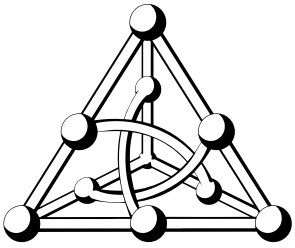
Metaheurísticas para o Problema de Roteamento de Veículos: Um Estudo Comparativo entre ILS e GRASP

Isaac Kosloski Oliveira

Trabalho de Conclusão de Curso
apresentado à Faculdade de Computação da Universidade Federal de Mato Grosso do Sul, como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação.



Orientadora: Prof. Bianca de Almeida Dantas

Campo Grande,
Dezembro de 2025

Resumo

O presente trabalho avaliou empiricamente as meta-heurísticas Busca Local Iterada (*ILS*) e Procedimento de Busca Adaptativa Gulosa Aleatória (*GRASP*) para resolver o Problema de Roteamento de Veículos Capacitados (*CVRP*). O *CVRP* é um problema *NP*-difícil crucial para a logística, visando a minimização do custo total das rotas, respeitando a capacidade dos veículos. A implementação modular utilizou dois operadores de busca local, como *2-Opt* e *Swap**, nas duas meta-heurísticas propostas. Os resultados comparativos em 100 instâncias do benchmark X revelaram que o *ILS* obteve soluções de custo superior em 98% das instâncias e foi, em média, $13,5$ vezes mais rápido que o *GRASP*. A diferença de desempenho foi atribuída à estratégia de intensificação do *ILS* com a perturbação *Double-Bridge*, contra o custo computacional da reconstrução gulosa aleatória do *GRASP* a cada iteração.

Palavras-chave: *CVRP*, *ILS*, *GRASP*, *Otimização Combinatória*

Abstract

This work empirically evaluated the Iterated Local Search (*ILS*) and Greedy Randomized Adaptive Search Procedure (*GRASP*) metaheuristics for solving the Capacitated Vehicle Routing Problem (*CVRP*). *CVRP* is an *NP*-hard problem critical for logistics, aiming to minimize the total cost of routes while respecting vehicle capacities. The modular implementation used two local search operators, such as 2-Opt and Swap*, in the two proposed metaheuristics. Comparative results on 100 instances of the X benchmark showed that *ILS* achieved superior cost solutions in 98% of the instances and was, on average, 13.5 times faster than *GRASP*. The performance difference was attributed to the *ILS* intensification strategy with the *Double-Bridge* perturbation, contrasting with the computational cost of *GRASP*'s randomized greedy reconstruction at each iteration.

Keywords: *CVRP*, *Metaheuristics*, *ILS*, *GRASP*, *Combinatorial Optimization*

Sumário

Capítulo 1

Introdução

O Problema de Roteamento de Veículos (VRP, do inglês *Vehicle Routing Problem*), introduzido por Dantzig e Ramser em 1959 [1], é uma das extensões mais importantes do Problema do Caixeiro Viajante (TSP, do inglês *Travelling Salesman Problem*) na área de otimização combinatória. Assim como o TSP, o VRP se classifica como um problema NP-difícil, mas ele generaliza a questão ao envolver uma frota de veículos baseada em um depósito central. O objetivo primário é determinar o conjunto de rotas de custo mínimo (distância, tempo ou consumo de combustível) que atendam a um grupo de clientes, garantindo que cada cliente seja visitado exatamente uma vez e que todas as rotas comecem e terminem no depósito.

A relevância prática do VRP é inegável, dado o seu papel central na logística e distribuição em escala global. As aplicações variam desde a coleta de lixo e serviços de manutenção até a entrega de mercadorias em comércio eletrônico e a distribuição de insumos em cadeias de suprimentos. Por exemplo, no Problema de Roteamento de Veículos Capacitados (CVRP, do inglês *Capacitated Vehicle Routing Problem*), a restrição de capacidade de carga de cada veículo é um fator crítico, forçando a divisão dos clientes em rotas que não excedam o limite físico dos veículos [2]. Resolver eficientemente o VRP, e suas numerosas variantes (como VRP com janelas de tempo, *time windows*), tem um impacto direto na redução de custos operacionais e na melhoria da pegada de carbono das empresas de transporte e distribuição.

Com o advento da teoria da NP-completude, algoritmos de aproximação e meta-heurísticas tornaram-se cruciais. Essa proeminência decorre da necessidade de gerar soluções quase ótimas para problemas de otimização NP-difíceis (*NP-hard*) e de contornar a intratabilidade computacional inerente. Para alguns problemas, é possível gerar soluções de alta qualidade rapidamente; para outros, no entanto, a dificuldade em gerar soluções subótimas provavelmente boas é comparável à dificuldade de encontrar soluções ótimas. Algoritmos de aproximação e meta-heurísticas foram desenvolvidos para resolver uma vasta gama de problemas, incluindo o Problema do Caixeiro Viajante [3] e Roteamento de Veículos. O valor desses algoritmos reside no seu tempo de execução, que geralmente possui complexidade de tempo polinomial (ainda que de alta ordem ou com constantes elevadas), garantindo viabilidade computacional para

aplicações no mundo real [4].

Ao projetar meta-heurísticas, uma abordagem simples em conceito e aplicação é frequentemente preferível. A Busca Local Iterada (ILS, do inglês *Iterated Local Search*), pode ser introduzida sob essa perspectiva. Seu procedimento central é a heurística incorporada *LocalSearch*, que, dada uma estrutura de vizinhança, fornece uma solução ótima local. Para alcançar soluções de maior qualidade, é possível iterar o procedimento *LocalSearch* e aplicar uma ligeira perturbação na solução. Essa modificação direciona a heurística a buscar novas soluções em uma bacia atração (i.e., uma região da vizinhança que quando otimizada, recai sempre na mesma solução) diferente no espaço de busca [5].

Uma outra abordagem, com similar simplicidade conceitual, é o Procedimento de Busca Adaptativa Gulosa Aleatória (GRASP, do inglês *Greedy Randomized Adaptive Search Procedures*). No GRASP, a *LocalSearch* também é utilizada, mas a diferença crucial reside na construção da solução de entrada. A *LocalSearch* funciona iterativamente, refinando a solução atual. A solução que a *LocalSearch* recebe é gerada por um módulo *GreedyRandomizedConstruction*, que constrói uma Lista Restrita de Candidatos (RCL, do inglês *Restricted Candidate List*) com base na seleção dos elementos com menor custo incremental. A solução é construída selecionando-se um elemento aleatório dessa RCL. A aplicação sucessiva dessa construção gulosa aleatória fornece uma solução inicial de qualidade, que é então otimizada pela *LocalSearch*. Ambos os procedimentos podem ser iterados para gerar soluções cada vez melhores [6].

O objetivo deste trabalho é avaliar de forma comparativa e testar as meta-heurísticas Busca Local Iterada e Procedimento de Busca Adaptativa Gulosa Aleatória na resolução do Problema de Roteamento de Veículos Capacitados, a fim de determinar a abordagem mais eficiente em termos de qualidade de solução e viabilidade computacional.

A Justificativa fundamental deste trabalho reside na urgência em mitigar o desafio da intratabilidade computacional inerente ao Problema de Roteamento de Veículos Capacitados. O CVRP é uma variante do VRP classicamente classificada como NP-difícil [1, 2], o que torna inviável a obtenção de soluções ótimas para instâncias de grande porte em tempo aceitável. Consequentemente, a pesquisa e aplicação de métodos heurísticos e meta-heurísticos são essenciais para gerar soluções de alta qualidade em tempo viável [4]. Além da necessidade teórica, a relevância prática é inegável: o setor de logística e distribuição global exige continuamente maior eficiência operacional e redução de custos. Adicionalmente, a otimização de rotas contribui diretamente para a sustentabilidade, minimizando o consumo de combustível e, consequentemente, a pegada de carbono das operações de transporte [7].

O restante deste trabalho está organizado em quatro capítulos adicionais, que detalham os fundamentos e o desenvolvimento do estudo. O Capítulo 2 apresenta o referencial teórico, cobrindo o formalismo do Problema de Roteamento de Veículos Capacitados, a classificação de problemas NP-difíceis e a função das meta-heurísticas de trajetória única. O Capítulo 3 detalha a metodologia, descrevendo a arquitetura de software modular, a implementação específica dos algoritmos ILS e GRASP, e os operadores de vizinhança utilizados. O Capítulo 4 é dedicado à apresentação e discus-

são dos resultados dos experimentos, com a análise comparativa de desempenho e a avaliação da qualidade das soluções. Por fim, o Capítulo 5 apresenta a conclusão do trabalho, sintetizando as contribuições e delineando propostas para futuras pesquisas.

Capítulo 2

Fundamentação Teórica

Ao longo deste capítulo, apresenta-se a fundamentação teórica do Problema de Roteamento de Veículos Capacitados, da Busca Local Iterada e do procedimento de Busca Adaptativa Gulosa Aleatória.

2.1 Problema de Roteamento de Veículos Capacitados

O Problema de Roteamento de Veículos Capacitados é um problema bem conhecido e amplamente estudado na otimização combinatória, sendo um membro notável da classe de problemas NP-difíceis. Fortemente relacionado ao Problema do Caixeiro Viajante, o CVRP é modelado com o intuito de obter rotas de entrega ótimas, nas quais cada veículo percorre uma única rota, com início e fim em um único depósito central. O objetivo principal é encontrar um conjunto de rotas em que cada cliente seja visitado exatamente uma vez por um único veículo, e a capacidade de cada veículo não seja excedida [8].

