Sistema para roteamento de veículos capacitados aplicando Métodos de Monte Carlo

Rômulo A. de Carvalho Oliveira[1], Karina V. Delgado[1], Daniel A. M. Moreira[1]

[1]Escola de Artes, Ciências e Humanidades – Universidade de São Paulo (USP) – São Paulo – SP – Brazil

{romulo.augusto.oliveira, daniel.augusto.moreira, kvd}@usp.br

Abstract. *The Vehicle Routing Problem (VRP) is one of the combinatorial optimization problems most studied in Computer Science and of great relevance to the areas of logistics and transport. This paper presents a new algorithm for solving the Capacitated Vehicle Routing Problem (CVRP). The proposed algorithm was developed based on Monte Carlo simulations and Clarke and Wright Savings heuristic and demonstrated results comparable to the best existing algorithms in the literature surpassing previous work with Monte Carlo methods when considering the general result. The comparison, analysis and evaluation of the algorithm were based on existing benchmarks in the literature.*

Resumo. *O Problema de Roteamento de Veículos (Vehicle Routing Problem, VRP) e um dos problemas de Otimização Combinatória mais estudados dentro da Computação e de grande relevância para as áreas de logística e transporte. Este trabalho apresenta um novo algoritmo para resolução do Problema de Roteamento de Veículos Capacitados (Capacitated Vehicle Routing Problem, CVRP). O algoritmo proposto foi desenvolvido baseado em Simulações de Monte Carlo e na heurística de Clarke & Wright Savings e demonstrou resultados comparáveis aos melhores algoritmos existentes na literatura, superando trabalhos anteriores com Métodos de Monte Carlo quando e considerado o resultado geral. A comparação, análise e avaliação do algoritmo foram feitas com base em benchmarks de problemas existentes na literatura.*

1. Introdução

O Problema de Roteamento de Veículos (*Vehicle Routing Problem*, VRP) é um problema de otimização combinatória que vem sendo estudado na área de Computação há mais de cinquenta anos [Dantzig and Ramser 1959]. É uma variação do Problema do Caixeiro Viajante [Gutin and Punnen 2002, Kanda 2014], portanto da classe NP-difícil [Lenstra and Rinnooy Kan 1981], e resume-se em atender um número determinado de clientes através de uma frota de veículos, utilizando a rota que ofereça o menor custo possível.

Além de ser um problema importante de otimização combinatória muito estudado em Computação [Toth and Vigo 2002], o roteamento de veículos é parte crítica e fundamental para execução bem-sucedida do processo de logística e transporte [Giaglis et al. 2004]. Em uma sociedade onde a eficiência energética e a redução de poluentes são preocupações globais, o roteamento de veículos feito de forma eficiente

pode reduzir custos, economizar energia e reduzir a emissão de gases poluentes [Christie et al. 2006].

Neste trabalho, será abordada a variação mais conhecida [Takes and Kosters 2010a] do VRP: O Problema de Roteamento de Veículos Capacitados (*Capacitated Vehicle Routing Problem*, CVRP). O CVRP consiste no Problema de Roteamento de Veículos com a restrição de que cada veículo possui uma capacidade máxima de carga. Métodos de Monte Carlo serão aplicados para resolver o CVRP. Métodos de Monte Carlo são métodos estatísticos que utilizam amostragem aleatória para resolver problemas probabilísticos e determinísticos [Bindel and Goodman 2009]. Essas técnicas são aplicadas quando não é viável empregar métodos exatos na resolução de problemas. O algoritmo proposto foi desenvolvido baseado em Simulações de Monte Carlo e na heurística de *Clarke & Wright Savings*.

A Seção 2 apresenta a definição formal de CVRP. A Seção 3 introduz Métodos de Monte Carlo. A Seção 4 apresenta os trabalhos relacionados. O algoritmo proposto neste trabalho é discutido na Seção 5. Na Seção 6 são apresentadas a metodologia e os experimentos realizados. A Seção 6.4 discute sobre os resultados obtidos e a Seção 8 conclui este trabalho.

2. Definição formal de CVRP

O Problema de Roteamento de Veículos Capacitados pode ser formalizado como um grafo não direcionado $G = (V, E)$, onde $V = \{v_0, v_1, \ldots, v_n | \, n \geq 1\}$ é o conjunto de vértices e $E = \{(v_i, v_j) : v_i, v_j \in V, v_i \neq v_j\}$ é o conjunto de arestas. Definimos m como o número de veículos a serem usados para servir todos os clientes, onde $1 \leq m \leq n$. Os veículos têm a mesma capacidade máxima Q. O vértice v_0 representa o depósito, de onde partem os veículos, e cada um dos demais vértices representa um cliente. Cada vértice $v_i \in V$ tem associada uma demanda q_i, onde $0 < q_i \leq Q$. Cada aresta $(v_i, v_j) \in E$ tem associada um custo não negativo c_{ij}, que representa o custo de se chegar do vértice v_i ao vértice v_j.

A solução consiste em encontrar um conjunto de rotas $R = \{r_1, r_2, \ldots, r_m\}$, de forma que todos os clientes sejam visitados exatamente uma vez, sem que as rotas excedam a capacidade máxima Q dos veículos e com o menor custo total $C(R)$ possível. Cada rota r_i é servida por exatamente um veículo, começa e termina no depósito, tem tamanho k_i e é definida por $r_i = (v_0^i, v_1^i, \ldots, v_{k_i}^i, v_{k_{i+1}}^i)$, onde $v_0^i = v_{k_{i+1}}^i = v_0$ e $v_j^i \neq v_l^j$ para $0 \leq j < l \leq k_i$. Cada rota r_i tem um custo $C(r_i) = \sum_{j=0}^{k_i} c_{v_j^i, v_{j+1}^i}$, que é o custo de percorrer a rota completamente (partindo do depósito, passando por todos os clientes e retornado ao depósito). O custo total da solução $C(R)$ é a soma do custo das rotas, definido como $C(R) = \sum_{i=1}^{m} C(r_i)$.

O CVRP consiste em um problema de minimização e a melhor solução é aquela que apresenta o menor custo $C(R)$.

3. Métodos de Monte Carlo

Métodos de Monte Carlo são uma classe de métodos que utilizam amostragem de números aleatórios para se obter resultados numéricos, isto é, utilizam métodos probabilísticos para resolver problemas probabilísticos e determinísticos [Bindel and Goodman 2009]. Definido em 1949 por Nicholas Metropolis e Stanislaw Ulam, foi desenvolvido por John Von Neumann e Stanislaw Ulam [Metropolis and Ulam 1949].

Simulação de Monte Carlo (*Monte Carlo Simulation*, MCS) é uma das técnicas usadas em análise probabilística para se entender distribuições e extrair seu comportamento de amostras aleatórias. Essas amostras são geradas a fim de permitir que o comportamento real possa ser simulado, estudado e entendido sem que novas amostras populacionais precisem ser colhidas [Mooney 1997]. O Método de Monte Carlo é muito utilizado, dentre outras coisas, para integrações em muitas dimensões, resolver sistemas lineares densos e simular modelos físicos e matemáticos complexos [Gentle 2003].

Amostragem aleatória, em inglês *random sampling*, é o processo estatístico de seleção de uma amostra populacional. O tipo de amostragem aleatória abordado neste trabalho é a amostragem aleatória simples (*plain random sampling* ou *simple random sampling*) [Bussab and Morettin 2004]. Amostragem aleatória simples é um tipo de amostragem probabilística onde os elementos da população são sorteados aleatoriamente. Todos os elementos têm a mesma probabilidade de serem selecionados. O sorteio é feito através de números aleatórios gerados por computadores ou por numeração e sorteio usando tabelas de números aleatórios [Bussab and Morettin 2004].

Em problemas de busca, *plain random sampling* pode ser usado como um método que começa no nó raiz e continuamente escolhe nós filhos aleatórios até que um nó folha seja alcançado. A cada nível da árvore, um nó é sorteado e escolhido para continuar com a busca. Esse método pode ser aplicado a fim de maximizar a diversidade das soluções encontradas quando o espaço de busca é muito grande para usar a busca em profundidade [Takes and Kosters 2010b].

