Universidade Federal do Paraná

Bruno Augusto Luvizott
Rodrigo Campos Serra Domingues

ANÁLISE COMPARATIVA DE ALGORITMOS META-HEURÍSTICOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE BIOBJETIVO

Curitiba PR
2025

Bruno Augusto Luvizott
Rodrigo Campos Serra Domingues

ANÁLISE COMPARATIVA DE ALGORITMOS META-HEURÍSTICOS PARA O PROBLEMA DO CAIXEIRO VIAJANTE BIOBJETIVO

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Eduardo Jaques Spinosa.

Curitiba PR
2025

Aos nossos professores, pais e amigos, que tornaram esta conquista possível através de seu apoio e orientação.

Agradecimentos

Agradecemos a todas as pessoas e amigos que nos apoiaram e ajudaram ao longo de todos esses anos. Um agradecimento especial ao nosso orientador, Professor Eduardo Spinosa, pela orientação, paciência e pelos conhecimentos compartilhados, fundamentais para a conclusão deste trabalho. Estendemos nossa gratidão aos professores Giovanni Venâncio e Lucas Ferrari de Oliveira pela participação na banca examinadora e pelas valiosas contribuições e sugestões.

À Universidade Federal do Paraná (UFPR) e ao Departamento de Informática, agradecemos pela infraestrutura disponibilizada e pelo ambiente propício ao aprendizado. Agradecemos também a todos os professores que contribuíram para nossa formação acadêmica durante o curso.

Eu, Rodrigo, agradeço imensamente aos meus pais e amigos pelo apoio incondicional e pelo incentivo constante.

Eu, Bruno, agradeço à minha mãe por todo o amor, apoio e incentivo que sempre me proporcionou.

Resumo

O Problema do Caixeiro Viajante Multiobjetivo (MOTSP) é um desafio canônico na otimização combinatória, servindo de modelo para problemas práticos que exigem o balanceamento de múltiplos critérios conflitantes, como custo e tempo. Dada a sua complexidade NP-difícil, meta-heurísticas são a abordagem padrão para encontrar boas aproximações da Fronteira de Pareto. Este trabalho apresenta uma análise comparativa abrangente de nove proeminentes algoritmos meta-heurísticos para o MOTSP, representando diferentes paradigmas: baseados em dominância de Pareto (NSGA-II, SPEA2, GDE3), em decomposição (MOEA/D, NSGA-III), em indicadores (SMS-EMOA) e em inteligência de enxame (MOCell, SMPSO, MOACO). A avaliação foi conduzida em instâncias de 100 a 800 cidades, utilizando uma metodologia multifacetada que considera a qualidade da solução (Hipervolume), a eficiência computacional (tempo de execução e consumo de memória) e um Índice de Desempenho Combinado (IDC) para uma classificação agregada. Os resultados demonstram uma superioridade esmagadora do MOACO em termos de qualidade da solução, alcançando valores de Hipervolume ordens de magnitude maiores que seus concorrentes, um resultado validado estatisticamente. Apesar de sua escalabilidade temporal ser inferior em problemas maiores, sua performance excepcional em qualidade o consagrou como o algoritmo de melhor desempenho global no IDC. Em contraste, o SMS-EMOA destacou-se pela excelente escalabilidade em tempo de execução, tornando-se o segundo melhor no ranking geral para a maior instância. O estudo também revelou que o SMPSO é computacionalmente inviável devido ao seu consumo de memória exponencial. A análise conclui que, embora o MOACO seja superior na busca por soluções de alta qualidade, a escolha do algoritmo ideal depende do *trade-off* entre a qualidade desejada e os recursos computacionais disponíveis, com o SMS-EMOA emergindo como uma alternativa eficiente para cenários com restrições de tempo.

Palavras-chave: Otimização Multiobjetivo, Problema do Caixeiro-Viajante, Meta-heurísticas.

Abstract

The Multi-objective Traveling Salesman Problem (MOTSP) is a canonical challenge in combinatorial optimization, serving as a model for practical problems that require balancing multiple conflicting criteria, such as cost and time. Given its NP-hard complexity, metaheuristics are the standard approach for finding good approximations of the Pareto Front. This work presents a comprehensive comparative analysis of nine prominent metaheuristic algorithms for the MOTSP, representing different paradigms: Pareto dominance-based (NSGA-II, SPEA2, GDE3), decomposition-based (MOEA/D, NSGA-III), indicator-based (SMS-EMOA), and swarm intelligence (MOCell, SMPSO, MOACO). The evaluation was conducted on instances ranging from 100 to 800 cities, using a multi-faceted methodology that considers solution quality (Hypervolume), computational efficiency (execution time and memory consumption), and a Combined Performance Index (IDC) for an aggregated ranking. The results demonstrate an overwhelming superiority of MOACO in terms of solution quality, achieving Hypervolume values orders of magnitude greater than its competitors, a result that was statistically validated. Although its time scalability is inferior on larger problems, its exceptional quality performance established it as the best overall algorithm in the IDC ranking. In contrast, SMS-EMOA stood out for its excellent scalability in execution time, becoming the second-best in the overall ranking for the largest instance. The study also revealed that SMPSO is computationally unfeasible due to its exponential memory consumption. The analysis concludes that while MOACO is superior in the search for high-quality solutions, the choice of the ideal algorithm depends on the trade-off between desired quality and available computational resources, with SMS-EMOA emerging as an efficient alternative for time-constrained scenarios.

Keywords: Multi-objective Optimization, Traveling Salesman Problem, Metaheuristics.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

ACO	Otimização por Colônia de Formigas (*Ant Colony Optimization*)
AG	Algoritmo Genético
ATSP	Problema do Caixeiro Viajante Assimétrico (*Asymmetric Traveling Salesman Problem*)
cGA	Algoritmo Genético Celular (*Cellular Genetic Algorithm*)
DE	Evolução Diferencial (*Differential Evolution*)
GDE3	*Generalized Differential Evolution 3*
HCP	Problema do Ciclo Hamiltoniano (*Hamiltonian Cycle Problem*)
HPP	Problema do Caminho Hamiltoniano (*Hamiltonian Path Problem*)
HV	Hipervolume
IDC	Índice de Desempenho Combinado
MaOP	Problema de Otimização com Muitos Objetivos (*Many-Objective Problem*)
MCDM	Tomada de Decisão Multicritério (*Multi-Criteria Decision Making*)
MOACO	*Multi-Objective Ant Colony Optimization*
MOCell	*Multi-Objective Cellular Genetic Algorithm*
MOEA	Algoritmo Evolutivo Multiobjetivo (*Multi-Objective Evolutionary Algorithm*)
MOEA/D	*Multi-Objective Evolutionary Algorithm based on Decomposition*
MOSP	Problema Padrão Mono-objetivo (*Mono-Objective Standard Problem*)
MOTSP	Problema do Caixeiro Viajante Multiobjetivo (*Multi-Objective Traveling Salesman Problem*)
NP	Tempo Polinomial Não-Determinístico (*Nondeterministic Polynomial time*)
NSGA-II	*Non-dominated Sorting Genetic Algorithm II*
NSGA-III	*Non-dominated Sorting Genetic Algorithm III*
P-ACO	Otimização por Colônia de Formigas Baseada em Pareto (*Pareto-Based Ant Colony Optimization*)

PSO	Otimização por Enxame de Partículas (*Particle Swarm Optimization*)
SI	Inteligência de Enxame (*Swarm Intelligence*)
SMPSO	*Speed-constrained Multi-objective Particle Swarm Optimization*
SMS-EMOA	*S-Metric Selection Evolutionary Multi-Objective Algorithm*
SPEA2	*Strength Pareto Evolutionary Algorithm 2*
TOPSIS	*Technique for Order of Preference by Similarity to Ideal Solution*
TSP	Problema do Caixeiro Viajante (*Traveling Salesman Problem*)
TSPLIB	*Traveling Salesman Problem Library*