Com base nisso, uma definição mais formal para o CVRP pode ser apresentada como um problema em grafos. Seja $G = (V, A)$ um grafo, em que $V = \{1, \ldots, n\}$ é um conjunto de vértices representando clientes, com o depósito localizado no vértice 1, e A é um conjunto de arcos. Cada arco (i, j), com $i \neq j$, é associado a uma matriz de distâncias não negativas $C = (c_{ij})$. Em alguns contextos, c_{ij} pode ser interpretado como um custo ou tempo de viagem. Quando C é simétrico, é conveniente considerar A como um conjunto E de arestas não direcionadas (i.e., $c_{ij} = c_{ji}$). Além disso, existe um número m de veículos disponíveis no depósito, limitado pelo intervalo $m_L \leq m \leq m_U$. Se $m_L = m_U$, o número m é considerado fixo; se $m_L = 1$ e $m_U = n - 1$, m é considerado livre. Os veículos são assumidos como tendo a mesma capacidade, embora existam variações que consideram uma frota heterogênea.

O CVRP consiste em formar um conjunto de rotas de custo mínimo, de modo que: (i) cada cliente em $V \setminus \{1\}$ seja visitado exatamente uma única vez por um único veículo; (ii) Todas as rotas iniciem e finalizem no depósito; (iii) uma demanda (custo), não negativa, d_i, esteja relacionada a cada cliente $i > 1$, e a soma das demandas de cada rota

não exceda a capacidade do veículo [2]. A Figura 2.1 ilustra uma instância do CVRP, com 9 clientes e suas respectivas demandas d_i (representados por nós circulares) e um depósito (nó quadrado), totalizando 3 rotas. Cada rota é percorrida por um veículo, que ao atingir seu limite de capacidade, retorna ao depósito e então, um novo veículo inicia a próxima rota, partindo do depósito.

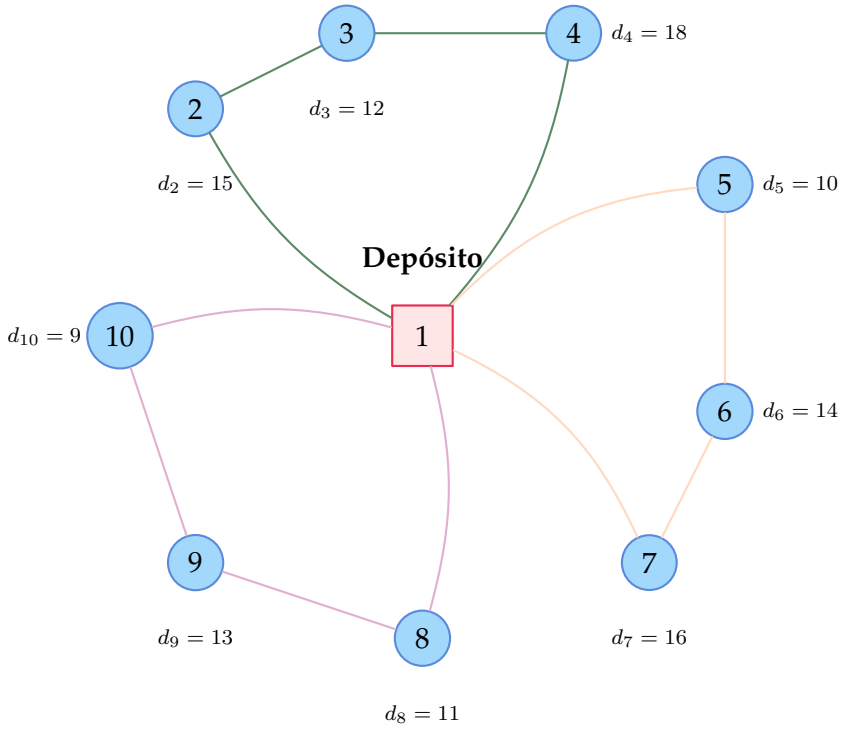


Figura 2.1: Uma instância para o problema do roteamento de veículos capacitados

2.1.1 Modelagem Matemática

Uma possível modelagem para o problema:

$$\min \sum_{v=1}^{m} \sum_{(i,j)} c_{ij}^{v} x_{ij}^{v} \tag{2.1}$$

Sujeito a:

$$\sum_{v=1}^{m} \sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ij}^{m} + \sum_{v=1}^{m} \sum_{\substack{k=0 \\ k \neq j}}^{n} x_{jk}^{v} = 2, \forall j \in \mathcal{N} - \tag{2.2a}$$

$$\sum_{j=1}^{n} x_{0j}^{v} = 1 \tag{2.2b}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ij}^{v} \leq 1, \forall j \in \mathcal{N} \tag{2.2c}$$

$$\sum_{i=1}^{n} x_{i0}^{v} = 1, \forall v \in \mathcal{V} \tag{2.2d}$$

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} x_{ij}^{v} - \sum_{\substack{k=0 \\ k \neq j}}^{n} x_{jk}^{v} = 0, \forall j \in \mathcal{N} \tag{2.2e}$$

$$\sum_{i \in \mathcal{S}} \sum_{\substack{j \in \mathcal{S} \\ j \neq i}} x_{ij}^{v} \leq |\mathcal{S}| - 1, \forall \mathcal{S} \subseteq \mathcal{N}, |\mathcal{S}| \geq 2, \forall v \in \mathcal{V} \tag{2.2f}$$

$$\sum_{(i,j)} a_{ij}^{v} x_{ij}^{v} \leq b_{v}, \forall v \in \mathcal{V} \tag{2.2g}$$

$$x_{ij}^{v} \in 0, 1, \forall (i,j) \in E, \forall v \in \mathcal{V} \tag{2.2h}$$

Onde:

- $\mathcal{N}$: Conjunto de clientes, $\mathcal{N} = \{1, 2, ..., n\}$;
- $\mathcal{V}$: Conjunto total de veículos, $V = \{1, 2, ..., m\}$;
- E: Conjunto de total de arestas, $E = \{(i,j) | i \neq j, i, j \in \{0, 1, ..., n\}\}$;
- c_{ij}^{v}: Custo de deslocamento do veículo v de i para j;
- x_{ij}^{v}: Variável binária que indica se o veículo v percorre a aresta (i,j);
- a_{ij}^{v}: Quantidade de recurso consumido pelo veículo v ao percorrer (i,j);
- b_{v}: Capacidade máxima do veículo v.
- $\mathcal{S} \subseteq \mathcal{N}$: Subconjunto de clientes usado na restrição de sub-rota.
- 0: Representa o depósito.

Esta modelagem possui como função objetivo a minimização do custo total das rotas (2.1). As restrições de (2.2a) a (2.2h) devem ser atendidas. A restrição (2.2a) assegura o balanço de fluxo para cada nó de cliente j, no qual a soma total das arestas que chegam a j e das arestas que saem de j deve ser igual a 2. Isso garante que cada cliente j seja visitado e subsequentemente deixado por um único veículo, confirmando que todos os clientes em $\mathcal{N}$ são atendidos. A restrição (2.2b) garante que cada veículo parta do depósito apenas uma única vez. A restrição (2.2c) garante que cada cliente seja visitado por no máximo um único veículo. A restrição (2.2d) garante que cada veículo finalize a rota no depósito. A restrição (2.2e) garante o balanço de fluxo em cada nó,

assegurando que um veículo que chega a um nó também saia dele (ou vice-versa), enquanto a (2.2f) inibe a criação de sub-rotas (ou *subtours*) entre subconjuntos de clientes. A restrição (2.2g) mantém a capacidade máxima de cada veículo respeitada, enquanto a restrição final (2.2h) define a variável de decisão x_{ij}^v como binária, indicando o uso ou não da aresta (i, j) pelo veículo v [9].

2.2 Busca Local Iterada

A Busca Local Iterada, é uma meta-heurística sofisticada projetada para resolver problemas de otimização. Ela opera iterativamente, gerando e aprimorando soluções através de uma busca local, comparando os resultados obtidos para convergir a uma solução final de alta qualidade.

A concepção do ILS parte da adaptação de um algoritmo heurístico de otimização simples, a Busca Local, para um problema específico. Este procedimento de Busca Local pode ser implementado como a rotina *LocalSearch*. Surge, então, a questão de se é possível otimizar um resultado de busca local iterativamente; se a resposta for afirmativa, a otimização obtida tende a ser significativa. Para que a Busca Local funcione, assume-se que o procedimento *LocalSearch* seja determinístico e sem memória.

Seja C a função de custo de um problema de otimização, cujo objetivo é minimizar o seu valor. Seja S o conjunto finito de todas as soluções s. Na Figura 2.2, um diagrama de fluxo de alta abstração da Busca Local é ilustrado. Dada uma solução de entrada s, o procedimento sempre gera uma mesma solução de saída s^* com um custo menor ou igual a $C(s)$. O procedimento *LocalSearch* define, portanto, um mapeamento $n : 1$ do conjunto S para um subconjunto menor S^* de soluções ótimas locais s^*. A característica principal de uma Busca Local é a estrutura de vizinhança; a partir dela, S pode ser entendido como uma estrutura topológica, e não apenas um conjunto, o que permite transitar de uma solução para outra [10].

Para compreender as limitações da otimização, é útil o conceito de bacia de atração (ou *basin of attraction*) de um mínimo local s^*. Essa bacia é o conjunto de todas as soluções s que são mapeadas para s^* sob a rotina de Busca Local. O procedimento *LocalSearch* recebe uma solução s e fornece, invariavelmente, um mínimo local s^* dentro da bacia de atração, independentemente do número de iterações realizadas.

Para explorar o espaço S de soluções fora da bacia de atração local, é necessário aplicar uma mudança, ou Perturbação (*Perturbation*), que leva a um estado intermediário s'. A partir de s', o procedimento *LocalSearch* é novamente aplicado, fornecendo uma nova solução de mínimo local $s^{*'}$.

O próximo passo é estabelecer um critério que decida qual das duas soluções de mínimo local, s^* e $s^{*'}$, deve ser a solução aceita. O procedimento Critério de Aceitação (*AcceptanceCriterion*) implementa essa tarefa, escolhendo a melhor solução para o próximo passo. Este critério pode ser definido de acordo com o problema e a solução desejada, priorizando, por exemplo, a solução de menor custo ou a que promova maior

diversidade de busca.

