Universidade Federal de Juiz de Fora
Departamento de Ciência da Computação
DCC059 - Teoria dos Grafos Semestre 2016-1

Problema do Roteamento de Veículos com Janela de Tempo (PRVJT)

Mateus Coutinho Marim

Professor: Stênio Sã Rosário F. Soares

Relatório do trabalho final de Teoria dos Grafos, parte integrante da avaliação da disciplina.

Juiz de Fora

Agosto de 2016

Sumário

1 Introdução

O problema de roteamento de veículos (PRV) é um dos mais estudados problemas na área da otimização combinatória. Consiste no atendimento de um conjunto de consumidores por intermédio de uma frota de veículos, que partem de um ou mais pontos denominados depósitos. A restrição presente no PRV é que cada veículo v possui uma capacidade C_v e o somatório de todas as demandas dos consumidores atendidos por um veículo v não pode ultrapassar C_v.

O PRV, apesar do seu enunciado relativamente simples, apresenta elevada complexidade computacional, pelo que é interessante como problema no teste de diversas heurísticas.

O PRV com janela de tempo (PRVJT) adiciona como restrição ao PRV que os clientes devem ser atendidos pelos veículos dentro de uma dada janela de tempo, tornando o problema ainda mais combinatório e restrito, necessitando a criação de uma heurística mais sofisticada para gerar soluções para o problema.

1.1 O problema e formulação

O Problema de Roteamento de Veículos (PRV) é um problema combinatório cujo conjunto base são as arestas de um grafo $G = (V, E)$. A notação utilizada por esse problema é a seguinte:

- $V = \{ v_0, v_1, \ldots v_n \}$ é um conjunto de vértices, onde:
  - Considere o deposito sendo localizado em v_0
  - Seja $V' = V \setminus \{ v_0 \}$ um conjuntode n cidades.
- $A = \{ (v_i, v_j) \in V; i \neq j \}$ é um conjunto de arestas.
- C é uma matrix de custos ou distâncias c_{ij} não negativas entre os clientes v_i e v_j.
- D o vetor de demandas dos clientes.
- R_i é a rota para o veículo i.
- m é o número de veículos (todos idênticos). Uma rota é designada para cada veículo.

Quando $c_{ij} = c_{ji}$ para todo $(v_i, v_j) \in A$ o problema é dito simétrico e é comum substituir A pelo conjunto de arestas $E = \{ (v_i, v_j) \mid v_i, v_j \in V; i < j \}$

Com cada vértice v_i em V' associado a uma quantidade q_i de bens a serem entregues por um veículo. Assim o PRV consiste de determinar o conjuto de m rotas para os veículos

de forma a minimizar o custo total, começando e terminando em um depósito, tal que cada vértice em V' é visitado exatamente uma vez por um veículo.

Para melhor computação, pode ser definido $b(V) = \lceil (\sum_{v_i \in V} d_i)/C \rceil$ como limite inferior óbvio para o número de veículos necessários para atender os clientes no conjunto V.

Nós vamos considerar um tempo de serviço δ_i (tempo necessário para descarregar todos bens), necessário para um veículo descarregar a quantidade q_i em v_i. É necessário que a duração de qualquer rota de veículo (viagem mais tempos de serviço) não ultrapassem um limite dado, então, nesse contexto o custo c_{ij} é o tempo de viajem entre as cidades. O problema definido acima é $NP - hard$.

O PRV com janela de tempo (PRVJT) é o mesmo problema que o PRV com a restrição adicional que no PRVJT uma janela de tempo é associada a cada cliente $v \in V$, definindo um intervalo $[e_0, l_0]$ onde os clientes tem que ser fornecidos. O intervalo $[e_0, l_0]$ no depósito é chamdo de horizonte de agendamento. Aqui está uma descrição formal do problema:

- **Objetivo:** O objetivo é minimizar a frota de veículos e a soma do tempo de viagem e tempo de espera necessário para fornecer todos clientes em seus horários, neste trabalho o objetivo será apenas minimizar a distância entre os clientes na rota.