Lista de Símbolos

G	Grafo $G = (V, E)$
V	Conjunto de vértices em um grafo
E	Conjunto de arestas em um grafo
N	Número de cidades (vértices) ou tamanho da população
D	Matriz de distâncias ou custos
$d_{i,j}$	Distância ou custo entre os nós i e j
π	Uma rota ou permutação de cidades
$C(p)$	Custo total de uma rota p
k	Número de objetivos em um problema multiobjetivo
$f_i(x)$	Valor do i-ésimo objetivo para a solução x
$\mathbf{F}(x)$	Vetor de funções objetivo para a solução x
$\preceq$	Relação de dominância de Pareto
$\mathcal{P}$	Conjunto Ótimo de Pareto (no espaço de decisão)
$\mathcal{PF}$	Fronteira de Pareto (no espaço de objetivos)
HV	Hipervolume
R	Ponto de referência para cálculo do Hipervolume
λ	Vetor de pesos em algoritmos de decomposição
t	Índice de iteração ou geração
P_t	População na geração t
p_c, p_m	Probabilidades de cruzamento e mutação
x_i	Vetor de posição da partícula i no PSO
v_i	Vetor de velocidade da partícula i no PSO
$pbest_i$	Melhor posição pessoal encontrada pela partícula i no PSO
$gbest$	Melhor posição global encontrada pelo enxame no PSO
w	Peso de inércia no PSO
c_1, c_2	Coeficientes de aceleração cognitivo e social no PSO
r_1, r_2	Valores aleatórios uniformemente distribuídos
χ	Fator de constrição no SMPSO
τ_{ij}	Intensidade do feromônio na aresta (i, j) no ACO
η_{ij}	Informação heurística entre os nós i e j no ACO
α, β	Parâmetros de ponderação no ACO
ρ	Taxa de evaporação do feromônio no ACO

Sumário

Capítulo 1

Introdução

A otimização combinatória é uma área fundamental da ciência da computação, e dentre seus problemas, o Problema do Caixeiro Viajante (TSP) destaca-se como um dos mais emblemáticos. Classificado por Garey e Johnson (1979) como NP-difícil, o TSP não apenas representa um desafio teórico para a teoria da complexidade, mas também serve como modelo para uma vasta gama de problemas práticos na indústria e na ciência, com aplicações que vão desde o roteamento de veículos em logística até o sequenciamento de tarefas em manufatura.

A intratabilidade computacional do TSP, que implica a inexistência de algoritmos capazes de encontrar a solução ótima em tempo polinomial para todas as instâncias, torna-o um campo fértil para o desenvolvimento e teste de abordagens heurísticas. No entanto, aplicações do mundo real frequentemente exigem a otimização simultânea de múltiplos critérios conflitantes, como minimizar a distância e, ao mesmo tempo, o tempo de viagem. Essa necessidade dá origem à versão multiobjetivo do problema (MOTSP), na qual a adição de novas funções objetivo eleva a complexidade e redefine o conceito de otimalidade. Em vez de uma única solução ótima, o objetivo passa a ser a identificação de um conjunto de soluções de compromisso, cuja imagem no espaço de objetivos constitui uma "Fronteira de Pareto", de acordo com Deb (2001).

Para lidar com a complexidade do MOTSP, a literatura consolidou o uso de meta-heurísticas, especialmente as de natureza populacional, como Algoritmos Genéticos e de Inteligência de Enxame. Segundo Coello Coello et al. (2007), tais algoritmos são inerentemente adequados para a otimização multiobjetivo, pois manipulam um conjunto de soluções candidatas em paralelo, permitindo uma exploração eficiente do espaço de busca e a aproximação da Fronteira de Pareto em uma única execução.

Nesse contexto, este trabalho realiza um estudo comparativo abrangente entre nove proeminentes algoritmos meta-heurísticos para o MOTSP. Foram selecionadas abordagens representativas dos principais paradigmas da área: algoritmos baseados em dominância de Pareto (NSGA-II, SPEA2, GDE3), em decomposição (MOEA/D, NSGA-III), em indicadores (SMS-EMOA) e em inteligência de enxame (MOCell, SMPSO, MOACO). As contribuições desta pesquisa são multifacetadas. Primeiramente, destaca-se o desenvolvimento de uma implementação própria do algoritmo MOACO, baseado na abordagem P-ACO, uma vez que tal algoritmo não estava disponível no *framework jmetalpy* utilizado. Em segundo lugar, a análise de desempenho vai além da tradicional avaliação da qualidade da solução (medida pelo Hipervolume), incorporando uma análise rigorosa da eficiência computacional. Em particular, a medição do consumo de memória, um fator crítico para a escalabilidade em ambientes com recursos limitados, mas frequentemente negligenciado em estudos comparativos. Finalmente, como principal contribuição metodológica, este trabalho propõe e utiliza o Índice de Desempenho Combinado (IDC), uma métrica agregada que consolida a qualidade da solução e os custos

computacionais (tempo e memória) em um único indicador. Essa abordagem permite uma classificação robusta dos algoritmos e revela de forma clara os *trade-offs* entre eficácia e eficiência.

Para guiar o leitor através deste projeto, o trabalho está organizado nos seguintes capítulos. O Capítulo 2, *Fundamentação Teórica*, detalha o TSP, sua extensão multiobjetivo, o conceito de Fronteira de Pareto e os paradigmas de Algoritmos Genéticos e Inteligência de Enxame. O Capítulo 3, *Materiais e Metodologia*, descreve as instâncias de teste, os algoritmos avaliados e as métricas de desempenho utilizadas. O Capítulo 4, *Implementação*, explica a arquitetura do software e os detalhes da execução dos experimentos. O Capítulo 5, *Resultados*, apresenta e analisa os dados coletados, incluindo a análise estatística. Finalmente, o Capítulo 6, *Conclusão*, resume as descobertas, discute suas implicações e sugere direções para trabalhos futuros.

Capítulo 2

Fundamentação Teórica

2.1 O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (TSP, na sigla em inglês) é um problema de otimização combinatória bastante estudado em matemática e ciência da computação. É um problema NP-difícil, o que significa que é um problema ao menos tão difícil quanto o problema mais difícil na classe NP. O TSP é um problema que faz a seguinte pergunta:

> Dado um conjunto de n cidades e suas distâncias, qual é o menor caminho que passa por todas as cidades apenas uma vez e volta à cidade de origem?

O TSP é um problema intrinsecamente relacionado aos problemas do caminho Hamiltoniano (HPP) e do ciclo Hamiltoniano (HCP). O primeiro é um problema que busca saber se, dado um grafo direcionado ou não-direcionado, este grafo contém um caminho que passa por todos os vértices apenas uma vez, enquanto o segundo problema busca saber se há um caminho que passa por todos os vértices e retorna à origem. É fácil perceber que o problema do ciclo Hamiltoniano é uma extensão do problema do caminho Hamiltoniano. Se um grafo contém ao menos um caminho Hamiltoniano, cujo último vértice tem uma aresta com o vértice de origem, então há um ciclo Hamiltoniano.

Os problemas do caminho e do ciclo Hamiltoniano são problemas de decisão e, portanto, apenas respondem se existem tais caminhos e ciclos. No entanto, um grafo pode conter diversos ciclos Hamiltonianos e, se o grafo tiver pesos nas arestas, esses ciclos podem ter diferentes somas para os seus pesos. Dessa forma, o TSP pode ser visto como um problema que busca o ciclo Hamiltoniano com a menor soma de seus pesos, o que implica na redução do TSP no HCP. Em 1972, Karp (1972) demonstrou que o HCP é NP-completo, implicando que o TSP seja um problema NP-difícil. Formalmente, podemos definir o TSP, usando a teoria dos grafos, da seguinte forma:

Modelo: O modelo é um grafo $G = (V, E)$, onde:

- V é o conjunto de vértices, representando as n cidades.
- E é o conjunto de arestas, onde a aresta (i, j) existe conectando qualquer par de vértices (cidades) i e j.
- Um peso $c(i, j)$ é associado a cada aresta, representando a distância ou custo, de ir da cidade i para a cidade j.

Objetivo: O objetivo do TSP é encontrar um ciclo Hamiltoniano com o peso total mínimo. Se p = (v1, v2,...,vn,v1) é uma permutação de um caminho no grafo, então o objetivo é minimizar:

$$C(p) = \sum_{i=1}^{n-1} c(v_i, v_{i+1}) + c(v_n, v_1) \tag{2.1}$$

A partir da definição acima, se considerarmos um grafo completo, então existem $(n-1)!$ ciclos hamiltonianos que são resultados possíveis para o Problema do Caixeiro Viajante. Sendo assim, um algoritmo de força bruta levaria um tempo assintótico de $O(n) = (n-1)!$ para verificar quais ciclos hamiltonianos satisfazem o Problema do Caixeiro Viajante, podendo ser potencialmente mais de um. No entanto, se considerarmos um grafo não completo, então não podemos afirmar que o grafo tenha algum ciclo hamiltoniano sendo necessário verificar se existe algum primeiro. Dessa forma, pode-se considerar que um algoritmo que resolva, de forma geral, o Problema do Caixeiro Viajante deve encontrar todos os ciclos hamiltonianos em um grafo. Se houver algum, deve-se calcular o custo total desses ciclos e filtrar quais ciclos têm o custo minimo, de modo a serem solução do problema.

