Universidade Federal do Espírito Santo

Centro Universitário Norte do Espírito Santo

Curso de Bacharelado em Matemática Industrial

Wisley Kenede Pereira de Jesus

UMA ABORDAGEM AO PROBLEMA DE ROTEAMENTO DE VEÍCULOS COM JANELAS DE TEMPO USANDO META-HEURÍSTICAS

São Mateus

2023

Wisley Kenede Pereira de Jesus

UMA ABORDAGEM AO PROBLEMA DE ROTEAMENTO DE VEÍCULOS COM JANELAS DE TEMPO USANDO META-HEURÍSTICAS

Trabalho submetido ao Colegiado do Curso de Bacharelado em Matemática Industrial da UFES (Campus São Mateus), como requisito parcial para a obtenção do grau de Bacharel em Matemática Industrial.

São Mateus

2023

Wisley Kenede Pereira de Jesus

UMA ABORDAGEM AO PROBLEMA DE ROTEAMENTO DE VEÍCULOS COM JANELAS DE TEMPO USANDO META-HEURÍSTICAS

Trabalho submetido ao Colegiado do Curso de Bacharelado em Matemática Industrial da UFES (Campus São Mateus), como requisito parcial para a obtenção do grau de Bacharel em Matemática Industrial.

Aprovada em 08 de Dezembro de 2023.

Comissão Examinadora

Prof. Leonardo Delarmelina Secchin
Universidade Federal do Espírito Santo
Orientador

Prof. Isaac Pinheiro dos Santos
Universidade Federal do Espírito Santo

Prof. Paulo Wander Barbosa
Universidade Federal do Espírito Santo

Resumo

O Problema de Roteamento de Veículos com Janelas de Tempo, em inglês, *Vehicle Routing Problem with Time Windows* (VRPTW) é um tópico de grande interesse na comunidade acadêmica pelo fato de ser um problema bastante presente em situações do mundo real. Por ser um problema NP-Difícil, resolver o VRPTW em um tempo de computação razoável é um desafio. O objetivo central do trabalho é estudar algoritmos meta-heurísticos (Simulated Annealing (SA) e Iterated Local Search (ILS)) para resolver o VRPTW. Utilizamos uma abordagem diferente da literatura para lidar com inviabilidades, comumente tratada por penalização, de modo a explorar o espaço de soluções do problema. Realizamos diversos testes numéricos utilizando as clássicas instâncias de *benchmark* feitas por Solomon. O trabalho é um primeiro contato aos algoritmos aleatórios e trabalhos futuros para melhorar o algoritmo que substitui a penalização apresentado são possíveis.

Palavras-chave: Meta-heurísticas, Problema de Roteamento de Veículos com Janelas de Tempo, Simulated Annealing, Iterated Local Search.

Lista de Figuras

Lista de Tabelas

Sumário

1 Introdução

1.1 Definição do VRPTW

O Problema de Roteamento de Veículos com Janelas de Tempo, em inglês, *Vehicle Routing Problem with Time Windows* (VRPTW), é um problema clássico de otimização em pesquisa operacional e logística. O VRPTW é uma extensão do Problema de Roteamento de Veículos, em inglês, *Vehicle Routing Problem* (VRP), que busca otimizar as rotas de uma frota de veículos para atender a um conjunto de clientes minimizando a distância total percorrida, que por sua vez é a generalização do conhecido Problema do Caixeiro Viajante. Um estudo realizado pela fundação Dom Cabral em 2018 [1] constatou que no Brasil os custos logísticos consomem cerca de 12,37% da receita das empresas, sendo que destes, 63,5% correspondem apenas ao transporte. Diante disso, otimizar rotas de veículos para diminuir gastos com transporte é um tópico de interesse da comunidade científica.

No VRPTW, cada cliente tem uma janela de tempo específica dentro da qual deve ser atendido. A janela de tempo representa o período permitido durante o qual o cliente pode receber entregas ou serviços. O objetivo é encontrar as rotas ótimas para os veículos, garantindo que todos os clientes sejam visitados dentro de suas respectivas janelas de tempo, minimizando a distância total de todas as rotas. Impor restrições de tempo às rotas de entrega é importante quando se trata de produtos perecíveis, onde diminuir os gastos não só gera lucros mas também conserva a integridade da mercadoria. O VRPTW é um problema combinatorial desafiador e é classificado como NP-Difícil, o que significa que encontrar uma solução exata para grandes instâncias pode ser impraticável em prazos razoáveis. Como resultado, vários métodos heurísticos e meta-heurísticos são frequentemente usados para encontrar boas soluções em tempos menores.

O VRPTW tem inúmeras aplicações do mundo real, incluindo serviços de entrega, planejamento de transporte, coleta de lixo e serviços móveis de saúde. Resolver o VRPTW com eficiência pode levar a economia de custos, tempo de viagem reduzido e maior

satisfação do cliente [2].

1.2 Modelo Matemático

O VRPTW é definido por uma frota de veículos homogênea V (todos os veículos são iguais), um conjunto de clientes C e um grafo direcionado completo G (isto é, existe um arco ligando cada par de vértices diferentes). O grafo consiste de $|C| + 2$ vértices (onde $|C|$ é a cardinalidade de C, que por sua vez é o número de elementos do conjunto C), onde os clientes são denotados pelos vértices $\{1, 2, ..., n\}$ e o depósito é representado pelos vértices $\{0\}$ (depósito de partida) e $\{n+1\}$ (depósito de retorno). Essencialmente, $\{0\}$ e $\{n+1\}$ representam o mesmo depósito, mas para uma melhor representação matemática do problema, é comum dividir em dois pontos diferentes no grafo G. O conjunto de todos os vértices é denotado por $N = \{0, 1, ..., n, n+1\}$, e o conjunto de arcos A representa uma conexão direta entre o depósito e os clientes e entre os próprios clientes. Para cada arco (i, j) onde $i \neq j$ existe uma distância c_{ij} de i a j e um tempo de transporte t_{ij}.

Cada veículo tem uma capacidade de carga idêntica q, e cada cliente i tem uma demanda d_i e uma janela de tempo $[a_i, b_i]$. As janelas de tempo para os dois depósitos são consideradas idênticas, ou seja, $[a_0, b_0] = [a_{n+1}, b_{n+1}] = [E, L]$, onde E e L representam a saída mais cedo possível do depósito e o último horário possível de chegada ao depósito, respectivamente. $[E, L]$ é chamado de *horizonte de planejamento*. O modelo possui dois grupos de variáveis de decisão, x e s. Para cada arco (i, j) e cada veículo k definimos x_{ijk} como: $x_{ijk} = 1$ se o veículo k passa pelo arco (i, j) e $x_{ijk} = 0$ caso contrário. A segunda variável s_{ik} é definida para cada vértice i e cada veículo k e representa o momento em que k chega ao cliente i.

O objetivo é projetar um conjunto de rotas que minimiza a distância total, tal que cada cliente é servido uma única vez, cada rota deve começar no vértice 0 e terminar no vértice $n + 1$ e as janelas de tempo dos clientes e as restrições de capacidade dos veículos devem ser respeitadas.

A função objetivo (1.1) minimiza a distância total percorrida por todos os veículos. A restrição (1.2) garante que cada cliente é visitado exatamente uma única vez, e (1.3) afirma que um veículo só pode ser carregado até sua capacidade. As equações (1.4), (1.5) e (1.6) indicam, respectivamente, que cada veículo deve sair do depósito 0; que após um veículo chegar a um cliente ele deve seguir para o próximo destino (conservação de fluxo); e que, finalmente, todos os veículos devem terminar sua rota no depósito $n + 1$. [3]

$$\textbf{Min} \sum_{k\in V}\sum_{i\in N}\sum_{j\in N} c_{ij}x_{ijk} \tag{1.1}$$

$$\textbf{sujeito a} \sum_{k\in V}\sum_{j\in N} x_{ijk} = 1 \quad \forall i \in C, \tag{1.2}$$

$$\sum_{i\in C}\sum_{j\in N} d_i x_{ijk} \leq q \quad \forall k \in V, \tag{1.3}$$

$$\sum_{j\in N} x_{0jk} = 1 \quad \forall k \in V, \tag{1.4}$$

$$\sum_{i\in N} x_{ihk} - \sum_{j\in N} x_{hjk} = 0 \quad \forall h \in C,\ \forall k \in V, \tag{1.5}$$

$$\sum_{i\in N} x_{i,n+1,k} = 1 \quad \forall k \in V, \tag{1.6}$$

$$x_{ijk}(s_{ik} + t_{ij} - s_{jk}) \leq 0 \quad \forall i,j \in N\ (i \neq j),\ \forall k \in V, \tag{1.7}$$

$$a_i \leq s_{ik} \leq b_i \quad \forall i \in N,\ \forall k \in V, \tag{1.8}$$

$$x_{ijk} \in \{0,1\} \quad \forall i,j \in N\ (i \neq j),\ \forall k \in V. \tag{1.9}$$

As inequações (1.7) estabelecem as relações entre a hora de partida dos veículos de um cliente e a hora de chegada no próximo: a hora de chegada do veículo k ao cliente i (s_{ik}) mais o tempo necessário para viajar de i a j (t_{ij}) não pode ser maior que a hora de chegada do veículo k ao cliente j (s_{jk}). Isso é razoável pois em (1.8) garantimos que as janelas de tempo devem ser obedecidas. Finalmente, (1.9) são as restrições de integralidade. Vale notar que os veículos são autorizados a permanecer no depósito, especialmente para os casos onde não temos um número fixo de veículos que devem ser utilizados gerando a necessidade de minimizar essa quantidade juntamente com a distância das rotas. Os veículos não utilizados são modelados passando pelo arco $(0, n+1)$ com $c_{0,n+1}$=$t_{0,n+1}$=0, que por sua vez tem que ser adicionado ao conjunto A. Para cada veículo, a variável s de tempo de chegada impõe um direcionamento e rota únicos, eliminando qualquer *subtour*.

As inequações (1.7) são não lineares, mas elas podem ser linearizadas como:

$$s_{ik} + t_{ij} - s_{jk} \leq M(1 - x_{ijk}) \quad \forall i,j \in N,\ \forall k \in V, \tag{1.10}$$

onde M é uma constante grande. A constante M pode ser qualquer número grande mas um bom valor seria $max\{b_i + t_{ij} - a_j\}$ para todo $(i,j) \in A$.

2 Revisão da Literatura sobre o VRPTW

O alto nível de complexidade do VRPTW e sua vasta utilidade e aplicação em situações da vida real são razões do imenso número de pesquisas no assunto [4]. Esse grande número de aplicações no mundo real em diversas partes do mundo mostra que o uso de procedimentos computacionais para o processo de planejamento de distribuição vem produzindo economias substanciais nos custos globais de transporte. O impacto dessas economias no sistema econômico global é significante, até porque o processo de transporte está envolvido em todos os estágios dos sistemas de produção e distribuição [3].

