Sistema para roteamento de veículos capacitados aplicando *Métodos de Monte Carlo*

Alternative Title: Capacitated vehicle routing system applying Monte Carlo methods

Rômulo Augusto de Carvalho Oliveira
Universidade de São Paulo
romulo.augusto.oliveira@usp.br

Karina Valdivia Delgado
Universidade de São Paulo
kvd@usp.br

RESUMO

O Problema de Roteamento de Veículos (*Vehicle Routing Problem*, VRP) é 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), utilizando Métodos de Monte Carlo. Métodos de Monte Carlo são métodos estatísticos que utilizam amostragem aleatória para resolver problemas probabilísticos e determinísticos. 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. A comparação, análise e avaliação do algoritmo foram feitas com base em *benchmarks* de problemas existentes na literatura.

Palavras-Chave

Problema de Roteamento de Veículos Capacitados, Métodos de Monte Carlo, Otimização Combinatória, Algoritmos Heurísticos, Algoritmos Meta-heurísticos

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) using Monte Carlo methods. Monte Carlo methods are statistical methods that use random sampling to solve probabilistic and deterministic problems. 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, it overcomes previous work with Monte Carlo methods. The comparison, analysis and evaluation of the algorithm were based on existing benchmarks in the literature.

SBSI 2015, May 26th-29th, 2015, Goiânia, Goiás, Brazil

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.4.2 [**Types of Systems**]: [Logistics]

General Terms

Theory, Algorithms

Keywords

Capacitated Vehicle Routing Problem, Monte Carlo Methods, Combinatorial Optimization, Heuristic Algorithms, Metaheuristic Algorithms

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 [12]. É uma variação do Problema do Caixeiro Viajante [17, 18], portanto da classe NP-difícil [21], 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 [32], o roteamento de veículos é parte crítica e fundamental para execução bem-sucedida do processo de logística e transporte [16]. 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 [9].

Neste trabalho, será abordada a variação mais conhecida [29] 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 [7]. 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 discute sobre algoritmos utilizados para resolver CVRP. A aplicação dos Métodos de Monte Carlo em CVRP e o algoritmo proposto neste trabalho são discutidos na Seção 5. Na Seção 6 são apresentadas a metodologia e os experimentos realizados. A Seção 7 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 $V = \{v_0, v_1, ..., v_n | n \geq 1\}$ 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 $R = \{r_1, r_2, ..., r_m\}$, 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 $C(R)$ possível. Cada rota r_i é servida por exatamente um veículo, começa e termina no depósito, tem tamanho k_i e é definida por $r_i = (v_0^i, v_1^i, ..., v_{k_i}^i, v_{k_i+1}^i)$, onde $v_0^i = v_{k_i+1}^i = v_0$, $v_j^i \neq v_l^i$ 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 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 [7]. Definido em 1949 por Nicholas Metropolis e Stanislaw Ulam, foi desenvolvido por John Von Neumann e Stanislaw Ulam [23].

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 [24]. 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 [15].

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*) [8].

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 [8, 11]. 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 [30].

Plain random sampling pode ser aplicado em Simulações de Monte Carlo para problemas de busca [30]. 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 [30]. 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 [20]. Entre os algoritmos exatos destacam-se *branch and bound* [19, 14] e *branch and cut* [22].

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 [26]. Os métodos heurísticos são divididos em duas classes principais: métodos heurísticos clássicos e meta-heurísticos [32].

Algoritmos heurísticos clássicos geralmente encontram soluções boas em tempo reduzido e sem explorar profundamente o espaço de busca [32]. Entre os métodos heurísticos clássicos destaca-se o *Clarke & Wright Savings* [10]. 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 [32]. Algoritmos meta-heurísticos utilizam conceitos de diversas áreas para desenvolver heurísticas que possam obter melhores soluções dos problemas [25]. Entre as áreas podem ser

destacadas Inteligência Artificial [4, 6] e Matemática [27].

4.1 Algoritmo Clarke & Wright Savings

O algoritmo *Clarke & Wright Savings* (CWS) é um dos algoritmos heurísticos mais conhecidos e estudados em CVRP [32]. Foi definido em 1964 [10] 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 $r = (0, ..., i, 0)$ e $s = (0, j, ..., 0)$ que são factíveis de uma fusão (*merge*) em uma nova rota $t = (0, ..., i, j, ...0)$, a fusão dessas rotas gera uma economia de custo S_{ij} [10], definida por:

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

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 [32] e será abordada neste trabalho. O Algoritmo 1 contém o pseudocódigo do algoritmo paralelo.