2.2 Caixeiro Viajante Multiobjetivo

Apesar de o Problema do Caixeiro Viajante ser computacionalmente interessante e bastante estudado em ciência da computação, problemas e aplicações práticas frequentemente exigem a consideração simultânea de múltiplos critérios de desempenho, que são frequentemente conflitantes. Por exemplo, considere um carro que precise percorrer todas as cidades em uma região no menor tempo possível. Porém, se o motorista fizer isso, o consumo de combustível aumenta consideravelmente, sendo, portanto, necessário minimizar tanto o tempo percorrido quanto o consumo de combustível. Uma extensão do Problema do Caixeiro Viajante que leva isso em consideração é a sua versão multiobjetivo (MOTSP na sigla em inglês).

Ao contrário do TSP mono-objetivo, onde se busca um único escalar ótimo global, o MOTSP visa otimizar um vetor de funções objetivo $F(x) = [f_1(x), f_2(x), \ldots, f_k(x)]$. Exemplos práticos incluem o balanceamento entre a minimização da distância percorrida, a minimização do tempo total de trânsito e a mitigação de riscos ou emissões de carbono.

Do ponto de vista da complexidade computacional, o TSP clássico é classificado como NP-difícil por Garey e Johnson (1979). O MOTSP, sendo uma generalização deste, herda essa intratabilidade e adiciona uma camada de complexidade: a inexistência, na maioria dos casos, de uma única solução que minimize todos os objetivos simultaneamente.

Consequentemente, o conceito de otimalidade é redefinido por meio da Dominância de Pareto. Uma solução x é dita dominar uma solução y se, e somente se, x for pelo menos tão boa quanto y em todos os objetivos e estritamente melhor em pelo menos um, de acordo com Deb (2001). O objetivo do MOTSP, portanto, não é encontrar uma única rota, mas sim aproximar o Conjunto Ótimo de Pareto — o conjunto de todas as soluções não-dominadas. A projeção dessas soluções no espaço de objetivos forma a Fronteira de Pareto.

2.3 Fronteira de Pareto

Como descrito na seção anterior, para o problema multiobjetivo do Problema do Caixeiro Viajante, a otimalidade é definida através da Dominância de Pareto, conceito introduzido por

Vilfredo Pareto. A formalização matemática dessa relação, amplamente utilizada em algoritmos de otimização, é definida por Deb (2001) da seguinte forma para um problema de minimização:

Diz-se que a solução x domina a solução y (denotado por $x \preceq y$) se e somente se:

- $f_i(x) \leq f_i(y)$ para todo objetivo $i \in \{1, \ldots, k\}$.
- $f_j(x) < f_j(y)$ para pelo menos um objetivo $j \in \{1, \ldots, k\}$.

Uma solução x^* é considerada Ótima de Pareto se nenhuma outra solução factível x a dominar.

A Fronteira de Pareto emerge da distinção entre os espaços de decisão e objetivo:

- Conjunto Ótimo de Pareto ($\mathcal{P}$): É o conjunto de todas as rotas ótimas de Pareto no espaço de decisão (o domínio das rotas).
- Fronteira de Pareto ($\mathcal{PF}$): É a imagem do Conjunto Ótimo de Pareto no espaço objetivo. Ou seja:

$$\mathcal{PF} = \{\mathbf{F}(x) \mid x \in \mathcal{P}\} \tag{2.2}$$

A $\mathcal{PF}$ constitui a fronteira de eficiência do problema, representando o *trade-off* máximo possível entre os objetivos conflitantes. Segundo Ehrgott (2005), para o MOTSP, devido à sua natureza combinatória e discreta, a $\mathcal{PF}$ é tipicamente composta por um conjunto finito e não-contínuo de pontos no espaço objetivo.

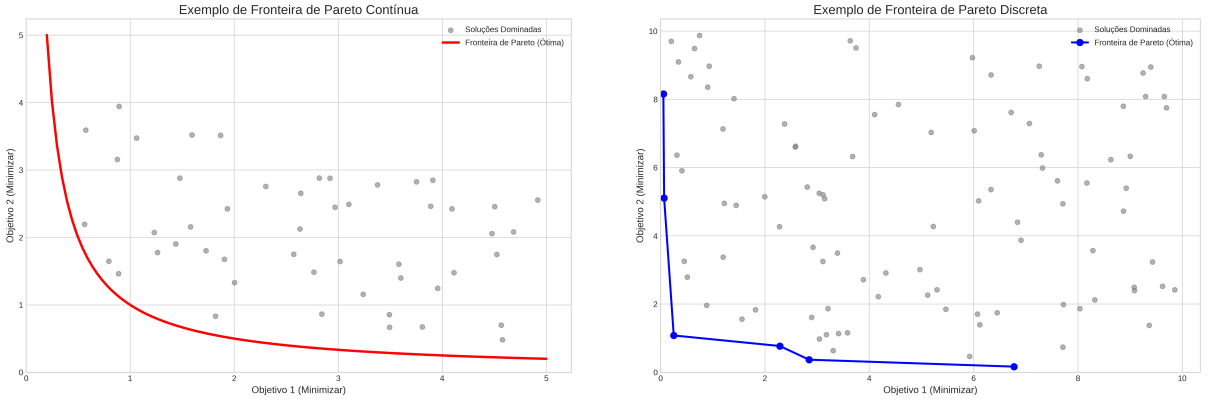
A complexidade do MOTSP (que é NP-difícil) implica que a determinação exata da $\mathcal{PF}$ é inviável para problemas de grande escala. O foco da pesquisa se concentra, portanto, na aproximação da Fronteira de Pareto, buscando uma alta qualidade em termos de:

- Convergência: A proximidade da Fronteira de Pareto aproximada em relação à $\mathcal{PF}$ verdadeira.
- Diversidade/Espalhamento: A capacidade de a solução aproximada cobrir uniformemente toda a extensão da $\mathcal{PF}$, oferecendo uma ampla gama de opções de *trade-off*.

Meta-heurísticas populacionais, como o NSGA-II (*Non-dominated Sorting Genetic Algorithm II*), proposto por Deb et al. (2002), são ferramentas padrão que utilizam a Dominância de Pareto como critério de aptidão (*fitness*) para evoluir a população de rotas, buscando soluções que maximizem simultaneamente a convergência e o espalhamento na $\mathcal{PF}$. Este algoritmo será apresentado na seção 3.3.1 do capítulo 3.

A utilidade da Fronteira de Pareto no MOTSP é prática e direta. Ela transforma um problema de otimização em um problema de suporte à decisão. Em vez de entregar uma única rota (que poderia ser a mais curta, mas a mais cara), a $\mathcal{PF}$ fornece ao gestor de logística o catálogo completo das melhores opções de *trade-off* possíveis. O decisor pode então escolher o ponto na fronteira que melhor se alinha com as prioridades operacionais atuais (por exemplo, priorizar o custo durante uma crise econômica ou priorizar o tempo de entrega durante um pico de demanda).

A Figura 2.1 ilustra exemplos de fronteiras de Pareto. A primeira imagem (a) mostra uma fronteira contínua, típica de problemas onde as variáveis de decisão podem assumir qualquer valor real. Nesse caso, a fronteira forma uma curva ou superfície contínua no espaço de objetivos. Em contraste, a segunda imagem (b) representa uma fronteira discreta, característica de problemas combinatórios como o MOTSP. Devido à natureza discreta do espaço de soluções, a fronteira é composta por um conjunto de pontos distintos, cada um correspondendo a uma solução de compromisso específica.



(a) Fronteira de Pareto Contínua.

(b) Fronteira de Pareto Discreta.

Figura 2.1: Exemplos de Fronteira de Pareto para um problema de minimização com dois objetivos. (a) Fronteira contínua, onde o espaço de soluções é contínuo. (b) Fronteira discreta, composta por um conjunto de pontos, típica de problemas combinatórios como o MOTSP.

2.4 Algoritmos Genéticos

A resolução do Problema do Caixeiro-Viajante Multiobjetivo (MOTSP) impõe desafios computacionais significativos devido à natureza NP-difícil do problema combinatório subjacente e à necessidade de otimizar simultaneamente funções objetivo conflitantes. Nesse cenário, os Algoritmos Genéticos (AGs), inseridos no campo da Computação Evolutiva, consolidaram-se como a abordagem meta-heurística predominante para a aproximação eficiente da Fronteira de Pareto.