Plain random sampling pode ser aplicado em Simulações de Monte Carlo para problemas de busca [Takes and Kosters 2010b]. Primeiro é selecionado o nó raiz na árvore de busca. Para cada um dos n nós filhos candidatos, são executadas r simulações aleatórias até que um nó folha seja alcançado. A simulação é feita utilizando o método *plain random sampling*, explicado anteriormente. Após esse processo, o melhor nó filho candidato é escolhido entre os nós existentes para fazer parte da solução. O melhor nó pode ser o que obteve melhor solução entre todas as simulações ou o que obteve a melhor média entre as simulações [Takes and Kosters 2010b]. A partir daí, esse nó filho passa ser o nó atual da árvore de busca e o método continua até que o nó atual seja uma folha. A vantagem da Simulação de Monte Carlo é que a busca é guiada de acordo com o melhor resultado obtido dentre as simulações. Esse processo pode ser repetido várias vezes para se obter um conjunto de soluções ou a melhor solução entre todas as iterações.

4. Algoritmos de CVRP

Há duas abordagens principais usadas para resolver o problema de roteamento de veículos: métodos exatos e métodos heurísticos. Os métodos exatos são soluções que garantem que a solução ótima será encontrada, mas que pode ser custoso computacionalmente devido à complexidade NP-difícil [Leeuewn 1990]. Entre os algoritmos exatos destacam-se *branch and bound* [Laporte 1992, Fischetti et al. 1994] e *branch and cut* [Lysgaard et al. 2004].

Métodos heurísticos são métodos que percorrem um caminho reduzido no espaço de busca seguindo uma estratégia guiada e não garantem que a solução ótima será encontrada, mas permitem encontrar uma aproximação com menor custo de tempo e recursos [Pearl 1984]. Os métodos heurísticos são divididos em duas classes principais: métodos heurísticos clássicos e meta-heurísticos [Toth and Vigo 2002].

Algoritmos heurísticos clássicos geralmente encontram soluções boas em tempo reduzido e sem explorar profundamente o espaço de busca [Toth and Vigo 2002]. Entre os métodos heurísticos clássicos destaca-se o *Clarke & Wright Savings* [Clarke and Wright 1964]. Esse método será abordado neste trabalho e usado como algoritmo base para aplicação dos Métodos de Monte Carlo.

Métodos meta-heurísticos geralmente incorporam métodos heurísticos clássicos, mas fazem uma exploração maior de regiões mais promissoras do espaço de busca. Esses métodos geralmente retornam soluções melhores do que os métodos tradicionais, ao custo de tempo de computação [Toth and Vigo 2002]. Algoritmos meta-heurísticos utilizam conceitos de diversas áreas para desenvolver heurísticas que possam obter melhores soluções dos problemas [Osman and Kelly 1996]. Entre as áreas podem ser destacadas Inteligência Artificial [Baker and Ayechew 2003, Bell and Mcmullen 2004] e Matemática [Perboli et al. 2011].

4.1. Algoritmo Clarke & Wright Savings

O algoritmo *Clarke & Wright Savings* (CWS) é um dos algoritmos heurísticos mais conhecidos e estudados em CVRP [Toth and Vigo 2002]. Foi definido em 1964 [Clarke and Wright 1964] e é baseado no princípio de economia (*savings*). É um método construtivo, ou seja, a solução vai sendo construída a cada passo, sempre se preocupando com as restrições do problema. O princípio do algoritmo se baseia na ideia de que se há duas rotas diferentes $r = (0, \dots, i, 0)$ e $s = (0, j, \dots, 0)$ que são factíveis de uma fusão (*merge*) em uma nova rota $t = (0, \dots, i, j, \dots, 0)$, a fusão dessas rotas gera uma economia de custo S_{ij} S_{ij} [Clarke and Wright 1964], definida por:

$$S_{ij} = C(i, 0) + C(0, j) - C(i, j). \quad (1)$$

Duas rotas r e s são factíveis de uma fusão quando a soma das demandas dos clientes das duas rotas não excede a capacidade máxima Q de um veículo:

$$\sum_{i \in r} q_i + \sum_{j \in s} q_j \leq Q. \quad (2)$$

Existem duas versões do algoritmo *Clarke & Wright Savings*: a versão paralela e a versão sequencial. A diferença entre as versões está na estratégia de seleção e no número de rotas construídas a cada passo do algoritmo. Enquanto a versão sequencial apenas trata de uma rota por vez, a versão paralela permite que mais de uma rota seja construída. A versão paralela do algoritmo supera os resultados da versão sequencial [Toth and Vigo 2002] e será abordada neste trabalho. O Algoritmo 1 contém o pseudocódigo do algoritmo paralelo.

Algoritmo 1: *Clarke & Wright Savings* [Clarke and Wright 1964]

Entrada: Conjunto de vértices V e conjunto de arestas E, capacidade máxima Q dos veículos
Saída: Conjunto de rotas

```
1 routes ← create_initial_routes(V);
2 savings_list = compute_savings_list(V);
3 foreach i, j ∈ savings_list do
4 |   if feasible_merge(i, j, Q) then
5 |   |   routes.add(merge_routes(i, j));
6 |   end
7 end
8 return routes
```

O Algoritmo 1 começa colocando cada cliente i em uma nova rota $r = (0, i, 0)$ que vai até o cliente e retorna para o depósito (Linha 1 do Algoritmo 1). Depois, é criada a lista de *savings* (*savings list*) com a economia obtida entre cada nó $i, j \in V - \{0\}$, ordenada de forma decrescente de economia. O Algoritmo 2 contém o pseudocódigo do algoritmo de criação da lista de *savings*.

Algoritmo 2: *compute_savings_list* [Clarke and Wright 1964]

Entrada: Conjunto de vértices V
Saída: Lista de *savings*, ordenada por ordem decrescente de economia

```
1 savings_list = {};
2 foreach i, j ∈ V − {0} do
3 |   S_ij = C(i, 0) + C(0, j) − C(i, j);
4 |   savings_list.add(S_i,j);
5 end
6 sort_decreasing(savings_list);
7 return savings_list
```

Nos passos das Linhas 3-6 do Algoritmo 1, a lista de *savings* é iterada e para cada par (i, j) o Algoritmo *CWS* tenta fundir as rotas r e s, as quais i e j pertencem

respectivamente (Linha 4). A Linha 4 verifica as condições para que duas rotas sejam consideradas factíveis de fusão, já citadas anteriormente:

1. i deve ser o primeiro (ou último) cliente a ser visitado na sua rota r,
2. j deve ser o último (ou o primeiro) cliente a ser visitado na sua rota s,
3. r deve ser diferente de s e
4. a capacidade das rotas juntas não deve exceder Q (ver Equação 2).

Quando duas rotas r e s podem ser fundidas, uma nova rota t ligando as rotas r e s é criada. As rotas r e s são eliminadas e a rota t é adicionada na lista de rotas (Linha 5). No final, a lista de rotas criada (*routes*) é retornada pelo algoritmo.

4.2. Algoritmo Clarke & Wright Savings

BinaryMCS-CWS [Takes and Kosters 2010a] é um algoritmo que combina Simulação de Monte Carlo e o algoritmo *Clarke & Wright Savings*. O algoritmo trabalha percorrendo a lista de *savings* do *CWS* e então efetuando um número determinado de simulações. A estratégia usada no algoritmo é explorar a árvore de busca de forma binária, isto é, para cada par de nós (i,j) são feitas r simulações com o par *pertencendo* à solução e r simulações *sem o par pertencer* à solução. O caminho que obtiver o melhor custo médio determina se o par deve ou não ser incluído na solução que está sendo construída [Takes and Kosters 2010a].