Determinada esta estrutura base, o diagrama de fluxo na Figura 2.3 fornece uma visão de como o ILS opera.

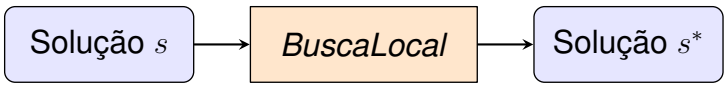


Figura 2.2: Diagrama de fluxo do procedimento de busca local.

A partir do diagrama de fluxo da Figura 2.3 , compreende-se como o ILS pode ser modularizado e como sua construção pode ser implementada, o que leva à definição do Algoritmo 1 proposto por [11].

Algoritmo 1 Iterated Local Search

$s_0 \leftarrow$ GenerateInitialSolution()
$s^* \leftarrow$ LocalSearch(s_0)
REPITA
  $s' \leftarrow$ Perturbation($s^*, history$)
  $s^{*'} \leftarrow$LocalSearch(s')
  $s^* \leftarrow$ AcceptanceCriterion($s^*, s^{*'}, history$)
ATÉ termination condition met

O procedimento *GenerateInitialSolution*() é executado apenas uma vez, fornecendo a solução inicial s_0, que serve de entrada para o procedimento *LocalSearch*(). Em seguida, uma *Perturbation*() é aplicada em s^*, gerando uma nova solução s' em uma bacia de atração diferente no espaço S, a qual é a nova entrada para uma subsequente chamada de *LocalSearch*(). O procedimento *AcceptanceCriterion*() define qual solução é a melhor candidata para a próxima iteração. Para obter uma estrutura de retroalimentação a cada ciclo, a solução aceita é definida como s^*. O algoritmo repete os passos de perturbação, busca e aceitação até que uma condição de término predefinida seja atendida.

2.3 Procedimento de Busca Adaptativa Gulosa Aleatória

O Procedimento de Busca Adaptativa Gulosa Aleatória é uma meta-heurística projetada para resolver problemas de otimização combinatória. O GRASP opera iterativamente em duas fases principais de multi-início: a construção e a busca local. A primeira fase constrói uma solução viável, e a segunda investiga a vizinhança dessa solução. A melhor solução obtida ao longo das iterações (determinada pelas condições desejadas) é escolhida como resultado final.

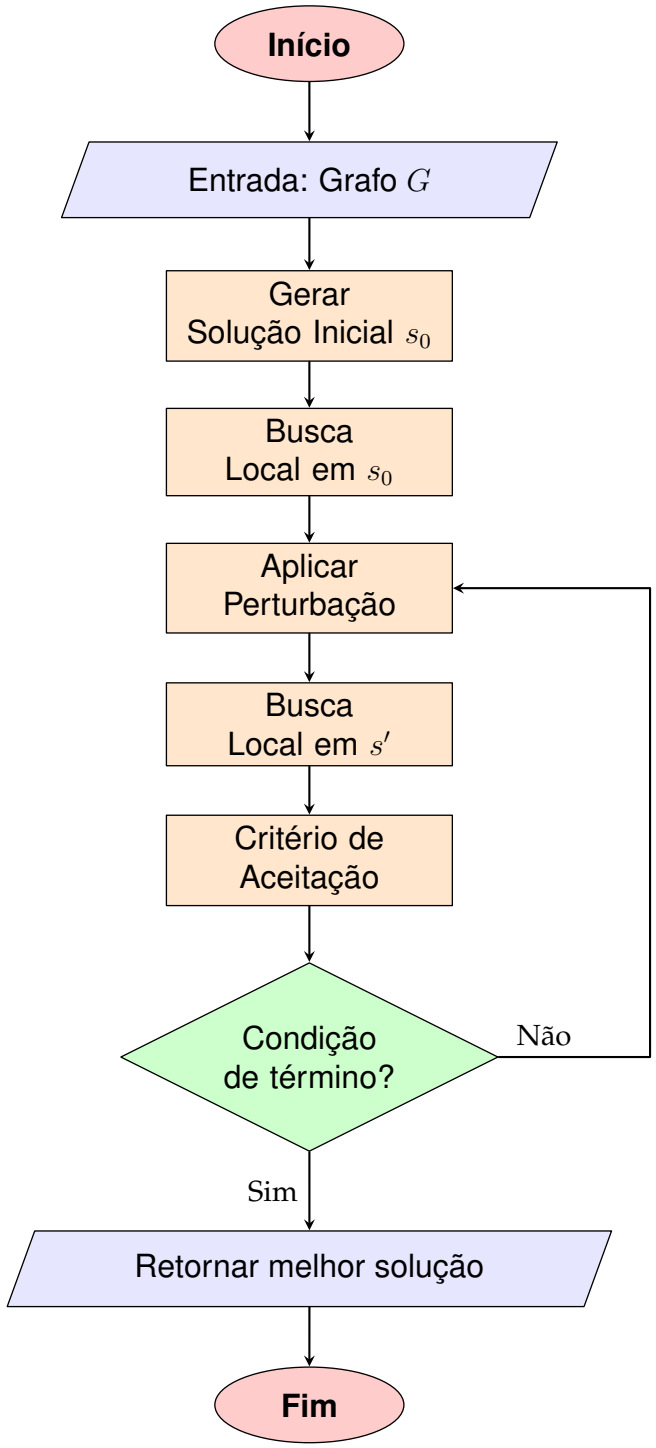


Figura 2.3: Diagrama de fluxo do procedimento ILS.

2.3.1 Fase de Construção Gulosa Aleatória

O procedimento *GreedyRandomizedConstruction* implementa a fase de construção. Dada uma solução inicial s_0, o procedimento avalia um conjunto de elementos candidatos, que são todos os elementos com potencial para serem incorporados em uma solução parcial s'. Essa avaliação é feita utilizando-se um procedimento guloso, geralmente influenciado pelo custo incremental resultante da incorporação de cada elemento candidato na solução parcial.

Para balancear a exploração e a explotação, é utilizada uma Lista Restrita de Candidatos (RCL, do inglês *Restricted Candidate List*). A RCL é formada pelos melhores elementos, ou seja, aqueles cuja incorporação na solução parcial produz o menor custo incremental.

- A seleção desses "melhores elementos" garante a faceta gulosa do algoritmo.
- A viabilidade de um elemento ser definitivamente incorporado na solução parcial é decidida por um procedimento que seleciona aleatoriamente um elemento da RCL, o que garante a faceta probabilística (ou Aleatória) do algoritmo.

Uma vez que o elemento é definido como o próximo a ser adicionado à solução parcial, a RCL é atualizada, e os custos incrementais são reavaliados. Esse processo garante a faceta adaptativa do algoritmo.

Essas ideias definem o Algoritmo 2, de construção gulosa aleatório proposto por [12].

Algoritmo 2 Greedy Randomized Construction

```
s ← ∅
Evaluate the incremental costs of the candidate elements
REPITA
    Build the restricted candidate list (RCL)
    Select an element s_r from the RCL at random
    s ← s ∪ s_r
    Reevaluate the incremental costs
ATÉ Solution is not complete
RETORNE s
```

O procedimento inicializa com uma solução vazia (s). Em seguida, cada elemento é avaliado para compor a RCL. Iterativamente, a RCL é construída, um elemento é escolhido aleatoriamente da RCL e incorporado à solução parcial s. Subsequentemente, cada elemento remanescente é reavaliado para compor a nova RCL. Esses passos iteram até que a solução parcial s esteja completa e viável.

2.3.2 Fase de Busca Local

As soluções geradas pela *GreedyRandomizedConstruction* não são necessariamente ótimas. Como passo subsequente, é possível melhorar a solução atual. Visto que a solução completa possui uma estrutura de vizinhança, um procedimento de busca local pode ser aplicado para o refinamento. Esta segunda fase deve funcionar exatamente como o procedimento previamente implementado e descrito na Subseção 2.2.

A ideia geral do GRASP, combinando essas duas fases, é ilustrada na Figura 2.4.

Assim, define-se o algoritmo completo do Procedimento de Busca Adaptativa Gulosa Aleatória. O Algoritmo 3 fornece uma arquitetura de alto nível proposta por [12].

Algoritmo 3 Greedy Randomized Adaptive Search Procedures

```
s_0 ← GenerateInitialSolution()
s ← s_0
REPITA
    s' ← GreedyRandomizedConstruction(s)
    s* ← LocalSearch(s')
    s ← UpdateSolution(s*, s)
ATÉ termination condition met
```

O procedimento *GenerateInitialSolution*() é executado uma única vez, fornecendo a solução inicial s_0. O procedimento *GreedyRandomizedConstruction*() recebe esta solução como entrada para a primeira fase, que fornece uma solução completa s'. A solução s' é a nova entrada para *LocalSearch*(). O procedimento *UpdateSolution*() define qual solução obtida é a melhor até o momento. Para iterar o processo de busca, a solução aceita é definida como s, e o algoritmo repete os passos de construção, busca local e atualização até que uma condição de término predefinida seja atendida.

Uma vez que o Problema de Roteamento de Veículos Capacitados e as meta-heurísticas, Busca Local Iterada e Procedimento de Busca Adaptativa Gulosa Aleatória, foram formalmente definidos no referencial teórico, o foco do trabalho passa para a metodologia de implementação. Esta transição é crucial, pois a eficiência e a validade dos resultados dependem diretamente da forma como os conceitos teóricos são traduzidos em algoritmos computacionais. Assim, é fundamental estabelecer as decisões de design da arquitetura de software, especificar as estruturas de dados para a representação da solução e do problema, e detalhar os operadores de vizinhança utilizados, como o *Swap**, antes de apresentar o código e os experimentos. O próximo capítulo, portanto, dedica-se a este rigor metodológico, que serve de ponte entre a fundação teórica e a análise prática dos resultados.