A principal vantagem teórica dos Algoritmos Genéticos na otimização multiobjetivo reside na sua natureza populacional. Diferentemente de métodos baseados em trajetória (como o *Simulated Annealing*), que processam uma única solução por iteração, os AGs mantêm uma população de soluções candidatas (P_t). Segundo Coello Coello et al. (2007), isso permite que o algoritmo explore e mantenha, em uma única execução, um conjunto diversificado de soluções não-dominadas, aproximando-se da Fronteira de Pareto ($\mathcal{PF}$) em paralelo.

Os Algoritmos Genéticos (AGs) constituem uma classe de meta-heurísticas de busca estocástica fundamentada nos princípios da evolução natural e da genética populacional. Conforme formalizado por Holland (1975), os AGs operam sob a premissa darwiniana da "sobrevivência do mais apto", onde uma população de soluções candidatas evolui iterativamente em direção a uma solução ótima global, conforme descrito por Coello Coello et al. (2007).

O funcionamento de um AG baseia-se na manipulação de uma população de indivíduos, onde cada indivíduo (genótipo) representa uma solução potencial (fenótipo) para o problema de otimização em questão. O processo de busca é governado por operadores probabilísticos que simulam mecanismos biológicos: seleção, recombinação (*crossover*) e mutação.

O ciclo evolutivo pode ser descrito através das seguintes etapas sequenciais:

- Inicialização: O processo inicia-se com a geração, tipicamente aleatória, de uma população inicial $P(0)$ composta por N indivíduos.
- Avaliação (Fitness): Cada indivíduo x_i da população é avaliado por uma função objetivo $f(x)$, que atribui um valor de aptidão (*fitness*) proporcional à qualidade da solução. Segundo Goldberg (1989), esta métrica guia o processo de busca.

- Seleção: Um subconjunto de indivíduos é selecionado para reprodução. O mecanismo de seleção (e.g., Roleta, Torneio) é estocástico, mas enviesado para favorecer indivíduos com maior aptidão, aplicando uma pressão seletiva que direciona a evolução para regiões promissoras do espaço de busca.

- Recombinação (Crossover): Pares de indivíduos selecionados (progenitores) trocam material genético para gerar novos indivíduos (prole). Este é o operador primário de exploração (exploitation), combinando características de soluções boas para tentar gerar soluções ainda melhores. A ocorrência deste evento é governada por uma probabilidade de cruzamento (p_c).

- Mutação: Alterações aleatórias são introduzidas nos genes dos novos indivíduos com uma baixa probabilidade (p_m). A mutação atua como operador de diversificação (exploration), essencial para manter a diversidade genética da população e evitar a estagnação em ótimos locais, segundo Mitchell (1996).

- Substituição e Critério de Parada: A nova geração de indivíduos substitui a anterior (total ou parcialmente, podendo haver elitismo). O ciclo repete-se até que um critério de parada seja satisfeito (e.g., número máximo de gerações, convergência da população ou alcance de uma solução satisfatória).

O Algoritmo 1 apresenta um pseudocódigo para o funcionamento de um algoritmo genético genérico, o qual recebe como entrada o tamanho da população N e retorna a solução ótima aproximada $S_{\text{ótima}}$.

Algoritmo 1 Estrutura geral de um Algoritmo Genético (AG). O pseudocódigo descreve o ciclo evolutivo, desde a inicialização da população até a seleção da melhor solução após o critério de parada.

Require: N (Tamanho da População), CritérioParada
Ensure: $S_{\text{ótima}}$ (Melhor solução aproximada)
$P \leftarrow$ InicializarPopulação(N)
AvaliarAptidão(P)
while ¬CritérioParada(P) **do**
  $P_{\text{selecionada}} \leftarrow$ Seleção(P)
  $P_{\text{nova}} \leftarrow$ Recombinação($P_{\text{selecionada}}$)
  $P_{\text{nova}} \leftarrow$ Mutação(P_{nova})
  AvaliarAptidão(P_{nova})
  $P \leftarrow$ SubstituiçãoGeracional(P, P_{nova})
end while
return ObterMelhorSolução(P)

A eficácia dos Algoritmos Genéticos reside no equilíbrio dinâmico entre a exploração de novas regiões do espaço de busca (via mutação) e o refinamento de soluções já conhecidas (via cruzamento e seleção). Embora não garantam a otimalidade global em tempo polinomial, eles oferecem soluções robustas e de alta qualidade para problemas complexos, não-lineares e não-diferenciáveis, onde, segundo Eiben e Smith (2015), métodos analíticos tradicionais falham.

2.5 Inteligência de Enxame

A Inteligência de Enxame (SI, do inglês *Swarm Intelligence*) é um paradigma da Inteligência Artificial que se inspira no comportamento coletivo de sistemas descentralizados e auto-organizados, como colônias de formigas, cardumes de peixes ou bandos de pássaros. Proposto formalmente no final da década de 1980, o conceito descreve como interações locais entre agentes simples, que seguem regras básicas, podem levar ao surgimento de um comportamento global "inteligente" e coordenado, capaz de resolver problemas complexos sem a necessidade de um controle centralizado, como definido por Bonabeau et al. (1999).

O funcionamento dos algoritmos de SI baseia-se em princípios fundamentais observados na natureza:

- **Auto-organização:** A ordem e a estrutura globais do sistema emergem espontaneamente das interações locais entre os agentes, sem um plano ou guia externo.
- **Comunicação Indireta (Estigmergia):** Em muitos sistemas de SI, os agentes comunicam-se de forma indireta, modificando o ambiente. Um exemplo clássico é a trilha de feromônio deixada por formigas, que guia outras formigas em direção a fontes de alimento. De acordo com Dorigo et al. (2006), esse mecanismo permite uma coordenação assíncrona e robusta.
- **Equilíbrio entre Exploração e Explotação:** O sucesso do enxame depende de um balanço dinâmico entre a exploração de novas áreas do espaço de busca (*exploration*) e a explotação de regiões promissoras já conhecidas (*exploitation*).

Dois dos algoritmos mais representativos da Inteligência de Enxame são a Otimização por Colônia de Formigas (ACO) e a Otimização por Enxame de Partículas (PSO).

Otimização por Colônia de Formigas (ACO) Proposto por Dorigo (1992), o ACO é uma meta-heurística inspirada no comportamento de forrageamento de formigas reais, sendo particularmente eficaz para problemas de otimização combinatória como o TSP. No ACO, agentes computacionais, denominados "formigas", constroem soluções de forma incremental e probabilística. A comunicação entre as formigas ocorre de forma indireta (estigmergia), através do depósito de uma substância análoga ao feromônio nas arestas do grafo que compõem as soluções. A probabilidade de uma formiga escolher uma determinada aresta é função tanto da intensidade do feromônio (reforço positivo) quanto de uma informação heurística (e.g., o inverso da distância). Esse mecanismo de realimentação positiva faz com que os caminhos pertencentes a soluções de alta qualidade sejam progressivamente reforçados, guiando a busca do enxame em direção a regiões ótimas do espaço de soluções.

Otimização por Enxame de Partículas (PSO) Proposto por Kennedy e Eberhart (1995) e inspirado no comportamento social de bandos de pássaros, o PSO é uma meta-heurística populacional geralmente aplicada a problemas de otimização em espaços contínuos. No PSO, cada "partícula" representa uma solução candidata e "voa" pelo espaço de busca. O movimento de cada partícula é influenciado por sua própria melhor experiência passada (componente cognitivo, $pbest_i$) e pela melhor experiência de seus vizinhos ou de todo o enxame (componente social, $gbest$).

A atualização da velocidade e da posição de cada partícula i a cada iteração t é governada, segundo Poli et al. (2007), pelas seguintes equações:

$$v_i(t+1) = w \cdot v_i(t) + c_1 \cdot r_1 \cdot (pbest_i - x_i(t)) + c_2 \cdot r_2 \cdot (gbest - x_i(t)) \tag{2.3}$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{2.4}$$

Onde w é o peso de inércia, que controla a influência da velocidade anterior; c_1 e c_2 são os coeficientes de aceleração cognitivo e social, respectivamente; e r_1, r_2 são números aleatórios que introduzem estocasticidade na busca.

Diferentemente dos Algoritmos Genéticos, a Inteligência de Enxame não utiliza operadores explícitos de cruzamento e mutação. A "inteligência" emerge da cooperação e da competição entre os agentes, que compartilham informações de forma direta ou indireta para guiar a busca coletiva.

O Algoritmo 2 descreve a estrutura geral de um algoritmo de Inteligência de Enxame.

Algoritmo 2 Estrutura geral de um algoritmo de Inteligência de Enxame (SI). O pseudocódigo ilustra o processo iterativo onde agentes atualizam suas posições com base em memória individual e coletiva.