Em cada simulação, um par de nós (i,j) escolhido da lista de *savings* só é processado com uma probabilidade q, com $0 \leq q \leq 1$. O algoritmo pula alguns pares de nós durante as simulações e gera soluções diversificadas. Valores altos de q fazem com que muitos pares sejam pulados, gerando certo “caos", enquanto valores menores não permitem tanta exploração do espaço de busca [Takes and Kosters 2010a]. O autor demonstra $0.05 \leq q \leq 0.4$ como o intervalo de q que gera soluções melhores para o algoritmo. A Figura 1 mostra o custo médio das soluções obtidas (eixo vertical) para as escolhas de q (eixo horizontal) para a instância E051-05E [Takes and Kosters 2010a].

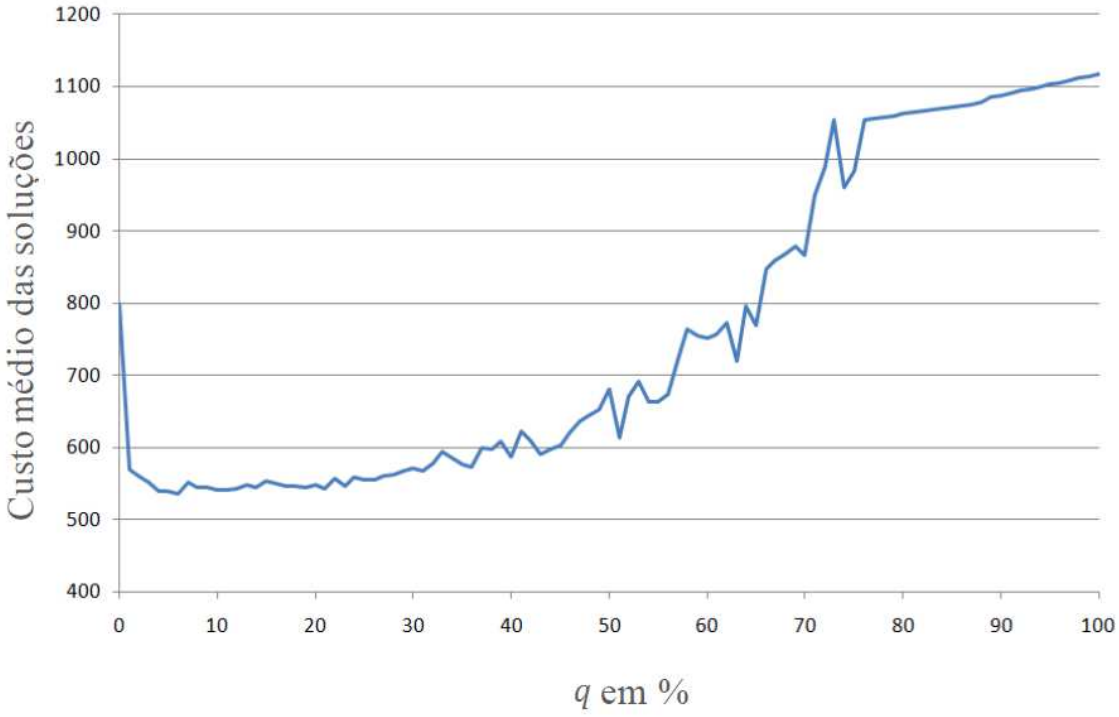


Figure 1. Custo médio x valores de q para a instância E051-05E [Takes and Kosters 2010a]

O algoritmo *BinaryMCS-CWS*, portanto, explora a árvore de busca e constrói uma solução de acordo com os resultados obtidos dentro das simulações. Isso guia a estratégia para o ramo de melhores resultados. Entretanto, a melhor solução pode ser encontrada em um ramo da árvore de busca percorrido durante uma simulação e não necessariamente ao término da iteração da lista de *savings* e da solução construída.

5. Monte Carlo Savings

Nesta seção, um novo algoritmo baseado no algoritmo *Clarke & Wright Savings* é proposto usando técnicas de Simulação de Monte Carlo. O algoritmo foi inspirado por aplicações anteriores dos Métodos de Monte Carlo e melhorias propostas no algoritmo de *CWS*.

Em todos os algoritmos que utilizam o algoritmo de *Clarke & Wright Savings*, as técnicas de Monte Carlo são aplicadas para aumentar a variabilidade das soluções a fim de superar os limites da heurística utilizada. Todos os algoritmos utilizam *MCS* durante a etapa de seleção de nós/arestas e fusão de rotas do algoritmo *CWS*.

Neste trabalho, uma abordagem diferente será feita utilizando a heurística de *Clarke & Wright Savings*. Simulações de Monte Carlo serão usadas para aumentar a variabilidade da lista de *savings*. A ideia é manter o algoritmo original de construção de soluções, mas utilizando as simulações para variar a ordem em que as arestas são percorridas, mudando as soluções finais e permitindo que diferentes rotas sejam geradas em cada simulação.

5.1. Algoritmo Monte Carlo Savings

O algoritmo, batizado de *Monte Carlo Savings*, é um algoritmo heurístico novo baseado no *Clarke & Wright Savings* [Clarke and Wright 1964], que utiliza Simulações de Monte Carlo para gerar lista de *savings* com certo grau de variabilidade e permitir que soluções diversificadas sejam encontradas. O algoritmo *Monte Carlo Savings* (Algoritmo 3) executa r simulações, onde em cada uma delas é gerada uma lista de *savings* utilizando *plain random sampling*. Para cada lista gerada, o algoritmo procede ao método padrão de *Clarke & Wright Savings*, percorrendo a lista de forma decrescente de economia e fundindo as rotas factíveis. Para permitir a variabilidade nas listas de *savings*, a fórmula de cálculo de economia recebe um componente aleatório p, definido em um intervalo $[-\lambda, +\lambda]$, usado para penalizar ou bonificar a economia gerada entre dois nós i e j. Uma economia é penalizada quando $-\lambda \leq p < 0$ e bonificada quando $0 < p \leq \lambda$. Quando $p = 0$, a economia não tem alteração. Seja $s = C(i,0) + C(0,j) - C(i,j)$, a Equação 3 mostra o cálculo da nova economia entre dois clientes i e j.

$$S_{ij} = s + s * p, \tag{3}$$

em que $-\lambda \leq p \leq \lambda$. Essas alterações na economia entre dois clientes permitem que a lista de *savings* tenha as arestas ordenadas de forma diferente do método padrão, fazendo com que a árvore de busca seja mais explorada e soluções diferentes sejam encontradas.

Algoritmo 3: *Monte Carlo Savings*

Entrada: Conjunto de vértices V e conjunto de arestas E, capacidade máxima Q dos veículos, número de simulações r e limite do intervalo aleatório λ

Saída: Conjunto de rotas

```
1  best = create_initial_routes(V);
2  for k ← 0 to r do
3  |   solution = create_initial_routes(V);
4  |   savings_list = compute_mcs_savings(V, λ);
5  |   foreach i, j ∈ savings_list do
6  |   |   if feasible_merge(i, j, Q) then
7  |   |   |   solution.add(merge_routes(i, j)); // CWS
8  |   |   end
9  |   end
10 |   if C(solution) < C(best) then
11 |   |   best = solution;
12 |   end
13 end
14 return best
```

Na Linha 1 do Algoritmo 3, é criada uma solução base com cada cliente sendo servido por seu próprio veículo. A função *create_initial_routes* é a mesma usada pelo *CWS* no Algoritmo 1. Nas Linhas 2-9 do Algoritmo 3, são efetuadas r simulações. Em cada simulação, é criada uma solução base (Linha 3), computada a lista de *savings* (Linha 4) usando o algoritmo *compute_mcs_savings* (Algoritmo 4), e calculada as rotas através do método clássico de *Clarke & Wright Savings* (Linhas 5-9). A função *merge_routes* é a mesma usada pelo *CWS* definida no Algoritmo 1. Finalmente, nas Linhas 10-12 do Algoritmo 3, compara-se o custo total da solução (definido na Seção 2) da simulação atual com o custo da melhor solução encontrada até agora e a solução com menor custo é escolhida como a melhor (*best*). O algoritmo termina quando r simulações são executadas e a melhor solução encontrada dentre as simulações é retornada. O Algoritmo 4 mostra a criação da lista de *savings*, utilizando o parâmetro λ para definir o intervalo em que o valor da componente aleatória p pode estar.