- **Viabilidade:** O PRVJT é, em relação ao PRV, caracterizado pela seguintes restrições adicionais:

  - Uma solução se torna inviável se um cliente é fornecido depois do limite superior de sua janela de tempo.

  - Um veículo chegando antes do limite inferior da janela de tempo causa um tempo de espera adicional na rota.

  - Cada rota deve começar e terminar na janela de tempo associada ao depósito.

  - No caso de janelas de tempo curtas, o término tardio do serviço não afeta a viabilidade da solução, mas é penalizado pela adição de um valor à função objetivo.

- **Formulação:** Deixe b_0 denotar o início do serviço em um cliente v. Agora para a rota $R_i = (v_0, v_1, \dots v_m, v_{m+1})$ ser viável ela deve manter $e_{v_i} < b_{v_i} < l_{v_i}$, $1 \leq i \leq m$, e $b_{v_i} + \delta_{v_i} + c_{v_i}, 0 \leq l_0$. Dado que um veículo viaja para o próximo cliente assim que seu serviço no cliente atual está finalizado, b_{v_i} pode ser recursivamente computado como $b_{v_i} = max\{\, e_{v_i}, b_{v_{i-1}} + \delta_{v_{i-1}} + c_{v_{i-1},v_i} \,\}$ com $b_0 = e_0$ e $\delta_0 = 0$. Assim, um tempo de espera $w_{v_i} = max\{\, 0, b_{v_i} - b_{v_{i-1}} - \delta_{v_{i-1}} - c_{v_{i-1},i} \,\}$ pode ser induzido no cliente v_i. Neste trabalho a função objetivo considerada será apenas a minimização da soma das distâncias entre os clientes na rota então o custo da rota i é agora dado por $C_{PRVJT}(R_i) = \sum_{i=0}^{m} c_{i,i+1}$. Para uma solução S com rotas $R_1, \dots R_m$, o custo

de S é dado por $F_{PRVJT}(S) = \sum_{i=1}^{m}(C_{PRVJT}(R_i)$. S é dito viável se todas rotas pertencendo a S são viáveis e seus clientes são fornecidos por exatamente uma rota. Como descrito por Solomon, nós assumimos que inicialmente todos veículos deixam o depósito no tempo mais cedo possível e_0. [4]

2 Metodologia utilizada

A solução do problema de roteamento de veículos é representada como um conjunto de listas de visitas aos clientes, cada lista correspondendo a uma única rota de veículo. O número máximo de listas corresponde ao número de veículos disponíveis. Como o conjunto de veículos é homogêneo a capacidade de carga q de todos é igual e sempre maior ou igual a demanda de cada cliente e nas instâncias que iremos tratar o tempo δ para fornecer cada cliente também é constante.

- **Abordagem baseada no algoritmo de construção de rotas em paralelo:** Potvin e Rousseau (1993) desenvolveu uma versão paralela da heurística de inserção de Solomon. Neste algoritmo os clientes candidatos à solução são atribuídas em paralelo para suas respectivas rotas. Em geral, a inserção de rotas em paralelo resulta em soluções de maior qualidade.

- **Custo da viagem:** O custo para sair do cliente v_i e chegar no v_j vai ser dado pela distância euclidiana dos pontos correspondentes às suas coordenadas no $\mathbb{R}^2$.

$$c(v_i, v_j) = \sqrt[2]{(v_j.x - v_i.x)^2 + (v_j.y - v_i.y)^2}$$

- **Atualização do tempo de cada rota:** A atualização do tempo da rota $r \in R$ vai ser dada pelo custo do veículo chegar do cliente v_{i-1} no cliente v_i recém adicionado na rota, o tempo de espera para poder atender o cliente dentro da janela de tempo e o tempo necessário para atender o cliente.

$$t(r) = c_{i-1,i} + w_i + \delta$$

- **Heurística de economia:** A heurística de Clarke e Wright (1964) surge, no campo da logística, como factor de simplicidade e flexibilidade na formulação da programação de rotas, no âmbito da gestão de transporte. Existem vários estudos que demonstram diferentes formas possíveis de programação de rotas. Para a sua implementação, são necessários conhecimentos matemáticos bastante complexos, restringindo deste modo, a sua utilização em algumas empresas (Carvalho, 2002, p. 206).