Require: N (Tamanho do Enxame), CritérioParada
Ensure: $S_{\text{ótima}}$ (Melhor solução encontrada pelo enxame)

```
E ← InicializarEnxame(N)    // Cria N agentes com posições/soluções aleatórias
AvaliarAptidão(E)
InicializarMemóriaColetiva(E)    // e.g., gbest no PSO, matriz de feromônio no ACO
while ¬CritérioParada(E) do
    for cada agente a ∈ E do
        AtualizarPosição(a, MemóriaColetiva)   // Movimento baseado na informação social/estigmergia
        AvaliarAptidão(a)
        AtualizarMemóriaIndividual(a)    // e.g., pbest no PSO
    end for
    AtualizarMemóriaColetiva(E)
end while
return ObterMelhorSolução(E)
```

Capítulo 3

Materiais e Metodologia

Este capítulo detalha os materiais e a metodologia empregados para a análise comparativa dos algoritmos de otimização multiobjetivo. Primeiramente, são apresentadas as fontes de dados utilizadas, com foco no formato TSPLIB e na metodologia para a geração de instâncias do Problema do Caixeiro-Viajante Multiobjetivo (MOTSP). Em seguida, são definidas as métricas de avaliação, que abrangem tanto a qualidade da solução (Hipervolume) quanto a eficiência computacional (tempo e memória). A seção subsequente descreve os fundamentos teóricos dos algoritmos selecionados para o estudo, organizados por seus paradigmas de funcionamento. Por fim, é detalhado o protocolo experimental, que estabelece as condições de teste e o procedimento de coleta de dados para garantir uma comparação justa e rigorosa.

3.1 Dados

O TSPLIB é um repositório amplamente reconhecido de instâncias de problemas de Otimização Combinatória, sendo o mais famoso o Problema do Caixeiro-Viajante (TSP). A estrutura padronizada de seus arquivos facilita o desenvolvimento e a avaliação comparativa de algoritmos, tanto para versões mono-objetivo quanto multiobjetivo.

Os arquivos de dados do TSPLIB (geralmente com extensão .tsp ou .par) são formatados em um cabeçalho simples seguido pela seção de dados. Essa estrutura é essencial para que os *parsers* de algoritmos possam interpretar corretamente a instância do problema.

Cabeçalho (*Header*)

O cabeçalho define as características cruciais da instância, conforme detalhado na Tabela 3.1.

Tabela 3.1: Descrição dos campos do cabeçalho (*header*) de um arquivo de instância do TSPLIB.

Campo	**Descrição**
NAME	Nome da instância (e.g., `berlin52`).
TYPE	Tipo de problema (e.g., `TSP`, `ATSP`, `SOP`).
COMMENT	Informações adicionais.
DIMENSION	Número total de nós (cidades) no problema.

Campo	Descrição
EDGE_WEIGHT_TYPE	Define como os pesos (distâncias) são fornecidos: `EXPLICIT` (matriz completa), `EUC_2D` (coordenadas euclidianas 2D), `CEIL_2D` (euclidianas arredondadas para cima), entre outros.
EDGE_WEIGHT_FORMAT	Aplicável se `TYPE` for `EXPLICIT`, define o formato da matriz de distâncias (e.g., `FULL_MATRIX`, `LOWER_DIAG_ROW`).

Seção de Dados

- **NODE_COORD_SECTION:** Usado quando `EDGE_WEIGHT_TYPE` é `EUC_2D` ou similar. Fornece as coordenadas (x, y) de cada nó.

```
NODE_COORD_SECTION
1 565 575
2 25 185
...
```

- **EDGE_WEIGHT_SECTION:** Usado quando `EDGE_WEIGHT_TYPE` é `EXPLICIT`. Fornece a matriz de custos (distâncias) diretamente, seguindo o formato especificado no cabeçalho.

```
EDGE_WEIGHT_SECTION
0 10 20 5
10 0 15 25
...
```

No TSP clássico, a meta é minimizar o custo total de uma única rota fechada que visita cada nó exatamente uma vez. **Matriz de Adjacência:** A matriz de distâncias gerada a partir dos dados do TSPLIB representa o único objetivo a ser minimizado.

O MOTSP envolve a otimização simultânea de dois ou mais objetivos conflitantes. No contexto do TSPLIB, isso é frequentemente tratado por arquivos de dados múltiplos. Tipicamente, é definido usando várias matrizes de adjacência para a mesma instância de cidade. Por exemplo, o MOTSP-2 (biobjetivo) usaria:

1. **Matriz 1 (Objetivo 1):** Custo ou Distância Euclidiana.
2. **Matriz 2 (Objetivo 2):** Tempo de Viagem, Emissão de Carbono, ou Risco (geralmente fornecida por um segundo arquivo ou calculada diferentemente).

Por exemplo, se o problema busca minimizar (Distância Total, Custo de Combustível Total), o algoritmo precisa de duas matrizes de adjacência (Distância e Custo) para cada par de cidades.

3.1.1 Geração da Matriz de Adjacência (Custos)

A matriz de adjacência D é a estrutura de dados fundamental para qualquer algoritmo de solução do TSP/MOTSP. Ela armazena o custo (ou distância) da aresta que conecta o nó i ao nó j, ou seja, $D_{i,j}$.

Caso 1: Dados de Coordenadas (`EUC_2D`, `CEIL_2D`, etc.)

O cálculo segue a distância Euclidiana, com a aplicação da regra de arredondamento:

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

- Para `EUC_2D`: O resultado é geralmente arredondado para o inteiro mais próximo.
- Para `CEIL_2D`: O resultado é arredondado para o próximo inteiro maior ($\lceil d_{i,j} \rceil$).

Caso 2: Matriz Explícita (`EXPLICIT`)

Quando o `EDGE_WEIGHT_TYPE` é `EXPLICIT`, a matriz é lida diretamente, seguindo o formato especificado em `EDGE_WEIGHT_FORMAT` (e.g., `FULL_MATRIX`, `UPPER_ROW`) para preencher a matriz D.

Matriz Resultante

A matriz de adjacência D terá dimensões $N \times N$, onde N é a dimensão.

- $D_{i,i}$ (custo do nó para ele mesmo) é geralmente 0 ou ∞ para evitar *loops* indesejados.
- Se o problema for simétrico (TSP), $D_{i,j} = D_{j,i}$.
- Se o problema for assimétrico (ATSP), $D_{i,j} \neq D_{j,i}$.

3.1.2 Geração de Problemas Multiobjetivo a partir de Instâncias Mono-objetivas do TSP

A construção de um Problema do Caixeiro-Viajante Multiobjetivo (MOTSP) a partir de uma instância padrão mono-objetivo (MOSP), tipicamente extraída de repositórios como o TSPLIB, exige a definição de um segundo objetivo que seja intrinsecamente conflitante com o objetivo primário de minimização de distância. O método mais robusto para garantir esse *trade-off* é definir o custo do segundo objetivo por meio de uma relação inversa ou recíproca com o custo original.

O Modelo Mono-objetivo Base (f_1)

O problema base é o TSP clássico. Dada uma rota (π) e a matriz de distâncias (D) da instância MOSP, o primeiro objetivo (f_1) é a minimização da distância total:

$$f_1(\pi) = \sum_{(i,j)\in\pi} d_{i,j} \tag{3.1}$$

Onde $d_{i,j}$ é a distância entre as cidades i e j.

Geração do Objetivo Conflitante f_2 através de Relações Inversas

Para que o problema seja um MOTSP válido, o algoritmo deve ser forçado a fazer escolhas difíceis. Uma maneira de criar um objetivo conflitante sintético é definir uma segunda matriz de custos que penalize as soluções que são boas para f_1. A matriz de custo para o segundo objetivo ($C^{(2)}$) pode ser definida pela relação recíproca, onde o custo da aresta (i, j) é modelado como o inverso da distância primária $d_{i,j}$:

A matriz de custo para o segundo objetivo ($C^{(2)}$) é definida pela relação recíproca/inversa:O custo da aresta (i, j) para o objetivo f_2 é modelado como o inverso da distância primária $d_{i,j}$:

$$c_{i,j}^{(2)} = g(d_{i,j}) = \frac{1}{d_{i,j}} \tag{3.2}$$

Esta relação inversa define o custo $c_{i,j}^{(2)}$ como o recíproco da distância $d_{i,j}$. Ao minimizar o somatório desses custos, o algoritmo é incentivado a escolher arestas de longa distância, criando um conflito direto com a minimização de f_1.

Como a distância $d_{i,j}$ é sempre positiva, a função $g(d_{i,j})$ garante que:

- Se a distância $d_{i,j}$ for baixa (bom para f_1), a utilidade $c_{i,j}^{(2)}$ será alta.
- Se a distância $d_{i,j}$ for alta (ruim para f_1), a utilidade $c_{i,j}^{(2)}$ será baixa.