Diversas heurísticas e algoritmos exatos foram desenvolvidos ao longo dos anos. Em 1959, Dantzig e Ramaser [5] foram os primeiros a introduzirem o problema de despacho de caminhões (posteriormente classificado como o VRP), que busca encontrar um conjunto ideal de rotas de entrega de gasolina para uma frota de caminhões, e também caracterizado como a generalização do Problema do Caixeiro Viajante, em inglês, *Traveling Salesman Problem* (TSP). Em 1964, Clarke e Wright [6] apresentaram um algoritmo para resolver o problema de despacho de caminhões generalizado que essencialmente é o mesmo problema de [5], mas considerando que os caminhões de entrega podem ter capacidades diferentes, ou seja, a frota de veículos não é homogênea. Em 1981, Schrage [7] classificou as características que são encontradas em problemas reais de roteamento de veículos e indicou quais delas causam maior dificuldade e quais abordagens de modelagem são adequadas. Esse estudo motivou outros pesquisadores a estudarem o problema e formas de resolvê-lo. Em 1987, Kolen, Rinnooy Kan e Trienekens [8] propuseram o primeiro método de otimização do VRPTW utilizando geração de colunas, apresentando uma abordagem de decomposição onde o subproblema é construído como uma estrutura de caminho mais curto. Kohl e Madsen em 1997 [9] apresentaram um algoritmo baseado na relaxação Lagrangiana, onde o problema mestre consiste em encontrar os multiplicadores de Lagrange e o subproblema é um Problema de Caminho Mais Curto com janelas de tempo e

restrições de capacidade.

Mesmo sendo importantes, esses métodos exatos clássicos de resolução do VRPTW acabam sendo pouco eficientes para instâncias grandes/complexas. Com isso, diversas abordagens usando meta-heurísticas foram surgindo, por exemplo, Feng [10] em 2011 discutiu uma variante prática do VRPTW que se originou do planejamento de transporte de derivados de petróleo da Corporação Nacional de Petróleo da China. Caminhões-tanque são programados para servir cada posto de gasolina em vários períodos de acordo com uma configuração de janelas de tempo dinâmica. Recarregar no depósito é sempre necessário após visitar um posto, então podemos assumir que os caminhões-tanque não possuem restrições de capacidade. O problema é formulado em um modelo de programação inteira-mista e caracterizado como NP-Hard, e segundo os autores só é resolvido para pequenos casos usando métodos exatos. Além disso, devido as janelas de tempo flutuantes impostas aos clientes, heurísticas tradicionais baseadas em busca local com operadores de troca de clientes não são aplicáveis. Dessa forma foi proposta uma heurística de particionamento de janelas de tempo iterativa que discretizam as janelas de tempo em múltiplos pontos temporais.

Os problemas de *pickup and delivery* (PDPs) constituem uma importante família de problemas de roteamento onde produtos ou passageiros devem ser transportados de diferentes origens para diferentes destinos [3], diferente do VRPTW onde temos apenas uma origem e um destino. Assim como outros problemas de roteamentos, os PDPs também possuem variações. Wang e Lang [11] usaram um algoritmo de *Tabu Search* para resolver uma dessas variações, onde cada cliente tem duas janelas de tempo e a demanda de cada um pode ser atendida se visitado uma ou duas vezes.

Uma variação do VRPTW que merece ser mencionada é aquela em que cada rota é associada a mais de um custo a ser minimizado, conhecida como *Multi-Objective* VRPTW. Baños em [12] explorou a resolução do *Multi-Objective* VRPTW com o objetivo de minimizar a distância percorrida e o desequilíbrio das rotas (esse desequilíbrio é analisado em duas perspectivas: o desequilíbrio na distância percorrida pelos veículos e o desequilíbrio nas cargas entregue por eles). Para resolver esse problema uma meta-heurística baseada no Simulated Annealing (SA) foi desenvolvida, o *Multiple Temperature Pareto Simulated Annealing* (MT-PSA). O SA também foi utilizado em um trabalho de Afifi [13] para resolver o VRPTWSyn (*VRP with time windows and synchronization constraints*), uma variação do VRPTW onde cada cliente poderia precisar de visitas simultâneas de diferentes veículos, usando uma implementação que incorpora várias técnicas de busca local para lidar com esse problema. Esses dois trabalhos ilustram bem o uso com sucesso

do Simulated Annealing para resolver o VRPTW e suas variações já que em ambos os casos os respectivos autores relatam bons resultados, fazendo o SA uma boa escolha de algoritmo.

Em 2012, Xiaobing [14] apresentou um algoritmo de MCPSO (*multi-swarm cooperative particle swarm optimizer*), uma meta-heurística inspirada no fenômeno da simbiose em ecossistemas naturais, para resolver o VRPTW. Para um visão geral da aplicação de heurísticas em problemas de agendamento com janelas de tempo, Ibaraki publicou um artigo no assunto [15], e em 2010, El-Sherbeny [16] fez um apanhado dos métodos utilizados para resolução do VRPTW.

Em 2022, Christian, Kerstin e Philipp [17] propuseram uma meta-heurística para o VRPTW que utiliza o subproblema constituído pela otimização apenas de uma janela de tempo selecionada do VRPTW, considerando que as rotas fora dessa janela de tempo são fixas. Esse problema recebeu o nome de Problema de Roteamento de Veículos com Janela de Tempo Única, em inglês, *Single Time Window Vehicle Routing Problem* (STWVRP). Em 2023, Bruce e Xingyin [4] fizeram um apanhado de artigos focados na resolução do VRP e suas variações de 2005 a 2022, considerando métodos exatos e heurísticas. Eles classificam esses artigos, fazem observações e identificam as maiores tendências na área.

A maioria das aplicações do VRPTW são encontradas no setor logístico, como por exemplo: entrega de produtos ou alimentos, serviços de entrega à bancos (transporte de dinheiro), serviços de correios, coleta de lixo, rotas de ônibus escolar, patrulhas de segurança e problemas de coleta e entrega. No entanto, as aplicações do VRPTW não se limitam a transporte de produtos e serviços gerais, podendo ser aplicado em uma ampla gama de setores industriais e variados contextos, como por exemplo o setor de assistência médica, tais como: escalonamento de médicos em hospitais, roteamento de enfermeiras para cuidados domésticos, transporte de sangue para central de doações e serviços de transporte de amostras biológicas para laboratórios de testes [2]. Além disso, com o aumento dos estudos sobre as mudanças climáticas e a preocupação com o meio ambiente, uma nova variação do VRP surgiu com o nome de *Green Vehicle Routing Problem* (G-VRP) [18]. O objetivo é minimizar emissões de CO_2, poluição sonora e acidentes ambientais.

3 Metodologia

3.1 Meta-heurísticas

Neste trabalho desenvolvemos e implementamos algoritmos meta-heurísticos para resolver o VRPTW. Meta-heurísticas são uma classe de algoritmos de otimização projetados para encontrar boas soluções para problemas complexos em tempo menor comparado com algoritmos exatos. Esses problemas geralmente possuem um grande espaço de soluções onde métodos tradicionais de Pesquisa Operacional são ineficientes ou impraticáveis. Mesmo que as meta-heurísticas sejam uma boa escolha para resolver o VRPTW, vale notar que diferente dos métodos exatos elas não produzem uma solução ótima, mas sim uma aproximação dessa solução ótima.

Meta-heurísticas são técnicas de uso geral que podem ser aplicadas a uma ampla gama de problemas, adaptando-se facilmente à problemas diversos. Elas são iterativas e fazem uso de escolhas aleatórias para explorar o espaço de soluções de forma eficiente. O principal objetivo das meta-heurísticas é encontrar um equilíbrio entre Intensificação: focando em regiões promissoras, concentrando a busca em uma área pequena e confinada; e Diversificação: pesquisando e explorando amplamente o espaço de soluções. As meta-heurísticas são inspiradas em fenômenos naturais ou sociais, como evolução, comportamento de enxames ou processos físicos. Alguns algoritmos meta-heurísticos bem conhecidos incluem:

- **Algoritmos Genéticos (do inglês *Genetic Algorithm* - GA):** Inspirados no processo de seleção natural e genética, os GAs desenvolvem uma população de soluções potenciais usando operadores como seleção, cruzamento e mutação.
- **Otimização de enxame de partículas (do inglês *Particle Swarm Optimization* - PSO):** Este algoritmo simula o comportamento de um enxame de partículas que se movem pelo espaço, atualizando suas posições com base em sua própria experiência .

- ***Simulated Annealing* (SA):** Inspirado no processo de recozimento em metalurgia, o SA começa com uma solução inicial e explora iterativamente o espaço de soluções aceitando soluções piores com uma certa probabilidade decrescente. Próximo do final da execução do algoritmo essa probabilidade é praticamente zero, fazendo o método convergir à uma solução.

- ***Tabu Search* (TS):** O TS mantém uma memória de curto prazo das soluções visitadas anteriormente (lista tabu) para evitar revisitá-las. Permite movimentos que levam a soluções piores temporariamente (desde que não estejam na lista tabu), permitindo escapar de ótimos locais.

- ***Iterated Local Search* (ILS):** ILS é um processo iterativo simples que combina algoritmos de busca local com perturbações aleatórias para explorar o espaço de soluções e encontrar melhores soluções.

Estes são apenas alguns exemplos, e existem inúmeros outros algoritmos meta-heurísticos, cada um com suas próprias características e variações. Para mais informações a respeito do assunto o leitor interessado pode consultar [19].

Neste trabalho iremos focar no uso de Meta-Heurísticas para a solução de problemas de Otimização Combinatória (OC). De acordo com Papadimitriou e Steiglitz [20], em problemas combinatórios procuramos por um objeto em um conjunto finito. Esse objeto é tipicamente um número inteiro, um subconjunto, uma permutação ou uma estrutura de grafo. Exemplos de problemas Combinatórios são o problema do Caixeiro Viajante (TSP), o Problema Quadrático de Alocação (QAP), e problemas de escalonamento e logística [21]. Em particular, o VRPTW é classificado como um problema combinatório, que neste trabalho iremos resolver usando duas Meta-Heurísticas: Simulated Annealing (SA) e Iterated Local Search (ILS). Tais algorítimos já foram utilizados para resolver o VRPTW em outros trabalhos [22, 23].

3.2 Estrutura de vizinhança e Representação de uma solução

Quando se trata de uma solução viável para o VRPTW o que esperamos ter é um número k de rotas para os k veículos que serão utilizados no processo de atendimento aos clientes. Cada rota especifica a ordem de clientes com que o veículo associado deve visitar, ordem essa que obviamente atende as restrições de janela de tempo dos clientes

e capacidade dos veículos. Para encontrar essa solução os métodos exatos de pesquisa operacional são um clássico exemplo de metodologia usada por muito tempo. Essencialmente a forma simplificada de como esses métodos operam é enumerando diversas soluções possíveis e avaliando a melhor delas. Como citamos anteriormente, a quantidade de possíveis soluções do VRPTW é gigantesca, o que faz o uso desses métodos exatos na maioria das vezes inviável para grandes instâncias.

Diante desse fato, as meta-heurísticas foram cada vez mais ganhando popularidade como metodologia escolhida para resolver o VRPTW. O motivo para isso é devido a forma com que esses algoritmos abordam o processo de busca de uma solução. Considere S o conjunto (ou espaço) de todas as soluções possíveis do VRPTW, diferente dos métodos exatos, as meta-heurísticas ao invés de enumerar diversas soluções $s \in S$ e depois avaliar qual é a melhor, elas procuram de forma aleatória novas soluções na vizinhança de s e avaliam sua qualidade a cada iteração, daí vem a classificação de meta-heurísticas serem métodos aleatórios.