Esta heurística baseia-se na troca de conjuntos de rotas em cada ponto de chegada, caso seja possível ou desejável, por forma a melhorar o desempenho global. Por conseguinte, uma das designações atribuída à formulação é a heurística de poupança de Clarke e Wright (, 2002, p. 207).

A formulação supracitada demonstra grande utilidade para frotas homogéneas, uma vez que admite a minimização da frota e da distância percorrida. No entanto, a heurística não possui a capacidade de manipulação de dados referentes a frotas heterogéneas, por não considerar os custos fixos e directos associados à variabilidade dos veículos e das distâncias percorridas (Teixeira, 2002, p. 4).

Primeiramente, são definidas as restrições básicas do problema, como por exemplo: por cada rota apenas é atendido um cliente. Com base nestas restrições, deve-se garantir a menor distância possível no atendimento de todos os clientes. Por conseguinte, elabora-se uma lista com as rotas (de ida e volta) desde o armazém até cada dois clientes, e calculam-se os custos inerentes às respectivas deslocações S_{ij}. [5]

$$S_{ij} = c_{0i} + c_{0j} - c_{ij}$$

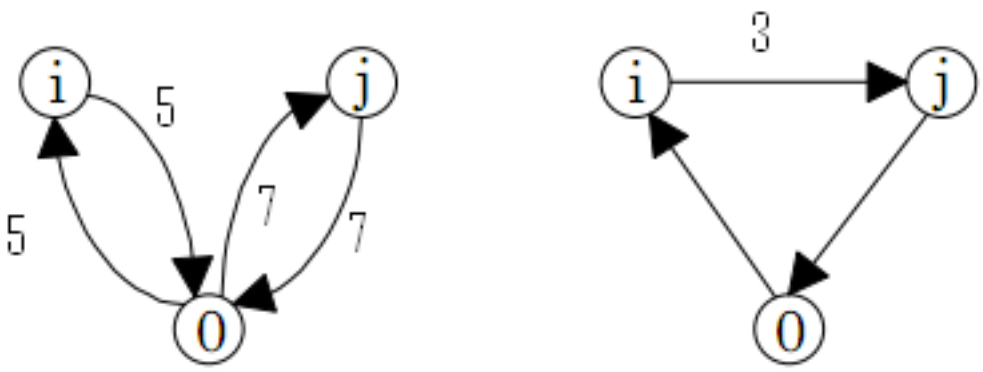

Figura 1: Representação da heurística de Clark & Wright (1964)

2.1 Estruturas de dados utilizadas

Como o grafo do PRVJT é um grafo completo não é necessário representar as arestas na memória, pois é possível calcular o custo c_{ij} da ida de um vértice v_i para um v_j através da distância euclidiana. Logo a estrutura que representa o grafo é composto por uma lista V que representa os vértices (clientes e depósito) e um vetor de coordenadas (x, y) para cada $v \in V$. Os clientes foram representados por uma estrutura com membros para

armazenar os limites $[e_0, l_0]$ da janela de tempo, o id do cliente, a demanda d_i e o tempo de serviço δ. A solução S é representada pelo valor da função objetivo, pelo α e pela semente utilizados para encontrar a solução.

2.2 Abordagens algoritmicas usadas na solução

Nesta seção vai ser apresentada uma abordagem para solução do PRVJT baseada no Greedy Random Reactive Algorithm (GRRA) e usando uma abordagem da construção de rotas em paralelo. Para aplicar o GRA no PRVJT, é utilizado o GRRA para construir uma solução na fase de construção e que é atualizada por soluções melhores durante as iterações do algoritmo.