Algorithm 1: *Clarke & Wright Savings*

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 ← merge_routes(i,j);
6 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.

Nos passos das Linhas 3-5 do Algoritmo 1, a lista de *savings* é iterada e para cada par (i, j) o Algoritmo *CWS* tenta fundir as rotas as quais i e j pertencem (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).

Algorithm 2: *compute_savings_list*

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(0, i) + C(0, j) − C(i, j);
4     savings_list ← S_i,j;
5 sort_decreasing(savings_list);
6 return savings_list
```

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 Algoritmos utilizando Métodos de Monte Carlo

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 [30]. O algoritmo *ALGACEA-1* foi proposto em [13] utilizando Técnicas de Monte Carlo sobre o algoritmo de Clarke & Wright Savings. Logo depois, os autores propuseram o *ALGACEA-2*, uma versão melhorada do *ALGACEA-1* que introduziu no algoritmo o conceito de Entropia [13, 31]. Em [29], 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 [29].

O algoritmo *BinaryMCS-CWS* 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 [29].

Em cada simulação, um par de nós (i, j) escolhido da lista de *savings* só é processado com uma probabilidade p, com $0 \leq p \leq 1$. O algoritmo pula alguns pares de nós durante as simulações e gera soluções diversificadas. Valores altos de p fazem com que muitos pares sejam pulados, gerando certo "caos", enquanto valores menores não permitem tanta exploração do espaço de busca [29]. O autor demonstra $0.05 \leq p \leq 0.4$ como o intervalo de p que gera soluções melhores para o algoritmo.

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 Carlos 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* [10], 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 si-mulações, 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 $s = C(0,i) + C(0,j) - C(i,j)$, a Equação 3 mostra o cálculo da nova economia entre dois clientes i e j.

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

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.

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 5-7). A função *merge_routes* é a mesma usada pelo *CWS* definida no Algoritmo 1. Finalmente, nas Linhas 8-9 do Algoritmo 3, compara-se o custo 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.

Algorithm 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 ← merge_routes(i,j); // CWS
8      if C(solution) < C(best) then
9          best = solution;
10 return best
```

Algorithm 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(0,i) + 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 ← S_{i,j};
7 sort_decreasing(savings_list);
8 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*.

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 representada 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 $m+m \cdot log_2(m)$. Os passos de fusão das rotas, representado nas Linhas 3-5 do Algoritmo 1 também têm complexidade linear m. Assim, podemos concluir que o consumo de tempo do algoritmo *Clarke & Wright Savings* é $f(m,n) = n + m + m \cdot \log_2(m) + m$.

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

O algoritmo *Monte Carlo Savings*, de forma simplificada, executa r vezes o algoritmo de *CWS* usando números aleatórios para variar 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 $O(r \cdot n^2 \cdot log_2(n))$.

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* [10] e *baseado em Centroide* [28, 5]; o algoritmo meta-heurístico *Busca Tabu Granular* [33, 5]; e o algoritmo que aplica Métodos de Monte Carlo *BinaryMCS-CWS* [29].

Para avaliar e comparar o desempenho dos algoritmos, foram usados um subconjunto de 15 instâncias de problemas definidos por Augerat [3] e um subconjunto de 13 instâncias de problemas utilizados por Takes [30] para avaliar o desempenho do seu algoritmo *BinaryMCS-CWS* frente ao *ALGACEA-2*. O conjunto de problemas e soluções dos *benchmarks* utilizados está no formato TSPLIB [1].

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 solução ótima disponível 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 [30] e [13]. Os resultados da *Busca Tabu Granular* e do algoritmo de *Centroide* são os mesmos mostrados em [5]. Os resultados do *BinaryMCS-CWS* e do *CWS* foram obtidos através de implementação próprias. Foram feitas 50 simulações por cada aresta visitada no algoritmo *BinaryMCS-CWS*, utilizando os mesmos parâmetros de [29]. O algoritmo também foi executado cinco vezes para cada instância e o seu melhor resultado selecionado.