Definição 3.1 ([24])**.** *Uma vizinhança de s é uma função $N : S \rightarrow 2^S = \{$subconjuntos de $S\}$ que associa a $s \in S$ um conjunto de vizinhos $N(s) \subseteq S$. $N(s)$ é chamado de vizinhança de s.*

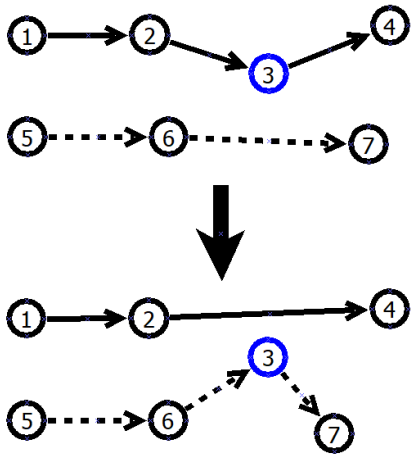
Para explorar essas vizinhanças das soluções usamos Operadores de Aprimoramento que alteram a solução atual de diferentes formas. Neste trabalho implementamos quatro operadores conhecidos da literatura. A funcionalidade desses operadores é basicamente fazer pequenas alterações na solução de forma aleatória com o objetivo de conseguir uma solução diferente e mais barata. Um exemplo básico seria mover um cliente de uma rota para outra. A particularidade aqui é: na maioria das vezes a solução resultante do operador não é viável. A forma com que esse problema é geralmente resolvido na literatura é usando penalização. Neste trabalho optamos por propor uma abordagem diferente para a solução desse problema, implementamos uma particularidade dentro de cada operador na forma de uma Função Exploradora (Algoritmo 1), que funciona da seguinte forma: dado uma rota R e um cliente C de uma rota diferente, percorremos R ordenadamente verificando se a inserção de C (função *Inserir*) gera uma rota viável. Caso afirmativo, inserimos C na primeira posição de R que torna a nova rota viável, gerando a rota *rotaNova*. Caso não seja possível inserir C em posição alguma de R, retornamos a rota R inalterada.

Algoritmo 1 Função Exploradora

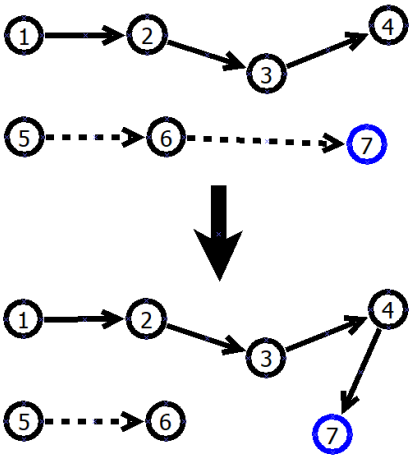
Entrada: R, C
  $novaRota \leftarrow R$
  para cada posição ***k*** em $novaRota$ **faça**
    Inserir(novaRota, k, C)
    se $novaRota$ for viável **então**
      return $novaRota$
    senão
      Remover(novaRota, k)
    fim se
  fim para
  return R

Os operadores utilizados são:

- **O1-*Insert*:** Escolhe-se aleatoriamente duas rotas $rota1$ e $rota2$ e um cliente da $rota1$ para retirá-lo e inseri-lo na $rota2$. A posição que ele vai ocupar na $rota2$ vai depender do resultado que *Optimize Route* retornar [25].



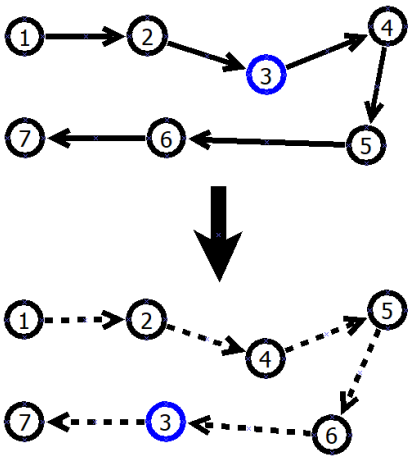
(a) O1-*Insert*: Retiramos um cliente aletório de uma rota aleatória (cliente 3 da rota não pontilhada) e o inserimos em outra rota (rota pontilhada).



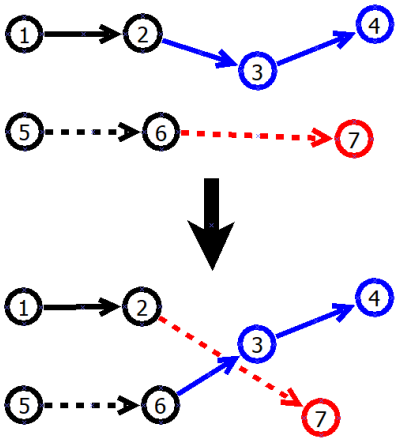
(b) O2-*Insert-RV*: Retiramos um cliente da menor rota (nesse caso cliente 7 da rota pontilhada) e o inserimos em outra rota (rota não pontilhada).

Figura 1: Operadores O1 e O2.

- **O2-*Insert - Redutor de Veículos*:** Semelhante ao operador *Insert*, com a diferença de que o cliente movido sai da rota que possui o menor número de clientes. Essencialmente o objetivo desse operador é diminuir a quantidade total de veículos utilizados [23].



(a) O3-*Intra*: Realocamos o cliente 3 da rota não pontilhada gerando uma nova rota (rota pontilhada).



(b) O4-*Cross Exchange*: Trocamos o pedaço da rota não pontilhada sinalizado pela subrota em azul pelo pedaço da rota pontilhada sinalizado pela subrota em vermelho.

Figura 2: Operadores O3 e O4

- **O3-*Intra Relocate*:** Escolhe-se aleatoriamente uma rota e um cliente dentro dessa rota para troca-lo de posição, utilizando a função *Optimize Route* para verificar todas as posições possíveis e retorna a primeira que for viável [23].
- **O4-*Cross Exchange*:** troca um pedaço de duas rotas diferentes e retornar as novas rotas se ambas forem viáveis, onde tamanho dessas subrotas é escolhido aleatoriamente [25].

Podemos classificar esses operadores que implementamos em duas categorias, oper-

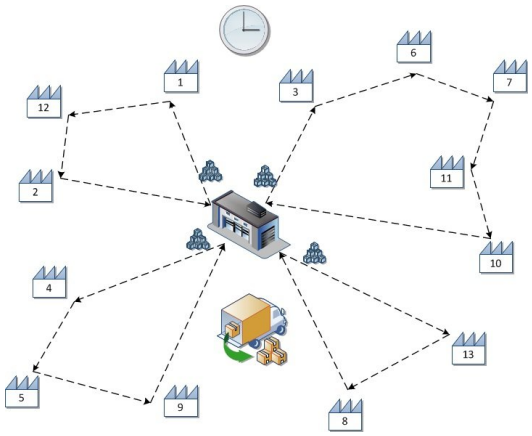


Figura 3: Ilustração de uma solução do VRPTW. [26]

adores que fazem movimentos *externos* e operadores que fazem movimentos *internos*. Movimentos *externos* são aqueles que mudam a rota de um cliente (operadores O1, O2 e O4), e movimentos *internos* são aquele que mudam a posição de um cliente dentro da mesma rota (operador O3).

3.3 Solução inicial

Parte do sucesso de uma meta-heurística é a solução inicial, e em particular para resolver o VRPTW utilizando tanto o SA quanto o ILS precisamos de uma solução inicial viável. Para isso iremos utilizar uma heurística de inserção desenvolvida por Solomon [27]. O Algoritmo 2 apresenta uma descrição simplificada do método. A função *ListaOrganizada()* ordena uma lista de clientes dada filtrando por aqueles que ainda não foram servidos, e os ordena de forma crescente pelo tempo de fechamento da janela de tempo b_i. Como entrada temos uma lista de clientes *cList* e em cada iteração do *loop* principal verificamos se *ListaOrganizada(cList)* não está vazia e caso não esteja, atribuímos à C o resultado. No *loop* interno percorremos a lista C cliente por cliente (denotado por k) para verificar se a rota concatenada $[R\ k]$ é viável, onde R inicialmente é a rota vazia. Caso $[R\ k]$ seja viável, atribuímos ao cliente atual k o *status de servido* e atualizamos R como $[R\ k]$. Ao final do *loop* interno atualizamos a solução S (que originalmente é uma lista vazia) como a lista de rotas concatenadas $[S\ R]$, e por fim retornamos S.

Algoritmo 2 Heurística de Inserção

Entrada: *cList*
  $S \leftarrow$ lista vazia
  enquanto *ListaOrganizada(cList)* não estiver vazia **faça**
    $C \leftarrow$ *ListaOrganizada(cList)*
    $R \leftarrow$ rota vazia
    para cada cliente $\boldsymbol{k}$ em C **faça**
      se $[\boldsymbol{R}\ \boldsymbol{k}]$ for viável **então**
        $k \leftarrow$ *servido*
        $R \leftarrow [R\ k]$
      fim se
    fim para
    $S \leftarrow [S\ R]$
  fim enquanto
  return S

Essencialmente o que essa heurística faz é distribuir o máximo de clientes possíveis para cada veículo levando em consideração as restrições do VRPTW, dessa forma temos

uma solução inicial de boa qualidade.

3.3.1 Exemplo Numérico

Para ilustrar o funcionamento do Algoritmo 2, apresentamos um exemplo numérico simples com 6 clientes e 2 veículos. Na Figura 4 podemos ver os vértices dos clientes e suas respectivas coordenadas.

Figura 4: Coordenadas dos vértices

A Tabela 1 traz as janelas de tempo e demanda dos clientes, onde vértice 0 representa o depósito. Além disso, os veículos são idênticos com capacidade de 20.