Algoritmo 4: *compute_mcs_savings*

Entrada: Conjunto de vértices V e limite do intervalo aleatório λ
Saída: Lista de *savings*, ordenada por ordem decrescente de economia

```
1 savings_list = {};
2 foreach i, j ∈ V − {0} do
3 |   s = C(i,0) + C(0,j) − C(i,j);
4 |   p = random(−λ, +λ); // random p ∈ [−λ, +λ]
5 |   S_{i,j} = s + (s * p);
6 |   savings_list.add(S_{i,j});
7 end
8 sort_decreasing(savings_list);
9 return savings_list
```

Note que para $r = 1$ e $\lambda = 0$, o algoritmo *Monte Carlo Savings* atua como o algoritmo padrão de *Clarke & Wright Savings*.

A Figura 2 contém as rotas iniciais e a lista de *savings* para uma instância de problema definida em [Larson and Odoni 1981] com 10 clientes e 3 veículos de capacidade $Q = 100$. O exemplo está disponível em [Larson and Odoni 1999]. Cada aresta representa uma rota, que inicialmente vai do depósito até o cliente e retorna. Ao lado do grafo, está a lista de *savings* gerada por uma determinada simulação para $p = 0.05$.

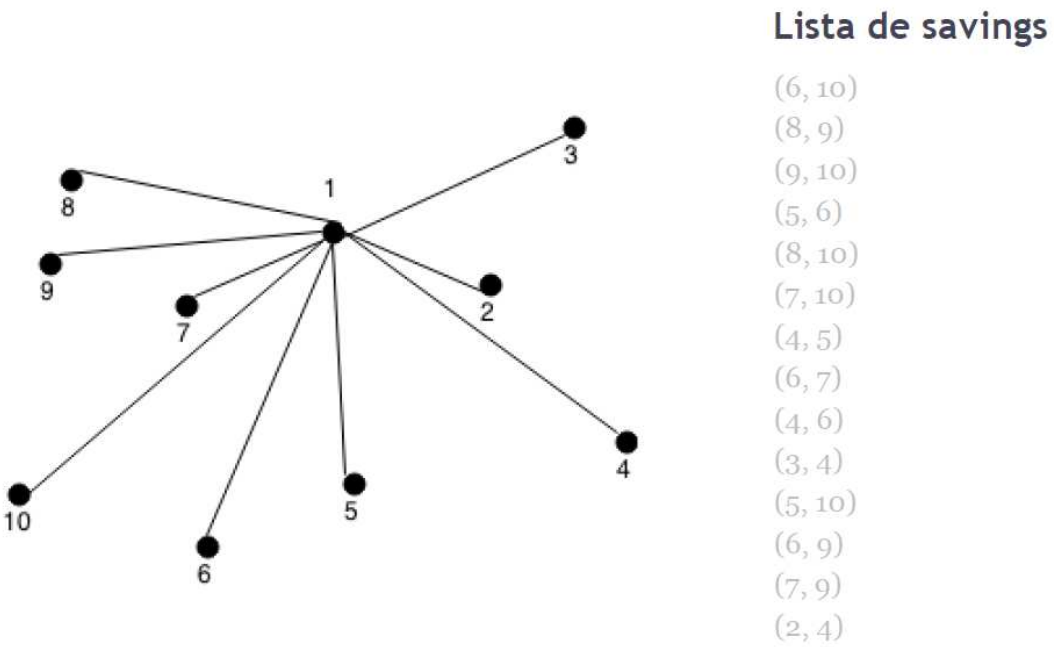


Figure 2. Rotas iniciais criadas pelo algoritmo e uma determinada lista de *savings* gerada para $p = 0.05$

A Figura 3 mostra a população de rotas geradas pelo algoritmo *Monte Carlo Savings* para $r = 3$ e $p = 0.05$ para a instância de exemplo, com custos 520, 453 e 522, respectivamente. Nessa execução, o melhor resultado obtido foi de custo 453. A Figura 4 mostra a rota completa gerada pela simulação com custo total de 453. Os pares em destaque são arestas que foram processadas por serem factíveis de junção. Comparativamente, a solução alcançada pelo algoritmo *Clarke & Wright Savings* tem custo 520 [Larson and Odoni 1981].

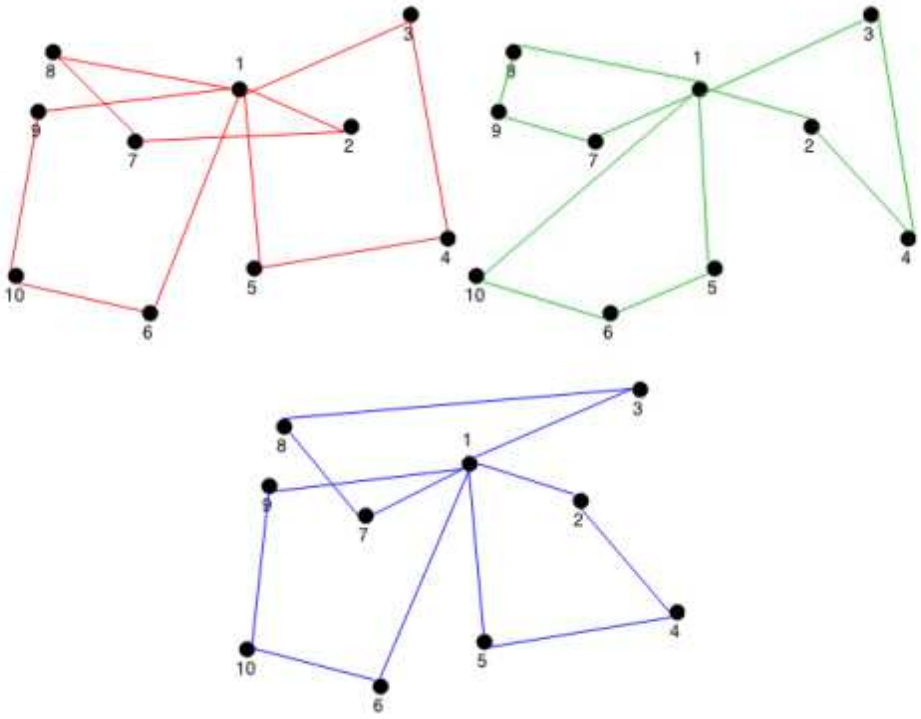
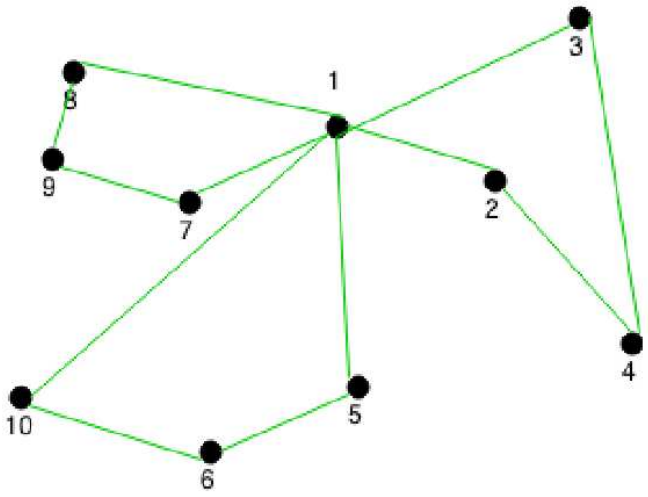


Figure 3. População de rotas geradas pelo algoritmo *Monte Carlo Savings* para $r = 3$ e $p = 0.05$