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 [3] para comparação dos resultados. As instâncias escolhidas são as mesmas utilizadas por [5]. 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 solução ótima existente na literatura para essas instâncias.

Nota-se que o algoritmo *Monte Carlo Savings* obtém os melhores resultados entre os algoritmos testados em 9 das 15 instâncias, incluindo 2 soluções ótimas e também o melhor resultado geral, com apenas 1,39% de variação da solução ótima. 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. Foi usado $\lambda = 0,05$ para 13 instâncias, mostrando ser esse um bom valor para o parâmetro. Na instância B-n31-k5 foi usado $\lambda = 0,0025$ e na instância *P-n19-k2* foi usado $\lambda = 0,1$.

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. O *Monte Carlo Savings* obteve melhor desempenho considerando o melhor resultado e a média de resultados.

TabelaTable 2: Resumo do desempenho dos algoritmos para as instâncias do Augerat

Algoritmos	Melhor resultado (%)	Média dos resultados (%)	Pior resultado (%)
Monte Carlo Savings	**0,00%**	**1,19%**	3,95%
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 [3] e 11 de [2]. Como mencionado anteriormente, as instâncias são as mesmas utilizadas por [30] 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 solução ótima existente na literatura para essas instâncias.

Novamente, o algoritmo *Monte Carlo Savings* obteve os melhores resultados, agora em 9 das 13 instâncias, com variação de 3,13% da solução ótima. Foi usado $\lambda = 0,0025$ para 10 instâncias, mostrando-se um bom valor para o parâmetro para instâncias maiores, em contrapartida de $\lambda = 0,05$ como melhor valor para o parâmetro do *benchmark* anterior. Nas instâncias *E101-14U* e *E200-17B* foi usado $\lambda = 0,05$ quanto na instância *E051-05E* foi usado $\lambda = 0,8$.

A Tabela 4 mostra o resumo do desempenho dos dois algoritmos.

TabelaTable 1: Comparação de resultados para as instâncias do Augerat

Instância	Solução ótima	Busca Tabu		Centroide		CWS		BinaryMCS-CWS		Monte Carlo 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%	787	0,38%
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%	**1155**	**0,79%**
A-n63-k10	1314	1355	3,12%	1476	12,33%	1382	5,18%	1342	2,13%	**1341**	**2,05%**
B-n31-k5	672	675	0,45%	700	4,17%	677	0,74%	**674**	**0,30%**	**674**	**0,30%**
B-n34-k5	788	**790**	**0,25%**	976	23,86%	806	2,28%	793	0,63%	795	0,89%
B-n38-k6	805	821	1,99%	835	3,73%	834	3,60%	**809**	**0,50%**	821	1,99%
B-n44-k7	909	946	4,07%	963	5,94%	939	3,30%	**926**	**1,87%**	**926**	**1,87%**
B-n66-k9	1316	1423	8,13%	1420	7,90%	1424	8,21%	**1358**	**3,19%**	1368	3,95%
P-n16-k8	450	452	0,44%	478	6,22%	(478)	6,22%	**450**	**0,00%**	**450**	**0,00%**
P-n19-k2	212	223	5,19%	242	14,15%	240	13,21%	**212**	**0,00%**	215	1,42%
P-n23-k8	529	573	8,32%	595	12,48%	(537)	1,51%	**529**	**0,00%**	533	0,76%
P-n40-k5	458	490	6,99%	595	29,91%	522	13,97%	470	2,62%	**464**	**1,31%**
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%	**11782**	**1,39%**

TabelaTable 3: Comparação de resultados para as instâncias usadas por Takes

Instância	Solução ótima	BinaryMCS-CWS		Monte Carlo Savings	
		Resultado	Diferença	Resultado	Diferença
A-n65-k9	1174	1224	4.26%	**1197**	**1,96%**
A-n80-k10	1764	**1805**	**2.32%**	1812	2,72%
E051-05E	525	536	2.10%	**535**	**1,90%**
E072-04F	242	265	9.50%	**254**	**4,96%**
E076-07S	691	703	1.74%	**701**	**1,45%**
E076-10E	837	860	2.75%	**859**	**2,63%**
E076-14U	1029	**1057**	**2.72%**	1068	3,79%
E101-08E	826	861	4.24%	**859**	**4,00%**
E101-10C	820	844	2.93%	**832**	**1,46%**
E101-14U	1091	**1101**	**0.92%**	1105	1,28%
E151-12C	1031	**1084**	**5.14%**	**1084**	**5,14%**
E200-17B	1291	**1346**	**4.26%**	1364	5,65%
E200-17C	1311	1360	3.74%	**1358**	**3,59%**
Total	12632	13046	3.28%	**13028**	**3.13%**