Vértice	**Janela de Tempo**	**Demanda**	Tempo de Serviço
0	[0, 20]	0	2
1	[2, 5]	4	2
2	[7, 10]	7	2
3	[12, 15]	6	2
4	[3, 6]	5	2
5	[7, 11]	8	2
6	[13, 16]	7	2

Tabela 1: Dados dos Clientes do Exemplo

Primeiro passo do algoritmo é ordenar os vértices que ainda não foram servidos pelo tempo de fechamento da janela de tempo. Fazendo isso, chegamos à seguinte lista L:

1	4	2	5	3	6

Próximo passo é percorrer essa lista e adicionar vértices um por um no final da rota de um veículo verificando se gera uma rota viável até preencher a capacidade. Sabendo que no começo os veículos possuem uma rota vazia e começando pelo vértice 1, o que queremos é saber se a rota $0 \rightarrow 1 \rightarrow 0$ é viável. Para isso verificamos as janelas de tempo e capacidade atual do veículo. Dado que $d_{a \rightarrow b}$ = "distância de a a b", t_i = "tempo de serviço de i", d_i="demanda de i" e $[a_i,b_i]$ = "janela de tempo de i", para verificar as janelas de tempo temos:

$$d_{0 \rightarrow 1} + t_1 \ \leq \ b_1 \quad \Rightarrow \quad 2,236 + 2 \ \leq \ 5 \quad \Leftrightarrow \quad 4,236 \ \leq \ 5.$$

Da mesma forma fazemos o cálculo para $1 \rightarrow 0$, lembrando que a distância percorrida e o tempo de serviço é acumulativo:

$$(d_{0 \rightarrow 1} + t_1) + d_{1 \rightarrow 0} \ \leq \ b_0 \quad \Rightarrow \quad 4,236 + 2,236 \ \leq \ 20 \quad \Leftrightarrow \quad 6,472 \ \leq \ 20.$$

Isso conclui que $0 \rightarrow 1 \rightarrow 0$ obedece às restrições de tempo, agora precisamos verificar a capacidade. Como a rota do veículo estava vazio a capacidade disponível é 20, subtraindo a demanda do vértice 1 temos 20 - d_1 = 20 - 4 = 16, o que significa que não estamos violando a capacidade. Concluímos então que é viável adicionar o vértice 1 na rota do primeiro veículo. Continuando a percorrer a lista L, próximo vértice a ser verificado é o vértice 4. Da mesma forma, queremos agora saber se a rota $0 \rightarrow 1 \rightarrow 4 \rightarrow 0$ é viável.

$$(d_{0 \rightarrow 1} + t_1) + (d_{1 \rightarrow 4} + t_4) \ \leq \ b_4 \quad \Rightarrow \quad 4,236 + 2,915 + 2 \ \leq \ 6 \quad \Leftrightarrow \quad 9.151 \ > \ 6.$$

Assim, sem precisar verificar a capacidade podemos concluir que adicionar o vértice 4 na rota não é viável, logo podemos ir para o próximo vértice da lista L, o vértice 2:

$$(d_{0 \rightarrow 1} + t_1) + (d_{1 \rightarrow 2} + t_2) \ \leq \ b_2 \quad \Rightarrow \quad 4,236 + 2,061 + 2 \ \leq \ 10 \quad \Leftrightarrow \quad 8,297 \ \leq \ 10,$$

$$(d_{0 \rightarrow 1} + t_1) + (d_{1 \rightarrow 2} + t_2) + d_{2 \rightarrow 0} \ \leq \ b_0 \quad \Rightarrow \quad 8,297 + 2,549 \ \leq \ 20 \quad \Leftrightarrow \quad 10,846 \ \leq \ 20.$$

Logo, a janela de tempo é viável, basta agora verificar a capacidade, que fica 20 - d_1 - d_2 = 20 - 4 - 7 = 9 o que significa que ainda temos capacidade disponível, fazendo a rota $0 \rightarrow 1 \rightarrow 2 \rightarrow 0$ viável. Continuando a percorrer a lista L temos o vértice 5 para verificar,

que fica:

$$(d_{0\to1} + t_1) + (d_{1\to2} + t_2) + (d_{2\to5} + t_5) = 8,297 + 4,5 + 2 = 14,797 \ > \ 11 = b_5.$$

Logo, adicionar o vértice 5 não é viável. O próximo na lista para verificar é o vértice 3, que fica:

$$\begin{aligned}
&(d_{0\to1} + t_1) + (d_{1\to2} + t_2) + (d_{2\to3} + t_3) \ \leq \ b_3 \\
\Rightarrow \quad &8,297 + 2,061 + 2 \ \leq \ 15 \\
\Leftrightarrow \quad &12,358 \ \leq \ 15,
\end{aligned}$$

$$\begin{aligned}
&(d_{0\to1} + t_1) + (d_{1\to2} + t_2) + (d_{2\to3} + t_3) + d_{3\to0} \ \leq \ b_0 \\
\Rightarrow \quad &12,358 + 3,535 \ \leq \ 20 \\
\Leftrightarrow \quad &15,893 \ \leq \ 20.
\end{aligned}$$

As janelas de tempo são viáveis. Para a capacidade fazemos 20 - $d_1 - d_2 - d_3 = 20 - 4 - 7 - 6 = 3$, logo adicionar o vértice 3 na rota é viável. Normalmente nesse ponto o algoritmo continuaria rodando e iria verificar o vértice 6, mas pelo propósito desse exemplo iremos apenas concluir que não seria uma adição viável simplesmente olhando para a capacidade disponível (3) e a demanda do vértice 6 (7).

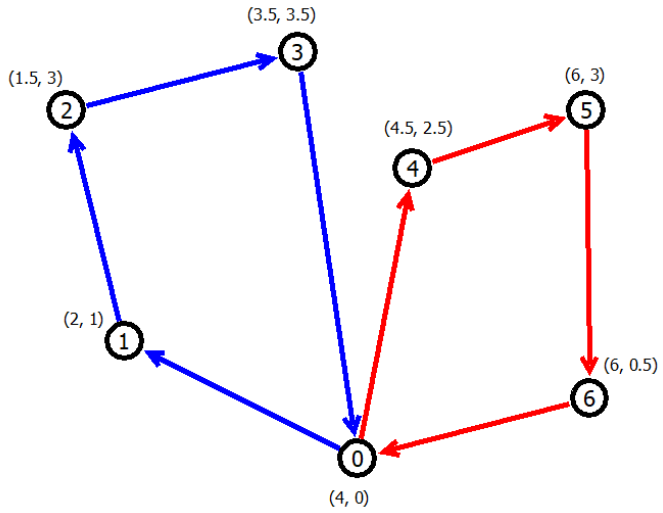


Figura 5: Solução do Exemplo

Após isso restam apenas 3 vértices que ainda não foram servidos: 4, 5 e 6. O processo de verificação é o mesmo e resultará na rota $0 \to 4 \to 5 \to 6 \to 0$, viável. Assim terminamos com a solução para o exemplo como ilustrado na Figura 5.

4 Algoritmos

4.1 Algoritmo SA

A metodologia do *Simulated Annealing* é inspirada pelo processo físico metalúrgico de recozimento (*annealing*), processo esse que altera as características físicas de um material metálico através do calor, onde tais metais são derretidos e resfriados lentamente. O calor permite que os átomos do material se libertem da sua posição inicial, posição essa que corresponde ao mínimo local de uso de energia. Os átomos então começam a perambular aleatoriamente explorando outros possíveis estados de uso de energia, até que no fim do resfriamento espera-se terminar com um material no melhor estado de uso de energia possível (mínimo global).

A analogia do SA à esse processo físico é que esses diferentes níveis de energia correspondem a soluções candidatas à ótima, e a forma com que o SA utiliza para explorar o espaço de soluções é aceitar uma dessas soluções mesmo que seja pior que a atual com uma certa probabilidade P, probabilidade essa que diminui de acordo com que a temperatura diminui (resfriamento). Esse processo é chamado de *up hill move*, isso faz com que no começo do processo é muito comum o SA aceitar uma solução pior, e no final do processo onde P estará bem pequeno as chances de um *up hill move* acontecer é reduzida, fazendo o método estacionar em uma solução.

Algoritmo 3 apresenta a estrutura do SA, onde os parâmetros do algoritmo são: $cList$ representa a lista de todos os clientes; t é a temperatura inicial; $maxIt$ é o número máximo de iterações para o *loop* principal; $maxItFalha$ é o máximo de iterações permitidas sem melhora da solução; t_{min} é o valor mínimo que a temperatura t pode atingir; $maxItInterna$ é o número máximo de iterações do *loop* interno e α é o fator de resfriamento que controla o decrescimento da temperatura t. Os valores para cada um desses parâmetros serão discutidos no próximo capítulo.

O algoritmo começa com o cálculo da solução inicial S dada pelo Algoritmo 2 que vimos anteriormente. O *loop* principal é executado até que um dos três critérios de

parada for atingido, são eles: o número máximo de iterações $maxIt$, o número máximo de iterações sem melhora na solução ou $t = t_{min}$. O *loop* interno é executado até $subiter = maxItInterna$, onde em cada iteração calculamos o custo total da solução atual S (d_1) e encontramos um vizinho de S (S_n) aplicando um dos operadores de aprimoramento que vimos no capítulo anterior através da função *Explorar(S)*, que utiliza as probabilidades 40%, 30%, 20% e 10% respectivamente para os operadores O1, O4, O3 e O2.

Algoritmo 3 Simulated Annealing

Entrada: $cList$, t, $maxIt$, $maxItFalha$, t_{min}, $maxItInterna$, α
  $S \leftarrow$ *Algoritmo2(cList)*
  $S_{best} \leftarrow S$
  $iterFalha \leftarrow 0$
  para $iter = 1 : maxIt$ **faça**
    se $iterFalha > maxItFalha$ **então**
      break
    fim se
    $iterFalha \leftarrow 1$
    se $t \leq t_{min}$ **então**
      break
    fim se
    para $subiter = 1 : maxItInterna$ **faça**
      $d_1 \leftarrow$ *CustoTotal(S)*
      $S_n \leftarrow$ *Explorar(S)*
      $d_2 \leftarrow$ *CustoTotal(S_n)*
      se $d_2 <= d_1$ **então**
        $S \leftarrow S_n$
        se $d_2 <$ *CustoTotal(S_{best})* **então**
          $S_{best} \leftarrow S_n$
          $iterFalha \leftarrow 0$
        fim se
      senão se $d_2 > d_1$ **então**
        $P \leftarrow exp(-((d_2 - d_1))/(d_1 t))$
        $r =$ *Random()*
        se $r < P$ **então**
          $S \leftarrow S_n$
        fim se
      fim se
    fim para
    $t \leftarrow t * \alpha$
  fim para
  return S_{best}
Saída: *Rotas*

Após isso calculamos o custo total de S_n (d_2), e se S_n for uma solução mais barata ou igual a S atualizamos S, e se S_n for mais barata que S_{best} atualizamos S_{best}. Se S_n for um solução pior que S, ela ainda pode ser aceita para tomar o lugar de S de acordo

com a probabilidade P (*up hill move*). Para fazer essa verificação primeiro perceba que o valor de P será sempre menor do que 1 já que estamos calculando a exponencial de um número negativo, logo usando a função *Random()* atribuímos à r um valor entre 0 e 1, e se r for menor que P aceitamos a solução pior S_n como solução atual. Ao final do *loop* interno atualizamos a temperatura t utilizando α e passamos para a próxima iteração do *loop* principal, que termina retornando S_{best}, uma lista de rotas otimizadas.

4.2 Algoritmo ILS

A ideia chave por detrás do ILS consiste em explorar o espaço de soluções por meio de perturbações em ótimos locais, onde esses ótimos locais são encontrados usando um algoritmo de busca local. A perturbação precisa ser suficientemente forte para permitir que a busca local explore diferentes soluções e fraca o suficiente para evitar um reinício aleatório, diversificando a busca [28]. Enquanto que a busca local geralmente é um algoritmo potente com o objetivo de intensificar a busca em uma pequena região. O processo de perturbação e busca local é repetido várias vezes, e a melhor solução encontrada até o momento é mantida como a solução atual. O algoritmo termina quando um critério de parada é atendido. O ILS foi aplicado a uma ampla gama de problemas de otimização, incluindo o VRPTW [22], *scheduling* [29] e *facility location problems* [30]. É um algoritmo simples e eficaz que pode ser facilmente adaptado a diferentes domínios de problemas usando diferentes operadores de busca local e estratégias de perturbação.

O Algoritmo 4 apresenta a estrutura básica do ILS. Os parâmetros para o algoritmo são: a lista de clientes $cList$ e $maxItFalha$, que assim como no SA é o número máximo de iterações permitidas sem melhora da solução. O algoritmo começa gerando uma solução inicial para o problema, que é obtida pelo Algoritmo 2. Em seguida entramos no *loop* principal e em cada iteração aplicamos o algoritmo de perturbação na melhor solução encontrada S_{best}. A solução resultante então passa pelo algoritmo de busca local para encontrar o próximo ótimo local. Se esse ótimo local for melhor que S_{best} atualizamos S_{best} e fazemos $iterFalha = 0$, basicamente reiniciando a execução do ILS a partir de uma solução melhor, caso contrário, incrementamos $iterFalha$. Por fim quando o critério de parada for atingido retornamos S_{best}.