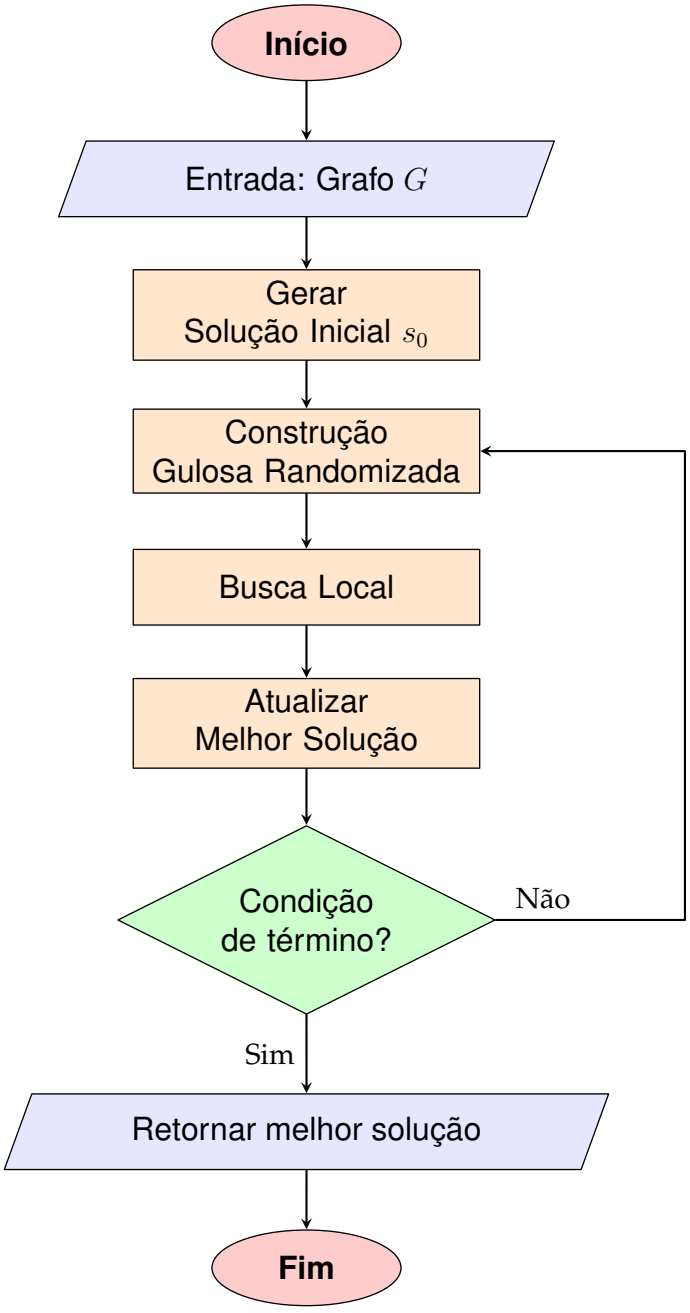


Figura 2.4: Diagrama de fluxo do procedimento GRASP.

Capítulo 3

Implementação

Este capítulo detalha as estratégias e os métodos empregados na implementação dos algoritmos propostos, visando a solução do Problema de Roteamento de Veículos Capacitados. Adotou-se uma abordagem modular com foco no reaproveitamento de código. Para facilitar a comparação entre as meta-heurísticas, foram utilizadas, sempre que possível, as mesmas estruturas de dados, heurísticas e funções de avaliação. Apesar das diferenças inerentes entre as meta-heurísticas avaliadas, as características e restrições do CVRP foram integralmente preservadas na modelagem.

Em busca de um equilíbrio entre abstração e desempenho, os algoritmos foram implementados de forma puramente sequencial, utilizando C++20 e seguindo o paradigma de Programação Orientada a Objetos. O *framework* geral das implementações é dividido em três fases distintas: (i) Inicialização: leitura das instâncias, criação das estruturas de dados e instanciação de objetos; (ii) Processamento: execução e implementação dos algoritmos heurísticos; e (iii) Saída e Análise: obtenção das soluções, análise de dados e apresentação em dashboard. A etapa (iii) é parcialmente integrada ao código principal e parcialmente implementada em Python 3, que é responsável pela persistência dos dados em SQLite, análise estatística e funcionalidades de back-end. Para a visualização e apresentação interativa (*front-end*) dos resultados, foi utilizada a biblioteca Streamlit.

3.1 Representações e estruturas

Com base na definição e modelagem apresentadas na Seção 2.1, foi definida uma estrutura de classes, conforme ilustrado no Diagrama de Classes (Figura 3.1). O projeto de cada classe buscou a otimização de recursos e o encapsulamento do problema. As classes principais são:

- **Component**: Classe base que encapsula os elementos da instância e é utilizada para o cálculo da matriz de adjacências (distâncias/custos).
- **Node**: Define um nó do problema, representando o depósito ou os clientes.

- **Vehicle**: Define um veículo, contendo atributos como capacidade máxima e demanda atual.
- **Scanner**: Responsável pela leitura dos arquivos de instância (entrada de dados) e pela alocação inicial da memória.
- **CVRP**: Implementa a lógica e as restrições gerais do problema, mantendo a relação com a classe *Scanner*.
- **Solution**: Representa uma solução viável (conjunto de rotas). É utilizada tanto para cálculos internos (Fase (ii)) quanto para a extração de dados brutos (Fase (iii)).
- **Solver**: Classe central que implementa os módulos solucionadores. A maior parte da Fase (ii) do framework é executada através dos métodos desta classe.

3.2 Solver

Uma vez definida a estrutura de dados principal, os módulos implementados na classe *Solver* merecem atenção. Os métodos desta classe são responsáveis por implementar os componentes tanto do ILS quanto do GRASP, aproveitando a similaridade estrutural entre as duas meta-heurísticas.

3.2.1 initialSolution_Greedy()

O método *initialSolution_Greedy*() tem por objetivo gerar uma solução inicial viável que sirva de ponto de partida para o ILS e o GRASP. Essa solução, mesmo que não seja ótima, é crucial para fornecer uma entrada válida aos demais módulos, permitindo o início do processo iterativo. O módulo constrói a solução utilizando uma heurística gulosa simples:

1. Inicia-se um veículo e uma rota a partir do depósito (nó 0).
2. Iterativamente, o algoritmo adiciona o cliente não visitado mais próximo à rota, respeitando a capacidade máxima do veículo.
3. Quando a inclusão do próximo cliente excede a capacidade do veículo atual, a rota é fechada (retornando ao depósito), e um novo veículo é instanciado para iniciar uma nova rota.

Esta abordagem tende a fornecer uma solução de qualidade localmente boa, servindo como uma base eficiente para ambas as meta-heurísticas.

A estratégia de construção se baseia nos seguintes passos:

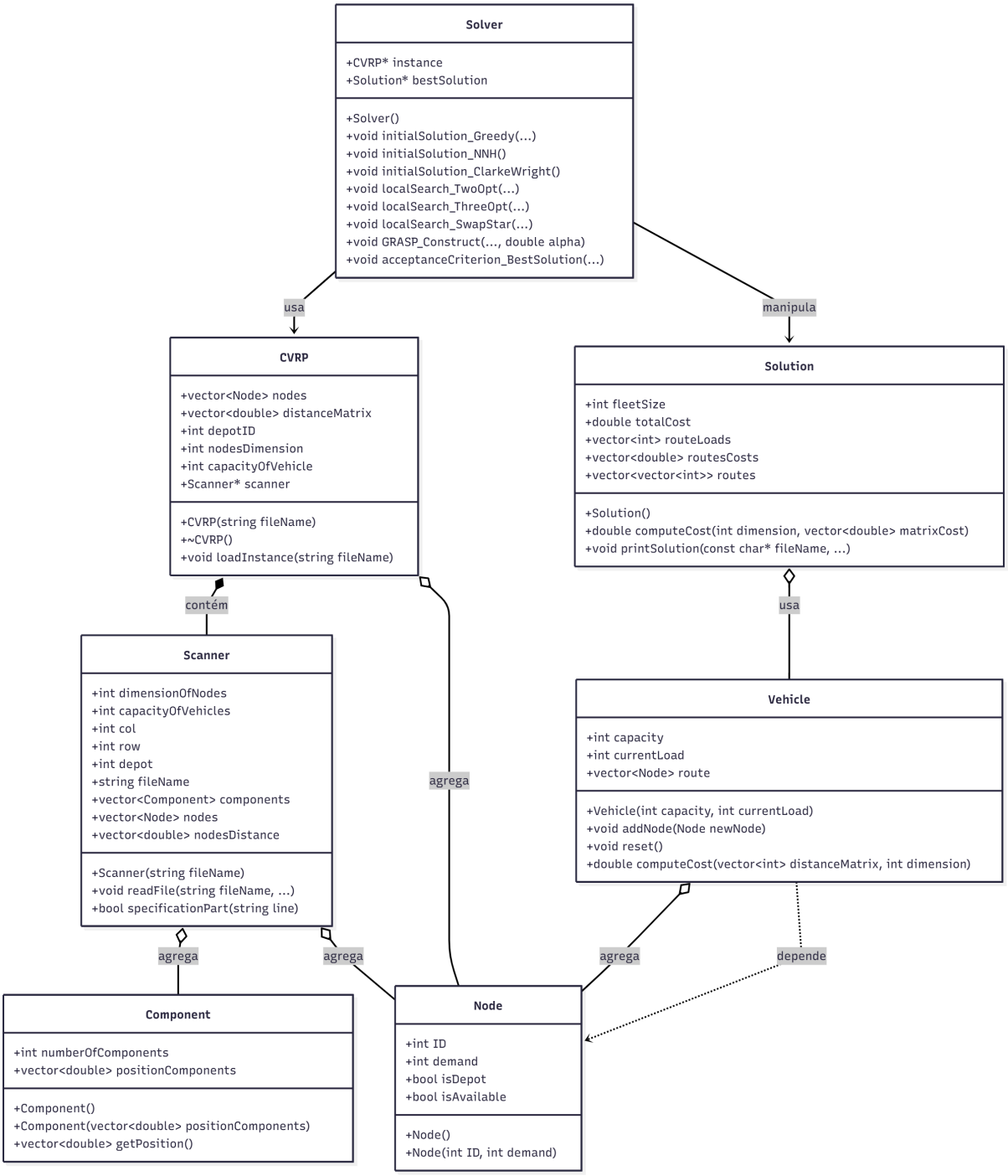


Figura 3.1: Diagrama de Classes para a implementação do CVRP

1. Marcação: Todos os clientes são inicialmente marcados como não visitados.
2. Início da Rota: Uma rota é iniciada a partir do depósito (nó 0).
3. Seleção Gulosa: O cliente não visitado mais próximo do nó atual é selecionado.
4. Validação: A viabilidade de adicionar o cliente à rota é validada, verificando-se a restrição de capacidade do veículo.
5. Fechamento e Novo Veículo: Se a adição for inviável, a rota atual é fechada (adicionando o depósito como último nó) e uma nova rota é iniciada com um novo veículo.
6. Término: O processo é repetido até que todos os clientes tenham sido visitados, garantindo uma solução inicial viável.