Lista de savings

(6, 10)
(8, 9)
(9, 10)
(5, 6)
(8, 10)
(7, 10)
(4, 5)
(6, 7)
(4, 6)
(3, 4)
(5, 10)
(6, 9)
(7, 9)
(2, 4)

Figure 4. Rota gerada pelo algoritmo *Monte Carlo Savings* ao final de uma simulação e a lista de *savings* processada

5.2. Análise de complexidade assintótica

Para analisarmos a complexidade assintótica do algoritmo *Monte Carlo Savings* é preciso primeiro analisar a complexidade do *CWS*, usado como base pelo nosso algoritmo. O consumo de tempo do algoritmo *Clarke & Wright Savings*, pode ser representado pela função $f(n,m)$, onde n é o número de clientes do problema e m o número de arestas. Essa função é obtida através da análise de complexidade dos 4 passos principais do algoritmo: inicialização das rotas, criação da lista de *savings*, ordenação da lista de *savings* e fusão das rotas. O primeiro passo, inicialização das rotas (*create_initial_routes* do Algoritmo 1) tem complexidade linear n. Os passos de criação e ordenação da lista de *savings* (*compute_savings_list* do Algoritmo 1) têm complexidade $m + m * \log_2(m)$. Os passos de fusão das rotas, representado nas Linhas 3-6 do Algoritmo 1 também têm complexidade linear m. Assim, podemos concluir que o consumo de tempo do algoritmo *Clarke & Wright Savings* é $f(n,m) = n + m + m * \log_2(m) + m$.

Para facilitar, podemos tomar $m = n^2$, embora o número máximo de arestas em um problema de CVRP seja de $n * (n-1)/2$. Portanto, o consumo de tempo do algoritmo CWS é $O(n^2 * \log_2(n))$.

O algoritmo *Monte Carlo Savings*, de forma simplificada, executa r vezes o algoritmo de *CWS* usando números aleatórios na criação da lista de *savings*. Note que a função *compute_mcs_savings* também tem a mesma complexidade da função *compute_savings_list* do Algoritmo 1. Portanto, o consumo de tempo do algoritmo de *Monte Carlo Savings* é $O(r * n^2 * \log_2(n))$.

É importante ressaltar que o número de simulações r pode ser muito maior do que o número de clientes n.

6. Experimentos

O algoritmo proposto é comparado com algoritmos de CVRP heurísticos, meta-heurísticos e que usam dos Métodos de Monte Carlo, entre eles os algoritmos heurísticos *Clarke & Wright Savings* [Clarke and Wright 1964] e *baseado em Centroide* [Shin and Han 2011]; o algoritmo meta-heurístico *Busca Tabu Granular* [Toth and Vigo 2003]; e o algoritmo que aplica Métodos de Monte Carlo *BinaryMCS-CWS* [Takes and Kosters 2010a].

Para avaliar e comparar o desempenho dos algoritmos, foram usados um subconjunto de 15 instâncias de problemas definidos por Augerat [Augerat 1995] e um subconjunto de 13 instâncias de problemas utilizados por Takes [Augerat 1995] para avaliar o desempenho do seu algoritmo *BinaryMCS-CWS* frente ao *ALGACEA-2* [Faulin and Juan 2007, Faulin and Juan 2008]. O conjunto de problemas e soluções dos *benchmarks* utilizados está no formato TSPLIB [Reinelt 2014].

O *benchmark* de Augerat foi usado para comparar os algoritmos de *Centroide*, *Busca Tabu Granular*, *Clarke & Wright Savings*, *BinaryMCS-CWS* e o *Monte Carlo Savings*. O *benchmark* de Takes foi usado para comparar os algoritmos *BinaryMCS-CWS* e o algoritmo proposto *Monte Carlo Savings*.

Foram avaliados os resultados baseados na melhor resposta obtida pelos algoritmos para aquela determinada instância de problema. As comparações também levam em conta a melhor solução conhecida na literatura. Para este trabalho, o algoritmo proposto *Monte Carlo Savings* foi implementado usando a linguagem de programação *Python*. Também foi implementada a versão paralela do algoritmo de *Clarke & Wright Savings* e o algoritmo *BinaryMCS-CWS*. O código fonte dessas implementações está disponível em `https://github.com/RomuloOliveira/monte-carlo-cvrp/`.

Foram feitas 2000 simulações em cada execução do algoritmo *Monte Carlo Savings*. O algoritmo foi executado cinco vezes para cada instância e o melhor resultado obtido foi selecionado. O tempo limite máximo estipulado por execução de instância foi de 5 minutos, o mesmo tempo usado por [Takes and Kosters 2010b] e [Takes and Kosters 2010b]. Os resultados da *Busca Tabu Granular* e do algoritmo de *Centroide* são os mesmos mostrados em [Barbosa and Takakura 2014]. Os resultados do *BinaryMCS-CWS* e do *CWS* foram obtidos através de implementações próprias. Foram feitas 50 simulações por cada aresta visitada no algoritmo *BinaryMCS-CWS*, utilizando os mesmos parâmetros de [Takes and Kosters 2010a]. O algoritmo também foi executado cinco vezes para cada instância e o seu melhor resultado selecionado.

Uma vez que um algoritmo robusto [Takes and Kosters 2010a]. deve usar o mesmo valor dos parâmetros para todas as instâncias, para definir o valor do parâmetro λ foi usada a ferramenta de configuração automática de algoritmos IRACE [López-Ibáñez et al. 2011]. Essa ferramenta encontra as configurações mais adequadas dos parâmetros utilizando comparações estatísticas. Dado que, métodos de configuração automática devem produzir uma configuração com bom desempenho em instâncias ainda não testadas, na etapa de ajuste de parâmetros foram fornecidas instâncias de [Augerat 1995] e de [DEIS 2014] diferentes das 28 utilizadas na etapa de teste. O IRACE foi executado com as configurações padrões, com exceção do tempo máximo de execução definido em 5 horas e tempo estimado de execução de cada teste definido em 60 segundos, sendo que o valor encontrado pela ferramenta foi 0,034. Nas Seções 6.1 e 6.2 são apresentados os resultados para as 15 e 13 instâncias de teste, respectivamente, usando esse valor de λ.

Os testes foram executados em um computador com processador *AMD Phenom II X4 840 @ 3,2 GHz* e 4GB de memória RAM.

6.1. Benchmark Augerat

Foram utilizadas 15 instâncias do *benchmark* Augerat [Augerat 1995] para comparação dos resultados. As instâncias escolhidas são as mesmas utilizadas por [Barbosa and Takakura 2014]. O número de clientes por instância varia de 16 a 66 clientes.

A Tabela 1 mostra a comparação de resultados entre os algoritmos e a melhor solução conhecida na literatura para essas instâncias.

Table 1. Comparação de resultados para as instâncias de Augerat

Instância	Melhor solução conhecida	Busca Tabu		Centroide		CWS		BinaryMCS-CWS		M.C. Savings	
		Resultado	Diferença	Resultado	Diferença	Resultado	Diferença	Resultado	Diferença	Resultado	Diferença
A-n32-k5	784	**784**	**0,00%**	874	11,48%	834	6,38%	796	1,53%	796	1,53%
A-n33-k6	742	760	2,43%	769	3,64%	880	18,60%	744	0,27%	**742**	**0,00%**
A-n36-k5	799	829	3,75%	899	12,52%	806	0,88%	**805**	**0,75%**	**805**	**0,75%**
A-n45-k7	1146	1197	4,45%	1257	9,69%	1203	4,97%	1178	2,79%	**1154**	**0,70%**
A-n63-k10	1314	1355	3,12%	1476	12,33%	1382	5,18%	1342	2,13%	**1325**	**0,84%**
B-n31-k5	672	675	0,45%	700	4,17%	677	0,74%	674	0,30%	**673**	**0,15%**
B-n34-k5	788	**790**	**0,25%**	976	23,86%	806	2,28%	793	0,63%	792	0,51%
B-n38-k6	805	821	1,99%	835	3,73%	834	3,60%	**809**	**0,50%**	819	1,74%
B-n44-k7	909	946	4,07%	963	5,94%	939	3,30%	**926**	**1,87%**	928	2,09%
B-n66-k9	1316	1423	8,13%	1420	7,90%	1424	8,21%	**1358**	**3,19%**	1360	3,34%
P-n16-k8	450	452	0,44%	478	6,22%	(478)	6,22%	**450**	**0,00%**	465	3,33%
P-n19-k2	212	223	5,19%	242	14,15%	240	13,21%	**212**	**0,00%**	(219)	3,30%
P-n23-k8	529	573	8,32%	595	12,48%	(537)	1,51%	**529**	**0,00%**	(534)	0,95%
P-n40-k5	458	490	6,99%	595	29,91%	522	13,97%	**470**	**2,62%**	474	3,49%
P-n50-k10	696	720	3,45%	765	9,91%	740	6,32%	718	3,16%	**706**	**1,44%**
Total	11620	12038	3,6%	12844	10,53%	12302	5,87%	11804	1,58%	**11792**	**1,48%**