Algoritmo 4 Iterated Local Search

Entrada: $cList$, $maxItFalha$
  $S \leftarrow Algoritmo2(cList)$
  $S_{best} \leftarrow S$
  $iterFalha \leftarrow 0$
  enquanto $iterFalha < maxItFalha$ **faça**
    $S \leftarrow Perturbação(S_{best})$
    $S \leftarrow BuscaLocal(S)$
    se S for melhor que S_{best} **então**
      $S_{best} \leftarrow S$
      $iterFalha \leftarrow 0$
    senão
      $iterFalha \leftarrow iterFalha + 1$
    fim se
  fim enquanto
  return S_{best}
Saída: *Rotas*

4.2.1 Perturbação

A ideia para o algoritmo de perturbação é bem simples, o que basicamente fazemos é aplicar o operador *Insert* que apresentamos anteriormente. Removemos um número aleatório de clientes da solução e os reinserimos aleatoriamente. Algoritmo 5 mostra o pseudocódigo para a perturbação, onde $removeMax$ é um parâmetro que controla o número máximo de clientes que podem ser removidos da solução usando uma pequena porcentagem do número total de clientes. O algoritmo começa escolhendo um número inteiro k aleatório entre 1 e $removeMax$, e então em cada iteração do *loop* principal aplica o operador O1-*Insert* e verifica se a rota resultante é diferente da original, caso verdadeiro diminuímos k, caso contrário passamos para a próxima iteração. Esse processo se repete enquanto $k > 0$

Algoritmo 5 Perturbação

Entrada: S
  $k \leftarrow$ número inteiro aleatório entre 1 e $removeMax$
  enquanto $k > 0$ **faça**
    aplica o operador *Insert*
    se a nova for diferente que a original **então**
      $k \leftarrow k - 1$
    fim se
  fim enquanto
  return solução perturbada

O objetivo da perturbação é alterar a solução o suficiente para sair dos mínimos

locais, mas ao mesmo tempo sem fugir muito da solução atual. Vale notar que essa implementação não permite nenhuma violação das restrições de janelas de tempo ou capacidade, o que significa que no fim sempre teremos uma solução viável.

4.2.2 Busca Local

A ideia da busca local que implementamos para o ILS é de olhar todas as possíveis combinações de clientes de diferentes rotas e procurar trocas de clientes que melhoram a solução. Algoritmo 6 mostra o pseudocódigo para a busco local, onde S é a solução inicial para ser melhorada onde o algoritmo iterativamente aplica uma sequência de movimentos externos para isso. Os movimentos consistem de trocas (*swapps*) entre os clientes c_1 e c_2 com o objetivo de diminuir o custo total da solução. O algoritmo considera todos os pares de clientes de diferentes rotas e usando a nossa Função Exploradora verificamos a viabilidade das novas rotas new_r_1 e new_r_2, se elas forem viáveis e o custo total da solução diminuir, atualizamos as rotas r_1 e r_2 e definimos a variável *improved* para *true*, o que garante mais uma execução do *loop* mais externo. O *loop* continua até nenhuma melhora é encontrada, fazendo o algoritmo terminar e retornar a solução melhorada S_{new}. Assim como a perturbação, essa implementação não permite nenhuma violação das restrições de janelas de tempo.

Algoritmo 6 Busca Local

Entrada: S
  $S_{new} \leftarrow S$
  $improved \leftarrow true$
  enquanto *improved* for *true* **faça**
    $improved \leftarrow false$
    para cada par de rotas diferentes r_1 e r_2 de S_{new} **faça**
      para cada par de clientes c_1 de r_1 e c_2 de r_2 **faça**
        remove c_1 de r_1
        remove c_2 de r_2
        $new\_r_1 \leftarrow FunçãoExploradora(r_1, c_2)$
        $new\_r_2 \leftarrow FunçãoExploradora(r_2, c_1)$
        se new_r_1 e new_r_2 for viável e diminui o custo da solução **então**
          $r_1 \leftarrow new\_r_1$
          $r_2 \leftarrow new\_r_2$
          $improved \leftarrow true$
        fim se
      fim para
    fim para
  fim enquanto
Saída: S_{new}

Como podemos ver, a busca local que implementamos para o ILS ficou razoavelmente forte, principalmente com o fator de *"restart"* que o parâmetro *improved* traz. Isso é importante pois enquanto a perturbação procura trazer diversificação para o método, a busca local traz intensificação concentrando a busca em uma área específica (vizinhança).

5 Resultados Computacionais

As implementações do SA e do ILS foram feita na linguagem Julia e os testes em um computador equipado com processador Intel(R) Xeon(R) Silver 4114 CPU 2.20GHz, 160Gb RAM e GNU/Linux Ubuntu Server 22.04.3 LTS.

5.1 Instâncias Utilizadas

Como o SA e o ILS são métodos aleatórios, cada execução gera uma solução diferente. Então, para termos um resultado justo, executamos os dois algoritmos um certo número de vezes em cada instância, e para esse número escolhemos uma quantidade razoável de 10 execuções. Com os resultados, apresentamos a melhor solução encontrada, a média das 10 execuções e o desvio padrão. A performance dos métodos propostos é avaliada usando as instâncias clássicas desenvolvidas por Solomon [31], que consiste de 100 clientes. Existem seis conjuntos de problemas: R1, R2, C1, C2, RC1 e RC2. Nos conjuntos R1 e R2 as posições dos clientes são criadas randomicamente de forma uniforme. Nos conjuntos C1 e C2 os clientes são posicionados em grupos. Nos conjuntos RC1 e RC2, parte dos clientes são posicionados randomicamente e parte em grupos. Os problemas R1, C1 e RC1 possuem um horizonte de planejamento de curto prazo, curto tempo de serviço e capacidade pequenas para os veículos permitindo apenas alguns clientes em cada rota. Por outro lado, os problemas R2, C2 e RC2 possuem um horizonte de planejamento de longo prazo, tempo de serviços mais longo e veículos com grande capacidade, fazendo com que cada rota seja capaz de acoplar um número maior de clientes.

5.2 Calibração SA

Assim como qualquer outro método heurístico, fixar valores apropriados para os parâmetros do algoritmo é crucial para conseguir bons resultados. Os parâmetros a serem ajustados para o SA são: t (temperatura inicial), $maxIt$ (número máximo de iterações

do *loop* principal), $maxItFalha$ (número máximo de iterações permitidas sem melhora da solução), t_{min} (temperatura mínima), $maxItInterna$ (número máximo de iterações do *loop* interno) e α (fator de redução da temperatura). Em particular, para os parâmetros $maxIt$, $maxItFalha$ e t_{min} atribuímos valores sem realizar calibração; o motivo é de que esses parâmetros não são tão sensíveis em relação aos resultados do algoritmo, a sua principal função é de deixar o algoritmo executar livremente para evitar estacionar em um mínimo local prematuramente. Os valores para esses parâmetros são $maxIt = 1000$, $maxItFalha = 100$ e $t_{min} = 10^{-6}$.

Para os demais parâmetros foram atribuídos 4 possíveis diferentes valores para cada um, $t \in \{0,5; 0,7; 1,2; 2\}$, $\alpha \in \{0,8; 0,9; 0,95; 0,995\}$ e $maxItInterna \in \{3000; 4500; 5500; 6000\}$. A ideia da calibração é de encontrar a melhor combinação desses valores, e para isso não testamos todas as combinações possíveis mas sim combinações razoáveis a partir da combinação inicial $(0,7; 0,9; 5500)$, resultando em cerca de 30 diferentes combinações testadas. Para decidir qual delas é a melhor, escolhemos executar cada uma das combinações 6 vezes nas instâncias R107, C202, RC108 e RC207 e analisamos os resultados, concluindo que a melhor combinação dos parâmetros é $t = 2$, $\alpha = 0,9$ e $maxItInterna = 5500$.

5.3 Calibração ILS

Como o ILS é um método simples, temos poucos parâmetros para ajustar, são eles: $maxItFalha$ e $removeMax$. Para a calibração, assim como o SA, atribuímos 4 possíveis valores para cada um, $maxItFalha \in \{15; 30; 50; 100\}$ e $removeMax \in \{2\%; 5\%; 10\%; 15\%\}$. Como nesse caso temos apenas dois parâmetros para calibrar, para todas as combinações possíveis escolhemos novamente executar 6 vezes o método sobre cada um das instâncias R107, C202, RC108 e RC207. O resultado da calibração revelou os melhores parâmetros sendo $maxItFalha = 30$ e $removeMax = 5\%$.

5.4 Resultados SA

As Tabelas 2, 3 e 4 apresentam os resultados dos testes. As colunas “NV” indicam o núméro de veículos na solução e “Dist” a distância total. Para comparação utilizamos o valor ótimo de cada problema; caso não seja conhecido, comparamos com o melhor resultado encontrado por heurísticas [31]. Em todas as tabelas os valores ótimos, quando conhecidos, são sinalizados com um “*”. As referências onde o melhor valor conhecido é

reportado são citadas na coluna "Ref".

Tabela 2: Melhores Resultados SA

Melhor Resultado				**SA**		
Problema	**NV**	**Dist**	**Ref**	**Dist**	**NV**	**Tempo(s)**
R101	20	1637,7 *	[32]	1653,8	19	1791
R103	14	1208,7 *	[33, 34]	1256,1	15	1788
R105	15	1355,3 *	[32]	1383,5	15	1795
R107	11	1064,6 *	[33, 35]	1121,1	11	1799
R112	9	982,1	[36]	1054,4	11	1796
R201	4	1252,4	[37]	1214,7	5	1868
R202	3	1191,7	[38]	1160,3	5	1864
R205	3	994,4	[38]	1066,1	4	1842
R207	2	893,3	[39]	953,8	4	1894
R211	2	892,7	[39]	950,1	4	1889
C101	10	827,3 *	[32]	828.9	10	1804
C105	10	827,3 *	[32]	828.9	10	1776
C202	3	589,1 *	[33, 35]	637,7	3	1942
C208	3	585,8 *	[35]	665,7	4	1907
RC101	15	1619,8 *	[32]	1658,6	15	1796
RC103	11	1258,1 *	[33, 35]	1349,8	12	1797
RC105	15	1513,7 *	[32]	1566,4	15	1801
RC106	11	1424,7	[40]	1397,9	13	1804
RC108	11	1114,2 *	[41]	1266,9	11	1806
RC201	9	1261,8 *	[35]	1400,4	5	1868
RC204	3	798,4	[42]	1002,1	4	1881
RC205	4	1297,2	[42]	1402,6	5	1863
RC207	3	1067,1	[39]	1190,5	4	1889
RC208	3	828,1	[43]	1009	4	1891

A Tabela 2 apresenta os melhores resultados encontrados pelo SA proposto. Como dito anteriormente, o SA foi executado 10 vezes para cada instância, e a média desses valores pode ser vistos na Tabela 3.