3.2.2 localSearch_TwoOpt()

O *localSearch_TwoOpt*() implementa o conhecido operador de Busca Local 2-Opt, projetado para refinar rotas individuais. A técnica consiste em remover duas arestas de uma rota e reconectá-las em ordem invertida, verificando se a nova rota resultante apresenta um custo total inferior. O processo é iterativo e é aplicado até que nenhuma troca de par de arestas resulte em melhoria no custo, convergindo a um ótimo local (*First or Best Improvement*, dependendo da implementação).

A estratégia de otimização 2-Opt se baseia no seguinte *framework* (assumindo a versão First ou Best Improvement):

1. A solução inicial é copiada para ser a melhor solução corrente (s_{best}).
2. É iniciada uma busca iterativa sobre cada rota na solução:
   - Avaliam-se todos os pares de segmentos a serem invertidos dentro da rota (definidos pelas arestas (i, j) e (k, l)).
   - Calcula-se a variação de custo (ΔC) gerada pela inversão do segmento.
   - Se $\Delta C < 0$ (ou seja, se o novo custo for estritamente menor), o movimento é aplicado e a solução corrente é atualizada.
3. O processo é repetido até que uma iteração completa não resulte em nenhuma melhoria de custo. [13]

3.2.3 localSearch_SwapStar()

O *localSearch_SwapStar*() é um operador de Busca Local mais sofisticado que o 2-Opt, pois permite a movimentação de sequências de clientes (*chains*) entre rotas distintas (*inter-route*) ou dentro da mesma rota (*intra-route*). Ao contrário do 2-Opt, o *Swap**

lida com sequências de clientes de tamanho pré-definido (*chain_length*, tipicamente entre 1 e 3). Essa capacidade de redistribuir a carga em um espaço de busca maior permite um melhor equilíbrio entre exploração e tempo de execução.

A estratégia de otimização do *Swap** é baseada na busca sistemática de movimentos inter-rotas (para $r1 \neq r2$) e intra-rotas (para $r1 = r2$):

1. Iteração de Rotas: O procedimento itera sobre cada par de rotas $(r1, r2)$ na solução.
2. Avaliação de Movimento: Para cada cadeia de tamanho *chain_length* selecionada em $r1$, e para cada posição de inserção em $r2$:
   - Viabilidade: É verificada a viabilidade de capacidade em $r1$ e $r2$ após o potencial movimento.
   - Custo: É calculada a variação de custo (ΔC) associada à troca.
   - Aceitação: Se $\Delta C < 0$, o movimento é aplicado à solução.
3. Atualização: As demandas (cargas) das rotas são atualizadas após a execução de movimentos inter-rotas ($r1 \neq r2$).
4. Término: O processo é repetido até que uma iteração completa não encontre movimentos que resultem em melhoria.

3.2.4 acceptanceCriterion_BestSolution()

O método *acceptanceCriterion_BestSolution*() implementa um critério de aceitação guloso e rigoroso. O módulo permite que apenas soluções estritamente melhores (ou seja, com custo total menor) que a melhor solução corrente (s_{best}) sejam aceitas. Esta estratégia, apesar de simples, garante a monotonia da melhoria ao longo das iterações. É uma escolha adequada, pois a diversificação necessária para escapar de ótimos locais é fornecida pelos módulos de *Perturbation*() (no ILS) e *GRASP_Construct*() (no GRASP).

O procedimento *acceptanceCriterion_BestSolution*() segue a lógica:

1. Comparação: Compara-se o custo da nova solução candidata (s') com o custo da melhor solução global encontrada até o momento (s_{global}).
2. Aceitação: Se $custo(s') < custo(s_{global})$ (critério de estrita melhoria):
   - s' é copiada completamente para a estrutura de s_{global}.

3.2.5 perturbation_DoubleBridge()

Exclusivo para o ILS, o método *perturbation_DoubleBridge*() implementa a perturbação Double-Bridge (ou Quadri-Opt). Esta é uma técnica elegante que visa gerar uma

modificação estrutural significativa na solução, dividindo uma rota em quatro segmentos e reconectando-os em uma ordem diferente ($A - B - C - D \rightarrow A - C - B - D$). A relevância deste operador reside na sua capacidade de retirar a solução de ótimos locais profundos, mantendo, no entanto, características estruturais da rota que foi otimizada.

O procedimento *perturbation_DoubleBridge*() é detalhado em:

1. Seleção e Verificação: Uma rota é selecionada aleatoriamente. É verificado se a rota possui um número mínimo de nós (geralmente $n \geq 8$ ou $n \geq 9$) para permitir a divisão em quatro segmentos.
2. Divisão: Quatro pontos de corte (*splits*) são selecionados aleatoriamente, dividindo a rota em cinco segmentos: A, B, C, D e E. (Nota: O segmento E representa o final da rota, do último split ao depósito).
   - Segmento A: Início da rota até $split_1$.
   - Segmento B: De $split_1$ até $split_2$.
   - Segmento C: De $split_2$ até $split_3$.
   - Segmento D: De $split_3$ até $split_4$.
   - Segmento E: De $split_4$ até o depósito.
3. Reconexão: Os segmentos são reconectados na ordem $A \rightarrow C \rightarrow B \rightarrow D \rightarrow E$ (a ordem clássica do Double-Bridge), formando a nova rota perturbada.
4. Custo: O custo da solução perturbada é recalculado.

3.2.6 GRASP_Construct()

Exclusivo para o GRASP, o método *GRASP_Construct*() implementa a fase construtiva, combinando elementos gulosos e aleatórios. Este equilíbrio entre exploração e explotação é controlado pelo parâmetro α. O algoritmo constrói a RCL, onde a dimensão e a qualidade dos candidatos são reguladas pelo valor de α. Isso permite a geração de soluções iniciais diversificadas a cada iteração do GRASP.

A estratégia de construção gulosa randomizada, realizada para cada nó de cliente a ser inserido na solução, é definida em:

1. Candidatos Viáveis: Identifica-se o conjunto completo de clientes não visitados (candidatos viáveis), respeitando as restrições de capacidade.
2. Avaliação: Para cada candidato, é calculado o custo incremental de inserção (ou custo de inserção).

3. Definição dos Limites: Os candidatos são ordenados por custo crescente para identificar o custo mínimo (*minCost*) e o custo máximo (*maxCost*). O limiar (*threshold*) é calculado com base em α:

$$threshold = minCost + \alpha \times (maxCost - minCost)$$

4. Construção da RCL: A RCL é formada por todos os candidatos cujo custo incremental de inserção é menor ou igual ao *threshold*.

5. Seleção: Um candidato é selecionado aleatoriamente da RCL e adicionado à rota atual, garantindo a natureza probabilística.

6. Atualização: As estruturas da rota e as listas de clientes são atualizadas, e o processo é repetido até que todos os clientes sejam alocados em rotas viáveis.

Todos os detalhes completos de implementação, incluindo o código-fonte em C++ e Python, estão disponíveis publicamente nos repositórios citados: [14], [15] e [16].

Capítulo 4

Resultados e Discussão

Este capítulo apresenta a descrição do ambiente experimental, os parâmetros de configuração e os resultados obtidos. Os ensaios foram realizados em um sistema com a seguinte configuração de hardware e software para garantir a reprodutibilidade: Processador: Intel Core i9-14900HX (2,20 GHz, 24 Núcleos, 32 Threads); Memória: 32GB DDR5 RAM; GPU: NVIDIA GeForce RTX 4070. O ambiente de execução foi o WSL2 (Subsistema Windows para Linux), utilizando a distribuição Ubuntu 24.04. O código foi compilado com G++ 13.3 e o parâmetro de otimização *-O3* foi empregado para maximizar o desempenho.

4.1 Ensaios e Comparação de resultados

Para a validação dos ensaios, foi empregado o conjunto de instâncias proposto por [17]. Este conjunto faz parte de um trabalho que introduziu 100 novas instâncias com um range de 100 a 1000 clientes, com o objetivo de mitigar a escassez de um conjunto robusto de benchmarks para o CVRP. As instâncias utilizadas estão disponíveis no repositório [18]. A partir dos resultados obtidos, o objetivo principal é comparar a qualidade e o tempo de execução dos algoritmos ILS e GRASP em diferentes subconjuntos dessas instâncias. A discussão visa apresentar os achados experimentais e analisar as possíveis causas para os comportamentos observados.

Para a avaliação de desempenho e análise estatística, cada instância foi executada 5 vezes de forma independente para ambos os algoritmos (ILS e GRASP), e a média dos custos das soluções foi utilizada como métrica de qualidade. Duas heurísticas de busca local foram testadas internamente: 2-Opt e *Swap**. No entanto, as soluções geradas utilizando o 2-Opt (apenas intra-rota) não demonstraram viabilidade ou qualidade competitiva para a comparação proposta. Portanto, optou-se por manter na análise comparativa apenas as execuções que utilizam o operador Swap* (que permite movimentos inter-rota e intra-rota), mais adequado para o refinamento de rotas no CVRP.