TabelaTable 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%	**3.12%**	**5.65%**
BinaryMCS-CWS	**0.92%**	3.59%	9.50%

O *Monte Carlo Savings* mostrou bom desempenho também em instâncias com grande número de clientes. Porém, nesse segundo *benchmark*, a diferença entre os algoritmos é menor. O resultado mais próximo da solução ótima 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.

7. 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.

Os dois algoritmos 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.

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, o algoritmo proposto obteve o melhor desempenho entre todos os algoritmos comparados, com variação de apenas 1,39% e 3,13% para as soluções ótimas 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, superando em ambos os *benchmarks* o algoritmo *BinaryMCS-CWS*. Portanto, o algoritmo proposto demonstrou resultados comparáveis aos melhores algoritmos disponíveis na literatura, podendo ser considerado 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 metaheurísticos, incluindo a Busca Tabu Granular e o BinaryMCS-CWS; estudos e variações da fórmula de cálculo de savings, assim como o estudo da sensibilidade da solução proposta em função dos novos parâmetros adicionados, r e λ.

9. REFERÊNCIAS

[1] TSPLIB, 2014. Disponível em <`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/`>. Acesso em: 24 ago 2014.

[2] VRPLIB: A Vehicle Routing Problem LIBrary, 2014. Disponível em <`http://www.or.deis.unibo.it/research_pages/ORinstances/VRPLIB/VRPLIB.html`>. Acesso em: 24 ago 2014.

[3] P. Augerat. *Approche polyèdrale du problème de tournées de véhicules*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 1995. Disponível em <`http://www.branchandcut.org/VRP/data`>. Acesso em: 24 ago 2014.

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

[5] V. P. Barbosa and R. M. Takakura. Implementação e Comparação de Algoritmos que Resolvam o Problema do Roteamento de Veículos Capacitados, 2014. Bachelor's thesis, Escola de Artes, Ciências e Humanidades, Universidade de São Paulo, São Paulo, Brasil. 44 f.

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

[7] D. Bindel and J. Goodman. *Principles of Scientific Computing*. Manuscript, 2009.

[8] W. d. O. Bussab and P. A. Morettin. *Estatística básica*. Saraiva, São Paulo, 5 edition, 2004.

[9] J. S. Christie, S. Satir, and T. P. Campus. 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, 2006.

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

[11] T. A. Cruse. *Reliability-based mechanical design*, volume 108. CRC Press, 1997.

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

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

[14] M. Fischetti, P. Toth, and D. Vigo. A branch-and-bound algorithm for the capacitated

vehicle routing problem on directed graphs. *Operations Research*, 42(5):846–859, 1994.

[15] J. E. Gentle. *Random number generation and Monte Carlo methods*. Springer Science & Business Media, 2003.

[16] G. M. Giaglis et al. 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, 2004.

[17] G. Gutin and A. P. Punnen, editors. *The traveling salesman problem and its variations*. Combinatorial optimization. Kluwer Academic, Dordrecht, London, 2002.

[18] J. Y. Kanda. 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, 2014.

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

[20] J. Leeuewn. *Handbook of theoretical computer science: Algorithms and complexity*, volume 1. Elsevier, 1990.

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

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

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

[24] C. Z. Mooney. *Monte Carlo Simulation*. Number 116. Sage, 1997.

[25] I. H. Osman and J. P. Kelly. Meta-heuristics: An overview. In *Meta-Heuristics*, pages 1–21. Springer, 1996.

[26] J. Pearl. *Heuristics: Intelligent search strategies for computer problem solving*. Addison-Wesley Pub. Co., Inc., Reading, MA, 1984.

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

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

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

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

[31] I. J. Taneja et al. 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, 1989.

[32] P. Toth and D. Vigo. *The Vehicle Routing Problem*, pages 1–26. 9 edition, 2002.

[33] P. Toth and D. Vigo. The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, 15(4):333–346, 2003.