Para avaliarmos os resultados utilizamos o Desvio Padrão em porcentagem apresentado na Tabela 4. O desvio padrão expressa o grau de dispersão de um conjunto de dados numéricos em relação à média. Quanto mais próximo de 0 for o desvio padrão, mais homogêneo são os dados. Para calcular o desvio padrão σ utilizamos a fórmula:

$$\sigma = \frac{\sqrt{\frac{\sum(x_i-\mu)^2}{N}}}{\mu} \times 100, \tag{5.1}$$

onde x_i é o valor da posição i no conjunto de dados, μ é a média aritmética dos dados e N é quantidade total de dados.

A partir dos resultados obtidos podemos ver que de forma geral o SA proposto teve

Tabela 3: Médias dos Resultados SA

Melhor Resultado				**SA Médias**		
Problema	**NV**	**Dist**	**Ref**	**Dist**	**NV**	**Tempo(s)**
R101	20	1637,7 *	[32]	1657,6	19	1789
R103	14	1208,7 *	[33, 34]	1280,5	14	1788
R105	15	1355,3 *	[32]	1407,9	15	1794
R107	11	1064,6 *	[33, 35]	1170,2	12	1797
R112	9	982,1	[36]	1081,8	11	1796
R201	4	1252,4	[37]	1303	5	1869
R202	3	1191,7	[38]	1216,9	4	1889
R205	3	994,4	[38]	1146,6	4	1890
R207	2	893,3	[39]	1028,4	4	1895
R211	2	892,7	[39]	997,4	3,5	1916
C101	10	827,3 *	[32]	828.9	10	1805
C105	10	827,3 *	[32]	867,7	10	1803
C202	3	589,1 *	[33, 35]	654,2	3	1931
C208	3	585,8 *	[35]	724,2	4	1902
RC101	15	1619,8 *	[32]	1685,9	15	1796
RC103	11	1258,1 *	[33, 35]	1386,4	12	1796
RC105	15	1513,7 *	[32]	1598	15	1801
RC106	11	1424,7	[40]	1426,3	13	1803
RC108	11	1114,2 *	[41]	1290,8	11.5	1806
RC201	9	1261,8 *	[35]	1481,1	5	1869
RC204	3	798,4	[42]	1034	4	1887
RC205	4	1297,2	[42]	1434	5	1862
RC207	3	1067,1	[39]	1252,6	4	1889
RC208	3	828,1	[43]	1054,2	4	1891

bons resultados. Não encontramos o valor ótimo para as instâncias mas ficamos próximos, com destaque para R101, R105, C101, C105, RC101 e RC105. A maior dificuldade foi resolver as instâncias dos conjuntos C2, R2 e RC2, onde um possível motivo para isso pode ser a capacidade dos veículos e o horizonte de planejamento dessas instâncias serem maiores em relação às outras. Isso resulta em um número menor de veículos a serem utilizados e consequentemente rotas maiores, o que particularmente é um tipo de problema difícil para o SA proposto resolver já que a maioria dos operadores de aprimoramento implementados fazem movimentos externos, e o interessante aqui são operadores que fazem movimentos internos.

Em particular, nas instâncias R201, R202 e RC106 o SA proposto obteve distâncias menores que o melhor valor conhecido, porém, o número de veículos utilizados é maior. Logo, a seguinte pergunta surge: qual solução é melhor? E a resposta é que não sabemos. Durante nossas pesquisas não encontramos informações sobre um senso comum que diz que a distância ou o número e veículos é o mais importante a se minimizar no VRPTW, isso

Tabela 4: Desvio Padrão SA

Melhor Resultado		**SA**	**SA Médias**	**Desvio Padrão**
Problema	**Dist**	**Dist**	**Dist**	$\sigma(\%)$
R101	1637,7 *	1653,8	1657,6	0,447
R103	1208,7 *	1256,1	1280,5	1,293
R105	1355,3 *	1383,5	1407,9	0,747
R107	1064,6 *	1121,1	1170,2	2,085
R112	982,1	1054,4	1081,8	2,153
R201	1252,4	1214,7	1303	3,594
R202	1191,7	1160,3	1216,9	3,275
R205	994,4	1066,1	1146,6	3,708
R207	893,3	953,8	1028,4	3,512
R211	892,7	950,1	997,4	3,247
C101	827,3 *	828.9	828.9	0
C105	827,3 *	828.9	867,7	4,164
C202	589,1 *	637,7	654,2	4,319
C208	585,8 *	665,7	724,2	3,529
RC101	1619,8 *	1658,6	1685,9	1,383
RC103	1258,1 *	1349,8	1386,4	1,258
RC105	1513,7 *	1566,4	1598	2,327
RC106	1424,7	1397,9	1426,3	1,491
RC108	1114,2 *	1266,9	1290,8	1,214
RC201	1261,8 *	1400,4	1481,1	3,194
RC204	798,4	1002,1	1034	3,684
RC205	1297,2	1402,6	1434	1,591
RC207	1067,1	1190,5	1252,6	3,438
RC208	828,1	1009	1054,2	3,744

vai depender da instância a ser resolvida. Por exemplo, pode existir uma instância em que só precisamos nos preocupar com a distância total, de modo que o número de veículos será minimizado na otimalidade. Conjecturamos que, de fato, ao minimizar distâncias estamos também minimizando o número de veículos, pelo menos se as distâncias satisfizerem a desigualdade triangular e se não houver restrições de capacidades e janelas de tempo. Veja a Figura 6.

Na solução em azul da Figura 6 estamos atribuindo um veículo para cliente, enquanto que na solução em vermelho estamos servindo os 3 clientes usando apenas um veículo. Em questão de distância total podemos ver que a solução em azul obviamente é mais cara que a em vermelho, o que podemos concluir que para esse caso diminuir o número de veículos utilizados reduz a distância total percorrida. Mas a realidade é que em instâncias de problemas reais a conclusão não é tão trivial assim pois as janelas de tempo e capacidade dos veículos influencia muito em como as rotas vão ser construídas. Para as instâncias de teste que utilizamos, ambos distância e número de veículos são objetivos a serem

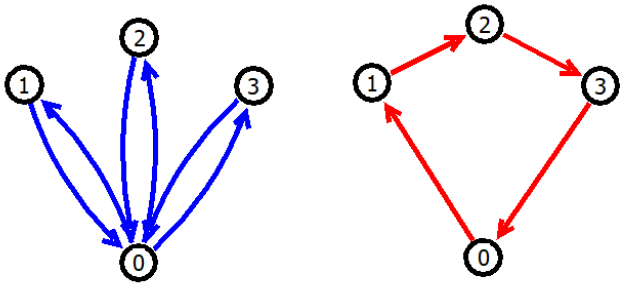


Figura 6: Exemplo da relação entre número de veículos utilizados e distância total.

minimizados, mas o autor não especifica se um é mais importante que o outro.

Ainda em relação ao número de veículos, é interessante notar que os operadores O1 *Insert* e O2 *Insert - Reduce Vehicles* cumpriram o seu objetivo. Na Tabela 2 podemos ver que o número de veículos utilizados nos resultados do SA proposto foi semelhante aos melhores valores conhecidos, até mesmo para os valores médios na Tabela 3, com apenas uma exceção: a instância RC201, onde o valor ótimo diz que 9 veículos foram utilizados e o SA proposto encontrou soluções com 4 ou 5 veículos, mas com distância total maior.

Olhando para o desvio padrão na Tabela 4 podemos ver que a maior porcentagem encontrada é por volta de 4,3%, o que nos diz que o SA proposto é um algoritmo robusto. Em relação ao tempo de execução do SA, observamos um alto custo. Acreditamos que melhorias na implementação, com menor uso de *loops*, melhore o tempo de execução. No entanto, os tempos de execução são parecidos em diferentes instâncias, o que é uma qualidade em relação à métodos exatos onde o tempo cresce (muitas vezes de forma não polinomial) com o aumento dos dados. Isso deve-se à forma como o SA é feito, a temperatura diminui uniformemente e cada vez que isso acontece fazemos 5500 iterações independentemente da instância. Serve como motivação para trabalhos futuros estudar formas otimizadas de implementação do algoritmo.

A Figura 7 mostra a solução gráfica do problema R101, onde podemos perceber rotas com "nós", ou seja, com diferentes arcos (i, j) se cruzando. Isso é um acontecimento comum para esse tipo de problema, principalmente levando em consideração que não temos nenhum operador específico para lidar com esses "nós". Na tentativa de resolver o problema implementamos um algoritmo de busca local *Two-Opt* para ser aplicado na solução resultante do SA. O *Two-Opt* é um algoritmo desenvolvido para o problema do caixeiro viajante que historicamente é conhecido por lidar com tais "nós". Porém, no nosso caso não apresentou efeito, talvez porque nosso algoritmo estacionou em um ótimo local. Na Figura 8 temos a representação gráfica de C101, uma das instâncias em que o

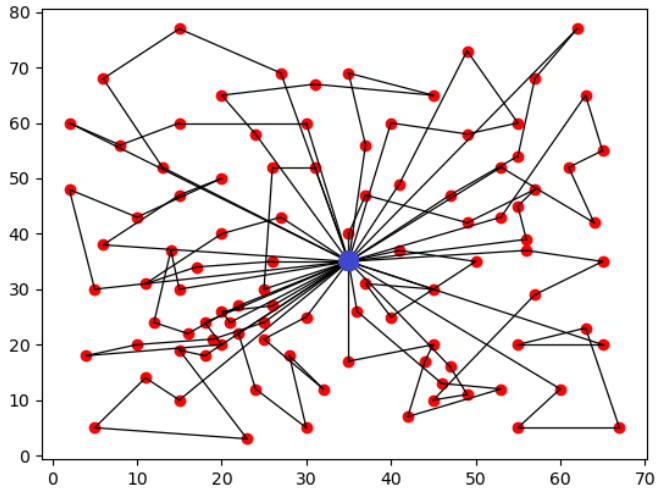


Figura 7: Solução Gráfica R101

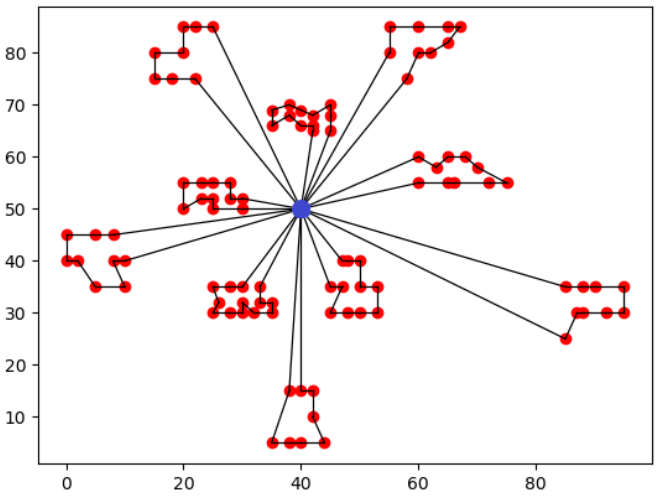


Figura 8: Solução Gráfica C101

SA chegou bem próximo do valor ótimo. Nela podemos ver o quão bem distribuídas estão as rotas.

Na Figura 9 temos a solução gráfica de RC207. Assim como discutimos antes, essa é uma das instâncias onde o SA teve mais dificuldade. Podemos ver que temos um número menor de rotas comparado com a solução da instância R101 e o quão extenso essas rotas são, gerando um grande número de "nós".