4.1.1 Conjunto de instâncias

O conjunto de instâncias utilizado neste estudo, disponível em [18], é a coleção original proposta por [17]. O nome de cada instância no CVRP segue uma convenção padrão que codifica suas principais características, facilitando a identificação do seu tamanho e restrições. O padrão de nomenclatura mais comum é: <Nome do conjuto da instância>-n<Número de Clientes>-k<Número Mínimo de Veículos>.

Exemplo: A instância X-n101-k25 é do cojunto "X", possui $\mathbf{n = 101}$ clientes e requer um mínimo de $\mathbf{k = 25}$ veículos.

4.1.2 Representação de Dados: Entrada e Saída

Os dados de **entrada** e **saída** para uma instância do CVRP são representados de forma tabular e sequencial, geralmente seguindo o formato de arquivos `.vrp` [18].

Dados de Entrada (O Problema)

O arquivo de entrada define todos os parâmetros necessários para resolver o problema:

Tabela 4.1: Estrutura de Dados de Entrada do CVRP

Componente	**Conteúdo**	**Exemplo de Dados**
Parâmetros Globais	Dimensão ($\mathcal{N}$ total de nós), Capacidade do Veículo ($\mathcal{Q}$).	`DIMENSION: 51`, `CAPACITY: 200`
Coordenadas dos Nós	ID do Nó, Coordenadas X e Y. (O nó 1 é geralmente o **Depósito**).	`NODE_COORD_SECTION` / `1 50 50` (Depósito) / `2 30 40` (Cliente)
Demandas dos Clientes	ID do Nó, Demanda (q_i).	`DEMAND_SECTION` / `1 0` (Depósito tem demanda zero) / `2 15` (Cliente 2 demanda 15)

Dados de Saída (A Solução)

A solução (**saída**) é o conjunto de rotas de custo mínimo (distância total), o qual representa uma sequência de nós que cada veículo deve percorrer:

Tabela 4.2: Estrutura de Dados de Saída da Solução do CVRP

Componente	**Conteúdo**	**Exemplo de Dados**
Valor Objetivo	O custo total da solução (distância total percorrida).	`Cost: 524.60` (Para a instância X-n101-k25)
Rotas	Sequência de nós visitados por cada veículo, iniciando e terminando no depósito (**0** ou **1**).	`Route 1: 1 -> 5 -> 12 -> 8 -> 1 / Route 2: 1 -> 20 -> 25 -> 1`

A Tabela 4.3, de resultados, compara o desempenho dos algoritmos ILS e GRASP utilizando o Swap* em todas as 100 instâncias do conjunto identificado como *X*. Ela apresenta:

- **Instância**: nome da instância
- **M. Sol.**: melhor solução conhecida (BKS) do CVRPLIB
- **M. Res.**: melhor valor obtido em 5 execuções
- **GAP (%)**: diferença percentual em relação à melhor solução conhecida
- **T (s)**: tempo médio de execução em segundos

Tabela 4.3: Comparação ILS vs GRASP com Swap* nas instâncias X.

		ILS			GRASP		
Instância	M. Sol.	M. Res.	GAP (%)	T(s)	M. Res.	GAP (%)	T(s)
X-n101-k25	27.591	39.265	42.31	0.15	33.806	22.53	2.57
X-n106-k14	26.362	28.217	7.04	2.03	28.901	9.63	4.04
X-n110-k13	14.971	18.788	25.50	2.05	19.298	28.90	4.57
X-n115-k10	12.747	13.956	9.48	6.79	15.195	19.20	16.34
X-n120-k6	13.332	14.866	11.51	5.20	15.849	18.88	12.99
X-n125-k30	55.539	66.985	20.61	0.88	63.668	14.64	4.51
X-n129-k18	28.940	34.486	19.16	2.15	35.967	24.28	6.44
X-n134-k13	10.916	14.419	32.09	2.52	14.527	33.08	8.97
X-n139-k10	13.590	16.854	24.02	3.23	17.101	25.84	9.24
X-n143-k7	15.700	17.546	11.76	8.04	18.994	20.98	23.34
X-n148-k46	43.448	56.517	30.08	0.19	56.579	30.22	5.24
X-n153-k22	21.220	25.559	20.45	16.17	24.430	15.13	36.02
X-n157-k13	16.876	18.107	7.29	4.75	18.600	10.22	13.57
X-n162-k11	14.138	15.703	11.07	8.80	17.552	24.15	24.65
X-n167-k10	20.557	23.519	14.41	11.03	25.399	23.55	21.91

Continua na próxima página...

Tabela 4.3 – Continuação

		ILS			GRASP		
Instância	M. Sol.	M. Res.	GAP (%)	T(s)	M. Res.	GAP (%)	T(s)
X-n172-k51	45.607	63.425	39.07	0.27	61.092	33.95	7.54
X-n176-k26	47.812	54.312	13.59	7.28	55.708	16.51	21.41
X-n181-k23	25.569	27.130	6.11	7.81	27.705	8.35	15.99
X-n186-k15	24.145	27.933	15.69	9.22	29.263	21.20	20.53
X-n190-k8	16.980	18.290	7.71	15.43	19.479	14.72	49.50
X-n195-k51	44.225	62.882	42.19	0.57	62.116	40.45	10.18
X-n200-k36	58.578	68.624	17.15	1.54	68.657	17.21	13.00
X-n204-k19	19.565	23.657	20.91	10.00	24.791	26.71	24.54
X-n209-k16	30.656	33.903	10.59	12.43	35.254	15.00	28.45
X-n214-k11	10.856	12.350	13.76	20.03	14.126	30.12	43.17
X-n219-k73	117.595	120.121	2.15	0.29	120.142	2.17	12.72
X-n223-k34	40.437	51.757	27.99	7.28	53.268	31.73	19.74
X-n228-k23	25.742	29.945	16.33	25.66	31.944	24.09	71.08
X-n233-k16	19.230	22.745	18.28	34.50	26.943	40.11	84.86
X-n237-k14	27.042	30.758	13.74	15.83	32.736	21.06	41.60
X-n242-k48	82.751	94.487	14.18	5.69	95.511	15.42	18.08
X-n247-k50	37.274	42.999	15.36	33.83	43.557	16.86	90.42
X-n251-k28	38.684	41.634	7.63	16.43	42.297	9.34	31.23
X-n256-k16	18.839	21.857	16.02	33.06	23.927	27.01	96.29
X-n261-k13	26.558	31.127	17.20	30.90	34.345	29.32	85.51
X-n266-k58	75.478	81.558	8.06	0.55	82.260	8.99	22.48
X-n270-k35	35.291	40.599	15.04	16.30	41.193	16.72	33.76
X-n275-k28	21.245	23.299	9.67	23.14	24.086	13.37	56.47
X-n280-k17	33.503	40.040	19.51	31.02	42.116	25.71	99.50
X-n284-k15	20.215	22.828	12.93	48.94	25.465	25.97	141.80
X-n289-k60	95.151	112.531	18.27	6.89	117.677	23.67	35.02
X-n294-k50	47.161	64.792	37.38	8.75	69.819	48.04	43.24
X-n298-k31	34.231	47.751	39.50	20.96	50.060	46.24	53.54
X-n303-k21	21.736	25.290	16.35	44.14	30.017	38.10	170.28
X-n308-k13	25.859	29.531	14.20	80.48	33.409	29.20	239.54
X-n313-k71	94.043	120.925	28.58	4.16	126.068	34.05	30.33
X-n317-k53	78.355	80.832	3.16	27.50	81.090	3.49	53.76
X-n322-k28	29.834	35.496	18.98	37.76	38.400	28.71	77.11
X-n327-k20	27.532	31.992	16.20	48.07	33.195	20.57	118.17
X-n331-k15	31.102	35.081	12.79	57.54	36.400	17.03	129.25
X-n336-k84	139.111	167.335	20.29	10.23	173.135	24.46	39.10
X-n344-k43	42.050	49.098	16.76	32.41	49.262	17.15	58.50
X-n351-k40	25.896	33.246	28.38	37.13	39.425	52.24	95.33
X-n359-k29	51.505	58.429	13.44	44.84	61.463	19.33	88.39
X-n367-k17	22.814	26.416	15.79	102.37	29.111	27.60	309.86

Continua na próxima página...