Nota-se que o algoritmo *Monte Carlo Savings* obtém os melhores resultados entre os algoritmos testados em 6 das 15 instâncias, incluindo uma solução igual à melhor solução conhecida e também o melhor resultado geral, com apenas 1,48% de variação da melhor solução conhecida da literatura. Nas instâncias *P-n16-k8* e *P-n23-k8* o algoritmo de *Clarke & Wright Savings* não obteve uma solução factível com o número mínimo desejado de veículos. O mesmo aconteceu com o algoritmo *Monte Carlo Savings* nas instâncias *P-n19-k2* e *P-n23-k8*.

Os resultados de ambos os algoritmos que utilizam Simulações de Monte Carlo mostram-se melhores que as outras abordagens, inclusive a *Busca Tabu Granular*.

A Tabela 2 mostra o resumo do desempenho dos algoritmos considerando o melhor resultado, a média dos resultados e o pior resultado. Os algoritmos *BinaryMCS-CWS* e *Monte Carlo Savings* obtiveram os melhores resultados.

Table 2. Resumo do desempenho dos algoritmos para as instâncias de Augerat

Algoritmos	Melhor resultado (%)	Média dos resultados (%)	Pior resultado (%)
Monte Carlo Savings	**0,00%**	1,61%	3,49%
BinaryMCS-CWS	**0,00%**	**1,32%**	**3,19%**
Busca Tabu	**0,00%**	3,54%	8,32%
CWS	0,74%	6,36%	18,60%
Centroide	3,64%	11,19%	29,91%

6.2. Benchmark Takes

Foram utilizadas 13 instâncias, sendo duas de [Augerat 1995] e 11 de [Augerat 1995]. Como mencionado anteriormente, as instâncias são as mesmas utilizadas por [Takes and Kosters 2010b] em sua comparação com o algoritmo *ALGACEA-2*. O número de clientes por instância varia de 51 a 200 clientes para esse subconjunto.

A Tabela 3 mostra a comparação de resultados entre os algoritmos e a melhor solução conhecida na literatura para essas instâncias.

Table 3. Comparação de resultados para as instâncias usadas por Takes

Instância	Melhor solução conhecida	BinaryMCS-CWS		Monte Carlo Savings	
		Resultado	Diferença	Resultado	Diferença
A-n65-k9	1174	1224	4.26%	**1196**	**1,87%**
A-n80-k10	1764	**1805**	**2.32%**	1806	2,38%
E051-05E	525	**536**	**2.10%**	537	2,29%
E072-04F	242	265	9.50%	**249**	**2,89%**
E076-07S	691	**703**	**1.74%**	**703**	**1,74%**
E076-10E	837	860	2.75%	**858**	**2,51%**
E076-14U	1029	1057	2.72%	**1054**	**2,43%**
E101-08E	826	861	4.24%	**857**	**3,75%**
E101-10C	820	844	2.93%	**831**	**1,34%**
E101-14U	1091	**1101**	**0.92%**	1105	1,28%
E151-12C	1031	1084	5.14%	**1080**	**4,75%**
E200-17B	1291	**1346**	**4.26%**	1365	5,73%
E200-17C	1311	**1360**	**3.74%**	1367	4,27%
Total	12632	13046	3.28%	**13008**	**2,98%**

O algoritmo *Monte Carlo Savings* obteve os melhores resultados, agora em 8 das 13 instâncias, com variação de 2,98% da melhor solução conhecida da literatura.

A Tabela 4 mostra o resumo do desempenho dos dois algoritmos. O *Monte Carlo Savings* mostrou bom desempenho também em instâncias com grande número de clientes. O resultado mais próximo da melhor solução conhecida foi obtido pelo *BinaryMCS-CWS*, enquanto o *Monte Carlo Savings* foi melhor nos outros dois quesitos (média dos resultados e pior resultado), além do resultado geral.

Table 4. Resumo do desempenho dos algoritmos para as instâncias usadas por Takes

Algoritmos	Melhor resultado (%)	Média dos resultados (%)	Pior resultado (%)
Monte Carlo Savings	1,28%	**2,86%**	**5,73%**
BinaryMCS-CWS	**0,92%**	3,59%	9,50%

6.3. Análise de comportamento do algoritmo

No trabalho feito por No trabalho feito por [Takes and Kosters 2010b], o parâmetro aleatório q tem relação quase direta com o custo das soluções obtidas pelo algoritmo *BinaryMCS-CWS* [Takes and Kosters 2010b], como pode ser visto na Figura 1. O algoritmo *BinaryMCS-CWS* utiliza q para pular arestas da solução com uma probabilidade $1 - q$. Isso faz com que valores muito altos de q gerem soluções com muito mais custos [Takes and Kosters 2010b]..

No algoritmo *Monte Carlo Savings*, o parâmetro p é um número aleatório definido no intervalo $[-\lambda, +\lambda]$ e utilizado para aumentar ou diminuir o valor de economia de uma aresta, alterando a ordenação original da lista de *savings*. Diferentemente do *BinaryMCS-CWS*, o parâmetro λ não tem relação direta com o custo das soluções. Isso se deve ao fato de que o algoritmo *Monte Carlo Savings* varia limitadamente a lista de *savings*, usando o parâmetro p como aditivo, penalizando ou bonificando as economias, com tendências de manter as arestas com maior economia no topo da lista e as com menor economia no fim. Esse comportamento tem relação direta com os bons resultados obtidos pelo algoritmo, que aproveita melhor a heurística desenvolvida por *Clarke & Wright* [Clarke and Wright 1964].

A Figura 5 mostra o custo médio de todas as rotas obtidas dentro de *uma mesma execução* (eixo vertical) para as escolhas de λ (eixo horizontal) para as instâncias A-n32-k5, A-n65-k9 e E051-05E, para $r = 2000$. Como esperado, não existe uma relação direta entre λ e o custo médio das rotas.

Também não foi encontrada relação entre λ e o tempo necessário para se encontrar a melhor solução, dentro do limite de 5 minutos. A Figura 6 mostra o instante de tempo (eixo vertical) em que a melhor solução entre as simulações foi encontrada para um determinado λ (eixo horizontal) na instância E051-05e. Note que a melhor solução encontrada dentre as simulações *numa mesma execução* não é, necessariamente, a melhor solução entre *todas as execuções* do algoritmo.