A Figura 10 mostra o comportamento clássico do SA através do gráfico de descida (distância total pelo número de iterações) da solução de R101. No começo aceitamos diversas soluções piores para fugirmos de ótimos locais, comportamento esse que diminui com o decorrer da execução do algoritmo devido a redução da probabilidade P explicado

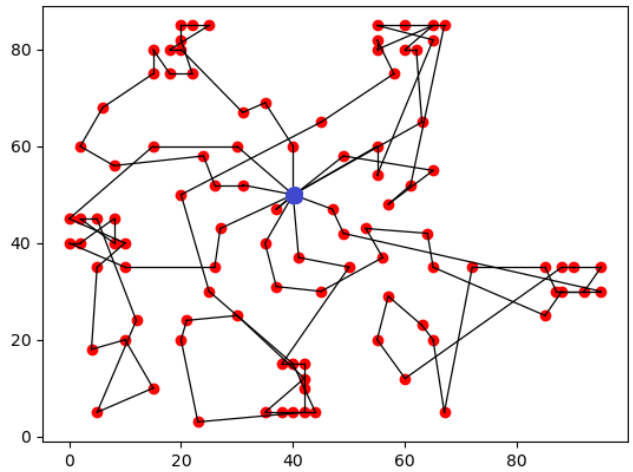


Figura 9: Solução Gráfica RC207

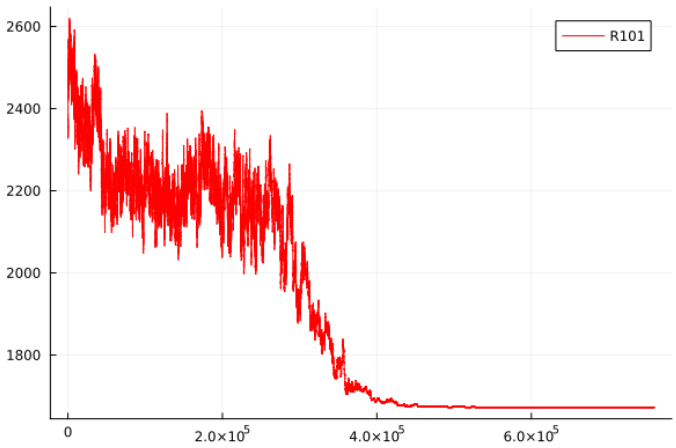


Figura 10: Gráfico de Descida R101 - SA

anteriormente.

5.5 Resultados ILS e Comparações

Para os testes computacionais do ILS utilizamos a mesma metodologia de testes que usamos no SA. Cada instância foi executada 10 vezes. Os melhores resultados encontrados estão na Tabela 5, a média das 10 execuções estão na Tabela 6 e o desvio padrão na Tabela 7.

De forma geral o ILS teve resultados razoáveis. O ILS é um algoritmo simples mas bastante sensível à busca local e perturbação utilizada. No ILS é muito importante calibrar as potências dos algoritmos de busca local e perturbação de forma que explore

bem o espaço de soluções. Serve como motivação para trabalhos futuros testar diferentes algoritmos para busca local e perturbação.

Diferentemente do SA, o ILS obteve piores distâncais que a melhor conhecida em todas as instâncias. Em relação ao tempo de computação, o ILS comportou-se diferente em instâncias distintas devido ao fato de que o algoritmo só é executado até não achar melhores soluções por um tempo, logo, se o algoritmo estaciona rapidamente em uma solução o tempo de execução será menor.

Tabela 5: Melhores Resultados ILS

Melhor Resultado				**ILS**		
Problema	**NV**	**Dist**	**Ref**	**Dist**	**NV**	**Tempo(s)**
R101	20	1637,7 *	[32]	1708,7	20	525
R103	14	1208,7 *	[33, 34]	1392,1	18	986
R105	15	1355,3 *	[32]	1453,4	18	1258
R107	11	1064,6 *	[33, 35]	1252,6	14	962
R112	9	982,1	[36]	1141,8	12	1230
R201	4	1252,4	[37]	1364,2	6	2886
R202	3	1191,7	[38]	1279,6	5	1903
R205	3	994,4	[38]	1221,2	5	1038
R207	2	893,3	[39]	1116,6	6	1579
R211	2	892,7	[39]	1072,8	4	2370
C101	10	827,3 *	[32]	919,1	12	626
C105	10	827,3 *	[32]	875,6	11	1722
C202	3	589,1 *	[33, 35]	856,6	5	2507
C208	3	585,8 *	[35]	764,5	3	2857
RC101	15	1619,8 *	[32]	1759,1	18	533
RC103	11	1258,1 *	[33, 35]	1480,9	15	1100
RC105	15	1513,7 *	[32]	1678,1	18	949
RC106	11	1424,7	[40]	1564,3	16	882
RC108	11	1114,2 *	[41]	1291,2	12	1692
RC201	9	1261,8 *	[35]	1495,9	6	2349
RC204	3	798,4	[42]	1181,4	6	1909
RC205	4	1297,2	[42]	1466,6	8	2485
RC207	3	1067,1	[39]	1345,6	6	2630
RC208	3	828,1	[43]	1176,4	5	3942

A Figura 11 mostra o comportamento típico do ILS. Podemos ver que ele converge de forma muito mais uniforme comparado ao SA. Os picos no gráfico representam as execuções do algoritmo de perturbação que serve para explorar soluções piores, que comparado ao SA, é uma busca menos intensa.

Para o desvio padrão na Tabela 7 podemos ver que o ILS ficou robusto, com apenas algumas instâncias maiores que 5%, e de forma geral, comparando os dois algoritmos

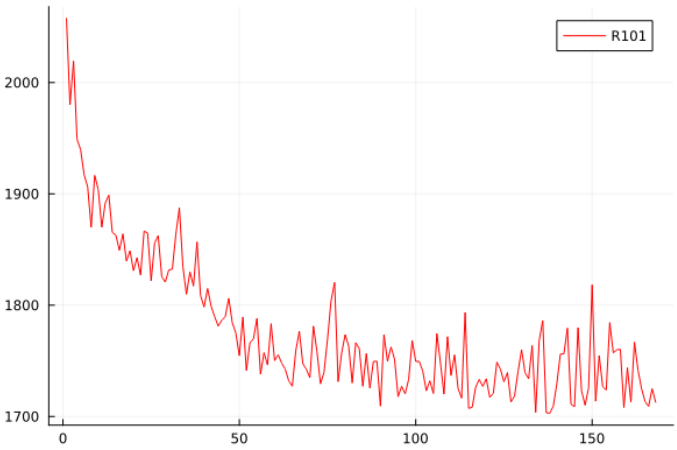


Figura 11: Gráfico de Descida R101 - ILS

podemos ver claramente que o SA teve um desempenho melhor. A principal diferença entre o SA e o ILS está na abordagem com que cada algoritmo explora o espaço de soluções. Perceba que no SA em cada iteração do *loop* interno do método nós realizamos um único movimento na solução procurando apenas um vizinho por vez, repetindo isso 5500 vezes antes de diminuir a temperatura. Já no ILS, em cada iteração a forma com que exploramos a vizinhança é muito mais profunda em uma determinada região do espaço.

Na Tabela 8 fizemos um resumo dos resultados encontrados pelos algoritmos. Nela apresentamos os melhores resultados da literatura de cada instância assim como os valores das médias da distância e do número de veículos do SA e do ILS (colunas M.D. e M.NV.). Além disso apresentamos uma medida em porcentagem da diferença entre as distâncias encontradas pelos algoritmos SA e ILS e os melhores valores da literatura (coluna C (%)).

Assim, podemos ver de forma clara como o SA se comportou melhor para resolver o VRPTW em relação ao ILS, tanto para as distâncias como para o número de veículos.

Tabela 6: Médias dos Resultados ILS

Melhor Resultado				**ILS**		
Problema	**NV**	**Dist**	**Ref**	**Dist**	**NV**	**Tempo(s)**
R101	20	1637,7 *	[32]	1749,3	22	479
R103	14	1208,7 *	[33, 34]	1464,6	19	504
R105	15	1355,3 *	[32]	1499,6	18	704
R107	11	1064,6 *	[33, 35]	1309,4	15,5	941
R112	9	982,1	[36]	1213,7	14	1161
R201	4	1252,4	[37]	1437,2	6	2536
R202	3	1191,7	[38]	1363,0	5	1713
R205	3	994,4	[38]	1260,1	5	1600
R207	2	893,3	[39]	1192,2	6	1634
R211	2	892,7	[39]	1135,1	5	1708
C101	10	827,3 *	[32]	991,1	12	672
C105	10	827,3 *	[32]	924,9	12	1003
C202	3	589,1 *	[33, 35]	957,8	5	2186
C208	3	585,8 *	[35]	843,3	4	1170
RC101	15	1619,8 *	[32]	1784,6	18,5	572
RC103	11	1258,1 *	[33, 35]	1524,0	15,5	914
RC105	15	1513,7 *	[32]	1728,9	19	846
RC106	11	1424,7	[40]	1600,9	17	685
RC108	11	1114,2 *	[41]	1385,1	14	1220
RC201	9	1261,8 *	[35]	1568,6	6	1967
RC204	3	798,4	[42]	1437,0	7	1126
RC205	4	1297,2	[42]	1508,4	7	1724
RC207	3	1067,1	[39]	1458,7	6	1811
RC208	3	828,1	[43]	1299,0	6	1959

Tabela 7: Desvio Padrão ILS

Melhor Resultado		**ILS**	**ILS Médias**	**Desvio Padrão**
Problema	**Dist**	**Dist**	**Dist**	σ
R101	1637,7 *	1708,7	1749,3	1,977
R103	1208,7 *	1392,1	1464,6	1,857
R105	1355,3 *	1453,4	1499,6	1,531
R107	1064,6 *	1252,6	1309,4	2,597
R112	982,1	1141,8	1213,7	4,015
R201	1252,4	1364,2	1437,2	3,413
R202	1191,7	1279,6	1363,0	3,327
R205	994,4	1221,2	1260,1	3,181
R207	893,3	1116,6	1192,2	5,185
R211	892,7	1072,8	1135,1	3,378
C101	827,3 *	919,1	991,1	3,240
C105	827,3 *	875,6	924,9	2,551
C202	589,1 *	856,6	957,8	7,235
C208	585,8 *	764,5	843,3	4,540
RC101	1619,8 *	1759,1	1784,6	1,272
RC103	1258,1 *	1480,9	1524,0	3,438
RC105	1513,7 *	1678,1	1728,9	1,990
RC106	1424,7	1564,3	1600,9	2,096
RC108	1114,2 *	1291,2	1385,1	6,064
RC201	1261,8 *	1495,9	1568,6	3,965
RC204	798,4	1181,4	1437,0	12,434
RC205	1297,2	1466,6	1508,4	2,290
RC207	1067,1	1345,6	1458,7	3,454
RC208	828,1	1176,4	1299,0	4,317