Tabela 4.3 – Continuação

		ILS			GRASP		
Instância	M. Sol.	M. Res.	GAP (%)	T(s)	M. Res.	GAP (%)	T(s)
X-n376-k94	147.713	150.344	1.78	0.72	150.642	1.98	48.30
X-n384-k52	65.928	72.619	10.15	43.69	72.952	10.65	67.00
X-n393-k38	38.260	44.036	15.10	48.78	45.689	19.42	110.53
X-n401-k29	66.154	72.984	10.32	92.94	76.866	16.19	177.46
X-n411-k19	19.712	24.292	23.23	204.03	28.539	44.78	271.58
X-n420-k130	107.798	133.006	23.38	1.43	135.097	25.32	57.44
X-n429-k61	65.449	73.445	12.22	61.16	73.774	12.72	89.88
X-n439-k37	36.391	40.847	12.24	94.39	41.569	14.23	208.88
X-n449-k29	55.233	67.064	21.42	72.70	73.638	33.32	156.36
X-n459-k26	24.139	29.465	22.06	97.26	31.710	31.36	258.52
X-n469-k138	221.824	242.255	9.21	1.25	242.387	9.27	71.68
X-n480-k70	89.449	97.236	8.71	66.73	97.759	9.29	129.58
X-n491-k59	66.483	89.084	34.00	85.00	97.862	47.20	137.50
X-n502-k39	69.226	71.618	3.46	133.78	72.365	4.53	288.03
X-n513-k21	24.201	29.182	20.58	234.40	32.003	32.24	285.51
X-n524-k153	154.593	181.244	17.24	76.90	180.653	16.86	480.15
X-n536-k96	94.846	125.302	32.11	22.64	127.659	34.60	154.42
X-n548-k50	86.700	92.086	6.21	114.97	92.662	6.88	225.36
X-n561-k42	42.717	56.520	32.31	162.00	59.271	38.75	380.87
X-n573-k30	50.673	55.188	8.91	299.34	57.335	13.15	290.09
X-n586-k159	190.316	206.118	8.30	1.80	206.408	8.46	141.38
X-n599-k92	108.451	117.496	8.34	149.48	117.921	8.73	190.38
X-n613-k62	59.535	79.619	33.73	151.26	88.302	48.32	286.82
X-n627-k43	62.164	68.614	10.38	199.25	69.260	11.41	280.66
X-n641-k35	63.682	69.938	9.82	172.80	71.188	11.79	474.96
X-n655-k131	106.780	109.752	2.78	1.84	109.891	2.91	259.58
X-n670-k130	146.332	175.224	19.74	438.29	199.976	36.66	337.98
X-n685-k75	68.205	92.417	35.50	201.16	97.839	43.45	457.83
X-n701-k44	81.923	90.741	10.76	281.48	94.157	14.93	576.44
X-n716-k35	43.373	49.920	15.09	501.45	60.012	38.36	261.07
X-n733-k159	136.187	164.699	20.94	52.97	166.257	22.08	295.19
X-n749-k98	77.269	99.778	29.13	245.39	106.651	38.03	464.33
X-n766-k71	114.417	137.304	20.00	336.40	140.538	22.83	426.08
X-n783-k48	72.386	85.770	18.49	442.55	95.071	31.34	634.03
X-n801-k40	73.305	79.516	8.47	320.96	81.235	10.82	696.37
X-n819-k171	158.121	170.773	8.00	8.45	171.225	8.29	445.87
X-n837-k142	193.737	206.293	6.48	326.10	206.523	6.60	480.62
X-n856-k95	88.965	95.780	7.66	434.48	96.187	8.12	732.93
X-n876-k59	99.299	111.817	12.61	536.80	120.995	21.85	459.96
X-n895-k37	53.860	62.422	15.90	969.71	64.450	19.66	543.18

Continua na próxima página...

Tabela 4.3 – Continuação

		ILS			GRASP		
Instância	M. Sol.	M. Res.	GAP (%)	T(s)	M. Res.	GAP (%)	T(s)
X-n916-k207	329.179	348.562	5.89	6.27	348.859	5.98	513.31
X-n936-k151	132.715	159.138	19.91	1365.60	192.824	45.29	320.18
X-n957-k87	85.465	91.580	7.15	483.46	92.117	7.78	890.78
X-n979-k58	118.976	130.190	9.43	1054.21	144.891	21.78	1084.20
X-n1001-k43	72.355	83.844	15.88	1126.35	85.961	18.80	653.88

A análise comparativa entre os algoritmos ILS e GRASP, ambos utilizando o operador de busca local *Swap**, revelou diferenças significativas no desempenho das metaheurísticas. Conforme apresentado na Tabela 4.3 , o ILS demonstrou superioridade consistente em termos de qualidade das soluções obtidas. Considerando o conjunto completo de 100 instâncias do benchmark X, o ILS obteve soluções de custo melhor ou igual ao GRASP em 98 das 100 instâncias testadas. As únicas exceções foram as instâncias X-n524-k153 e X-n876-k59, nas quais o GRASP apresentou resultados ligeiramente superiores. Esta tendência clara de superioridade do ILS pode ser atribuída à sua estratégia de intensificação (busca local profunda) combinada com um mecanismo de perturbação (*DoubleBridge*) que permite escapar de ótimos locais de forma mais eficaz, realocando a busca para novas bacias de atração.

Em relação ao tempo de execução, observa-se uma diferença substancial entre os dois algoritmos. O ILS apresentou tempos de execução significativamente menores, sendo, em média, aproximadamente 1,46 vezes mais rápido que o GRASP. Esta discrepância de desempenho pode ser explicada pela diferença fundamental nas suas fases de busca: enquanto o ILS parte de uma única solução inicial e a refina iterativamente (foco na intensificação), o GRASP exige a reconstrução completa de uma solução (*GRASP_Construct*) a cada iteração, um processo que é mais custoso do ponto de vista computacional.

Apesar do significativo tempo de execução adicional, o GRASP não demonstrou capacidade de compensar essa desvantagem com soluções de melhor qualidade. A diferença média entre os custos das soluções obtidas pelos dois algoritmos foi de aproximadamente 7,2% em favor do ILS, indicando que o esforço computacional extra demandado pela fase de construção randomizada do GRASP não se traduziu em ganhos de qualidade que justificassem a sua complexidade ou tempo de processamento.

4.1.3 Comparação com outros trabalhos

A comparação dos resultados obtidos neste trabalho com aqueles apresentados por [9], conforme mostrado na Tabela 4.4 , revela diferenças substanciais no desempenho entre os paradigmas de otimização aplicados ao CVRP. Os Algoritmos Evolutivos implementados por [9] demonstraram desempenho consistentemente superior aos métodos ILS e GRASP desenvolvidos neste TCC. O Algoritmo Memético (AM),

em particular, apresentou os melhores resultados gerais, obtendo soluções de custo inferior em 98 das 100 instâncias quando comparado ao ILS com *Swap**. De maneira similar, o Algoritmo Genético (AG) também superou tanto o ILS quanto o GRASP na maioria das instâncias testadas.

A Tabela 4.4 compara os resultados obtidos pelos algoritmos ILS e GRASP (ambos com Swap*) com os resultados do Algoritmo Genético (AG) e Algoritmo Memético (AM) apresentados por Rohwedder e Dantas (2024) nas instâncias do conjunto X. Ela apresenta:

M. Res.: melhor valor obtido nas execuções

Tabela 4.4: Comparação dos resultados com Rohwedder e Dantas (2024).

	Este Trabalho		Rohwedder e Dantas (2024)	
Instância	ILS Swap*	GRASP Swap*	AG	AM
X-n101-k25	39.265,00	33.806,00	30.540,00	28.668,40
X-n106-k14	28.217,00	28.901,00	28.259,60	27.118,20
X-n110-k13	18.788,00	19.298,00	17.407,90	16.249,40
X-n115-k10	13.956,00	15.195,00	14.990,40	14.084,90
X-n120-k6	14.866,00	15.849,00	16.213,10	14.454,90
X-n125-k30	66.985,00	63.668,00	63.010,10	58.642,60
X-n129-k18	34.486,00	35.967,00	33.666,40	31.166,00
X-n134-k13	14.419,00	14.527,00	12.794,30	11.660,50
X-n139-k10	16.854,00	17.101,00	16.615,40	15.014,70
X-n143-k7	17.546,00	18.994,00	19.616,40	18.048,70
X-n148-k46	56.517,00	56.579,00	49.303,60	45.266,50
X-n153-k22	25.559,00	24.430,00	26.990,00	22.970,50
X-n157-k13	18.107,00	18.600,00	19.063,20	17.498,50
X-n162-k11	15.703,00	17.552,00	16.947,30	15.483,10
X-n167-k10	23.519,00	25.399,00	24.697,10	23.139,50
X-n172-k51	63.425,00	61.092,00	50.152,30	47.170,30
X-n176-k26	54.312,00	55.708,00	59.047,20	51.061,90
X-n181-k23	27.130,00	27.705,00	28.200,30	26.206,10
X-n186-k15	27.933,00	29.263,00	28.469,80	26.542,40
X-n190-k8	18.290,00	19.479,00	19.550,80	18.292,70
X-n195-k51	62.882,00	62.116,00	49.630,30	46.717,70
X-n200-k36	68.624,00	68.657,00	64.380,10	61.358,60
X-n204-k19	23.657,00	24.791,00	22.868,30	21.267,70
X-n209-k16	33.903,00	35.254,00	36.184,00	33.298,80
X-n214-k11	12.350,00	14.126,00	13.304,80	12.565,10
X-n219-k73	120.121,00	120.142,00	119.649,00	117.769,00
X-n223-k34	51.757,00	53.268,00	45.662,00	43.918,60

Continua na próxima página...

Tabela 4.4 – Continuação

	Este Trabalho		Rohwedder e Dantas (2024)	
Instância	ILS	GRASP	AG	AM
X-n228-k23	29.945,00	31.944,00	31.272,60	28.722,30
X-n233-k16	22.745,00	26.943,00	23.332,10	22.585,30
X-n237-k14	30.758,00	32.736,00	32.171,50	29.391,70
X-n242-k48	94.487,00	95.511,00	92.476,00	88.343,90
X-n247-k50	42.999,00	43.557,00	44.298,10	40.043,30
X-n251-k28	41.634,00	42.297,00	43.784,00	41.435,00
X-n256-k16	21.857,00	23.927,00	21.568,20	20.743,10
X-n261-k13	31.127,00	34.345,00	32.291,50	30.961,40
X-n266-k58	81.558,00	82.260,00	83.181,50	79.545,60
X-n270-k35	40.599,00	41.193,00	40.079,10	37.696,00
X-n275-k28	23.299,00	24.086,00	24.069,40	22.634,70
X-n280-k17	40.040,00	42.116,00	42.037,60	38.285,80
X-n284-k15	22.828,00	25.465,00	24.398,10	23.243,90
X-n289-k60	112.531,00	117.677,00	107.015,00	102.504,00
X-n294-k50	64.792,00	69.819,00	54.006,40	51.772,70
X-n298-k31	47.751,00	50.060,00	40.282,80	39.008,90
X-n303-k21	25.290,00	30.017,00	25.980,30	24.857,00
X-n308-k13	29.531,00	33.409,00	33.479,50	30.121,10
X-n313-k71	120.925,00	126.068,00	106.346,00	100.169,00
X-n317-k53	80.832,00	81.090,00	80.669,90	79.183,00
X-n322-k28	35.496,00	38.400,00	34.940,40	32.659,30
X-n327-k20	31.992,00	33.195,00	33.292,30	31.638,80
X-n331-k15	35.081,00	36.400,00	37.259,30	33.815,70
X-n336-k84	167.335,00	173.135,00	159.811,00	150.230,00
X-n344-k43	49.098,00	49.262,00	48.117,00	45.318,70
X-n351-k40	33.246,00	39.425,00	29.772,90	28.897,20
X-n359-k29	58.429,00	61.463,00	59.178,60	57.815,70
X-n367-k17	26.416,00	29.111,00	28.451,30	26.917,50
X-n376-k94	150.344,00	150.642,00	152.308,00	148.801,00
X-n384-k52	72.619,00	72.952,00	73.951,40	70.969,50
X-n393-k38	44.036,00	45.689,00	43.988,10	42.638,30
X-n401-k29	72.984,00	76.866,00	73.428,50	71.636,90
X-n411-k19	24.292,00	28.539,00	24.996,80	23.865,50
X-n420-k130	133.006,00	135.097,00	122.972,00	114.416,00
X-n429-k61	73.445,00	73.774,00	73.143,50	70.910,80
X-n439-k37	40.847,00	41.569,00	41.491,60	39.898,40
X-n449-k29	67.064,00	73.638,00	64.594,80	63.397,20
X-n459-k26	29.465,00	31.710,00	29.325,80	28.338,20
X-n469-k138	242.255,00	242.387,00	250.757,00	235.472,00
X-n480-k70	97.236,00	97.759,00	98.697,40	95.788,70

Continua na próxima página...