Geração do Objetivo f_2 através de outra Instância Mono-objetiva

A metodologia mais comum e controlada para gerar instâncias de teste para o MOTSP, amplamente adotada na literatura, conforme descrito por Lust e Teghem (2010) e Ehrgott (2005), envolve a combinação de duas instâncias mono-objetivo distintas, mas com o mesmo número de cidades (N). Essa abordagem permite que os dois objetivos sejam definidos por conjuntos de dados independentes, modelando cenários realistas onde os custos não são correlacionados (e.g., distância vs. tempo, custo vs. risco).

Definição da Estrutura do Problema

A geração do MOTSP requer que ambas as instâncias base possuam a mesma topologia e o mesmo número de nós (N), garantindo que o espaço de soluções (Π) seja idêntico.

Sejam $\mathcal{I}_1$ e $\mathcal{I}_2$ as duas instâncias mono-objetivo selecionadas (ambas com N cidades).

- Objetivo Primário (f_1): Definido pela matriz de adjacência de custos $D^{(1)} = [d_{i,j}^{(1)}]$ de $\mathcal{I}_1$. Tipicamente, $d_{i,j}^{(1)}$ representa a distância geográfica (extraída de um arquivo TSPLIB).
- Objetivo Secundário (f_2): Definido pela matriz de adjacência de custos $D^{(2)} = [d_{i,j}^{(2)}]$ de $\mathcal{I}_2$. Este objetivo representa um custo conflitante, como tempo de viagem, custo financeiro, ou emissão de poluentes.

O problema biobjetivo resultante é formulado como a otimização simultânea de f_1 e f_2:

$$\text{Minimizar} \quad \mathbf{F}(\pi) = (f_1(\pi), f_2(\pi)) \tag{3.3}$$

Onde, para qualquer rota válida π:

$$f_1(\pi) = \sum_{(i,j)\in\pi} d_{i,j}^{(1)}$$

$$f_2(\pi) = \sum_{(i,j)\in\pi} d_{i,j}^{(2)}$$

O conflito entre os objetivos é uma propriedade emergente da independência das matrizes de custo. Uma aresta que é curta (baixo custo) em $D^{(1)}$ pode ser longa (alto custo) em $D^{(2)}$, forçando os algoritmos a encontrar soluções de compromisso. A busca pela solução ótima para f_1 levará, em geral, a uma solução subótima para f_2, e vice-versa, o que garante a existência de uma Fronteira de Pareto não-trivial a ser explorada.

Para implementar essa abordagem, o processo requer a leitura de dois arquivos de instância distintos.

- Seleção de Instâncias: As matrizes $D^{(1)}$ e $D^{(2)}$ devem ser dissimilares. Por exemplo, $D^{(1)}$ pode ser gerada a partir de coordenadas euclidianas (distância mais curta), enquanto $D^{(2)}$ pode ser gerada por uma matriz de custos assimétrica (ATSP) que simula um custo maior em certas direções ou horários de pico (tempo de viagem).
- Conflito: O conflito surge porque uma aresta que é de baixo custo em $D^{(1)}$ (e, portanto, favorecida por f_1) pode ser de alto custo em $D^{(2)}$ (e, portanto, penalizada por f_2).

A independência das matrizes garante que a busca pela solução mínima de f_1 levará a soluções subótimas para f_2, resultando em uma Fronteira de Pareto não-trivial que os algoritmos multiobjetivo precisam explorar.

Metodologia de Implementação

A implementação requer um processo de parsing duplo:

1. Processamento da Instância 1 ($\mathcal{I}_1$): Leitura de um arquivo TSPLIB para gerar a primeira matriz de custos ($D^{(1)}$).
2. Processamento da Instância 2 ($\mathcal{I}_2$): Leitura de um segundo arquivo (geralmente com o mesmo número de nós) para gerar a segunda matriz de custos ($D^{(2)}$).

A geração do MOTSP por combinação de duas instâncias independentes é o método padrão de modelagem para problemas de otimização que envolvem o balanceamento de recursos heterogêneos.

3.2 Métricas

A avaliação e comparação de algoritmos para o MOTSP requer uma abordagem multifacetada, que considere não apenas a qualidade das soluções encontradas, mas também a eficiência computacional com que são obtidas. Para este fim, foram utilizadas três categorias de métricas: uma para a qualidade da solução (Hipervolume), duas para a eficiência (tempo e memória) e uma métrica agregada (Índice de Desempenho Combinado) para uma análise global.

3.2.1 Qualidade da Solução: Hipervolume

O Hipervolume (HV) é a métrica mais robusta e teoricamente sólida para avaliar a qualidade de um conjunto de soluções em otimização multiobjetivo. Ele mede o volume (ou área, em problemas biobjetivo) no espaço objetivo que é dominado pela Fronteira de Pareto aproximada (P^*) e limitado por um Ponto de Referência (*Reference Point*, R) predefinido.

$$HV(P^*) = \text{Volume}(\bigcup_{x \in P^*} \{y \in \mathbb{R}^k \mid x \preceq y \preceq R\}) \tag{3.4}$$

Onde $x \preceq y$ denota que x domina ou é igual a y. O Ponto de Referência R deve ser escolhido de modo que seja dominado por todas as soluções potenciais na verdadeira Fronteira de Pareto.
Segundo Zitzler et al. (2003), o principal aspecto do Hipervolume reside em sua capacidade de integrar, em um único escalar, os dois objetivos centrais da Otimização Multiobjetivo:

- Convergência (*Proximity*): Quanto mais próximo o conjunto P^* estiver da Fronteira de Pareto verdadeira ($\mathcal{PF}$), maior será o volume que ele dominará.
- Diversidade (*Spread*): O Hipervolume maximiza-se apenas quando as soluções estão bem distribuídas ao longo de toda a Fronteira de Pareto verdadeira ($\mathcal{PF}$). Um conjunto de soluções aglomeradas em uma única região resulta em um Hipervolume menor, mesmo que essa região seja ótima.

Esta propriedade de ser estritamente monotônica com a relação de Dominância de Pareto garante que, se um conjunto de soluções A domina um conjunto B, então o Hipervolume de A será inequivocamente superior ao de B.

No entanto, apesar de sua superioridade teórica, o cálculo exato do Hipervolume enfrenta intratabilidade computacional à medida que o número de objetivos (k) aumenta. O tempo de cálculo cresce exponencialmente com k (para $k \geq 3$), tornando-o impraticável para a Otimização de Muitos Objetivos (Many-Objective Optimization, $k > 3$).

3.2.2 Eficiência Computacional: Tempo e Memória

Além da qualidade da solução, a eficiência computacional é um fator crítico na avaliação de algoritmos, especialmente para problemas de grande escala. Neste trabalho, a eficiência foi avaliada por meio de duas métricas:

- **Tempo de Execução:** Mede o tempo de processamento total (em segundos) que um algoritmo leva para completar sua execução, desde a inicialização até atingir o critério de parada. Esta métrica reflete a complexidade computacional e a sobrecarga operacional (*overhead*) de cada abordagem.
- **Consumo de Memória:** Mede o pico de memória RAM (em megabytes) utilizado pelo algoritmo durante sua execução. Esta métrica é crucial para avaliar a escalabilidade do algoritmo em ambientes com recursos limitados.

Devido à natureza estocástica das meta-heurísticas e a pequenas variações no ambiente de execução, tanto o tempo quanto a memória são medidos em múltiplas execuções independentes. Os resultados são então agregados, calculando-se a média e o desvio padrão para fornecer uma medida robusta e representativa do desempenho de cada algoritmo.

3.2.3 Análise Agregada: Índice de Desempenho Combinado (IDC)

Para consolidar a análise multifacetada de desempenho em uma única medida, foi utilizado o Índice de Desempenho Combinado (IDC). Esta métrica sintética, fundamentada em técnicas de Tomada de Decisão Multicritério (MCDM) como o TOPSIS, proposto por Hwang e Yoon (1981), permite classificar os algoritmos considerando simultaneamente a qualidade da solução (maximização do Hipervolume) e a eficiência computacional (minimização de tempo e memória). O IDC transforma o problema de avaliação de algoritmos em um problema de distância a um ponto ideal, onde um melhor desempenho agregado corresponde a uma menor distância a esse ponto de referência ótimo.