Tabela 8: Tabela de Comparações do SA e ILS com os Melhores Resultados

Melhor Resultado			SA			ILS		
Problema	Dist.	NV	M.D.	M.NV.	C (%)	M.D.	M.NV	C (%)
R101	1637,7 *	20	1657,6	19	1.2	1749,3	22	6.37
R103	1208,7 *	14	1280,5	14	5.60	1464,6	19	17.47
R105	1355,3 *	15	1407,9	15	3.73	1499,6	18	9.62
R107	1064,6 *	11	1170,2	12	9.02	1309,4	15,5	18.69
R112	982,1	9	1081,8	11	9.21	1213,7	14	19.08
R201	1252,4	4	1303	5	3.88	1437,2	6	12.85
R202	1191,7	3	1216,9	4	2.07	1363,0	5	12.56
R205	994,4	3	1146,6	4	13.27	1260,1	5	21.08
R207	893,3	2	1028,4	4	13.13	1192,2	6	25.07
R211	892,7	2	997,4	3,5	10.49	1135,1	5	21.35
C101	827,3 *	10	828.9	10	0.19	991,1	12	16.52
C105	827,3 *	10	867,7	10	4.65	924,9	12	10.55
C202	589,1 *	3	654,2	3	9.95	957,8	5	38.49
C208	585,8 *	3	724,2	4	19.11	843,3	4	30.53
RC101	1619,8 *	15	1685,9	15	3.92	1784,6	18,5	9.23
RC103	1258,1 *	11	1386,4	12	9.25	1524,0	15,5	17.44
RC105	1513,7 *	15	1598	15	5.27	1728,9	19	12.44
RC106	1424,7	11	1426,3	13	0.11	1600,9	17	11.00
RC108	1114,2 *	11	1290,8	11.5	13.68	1385,1	14	19.55
RC201	1261,8 *	9	1481,1	5	14.80	1568,6	6	19.55
RC204	798,4	3	1034	4	22.78	1437,0	7	44.43
RC205	1297,2	4	1434	5	9.53	1508,4	7	14.00
RC207	1067,1	3	1252,6	4	14.80	1458,7	6	26.84
RC208	828,1	3	1054,2	4	21.44	1299,0	6	36.25

Conclusões

Este trabalho abordou a resolução do VRPTW usando as meta-heurísticas Simulated Annealing e Iterated Local Search. Por serem algoritmos bastante amplos e utilizados em diferentes problemas, fizemos adaptações para resolver VRPTW. Diferentemente da literatura, onde durante a execução dos algoritmos penalizamos soluções inviáveis, implementamos os operadores de aprimoramento de forma que sempre geramos soluções viáveis, retirando a necessidade de penalização.

Testes computacionais foram feitos usando as clássicas instâncias de *benchmark* feitas por Solomon [31]. Ambos os algoritmos foram robustos. O SA teve uma vantagem em relação ao ILS no setindo dos resultados encontrados serem mais próximos ao melhores valores conhecidos das instâncias.

Como trabalho futuro refazer a *Função Exploradora* que substitui a penalização é possível, por exemplo, ao invés de percorrer as rotas inserindo clientes de forma ordenada procurando por soluções viáveis, podemos inserir os clientes de forma aleatória trazendo mais aleatoriedade para o método.

Referências Bibliográficas

1 CABRAL, F. D. *Custo Logístico tem um Aumento de Cerce de 15,5 Bilhões da Receita das Empresas Entre 2015 E 2017*. 2018. Disponível em: `https://www.fdc.org.br/conhecimento-site/nucleos-de-pesquisa-site/centro-de-referencia-site/Materiais/Custos_Logisticos_2018.pdf`.

2 ERDOĞAN, G. An open source spreadsheet solver for vehicle routing problems. *Computers &amp Operations Research*, Elsevier BV, v. 84, p. 62–72, aug 2017.

3 TOTH, P.; VIGO, D. (Ed.). *The Vehicle Routing Problem*. [S.l.]: Society for Industrial and Applied Mathematics, 2002.

4 GOLDEN, B.; WANG, X.; WASIL, E. The evolution of the vehicle routing problem—a survey of VRP research and practice from 2005 to 2022. In: *The Evolution of the Vehicle Routing Problem*. [S.l.]: Springer Nature Switzerland, 2023. p. 1–64.

5 DANTZIG, G. B.; RAMSER, J. H. The truck dispatching problem. *Management Science*, Institute for Operations Research and the Management Sciences (INFORMS), v. 6, n. 1, p. 80–91, oct 1959.

6 CLARKE, G.; WRIGHT, J. W. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, Institute for Operations Research and the Management Sciences (INFORMS), v. 12, n. 4, p. 568–581, aug 1964.

7 SCHRAGE, L. Formulation and structure of more complex/realistic routing and scheduling problems. *Networks*, Wiley, v. 11, n. 2, p. 229–232, 1981.

8 KOLEN, A. W. J.; KAN, A. H. G. R.; TRIENEKENS, H. W. J. M. Vehicle routing with time windows. *Operations Research*, Institute for Operations Research and the Management Sciences (INFORMS), v. 35, n. 2, p. 266–273, apr 1987.

9 KOHL, N.; MADSEN, O. B. G. An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. *Operations Research*, Institute for Operations Research and the Management Sciences (INFORMS), v. 45, n. 3, p. 395–406, jun 1997.

10 ZHEN, F. Multi-period vehicle routing problem with recurring dynamic time windows. In: *ICSSSM11*. [S.l.]: IEEE, 2011.

11 WANG, Y.; MAOXIANG. Study on the model and tabu search algorithm for delivery and pickup vehicle routing problem with time windows. In: *2008 IEEE International Conference on Service Operations and Logistics, and Informatics*. [S.l.]: IEEE, 2008.

12 BAÑOS, R. et al. A simulated annealing-based parallel multi-objective approach to vehicle routing problems with time windows. *Expert Systems with Applications*, Elsevier BV, v. 40, n. 5, p. 1696–1707, apr 2013.

13 AFIFI, S.; DANG, D.-C.; MOUKRIM, A. A simulated annealing algorithm for the vehicle routing problem with time windows and synchronization constraints. In: *Lecture Notes in Computer Science*. [S.l.]: Springer Berlin Heidelberg, 2013. p. 259–265.

14 GAN, X. et al. Vehicle routing problem with time windows and simultaneous delivery and pick-up service based on MCPSO. *Mathematical Problems in Engineering*, Hindawi Limited, v. 2012, p. 1–11, 2012.

15 IBARAKI, T. et al. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, Institute for Operations Research and the Management Sciences (INFORMS), v. 39, n. 2, p. 206–232, may 2005.

16 EL-SHERBENY, N. A. Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University - Science*, v. 22, p. 123–131, 2010. Disponível em: <https://api.semanticscholar.org/CorpusID:120630112>.

17 TRUDEN, C.; MAIER, K.; ARMBRUST, P. Decomposition of the vehicle routing problem with time windows on the time dimension. *Transportation Research Procedia*, Elsevier BV, v. 62, p. 131–138, 2022.

18 SABET, S.; FAROOQ, B. Green vehicle routing problem: State of the art and future directions. arXiv, 2022.

19 GLOVER, F.; KOCHENBERGER, G. A. (Ed.). *Handbook of Metaheuristics*. [S.l.]: Springer US, 2003.

20 RABINER, L. Combinatorial optimization:algorithms and complexity. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Institute of Electrical and Electronics Engineers (IEEE), v. 32, n. 6, p. 1258–1259, dec 1984.

21 BLUM, C.; ROLI, A. Metaheuristics in combinatorial optimization. *ACM Computing Surveys*, Association for Computing Machinery (ACM), v. 35, n. 3, p. 268–308, sep 2003.

22 TANER, F.; GALIĆ, A.; CARIĆ, T. Solving practical vehicle routing problem with time windows using metaheuristic algorithms. *PROMET - Traffic&ampTransportation*, Faculty of Transport and Traffic Sciences, v. 24, n. 4, p. 343–351, jan 1970.

23 MAHMUDY, W. Improved simulated annealing for optimization of vehicle routing problem with time windows (vrptw). *Kursor*, v. 7, p. 109–116, 01 2014.

24 ROLI, A. *Meta-heurístias*. Disponível em: `https://www.researchgate.net/publication/228746637_An_introduction_to_Metaheuristics`.

25 KILBY, P.; PROSSER, P.; SHAW, P. Guided local search for the vehicle routing problem with time windows. In: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. [S.l.]: Springer US, 1999. p. 473–486.

26 DEMIR, E. *A graphical representation of the VRPTW*. Disponível em: `https://www.researchgate.net/figure/A-graphical-representation-of-the-VRPTW_fig1_273112998`.

27 SOLOMON, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Oper. Res.*, v. 35, p. 254–265, 1987. Disponível em: <https://api.semanticscholar.org/CorpusID:15346313>.

28 SOUZA, P. H. V. P. M. J. F. *Iterated Local Search (ILS)*. 2021. Disponível em: `http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/ILS.pdf`.

29 XIE, F.; POTTS, C. N.; BEKTAŞ, T. Iterated local search for workforce scheduling and routing problems. *Journal of Heuristics*, Springer Science and Business Media LLC, v. 23, n. 6, p. 471–500, jul 2017.

30 SEYEDKOLAEI, A. A.; NASSERI, S. H. Facilities location in the supply chain network using an iterated local search algorithm. *Fuzzy Information and Engineering*, Tsinghua University Press, v. 15, n. 1, p. 14–25, mar 2023.

31 SOLOMON, M. *VRPTW Benchmark Problems*. Disponível em: `http://web.cba.neu.edu/~msolomon/problems.htm`.

32 KOHL, N. et al. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, Institute for Operations Research and the Management Sciences (INFORMS), v. 33, n. 1, p. 101–116, feb 1999.

33 COOK, W.; RICH, J. A parallel cutting-plane algorithm for the vehicle routing problem with time windows. 09 2001.

34 LARSEN, J. Parallelization of the vehicle routing problem with time windows. 07 2004.

35 KALLEHAUGE, B.; LARSEN, J.; MADSEN, O. Lagrangean duality applied on vehicle routing with time windows. *Computers Operations Research*, v. 33, p. 1464–1487, 01 2001.

36 GAMBARDELLA, L. M.; TAILLARD, R.; AGAZZI, G. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. 08 2001.

37 HOMBERGER, J.; GEHRING, H. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR: Information Systems and Operational Research*, Informa UK Limited, v. 37, n. 3, p. 297–318, aug 1999.

38 ROUSSEAU, L.-M.; GENDREAU, M.; PESANT, G. *Journal of Heuristics*, Springer Science and Business Media LLC, v. 8, n. 1, p. 43–58, 2002.

39 BENT, R.; HENTENRYCK, P. V. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, Institute for Operations Research and the Management Sciences (INFORMS), v. 38, n. 4, p. 515–530, nov 2004.

40 BERGER, J.; BARKAOUI, M. A parallel hybrid genetic algorithm for the vehicle routing problem with time windows. *Computers &amp Operations Research*, Elsevier BV, v. 31, n. 12, p. 2037–2053, oct 2004.

41 IRNICH, S.; VILLENEUVE, D. The shortest-path problem with resource constraints and ik/i-cycle elimination for ik/i $\geq$ 3. *INFORMS Journal on Computing*, Institute for Operations Research and the Management Sciences (INFORMS), v. 18, n. 3, p. 391–406, aug 2006.

42 MESTER, D.; BRäYSY, O. Active-guided evolution strategies for large-scale capacitated vehicle routing problems. *Computers &amp Operations Research*, Elsevier BV, v. 34, n. 10, p. 2964–2975, oct 2007.

43 IBARAKI, T. et al. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, v. 39, p. 206–232, 05 2005.