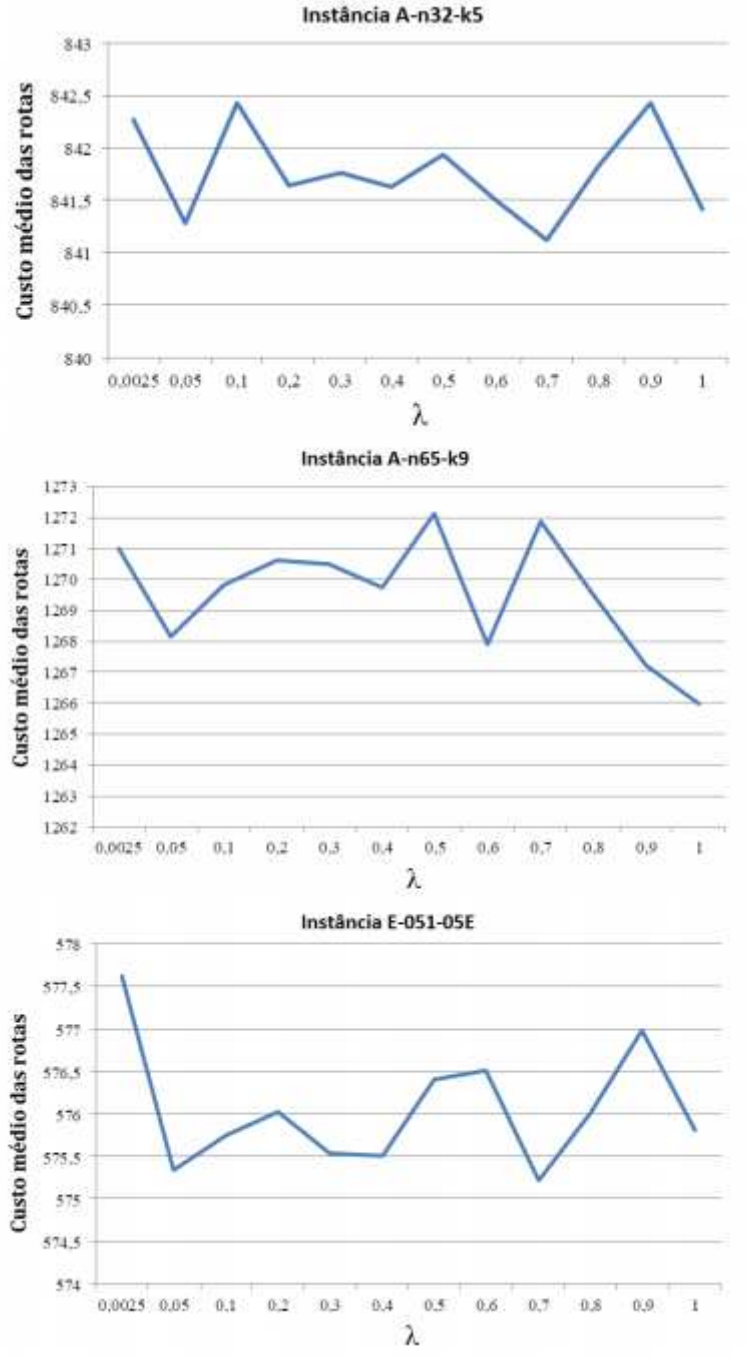


Figure 5. Custo médio das rotas x Valores de λ para as instâncias A-n32-k5, A-n65-k9 e E-051-05E

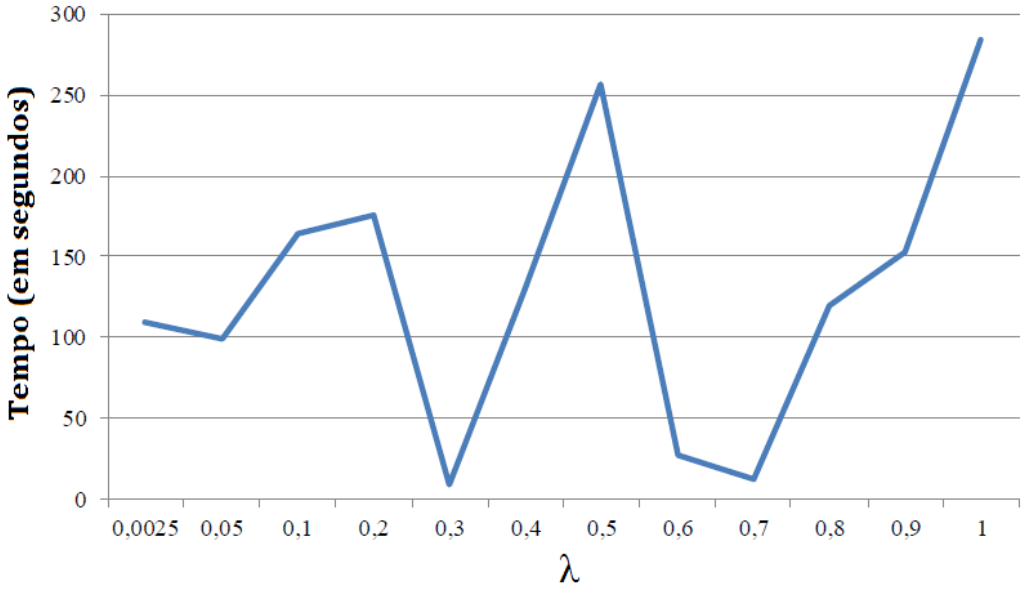


Figure 6. Tempo necessário para encontrar a melhor solução x Valores de λ para a instância E-051-05E

6.4. Discussão

A aplicação das técnicas de Monte Carlo para resolver problemas de roteamento de veículos reais é limitada pelo tempo de computação necessário. Por se basear em Simulações de Monte Carlo, diversas soluções são geradas pelo algoritmo para se encontrar soluções ótimas ou próximas de ótimas. Nos testes, foi estipulado um limite de 5 minutos que se mostrou suficiente para encontrar boas soluções, mas que pode impossibilitar a aplicação dessas técnicas em ambientes onde há restrições de tempo. Podemos notar, entretanto, o desempenho superior dos dois algoritmos que utilizam essas técnicas, o *BinaryMCS-CWS* e o *Monte Carlo Savings*, esse último proposto neste trabalho.

7. Trabalhos Correlatos

Comparado com a literatura existente para o Problema de Roteamento de Veículos, a aplicação dos Métodos de Monte Carlo em VRP e CVRP é relativamente pequena [Takes and Kosters 2010b]. [Buxey 1979] foi provavelmente o primeiro trabalho que combina Simulação de Monte Carlo e o algoritmo *Clarke & Wright Savings*. Esse trabalho foi posteriormente revisado em [Faulin and Juan 2007, Faulin and Juan 2008]. O algoritmo *ALGACEA-1* foi proposto em [Faulin and Juan 2007]. Logo depois, os autores propuseram o *ALGACEA-2*, uma versão melhorada do *ALGACEA-1* que introduziu no algoritmo o conceito de Entropia [Faulin and Juan 2007, Faulin and Juan 2008, Taneja et al. 1989]. Em [Takes and Kosters 2010a], foram propostas diversas formas de aplicar *MCS* em problemas de roteamento de veículos, entre elas um algoritmo baseado em *Nearest Neighbor Insertion* e dois algoritmos baseados em *CWS*, o *BestX-CWS* e o *BinaryMCS-CWS*. O algoritmo *BinaryMCS-CWS* supera os resultados de todos os trabalhos anteriores com *MCS* com resultados comparáveis aos melhores algoritmos de CVRP existentes na literatura [Takes and Kosters 2010a].

Os algoritmos *BinaryMCS-CWS* e *Monte Carlo Savings* utilizam o *CWS* como base. Enquanto o *BinaryMCS-CWS* explora a árvore de busca fazendo simulações com a mesma lista de *savings*, o algoritmo *Monte Carlo Savings* utiliza as simulações para gerar uma população de lista de *savings* diferentes, se baseando na eficiência bem estudada do algoritmo de *Clarke & Wright Savings* na geração da lista e obtendo, no geral, resultados melhores que todos os outros algoritmos analisados.

Em [de Carvalho Oliveira and Delgado 2015] foi apresentado o algoritmo *Monte Carlo Savings* e foram realizados experimentos com diferentes valores fixos de λ definidos manualmente para as diversas instâncias de teste. Neste trabalho, foi usada a ferramenta IRACE [López-Ibáñez et al. 2011] para encontrar um valor de λ de maneira automática para todas as instâncias. Além disso, o algoritmo proposto foi ilustrado através de um exemplo; foi comparado o comportamento do parâmetro q do algoritmo *BinaryMCS-CWS* e o parâmetro λ do algoritmo proposto; e foi analisada a relação entre λ e o tempo para encontrar a melhor solução.