Tabela 4.4 – Continuação

	Este Trabalho		Rohwedder e Dantas (2024)	
Instância	ILS	GRASP	AG	AM
X-n491-k59	89.084,00	97.862,00	75.249,00	74.054,30
X-n502-k39	71.618,00	72.365,00	73.470,80	71.527,60
X-n513-k21	29.182,00	32.003,00	30.441,00	29.703,20
X-n524-k153	181.244,00	180.653,00	186.553,00	162.966,00
X-n536-k96	125.302,00	127.659,00	105.954,00	102.952,00
X-n548-k50	92.086,00	92.662,00	94.254,20	91.856,20
X-n561-k42	56.520,00	59.271,00	50.634,30	49.658,90
X-n573-k30	55.188,00	57.335,00	58.696,60	56.956,40
X-n586-k159	206.118,00	206.408,00	212.787,00	204.666,00
X-n599-k92	117.496,00	117.921,00	120.824,00	117.787,00
X-n613-k62	79.619,00	88.302,00	69.336,60	68.299,50
X-n627-k43	68.614,00	69.260,00	69.572,60	68.076,70
X-n641-k35	69.938,00	71.188,00	73.873,60	72.253,00
X-n655-k131	109.752,00	109.891,00	110.372,00	108.759,00
X-n670-k130	175.224,00	199.976,00	181.131,00	166.589,00
X-n685-k75	92.417,00	97.839,00	78.527,40	77.462,00
X-n701-k44	90.741,00	94.157,00	93.597,30	92.730,10
X-n716-k35	49.920,00	60.012,00	50.442,10	50.048,50
X-n733-k159	164.699,00	166.257,00	153.632,00	151.463,00
X-n749-k98	99.778,00	106.651,00	87.303,90	87.145,00
X-n766-k71	137.304,00	140.538,00	138.460,00	134.554,00
X-n783-k48	85.770,00	95.071,00	84.082,90	84.229,40
X-n801-k40	79.516,00	81.235,00	84.086,50	82.610,10
X-n819-k171	170.773,00	171.225,00	174.334,00	174.012,00
X-n837-k142	206.293,00	206.523,00	212.812,00	211.000,00
X-n856-k95	95.780,00	96.187,00	97.372,20	95.908,30
X-n876-k59	111.817,00	120.995,00	110.018,00	109.896,00
X-n895-k37	62.422,00	64.450,00	65.631,00	64.389,10
X-n916-k207	348.562,00	348.859,00	364.024,00	358.095,00
X-n936-k151	159.138,00	192.824,00	166.842,00	160.901,00
X-n957-k87	91.580,00	92.117,00	94.129,00	92.984,90
X-n979-k58	130.190,00	144.891,00	129.897,00	130.012,00
X-n1001-k43	83.844,00	85.961,00	85.128,40	84.734,60

Essa superioridade pode ser atribuída a diversos fatores metodológicos intrínsecos aos algoritmos populacionais: Primeiro, os algoritmos evolutivos (AG e AM) mantêm uma população de soluções que evolui ao longo das gerações, o que naturalmente permite uma exploração mais diversificada do espaço de busca em comparação com a busca baseada em trajetória única (ILS e GRASP). Segundo, a estratégia de inicialização híbrida empregada por [9] (combinando métodos aleatórios com heurísticas cons-

trutivas baseadas em *K-Means* e vizinho mais próximo) proporcionou uma população inicial de melhor qualidade. Adicionalmente, o AM incorporou o operador *Swap** em uma etapa de busca local baseada em *Simulated Annealing*. Essa combinação pode ter permitido uma explotação mais eficaz da vizinhança, facilitando o escape de ótimos locais de forma mais eficiente.

Capítulo 5

Conclusão

Este trabalho teve como objetivo a implementação e avaliação comparativa das meta-heurísticas Busca Local Iterada (ILS) e Procedimento de Busca Adaptativa Gulosa Aleatória (GRASP) na solução do Problema de Roteamento de Veículos Capacitados (CVRP). Ambos os algoritmos são notáveis por sua consistência, simplicidade conceitual e eficácia. Eles se baseiam na aplicação iterativa de um procedimento de busca local sobre soluções. A diferença crucial entre as abordagens reside na geração das soluções: o ILS perturba uma solução já ótima localmente para explorar novas bacias de atração, enquanto o GRASP gera um tour guloso e aleatório a cada iteração antes de refiná-lo com a busca local.

Das principais contribuições alcançadas por este trabalho destaca-se o desenvolvimento de uma arquitetura de software modular em C++20 para o CVRP, com ênfase no reaproveitamento de componentes. O estudo forneceu ainda a validação empírica do *trade-off* entre intensificação (ILS) e diversificação (GRASP).

As meta-heurísticas alcançaram soluções de alta qualidade e próximas do ótimo nas 100 instâncias do conjunto X. Os resultados quantitativos destacam que o ILS obteve soluções de custo superior em 98% das instâncias, em relação ao GRASP. Além da superioridade na qualidade, o ILS demonstrou um tempo de execução menor em comparação com o GRASP, embora ambos os métodos tenham sido superados em qualidade de custo pelos Algoritmos Evolutivos da literatura comparada.

Sugerem-se alguns seguimentos de extensão e melhorias para trabalhos futuros. A primeira delas foca na Otimização de Hardware: implementar as meta-heurísticas com paralelização em GPGPUs (CUDA) ou FPGAs, visando obter ganhos de otimização dos operadores de Busca Local para instâncias de grande porte. Ainda é interessante focar no refinamento das soluções propostas, mantendo avaliações mais robustas a fim de serem feitos testes de validação estatística. Em suma, os resultados deste trabalho reforçam o papel das meta-heurísticas de busca local como ferramentas indispensáveis para a otimização de problemas NP-Difíceis, como o CVRP.

Referências Bibliográficas

[1] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959. 1

[2] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 1992. 1, 2.1

[3] Isaac Kosloski Oliveira, Bianca de Almeida Dantas, and Graziela Santos Araújo. A multi-core iterated local search for the traveling salesman problem using openmp. In *Anais da 7. Escola Regional de Alto Desempenho do Centro-Oeste (ERAD-CO)*, pages 30–32, Brasília/DF, 2024. Sociedade Brasileira de Computação (SBC). 1

[4] Fred Glover and Gary A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2002. Para o contexto geral de metaheurísticas e NP-hard. 1

[5] Helena R. Lourenço, Olivier C. Martin, and Thomas Stützle. Iterated local search. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, pages 321–353. Kluwer Academic Publishers, 2002. Artigo seminal de revisão do ILS. 1

[6] Mauricio G. C. Resende and Celso C. Ribeiro. *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. Springer, 2016. Livro de referência sobre o GRASP. Alternativamente, a citação seminal de 1995 (Resende and Feo) pode ser usada. 1

[7] Paolo Toth and Daniele Vigo, editors. *Vehicle Routing: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2014. 1

[8] Katrien Ramaekers Kris Braekers and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers Industrial Engineering*, 2015. 2.1

[9] Luan Sandim Rohwedder and Bianca de Almeida Dantas. Um estudo do problema de roteamento de veículos e implementações de algoritmos genéticos e meméticos. 2024. 2.1.1, 4.1.3, 4.1.3

[10] Olivier Martinz Helena R. Lorenço and THOMAS STUTZLE. Iterated local search. 2.2

[11] Thomas STUTZLE and Marco DORIGO. Aco algorithms for the traveling salesman problem. 1999. 2.2

[12] Mauricio G.C. Resende and Celso C. Ribeiro. *Optimization by GRASP*. Springer Science+Business Media LCC, 2016. 2.3.1, 2.3.2

[13] G. A. Croes. A method for solving traveling salesman problems. *Operations Research*, 6(6):791–812, 1958. 3

[14] Isaac Kosloski. Capacitated Vehicle Routing Problem: Iterated Local Search (ILS). GitHub repository, 2025. 3.2.6

[15] Isaac Kosloski. Capacitated Vehicle Routing Problem: Greedy Randomized Adaptive Search Procedure (GRASP). GitHub repository, 2025. 3.2.6

[16] Isaac Kosloski. Capacitated Vehicle Routing Problem: Dashboard. GitHub repository, 2025. 3.2.6

[17] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. New benchmark instances for the capacitated vehicle routing problem. *European Journal of Operational Research*, 257(3):845–858, 2017. 4.1, 4.1.1

[18] Disponível em: https://galgos.inf.puc-rio.br/cvrplib/en/instances, 2025. 4.1, 4.1.1, 4.1.2