- **Dedução do número inicial de rotas:** Neste trabalho é utilizada uma simples heurística para computar o número inicial de rotas. Dada uma demanda determinística para cada cliente d_i e a capacidade do veículo C, nós podemos calcular o número de rotas iniciais aplicando a equação abaixo.

$$b(V) = \lceil (\sum_{v_i \in V} d_i)/C \rceil$$

Algorithm 1: *deduzirVeiculos*

Data: $V \in G(V, E)$ - Conjunto de clientes a serem atendidos;
C - Capacidade dos veículos.
Result: b - Número inicial de rotas

$sum = 0$;
for $v_i \in V$ **do**
  $sum \leftarrow sum + d(v_i)$
return $\lceil sum/C \rceil$

- **Inicialização das rotas:** Uma vez determinado o número inicial de rotas é associado um ângulo θ_i para cada rota $R_i \in S$ definido por $\theta_{R_i} = \theta_{R_{i-1}} + \theta$ sendo $\theta_{R_0} = 0^o$ e $\theta = \lfloor 360/b \rfloor$. Após definidos os ângulos para cada rota são definidos os clientes iniciais para cada rota como sendo os que estiverem mais próximos e com o ângulo em relação ao depósito entre θ_{R_i} e $\theta_{R_{i+1}}$. O ângulo θ_{i0} do cliente v_i em relação ao depósito v_0 é

$$\theta_{i0} = arccos\left(\frac{< coord(v_i), coord(v_0) >}{||coord(v_i)|| \cdot ||coord(v_0)||}\right) \cdot 360$$

Algorithm 2: *inicializarRotas*

Data: $V \in G(V, E)$ - Conjunto de clientes a serem atendidos;
b - Número de veículos;
T - Lista dos tempos de cada rota;
Result: R - Conjunto de rotas iniciais

$R \leftarrow$ Lista com b novas rotas $r \in R : |r| = \varnothing$;
$\theta \leftarrow \lfloor 360/b \rfloor$;
$\theta_{R_0} \leftarrow 0^o$;

.

for $R_i \in R \backslash R_0$ **do**
    $\theta_{R_i} \leftarrow \theta_{R_{i-1}} + \theta$;
for $v_i \in V$ **do**
    $pos \leftarrow \theta_{v_i v_0}/\theta$;
    if $c_{i0} \leq c_{R_{pos}0}$ **then**
        $R_{pos} \leftarrow R_{pos} \cup \{ v_i \}$;
        $T_{pos} \leftarrow c_{i-1,i} + w_{i-1,i} + \delta_{i-1,i}$;

return R

- **Atualização dos candidatos a solução:** O algoritmo *atualizarCandidatos* recebe como parâmetro a lista de clientes V e retorna a lista de listas de candidatos para todo $R_i \in S$ rankeando os candidatos pela heurística de Clark e Wright (1964) descrita na seção anterior.

Algorithm 3: *atualizarCandidatos*

Data: $V \in G(V, E)$ - Conjunto de clientes a serem atendidos;
S - Solução até o momento;
T - Lista de tempos;
visited - Lista de visitas;
Result: L - Lista de candidatos paras rotas da solução

$L \leftarrow$ Lista de tamanho $|S|$ de listas de clientes;

for $r_i \in S$ **do**
    for $v_j \in V$ **do**
        if v_j *não foi visitado e diferente do ultimo adicionado em* r_i **then**
            $v_k \leftarrow$Ultimo cliente adicionado a rota;
            if $T_{r_i} + c_{v_k} \leq l_{v_j}$ **then**
                $L_i \leftarrow L_i \cup \{ v_j \}$;

for *each* $r \in L$ **do**
    Calcula economia na inserção para cada $v \in r$;
    Ordena r em ordem não crescente de economia;

if *se todas listas de candidatos* $l \in L$ *estão vazias* **then**
    $r \leftarrow \varnothing$;
    $u \leftarrow$cliente mais proximo do depósito;
    $r \leftarrow r \cup \{ u \}$;
    $L \leftarrow L \cup \{ r \}$;
    $T_r \leftarrow c_{0u}$;
    $visited_u \leftarrow$ *true*;
return L

- **Seleção de candidatos:** O algoritmo *selecionarCandidatos* seleciona o candidato para a rota $r_i \in S$ rankeado na posição $rand_i$.