8. Conclusão

Neste trabalho, foi proposto, implementado e analisado o algoritmo *Monte Carlo Savings*, que utiliza uma abordagem simples para aplicação dos Métodos de Monte Carlo, mais especificamente a Simulação de Monte Carlo, em problemas de roteamento de veículos capacitados, baseado no algoritmo heurístico *Clarke & Wright Savings*.

Nos dois *benchmarks* usados, no geral, o algoritmo proposto obteve o melhor desempenho entre todos os algoritmos comparados, com variação de apenas 1,48% e 2,98% para as melhores soluções conhecidas da literatura de cada um dos *benchmarks*, superando métodos bem conhecidos, como a *Busca Tabu Granular*. Em comparação com outros algoritmos que utilizam Simulações de Monte Carlo, o *Monte Carlo Savings* também se mostrou uma boa opção, além de ser um algoritmo robusto e simples de ser implementado. Portanto, o algoritmo proposto demonstrou resultados comparáveis aos melhores algoritmos disponíveis na literatura, podendo ser considerado junto com o *BinaryMCS-CWS* o *estado-da-arte* quando o assunto é Métodos de Monte Carlo para CVRP.

Trabalhos futuros podem envolver a utilização do algoritmo proposto como entrada para outros métodos meta-heurísticos, incluindo a Busca Tabu Granular e o BinaryMCS-CWS; estudos e variações da fórmula de cálculo de savings; e a escolha automática do valor do parâmetro λ baseada nas características da instância.

Referências

Augerat, P. (1995). Approche polyèdrale du problème de tournées de véhicules. PhD thesis, Institut National Polytechnique de Grenoble-INPG.

Baker, B. M. and Ayechew, M. (2003). A genetic algorithm for the vehicle routing problem. Computers & Operations Research, 30(5):787–800.

Barbosa, V. P. and Takakura, R. M. (2014). Implementação e Comparação de Algoritmos que Resolvam o Problema do Roteamento de Veículos Capacitados.

Bachelor's thesis, Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo, Brasil. 44 f.

Barr, R., Golden, B., Kelly, J., Resende, M., and Stewart, William R., J. (1995). Designing and reporting on computational experiments with heuristic methods. Journal of Heuristics, 1(1):9–32.

Bell, J. E. and Mcmullen, P. R. (2004). Ant colony optimization techniques for the vehicle routing problem. Advanced Engineering Informatics, 18(1):41–48.

Bindel, D. and Goodman, J. (2009). Principles of Scientific Computing. Manuscript.

Bussab, W. d. O. and Morettin, P. A. (2004). Estatística básica. Saraiva, São Paulo, 5 edition.

Buxey, G. M. (1979). The vehicle scheduling problem and Monte Carlo simulation. Journal of the Operational Research Society, 30:563–573.

Christie, J. S., Satir, S., and Campus, T. P. (2006). Saving our energy sources and meeting Kyoto emission reduction targets while minimizing costs with application of vehicle logistics optimization. In 2006 Annual conference and exhibition of the Transportation Association of Canada: Transportation without boundaries, Charlottetown, Prince Edward Island.

Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. Operations research, 12(4):568–581.

Cruse, T. A. (1997). Reliability-based mechanical design, volume 108. CRC Press.

Dantzig, G. B. and Ramser, J. H. (1959). The Truck Dispatching Problem. Management Science, 6(1):80–91.

de Carvalho Oliveira, R. A. and Delgado, K. V. (2015). Capacitated vehicle routing system applying Monte Carlo Methods. In Anais do XI Simpósio Brasileiro de Sistemas de Informação (SBSI), Goiânia, pages 1–8.

DEIS (2014). VRPLIB: A Vehicle Routing Problem LIBrary. Disponível em http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html. Acesso em: 24 ago 2014.

Faulin, J. and Juan, A. (2007). ALGACEA-2: An entropy-based heuristics for the Capacitated Vehicle Routing Problem. In Seventh Metaheuristics International Conference. 3 p.

Faulin, J. and Juan, A. A. (2008). The ALGACEA-1 method for the capacitated vehicle routing problem. International Transactions in Operational Research, 15(5):599–621.

Fischetti, M., Toth, P., and Vigo, D. (1994). A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. Operations Research, 42(5):846– 859.

Gentle, J. E. (2003). Random number generation and Monte Carlo methods. Springer Science & Business Media.

Giaglis, G. M. et al. (2004). Minimizing logistics risk through real-time vehicle routing and mobile technologies: Research to date and future trends. International Journal of Physical Distribution & Logistics Management, 34(9):749–764.

Gutin, G. and Punnen, A. P., editors (2002). The traveling salesman problem and its variations. Combinatorial optimization. Kluwer Academic, Dordrecht, London.

Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. In Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges, volume 59 of DIMACS, pages 215–250. American Mathematical Society.

Kanda, J. Y. (2014). Sistema de meta-aprendizado para a seleção de meta-heurística para o problema do caixeiro viajante. In Anais do X Simpósio Brasileiro de Sistemas de Informação (SBSI), Londrina, pages 651–662.

Laporte, G. (1992). The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research, 59(3):345–358.

Larson, R. C. and Odoni, A. R. (1981). Urban operations research. Number Monograph.

Larson, R. C. and Odoni, A. R. (1999). 6.4.12 Single-Depot VRP. http://web.mit.edu/urban_or_book/www/book/chapter6/6.4.12.html. Acesso em: 24 julho 2015.

Leeuewn, J. (1990). Handbook of theoretical computer science: Algorithms and complexity, volume 1. Elsevier.

Lenstra, J. K. and Rinnooy Kan, A. H. G. (1981). Complexity of Vehicle Routing and Scheduling Problems. Networks, 11:221–227.

López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., and Birattari, M. (2011). The IRACE package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium.

Lysgaard, J., Letchford, A. N., and Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. Mathematical Programming, 100(2):423– 445.

Metropolis, N. and Ulam, S. (1949). The Monte Carlo Method. Journal of the American statistical association, 44(247):335–341.

Mooney, C. Z. (1997). Monte Carlo Simulation. Number 116. Sage.

Osman, I. H. and Kelly, J. P. (1996). Meta-heuristics: An overview. In Meta-Heuristics, pages 1–21. Springer.

Pearl, J. (1984). Heuristics: Intelligent search strategies for computer problem solving. Addison-Wesley Pub. Co., Inc., Reading, MA.

Perboli, G., Tadei, R., and Vigo, D. (2011). The two-echelon capacitated vehicle routing problem: Models and math-based heuristics. Transportation Science, 45(3):364–380.

Reinelt, G. (2014). TSPLIB. Disponível em http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. Acesso em: 24 ago 2014.

Shin, K. and Han, S. (2011). A Centroid-based Heuristic Algorithm for the Capacitated Vehicle Routing Problem. Computing and Informatics, 30(4):721–732.

Takes, F. and Kosters, W. (2010a). Applying Monte Carlo Techniques to the Capacitated Vehicle Routing Problem. Master's thesis, Leiden University, Leiden, Holanda. 61 f.

Takes, F. and Kosters, W. (2010b). Applying Monte Carlo techniques to the Capacitated Vehicle Routing Problem. In Proceedings of 22th Benelux Conference on Artificial Intelligence (BNAIC 2010). 8 p.

Taneja, I. J. et al. (1989). On generalized information and divergence measures and their applications: A brief review. Questiió: Quaderns d'Estadística, Sistemes, Informatica i Investigació Operativa, 13(1):47–73.

Toth, P. and Vigo, D. (2002). The Vehicle Routing Problem, pages 1–26. 9 edition.

Toth, P. and Vigo, D. (2003). The granular tabu search and its application to the vehiclerouting problem. Informs Journal on computing, 15(4):333–346.