.71	5.85
δ	Real	7.0	8.0	7.7	7.18
ε	Real	9.0	10.0	9.14	9.59
p	Real	0.1	1.0	0.88	0.86
p-a	Real	0.1	1.0	0.13	0.46
p-d	Real	0.1	1.0	0.24	0.43

Foi utilizado um teste estatístico não-paramétrico na análise dos resultados das meta-heurísticas. O teste estatístico foi o de Mann e Whitney (1947), também conhecido como U-Teste (Conover, 1999). Esse teste: utilizou como nível de significância o valor 0,05; foi realizado utilizando o programa R (R Core Team, 2018) versão 3.6.2, 64 bits; foi realizado para cada instância; e utilizou as soluções das 30 execuções de cada algoritmo meta-heurístico em cada instância.

7 RESULTADOS

Os resultados das execuções do modelo matemático, da heurística e das meta-heurísticas são apresentados nas Tabelas 3, 4 e 5. Os resultados dos testes estatísticos são apresentação na Tabela 6. A coluna Instância apresenta o nome da instância, que relaciona suas características. Nas Tabelas 3 e 4, a coluna Lucro exibe o melhor lucro obtido e, a coluna Tempo exibe o tempo de execução em segundos. Na tabela 5, as colunas ACO e GRASP apresentam os valores médios dos lucros e tempos médios de execução em segundos obtidos nas 30 execuções de cada algoritmo meta-heurístico. Na Tabela 6, a segunda coluna apresenta os *p-valor* dos testes estatísticos.

Tabela 3: Resultados do *solver*

Instância	Gurobi	
	Lucro	Tempo
10-16	5065,35	223,22
20-40	27277,80	80000,00
10-20	5313,38	151,22
20-43	26735,50	80000,00

Tabela 4: Resultados do algoritmo heurístico

Instância	NAÍVE	
	Lucro	Tempo
10-16	3884,24	0,01
20-40	17352,40	0,01
30-78	34475,90	0,02
40-75	66906,00	0,02
50-107	97798,50	0,03
100-224	394773,00	0,22
10-20	3498,68	0,01
20-43	16652,00	0,01
30-59	35043,20	0,01
40-88	62024,70	0,02
50-118	102214,00	0,03
100-198	379201,00	0,31

Tabela 5: Resultados dos algoritmos meta-heurísticos

Instância	ACO		GRASP	
	Lucro	Tempo	Lucro	Tempo
10-16	4781,89	0,86	4815,77	2,06
20-40	**26684,62**	3,58	23601,32	7,51
30-78	**60465,06**	14,18	52031,62	17,04
40-75	**116020,60**	44,00	101095,03	29,19
50-107	**166189,03**	78,47	147545,47	66,66
100-224	**686116,10**	968,00	554424,47	820,65
10-20	5034,97	0,98	5047,24	1,88
20-43	**26186,60**	2,16	24419,46	8,62
30-59	**60814,48**	16,51	53228,78	15,14
40-88	**108400,57**	36,72	97031,65	36,82
50-118	**171731,77**	75,69	154270,40	79,08
100-198	**641158,10**	1012,48	492506,63	719,59

Tabela 6: Resultados dos testes estatístico U-Teste

Instância	*p-valor*
10-16	0.02607
20-40	< 2.2e-16
30-78	3.012e-11
40-75	2.72e-11
50-107	2.999e-11
100-224	3.018e-11
10-20	**0.8644**
20-43	< 2.2e-16
30-59	3.018e-11
40-88	3.016e-11
50-118	3.018e-11
100-198	< 2.2e-16

A Tabela 3 mostra que o *solver* atingiu o tempo limite para as instâncias de tamanho 20. Nestes casos, não se tem garantia que a solução ótima foi encontrada. O limite superior para as soluções das instâncias 20-40 e 20-43 era 31097,40 e 31091,40, respectivamente. As soluções ótimas para as instâncias de tamanho 10 foram encontradas.

Analisando e comparando os resultados, observa-se que em todas as instâncias as meta-heurísticas encontram melhores soluções que a heurística Naíve. As

meta-heurísticas utilizam maior tempo computacional que a heurística, mas conclusivamente encontram melhores soluções. O *solver* encontrou melhores soluções que a Naíve, o ACO e o GRASP, nas instâncias de tamanho 10 (10-16 e 10-20) e nas instâncias de tamanho 20 (20-40 e 20-43).