Algorithm 4: *selecionarCandidatos*

Data: S - Solução até o momento;
L - Lista das listas de candidatos das rotas;
T - Lista com os tempos de cada rota;
rand - Posições dos candidatos selecionados;
visited - Lista de visitados;
Result: S - Solução atualizada

for $r_i \in S$ **do**
  if $L_i \neq \varnothing$ **then**
    $v \leftarrow r_{i,i-1}$;
    $u \leftarrow L_{i,rand_i}$;
    $S_i \leftarrow S_i \cup \{ u \}$;
    $T_i \leftarrow c_{v,u} + w_{v,u} + \delta_{v,u}$;
    $visited_u \leftarrow true$;
return S

- **Algoritmo construtivo guloso randomizado (GRA):** O *algoritmoConstrutivoGuloso* no pseudo-código 5 tem como parâmetro, uma lista V de clientes a serem atendidos pelos veículos nas rotas, uma restrição da lista de candidatos α ($0 \leq \alpha \leq 1$), e uma semente *seed* para o gerador de números pseudo aleatórios.

Algorithm 5: *algoritmoConstrutivoGuloso*

Data: V - Lista de clientes;
C - Capacidade dos veículos;
α - Restrição da lista de candidatos;
seed - Semente para o gerador de números aleatórios;
Result: S - Solução construtiva

Inicializar gerador pseudo aleatório com *seed*;
$b \leftarrow deduzirVeiculos(A, C)$;
times $\leftarrow$ Lista de tamanho b para os tempos das rotas;
$S \leftarrow inicializarRotas(A, b, times)$;
visited $\leftarrow$ Lista de booleanos de tamanho $|A|$ inicialmente como false;
candidatos $\leftarrow$Lista com b listas vazias;

while *todos clientes não foram visitados* **do**
    rand $\leftarrow$ Lista de tamanho $|candidatos|$ para as posições aleatórias para os candidatos de cada rota;
    $candidatos \leftarrow atualizarCandidatos(A, S, times, visited)$;

    for *i = 1 to |candidatos|* **do**
        $rand_i \leftarrow random(seed, \alpha)$;
        //Posição aleatória até tamanho da lista de candidatos da rota R_i

    $S \leftarrow selecionarCandidatos(candidatos, S, rand, times)$;
    if *capacidade do veiculo foi atingida* **then**
        $S \leftarrow$Nova rota com cliente mais proximo do depósito;
        Atualiza tempo e seta cliente adicionado como visitado;

$S.funcValue \leftarrow computeFunctionValue(S)$;
$S.alpha \leftarrow \alpha$;
$S.seed \leftarrow seed$;
return S

- **Algoritmo construtivo guloso randomizado reativo GRRA:** O parâmetro de restrição da lista de candidatos α é basicamente o único parâmetro a ser definido em uma implementação prática de um GRA. Feo e Resende discutiram o efeito da escolha do valor de α em termos da qualidade e diversidade da solução e como impacta a saída de um procedimento GRASP (Neste trabalho iremos ver apenas com o GRA Reativo).

  Ao invés de usar um valor fixo para o parâmetro α que determina quais elementos vão ser colocados na lista de candidatos restrita em cada iteração da fase de construção,

é proposto selecioná-lo aleatóriamente a partir do conjunto discreto $\{\alpha_1, \dots \alpha_n\}$ contendo m valores aceitáveis pré-determinados.

Usar diferentes valores de alpha em cada iteração permite a construção de diferentes listas de candidatos restritas, eventualmente levando para construção de diferentes soluções ao qual nunca poderiam ser construídas se fosse utilizado um único valor de α fixo. Por exemplo, considere $\alpha_1 = 0.1, \alpha_2 = 0.2, \dots$ e $\alpha_{10} = 1$. Seja p_i a probabilidade associada com a escolha de αi, para $i = 1, \dots m$, correspondendo à uma distribuição uniforme.