O cálculo do IDC para cada algoritmo em uma dada instância do problema segue três passos:

1. **Normalização das Métricas:** Devido à incompatibilidade de unidades (volume, segundos, bytes), os valores médios de cada métrica — Hipervolume (HV), Tempo (T) e Memória (M) — são normalizados para o intervalo [0, 1] usando a mesma fórmula de normalização *Min-Max*:
$$\hat{x} = \frac{x - x_{\min}}{x_{\max} - x_{\min}} \tag{3.5}$$
Com esta normalização, um valor maior de Hipervolume se aproxima de 1 (ideal), enquanto um valor menor de Tempo ou Memória se aproxima de 0 (ideal).

2. **Definição do Ponto Ideal e Medida de Distância:** Após a normalização, o desempenho ideal para cada métrica é diferente. O Ponto Ideal ($\mathbf{P}_{\text{ideal}}$) no espaço normalizado, que representa um algoritmo hipotético perfeito, é definido como $\mathbf{P}_{\text{ideal}} = (\hat{HV} = 1, \hat{T} = 0, \hat{M} = 0)$. O IDC é então calculado com base na proximidade de um algoritmo a este ponto ideal.

3. **Cálculo do IDC:** A fórmula do IDC incorpora pesos (w_i) que refletem a importância relativa de cada métrica ($\sum w_i = 1$). Para que a interpretação seja "quanto maior, melhor", utiliza-se a distância inversa. Neste trabalho, foi adotada uma ponderação equilibrada: $w_{HV} = 0.5$, $w_T = 0.25$ e $w_M = 0.25$. O IDC para um algoritmo a com valores normalizados $A = (\hat{HV}_a, \hat{T}_a, \hat{M}_a)$ é dado por:
$$\text{IDC}_a = 1 - \sqrt{w_{HV} \cdot (1 - \hat{HV}_a)^2 + w_T \cdot (\hat{T}_a)^2 + w_M \cdot (\hat{M}_a)^2} \tag{3.6}$$
A expressão sob a raiz quadrada representa a distância Euclidiana ponderada do algoritmo a ao ponto ideal. O termo $(1 - \hat{HV}_a)^2$ calcula a distância ao ideal para o Hipervolume (1), enquanto os termos $(\hat{T}_a)^2$ e $(\hat{M}_a)^2$ calculam a distância ao ideal para Tempo e Memória (0). O algoritmo com o maior IDC é aquele que possui a menor distância ao ponto ideal, sendo considerado o de melhor desempenho combinado.

Apesar de sua utilidade para uma análise agregada, é crucial reconhecer as limitações do IDC. A principal delas é a subjetividade na atribuição dos pesos (w_i). A escolha de dar 50% de importância ao Hipervolume ($w_{HV} = 0.5$) e 25% para o tempo ($w_T = 0.25$) e 25% para a memória ($w_M = 0.25$) reflete a premissa de que a qualidade da solução é o objetivo primário de um algoritmo de otimização, sendo a eficiência computacional (tempo e memória combinados) um fator secundário, mas igualmente crítico para a viabilidade prática. Uma ponderação diferente (e.g., priorizando o tempo de execução) poderia alterar significativamente o ranking final dos algoritmos. Além disso, a consolidação de múltiplas métricas em um único índice, embora

conveniente, inevitavelmente mascara os *trade-offs* específicos de cada algoritmo. Portanto, o IDC deve ser interpretado como uma ferramenta de classificação geral, complementada pela análise individual de cada métrica de desempenho.

O IDC fornece, portanto, uma ferramenta objetiva para analisar o *trade-off* entre a busca por soluções de alta qualidade e a necessidade de algoritmos computacionalmente eficientes.

3.3 Algoritmos

Para a resolução do Problema do Caixeiro Viajante, tanto na versão mono-objetivo quanto na multiobjetivo, existem diversos algoritmos que utilizam heurísticas e meta-heurísticas. As abordagens selecionadas para este trabalho são apresentadas a seguir, classificadas de acordo com seu paradigma de funcionamento.

3.3.1 Algoritmos Baseados em Dominância de Pareto

Esta classe de algoritmos utiliza o conceito de dominância de Pareto como o principal motor de pressão seletiva para guiar a busca em direção à Fronteira de Pareto.

NSGA-II (*Non-dominated Sorting Genetic Algorithm II*)

O *Non-dominated Sorting Genetic Algorithm II* (NSGA-II), proposto por Deb et al. (2002), é amplamente reconhecido na literatura de computação evolutiva como um algoritmo de referência para otimização multiobjetivo. Sua concepção surgiu como uma resposta direta às ineficiências computacionais de seu predecessor, o NSGA, introduzindo três melhorias críticas: uma abordagem de classificação não-dominada rápida, um mecanismo de preservação de diversidade e a incorporação de elitismo explícito.

A mecânica de seleção do NSGA-II é baseada em dois atributos principais: o *Rank* de Não-Dominância e a Distância de Aglomeração (*Crowding Distance*). O motor primário de convergência do algoritmo é a classificação rápida não-dominada, que divide a população em fronteiras de Pareto hierárquicas ($F_1, F_2, \dots$). A primeira fronteira, F_1, contém as soluções que são completamente não-dominadas na população e, portanto, representa o melhor conjunto de soluções. As soluções desta fronteira recebem o *Rank* 1. O processo é repetido, removendo-se virtualmente a fronteira anterior, para encontrar as fronteiras subsequentes, atribuindo-lhes *Ranks* crescentes.

Para manter a diversidade e evitar a convergência prematura para uma única região da Fronteira de Pareto, o NSGA-II emprega a *Crowding Distance*. Para cada indivíduo dentro de uma mesma fronteira, calcula-se a distância média no espaço de objetivos entre ele e seus vizinhos imediatos. Soluções localizadas em regiões menos povoadas recebem um valor de distância maior, sendo preferidas durante a seleção. Indivíduos nas extremidades da Fronteira recebem uma distância infinita para garantir sua preservação.

O ciclo evolutivo do NSGA-II é elitista. A cada geração, a população de pais (P_t) e a de filhos recém-gerados (Q_t) são combinadas em uma população unificada de tamanho $2N$. Esta população unificada é então ordenada de acordo com o Rank de Não-Dominância. A nova população (P_{t+1}) é construída transferindo-se as melhores frentes ($F_1, F_2, \dots$) até que o tamanho da população, N, seja atingido. Se a última fronteira a ser incluída, F_L, exceder o espaço restante, o algoritmo utiliza a *Crowding Distance* como critério de desempate, selecionando os indivíduos de F_L com maior distância para compor a nova geração. Este mecanismo garante que tanto as melhores soluções (elitismo) quanto as mais isoladas (diversidade) sejam preservadas.

A Figura 3.1 ilustra o ciclo de seleção do NSGA-II. A população de pais (P_t) e a de filhos (Q_t) são unidas em uma população intermediária (R_t) de tamanho $2N$. Essa população é então ordenada em frentes de não-dominância ($F_1, F_2, \ldots$). A nova população (P_{t+1}) é formada pelas melhores frentes, e a *Crowding Distance* é usada como critério de desempate para selecionar indivíduos da última frente que não cabe inteiramente, garantindo a diversidade.

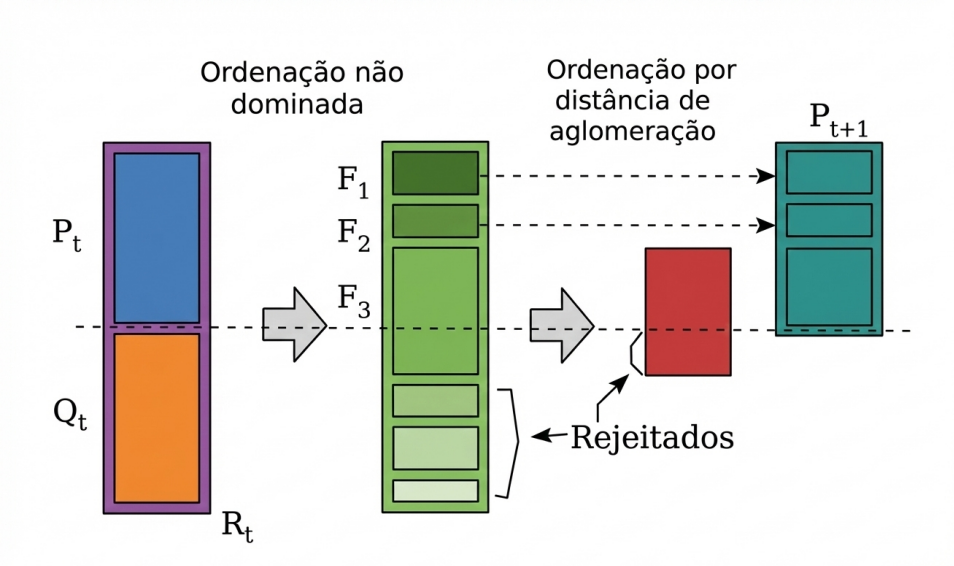


Figura 3.1: Procedimento do NSGA-II, de acordo com Deb et al. (2002).