O GRASP obteve melhor solução que o ACO em apenas 2 instâncias (10-16 e 10-20), ambas de tamanho 10. O ACO encontra melhores soluções em uma maior quantidade de instâncias, o que significa que o ACO possui uma maior eficiência. Em relação ao tempo de processamento, o ACO obteve menor tempo em mais da metade das instâncias. O GRASP consumiu mais tempo nas instâncias 10-16, 20-40, 30-78, 10-20, 20-43, 40-88 e 50-118.

O U-Teste apresentou que em apenas 1 das 12 instâncias não há diferença significativa nos valores das soluções. Nas instâncias (10-16 e 10-20) em que o GRASP obteve melhor solução, apenas na instância 10-16 há diferença significativa e, nas 10 instâncias em que o ACO obteve melhor solução há diferença significativa. Também confirmou conclusões anteriormente citadas.

8 CONCLUSÕES

Este artigo apresentou o Problema do Caixeiro Viajante com Múltiplos Passageiros Bônus Optativos Quota e Tempo – PCV-MPBOQT. Uma formulação matemática foi proposta, a qual forneceu as primeiras soluções ótimas do problema. Foram apresentadas para o problema uma heurística Naíve, uma meta-heurística ACO e uma meta-heurística GRASP. Também foi proposto um conjunto de 12 instâncias com até 100 vértices e, relatados resultados e analises de um experimento computacional.

A partir dos resultados do experimento computacional podemos concluir que o *solver* possui dificuldades em encontrar solução ótima para o problema (com exceção das instâncias 10-16 e 10-20 propostas). A Naíve encontrou as piores soluções e consumiu o menor tempo computacional. Enquanto o ACO encontrou melhores resultados no que se refere à qualidade das soluções encontradas pela heurística e pelas meta-heurísticas e, comparado ao GRASP consumiu menor tempo computacional em um maior número de instância.

Como atividades futuras será realizado o desenvolvimento de novas instâncias para o problema com características ainda não investigadas, e soluções a partir dessas instâncias. Também serão desenvolvidos e analisados: novos algoritmos heurísticos que realizem analises na seleção de localidades e ajustes nessa seleção ainda não abordadas;

e novos algoritmos meta-heurísticos baseados na evolução endossimbiótica intracelular mutualista.

REFERENCIAS

Calheiros, Z. S. A. (2017). *O Problema do Caixeiro Viajante Com Passageiros.* Dissertação de mestrado, Pós-graduação em Sistemas e Computação, Universidade Federal do Rio Grande do Norte.

Bastos, R. E. M. (2017). *O Problema do Caixeiro Viajante com Passageiros e Lotação.* Dissertação de mestrado, Pós-graduação em Sistemas e Computação, Universidade Federal do Rio Grande do Norte.

Carvalho, A. V. (2018). *O Problema do Caixeiro Viajante com Múltiplos Passageiros e Quota*. Dissertação de mestrado, Pós-graduação em Sistemas e Computação, Universidade Federal do Rio Grande do Norte.

Petch, V. A. (2018). *Problema do Passeio Lucrativo com Passageiros e Restrições de Tempo-PPL-RT*. Dissertação de mestrado, Pós-graduação em Sistemas e Computação, Universidade Federal do Rio Grande do Norte.

Awerbuch, B., Azar, Y., Blum, A. & Vempala, S. (1998). New approximation guarantees for minimum-weight k-trees and prize-collecting salesman. *SIAM Journal on Computing* 28(1), 254– 262.

Miller, C. E., Tucker, A. W. & Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM* 7(4), 326–329.

Helsgaun, K. (2000). An effective implementation of the Lin–Kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*(1), 106-130.

Dorigo, M. & Di Caro, G. (1999). Ant colony optimization: a new meta-heuristic. In *IEEE CEC99 Congress on Evolutionary Computation* 2, 1470–1477.

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

Feo, T. A., & Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*(2), 67-71.

R Core Team (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Mann, H. B., & Whitney, D. R. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, 50-60.

Conover, W. J. (1999). Practical nonparametric statistics. John Wiley & Sons.