É proposto atualizar periódicamente a distribuição de probabilidades p_i, $i = 1, \dots m$, usando informações coletadas durante a busca. Diferentes estratégias para essa operação de atualização podem ser exploradas.

É descrito abaixo uma regra de qualificação absoluta, baseada no valor médio das soluções obtidas com cada valor de α, que é apenas uma entre muitas estratégias. Lembre que em cada iteração algum valor de $\alpha = \alpha_i$ é aleatóriamente selecionado de V usando a distribuição de probabilidades $p_i = 1, \dots m$. Em cada iteração do GRRA, seja $F(S^*)$ ser o valor da melhor solução já encontrada. Além disso, deixe A_i ser o valor médio das soluções obtidos com $\alpha = \alpha_i$ na fase de construção. A distribuição de probabilidades vai ser atualizada após a execução de cada bloco de *blockIter* iterations (foi usado $blockIter = 100$ na implementação). Para isso, compute

$$q_i = \left(\frac{F(S^*)}{A_i} \right)^{\delta}$$

para todo $i = 1, \dots m$, e obtenha pela normalização dos q_i os novos valores de probalidades $p_i, i = 1, \dots m$ dados por

$$p_i = \frac{q_i}{\left(\sum_{j=1}^{j=m} q_j \right)}$$

É possível notar que o valor mais adequado $\alpha = \alpha_i$ se revela, quanto maior o valor associado de q_i, e, consequentemente, maior é o valor atualizado de p_i. Por conseguinte, no próximo bloco de *blockIter* iterações, os valores de α que levam a melhores soluções terão maiores probabilidades e, consequentemente, vão ser mais frequentemente utilizados na fase de construção do procedimento GRRA. O expoente δ pode ser usado e explorado para atenuar diferentemente os valores atualizados das probabilidades p_i. A abordagem descrita abaixo é chamada de GRA Reativo porque o valor do parâmetro α não é fixo mas, ao invés disso, é alto-ajustável. [2]

Algorithm 6: *algoritmoConstrutivoGulosoReativo*

Data: *maxIter* - Máximo de iterações do algoritmo;
blockIter - Tamanho do bloco de iterações para atualizar vetor de probabilidades;
δ - Expoente para normalização;
gran - Granularidade para gerar os alphas;
seed - Semente para o gerador de números aleatórios;
Result: S^* - Melhor solução encontrada

Inicializar gerador pseudo aleatório com *seed*;
$A \leftarrow$ Lista de alphas de acordo com a granularidade;
$P \leftarrow$ Lista de probabilidades com distribuição uniforme de tamanho $|A|$;
$Q \leftarrow$ Lista de tamanho $|A|$ inicializada com 0.0;
$sA \leftarrow$ Lista de somatórios para os alphas inicializados em 0.0;
$nA \leftarrow$ Lista com frequência de utilização dos alphas inicializado com 0;
$S^* = \varnothing$;
$S^*.funcVal = \infty$;

for *itr = 0 to maxIter* **do**
    $\alpha \leftarrow chooseAlpha(A)$;
    $nA_\alpha \leftarrow nA_\alpha + 1$;
    $S \leftarrow algoritmoConstrutivoGuloso(\alpha, seed)$;
    if $S.funcVal < S^*.funcVal$ **then**
        $sA_\alpha \leftarrow sA_\alpha + S.funcVal$;
        $S^* \leftarrow S$;
    if *itr mod blockIter == 0* **then**
        $sum \leftarrow 0.0$;
        for $q_i \in Q$ **do**
            $q_i \leftarrow (S^*.funcVal/(sA_i/nA_i))^\delta$;
            $sum \leftarrow sum + q_i$;
        for $p_i \in P$ **do**
            $p_i = q_i/sum$;

return S^*

3 Experimentos computacionais

Nos experimentos computacionais, os problemas de teste utilizados no trabalho são os benchmarks do Solomon, que são PRVJT com um único depósito e frota homogênea de veículos. O objetivo é minimizar a distância total percorrida pelos veículos nas rotas, cada veículo tem uma capacidadeque não deve ser excedida, e cada cliente deve ser visitado dentro de uma janela de tempo em particular.

Os problemas do Solomon são de 25,50 e 100 clientes com uma dada restrição de capacidade nos veículos, e janela de tempo na hora da entrega nos clientes. Esses problemas são separados em 6 classes: C1, C2, R1, R2, RC1 e RC2. Cada classe contém entre 8 e 12 instâncias de problemas individuais. O C1 e C2 tem os clientes localizados em clusters. Nas classes R1 e R2 os clientes estão em posições aleatórias. As classes RC1 e RC2 contém uma mistura de clientes em posições aleatórias e em clusters. Os problemas C1, R1 e RC1 tem um horizonte de agendamento mais curto, e requerem de 9 à 19 veículos. As classes C2, R2 e RC2 tem um horizonte de agendamento maior e requerem menos veículos (2-4). Tanto a distância como o tempo de viagem são dados pela distância euclidiana entre pontos.[3]

Os testes focam na variação dos parâmetros θ_1, θ_2 *e* θ_3, da fase de construção. No experimento 1 cada instância é executada uma vez com o algoritmo construtivo guloso sem o fator de aleatoriedade. São feitas 500 iterações em cada instância em 30 execuções com sementes diferentes para cada θ para o experimento 2. E no experimento 3 são feitas 30 execuções do GRRA com 1500 iterações cada, bloco de iterações de tamanho 100 e expoente de normalização δ igual a 10 e granularidade de 0.1, logo o conjunto de alphas será $A = \{\, 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1 \,\}$.

Todos experimentos foram executados em um computador Intel Core i5-4200M 2.5Ghz de 4 núcleos e com 4GB de memória RAM e sistema operacional Ubuntu 16.04.1 e arquitetura 64bits.

Nas tabelas as melhores soluções encontradas estão em negrito e as soluções ótimas das instâncias estão na primeira tabela.

Tabela 1: Soluções ótimas S^* (Solomon's Benchmark)

Instância	Function Value*	Number of Vehicles*
C101_25	191.3	3
C101_50	362.4	5
C101_100	827.3	10
C208_25	214.5	2
C208_50	350.5	2
C208_100	585.8	3
R201_25	463.3	4
R201_50	791.9	6
R201_100	1143.2	8
RC102_25	351.8	3
RC102_50	822.5	7
RC102_100	1457.4	14

Tabela 2: Experimento 1 ($\alpha = 0$)

Instância	Function Value	Number of Vehicles	Seed	Time
C101_25	529.822	9	0.000838	529.822
C101_50	1121.34	14	0.002809	1121.34
C101_100	2910.34	24	0.025368	2910.34
C208_25	240.392	2	0.000302	240.392
C208_50	646.045	5	0.001263	646.045
C208_100	1413.18	7	0.006544	1413.18
R201_25	679.395	5	0.000664	679.395
R201_50	1386.74	8	0.001798	1386.74
R201_100	2283.74	10	0.007508	2283.74
RC102_25	716.746	7	0.000533	716.746
RC102_50	1929.22	15	0.002717	1929.22
RC102_100	2647.95	23	0.014502	2647.95

Tabela 3: Experimento 2 ($\alpha = 0.1$)

Instância	Function Value	Number of Vehicles	Seed	Time
C101_25	481.273	7	8078	0.258662
C101_50	994.211	13	9047	1.41195
C101_100	2625.74	24	23844	8.50527
C208_25	**240.392**	**2**	**404**	0.136637
C208_50	646.045	5	3570	0.579558
C208_100	1351.98	7	6281	2.95206
R201_25	653.302	4	26482	0.190305
R201_50	1335.34	6	9795	0.909466
R201_100	2140.8	11	28685	4.2492
RC102_25	**584.041**	**7**	**11081**	0.252376
RC102_50	1538.41	13	31464	1.29792
RC102_100	**2656.12**	**23**	**13320**	8.44656

Tabela 4: Experimento 2 ($\alpha = 0.2$)

Instância	Function Value	Number of Vehicles	Seed	Time
C101_25	469.487	6	8078	0.247835
C101_50	994.153	13	5078	1.43289
C101_100	2684.26	24	23016	8.80874
C208_25	240.392	2	414	0.156324
C208_50	646.045	5	3570	0.575213
C208_100	1358.36	8	14096	3.35209
R201_25	**637.888**	**4**	**19641**	0.198958
R201_50	1335.34	6	6003	0.874592
R201_100	2140.8	11	7217	4.49449
RC102_25	584.041	7	30817	0.234859
RC102_50	1522.8	13	19467	1.23163
RC102_100	2627.87	21	3487	8.42519

Tabela 5: Experimento 2 ($\alpha = 0.3$)

Instância	Function Value	Number of Vehicles	Seed	Time
C101_25	442.936	6	8148	0.251021
C101_50	**951.179**	**14**	**16968**	1.3905
C101_100	**2639.65**	**25**	**12604**	9.03056
C208_25	240.392	2	494	0.149848
C208_50	**646.045**	**5**	**3690**	0.603782
C208_100	**1318.65**	**8**	**31311**	3.55091
R201_25	653.302	4	26432	0.220323
R201_50	**1317**	**7**	**8265**	0.907573
R201_100	**2087.24**	**11**	**8847**	4.60259
RC102_25	584.041	7	9118	0.245876
RC102_50	**1514.41**	**13**	**31534**	1.27907
RC102_100	2586.28	25	4144	8.64595

Tabela 6: Experimento 3

Instância	Function Value	Number of Vehicles	Alpha	Seed	Time
C101_25	**437.3**	**6**	**0.4**	**27218**	0.651308
C101_50	994.153	13	0.2	26457	4.0879
C101_100	2676	24	0.1	20247	240.14
C208_25	240.392	2	0	13560	0.504238
C208_50	646.045	5	0	18997	1.79428
C208_100	1413.18	7	0	26297	10.6729
R201_25	653.302	4	0.3	2183	0.618208
R201_50	1317	7	0.3	1933	2.59106
R201_100	2183.19	11	0.1	17417	11.3529
RC102_25	584.041	7	0.1	1887	0.680886
RC102_50	1522.93	13	0.3	31899	3.7333
RC102_100	2647.95	23	0	11989	25.1488

4 Conclusões

Neste trabalho, foi apresentado um algoritmo construtivo aleatorizado reativo utilizando uma heurística de economia e uma abordagem paralela com o objetivo de minimizar o somatório das distâncias dos clientes das rotas. Pela análise dos experimentos foi possível observar que a escolha do parâmetro α é de extrema importância para se encontrar uma boa solução junto também com o uso de sementes diferentes para o algoritmo randômico, pois pode ajudar ao algoritmo a sair de um mínimo local. Também é possível observar que os valores de α não passam de 0.5 para as melhores soluções o que pode indicar que no intervalo de $\{ 0, \ldots 0.5 \}$ existe maior probabilidade de se conseguir melhores resultados. O algoritmo proposto neste trabalho consegue resultados mais próximos do ótimo com instâncias com janela de tempo maior como nas C2 e R2 em que os resultados se afastam da solução ótima em média para um valor 76% maior que o ótimo, enquanto que, as instâncias C1 e RC1 variam em média para um valor 130% maior.

Referências

[1] José Mexia Crespo de Carvalho. [*Logística. 3ª ed*]. 2002.

[2] Marcelo Prais e Celso C. Ribeiro. *Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment* [*INFORMS Jurnal on Computing*]. 174-176, 1998.

[3] Marius M. Solomon website,
`http://w.cba.neu.edu/ msolomon/`

[4] The VRP Web,
`http://www.bernabe.dorronsoro.es/vrp/`

[5] Wikipedia - Heurística de Clarke e Wright,
`https://pt.wikipedia.org/wiki/Heurística_de_Clarke_e_Wright/`