SPEA2 (*Strength Pareto Evolutionary Algorithm* 2)

O *Strength Pareto Evolutionary Algorithm 2* (SPEA2), proposto por Zitzler et al. (2001), é um algoritmo evolutivo multiobjetivo que se destaca pelo uso de um arquivo externo para preservar soluções de elite e por uma sofisticada estratégia de atribuição de aptidão (*fitness*).

A mecânica do SPEA2 envolve uma população principal e um arquivo externo que armazena as soluções não-dominadas encontradas. A cada geração, a aptidão de todos os indivíduos (da população e do arquivo) é calculada. A principal inovação do algoritmo é sua função de *fitness*, que considera dois componentes: a força de uma solução, que é o número de outras soluções que ela domina; e uma aptidão bruta, que é a soma das forças das soluções que a dominam. Uma solução não-dominada tem, por definição, uma aptidão bruta igual a zero.

Para manter a diversidade e diferenciar soluções com a mesma aptidão bruta (por exemplo, múltiplas soluções não-dominadas), o SPEA2 incorpora uma estimativa de densidade. Diferente da *crowding distance* do NSGA-II, este mecanismo se baseia na distância para o k-ésimo vizinho mais próximo (k-NN), oferecendo uma medida mais robusta da densidade local. A aptidão final de uma solução é a soma de sua aptidão bruta e de sua densidade, favorecendo soluções que são não-dominadas e que se encontram em regiões menos povoadas da fronteira.

A seleção ambiental atualiza o arquivo externo com as melhores soluções. Se o número de soluções não-dominadas exceder a capacidade do arquivo, um operador de truncamento é ativado, removendo iterativamente as soluções de regiões mais densas para garantir uma cobertura uniforme da fronteira de Pareto. Embora seu mecanismo de densidade seja computacionalmente mais custoso que o do NSGA-II, o SPEA2 é reconhecido por produzir fronteiras de Pareto com excelente distribuição.

GDE3 (*Generalized Differential Evolution* 3)

O *Generalized Differential Evolution 3* (GDE3), proposto por Kukkonen e Deb (2005), estende o algoritmo de Evolução Diferencial (DE) para problemas multiobjetivo. Ele se destaca por utilizar vetores de diferença para gerar perturbações e guiar a busca, combinando a robustez dos operadores de variação da DE com regras de seleção baseadas na dominância de Pareto e um mecanismo de gerenciamento de diversidade, permitindo que a população evolua em direção à Fronteira de Pareto e mantenha um bom espalhamento das soluções.

A mecânica do GDE3 envolve a geração de novos indivíduos através de operadores de mutação e cruzamento. Para cada indivíduo "alvo" ($\vec{x}_i$), um vetor mutante ($\vec{v}_i$) é criado utilizando-se outros três vetores distintos da população ($\vec{x}_{r1}, \vec{x}_{r2}, \vec{x}_{r3}$), conforme a equação:

$$\vec{v}_i = \vec{x}_{r1} + F \cdot (\vec{x}_{r2} - \vec{x}_{r3}) \tag{3.7}$$

Onde F é o fator de escala que controla a amplitude da perturbação. Em seguida, um vetor "teste" ou experimental ($\vec{u}_i$) é formado combinando componentes do vetor alvo e do vetor mutante, através de um cruzamento binomial governado por uma taxa de cruzamento (CR).

A seleção de indivíduos no GDE3 é baseada na dominância de Pareto entre o pai ($\vec{x}_i$) e o filho ($\vec{u}_i$). Se o filho dominar o pai, ele o substitui na próxima geração. Se o pai dominar o filho, o pai é mantido. Caso sejam incomparáveis (não-dominados entre si), ambos são mantidos, o que pode levar a um aumento temporário da população. Para controlar o tamanho da população de volta ao seu valor original (N), o GDE3 aplica um processo de truncamento que classifica a população expandida em fronteiras de Pareto não-dominadas e utiliza a métrica de *Crowding Distance* para eliminar indivíduos aglomerados, similar ao NSGA-II.

Para a aplicação a problemas combinatórios discretos, como o MOTSP, o GDE3, sendo um algoritmo naturalmente contínuo, utiliza a representação de *Random Keys*, proposta por Bean (1994), para traduzir suas soluções do espaço contínuo para permutações válidas. Nesta abordagem, a posição de cada partícula é um vetor de números reais, e a rota do TSP é obtida ordenando-se os índices desse vetor com base nos valores que ele contém. Contudo, a aplicação da Evolução Diferencial com essa técnica enfrenta o desafio da causalidade fraca. A premissa de que a diferença vetorial entre boas soluções aponta para direções promissoras não se mantém linearmente no espaço de permutações, o que pode dificultar a convergência suave do algoritmo em problemas de alta dimensionalidade.

3.3.2 Algoritmos Baseados em Decomposição

Estes algoritmos transformam o problema multiobjetivo em um conjunto de subproblemas escalares (mono-objetivo) que são otimizados simultaneamente de forma cooperativa.

MOEA/D (Multi-Objective Evolutionary Algorithm based on Decomposition)

O *Multi-Objective Evolutionary Algorithm based on Decomposition* (MOEA/D), proposto por Zhang e Li (2007), representa uma mudança de paradigma na computação evolutiva multiobjetivo. Diferente de algoritmos baseados em dominância de Pareto, como o NSGA-II, o MOEA/D decompõe o problema multiobjetivo em um conjunto de subproblemas de otimização escalar (mono-objetivo), que são resolvidos simultaneamente e de forma cooperativa.

A premissa central é que, ao definir um conjunto de vetores de peso uniformemente distribuídos, cada vetor pode guiar a busca para uma porção específica da fronteira de Pareto. O algoritmo define N subproblemas, cada um associado a um vetor de peso λ^i. Para converter o

vetor de objetivos em um valor escalar, o MOEA/D utiliza uma função de agregação, como a de Tchebycheff:

$$g^{te}(x|\lambda^i, z^*) = \max_{j=1\ldots M} \{\lambda^i_j \cdot |f_j(x) - z^*_j|\} \tag{3.8}$$

Onde z^* é um ponto de referência ideal. A eficiência do algoritmo reside na sua estrutura de otimização cooperativa e local. Para cada subproblema, define-se uma vizinhança composta pelos subproblemas cujos vetores de peso são mais próximos. A reprodução para gerar novas soluções ocorre, preferencialmente, entre indivíduos dessa vizinhança.

A cada geração, uma nova solução é gerada e avaliada em relação aos subproblemas vizinhos. Se a nova solução apresentar um valor escalar melhor para um subproblema vizinho, ela substitui a solução atual daquele vizinho. Essa cooperação local permite que o MOEA/D explore o espaço de busca de forma eficiente, mantendo a diversidade intrinsecamente através da distribuição dos vetores de peso e evitando o alto custo computacional do ranqueamento por não-dominância.

NSGA-III (*Non-dominated Sorting Genetic Algorithm III*)

O *Non-dominated Sorting Genetic Algorithm III* (NSGA-III), proposto por Deb e Jain (2014), é uma evolução do NSGA-II projetada para resolver problemas de Otimização de Muitos Objetivos (*Many-Objective Optimization Problems*, MaOPs), tipicamente com quatro ou mais objetivos. Em cenários de alta dimensionalidade, a maioria das soluções tende a ser não-dominada, tornando o critério de *ranking* de Pareto ineficaz para guiar a busca.

Para superar essa limitação, o NSGA-III substitui o mecanismo de diversidade baseado em *Crowding Distance* por uma abordagem estruturada em Pontos de Referência. O algoritmo distribui um conjunto de pontos de referência em um hiperplano normalizado no espaço objetivo. A seleção ambiental, então, prioriza soluções que estão próximas a esses pontos de referência, garantindo uma cobertura mais estruturada e uniforme da fronteira de Pareto.

A estrutura geral do NSGA-III é similar à do NSGA-II (geracional e elitista), mas a diferença fundamental reside na etapa de seleção, quando a população precisa ser truncada. Primeiramente, o algoritmo gera um conjunto de pontos de referência usando a abordagem sistemática proposta por Das e Dennis (1998). Para um problema de M objetivos e p divisões por eixo, o número total de pontos de referência (H) é dado por:

$$H = \binom{M + p - 1}{p} \tag{3.9}$$

A cada geração, os valores dos objetivos são normalizados dinamicamente, e cada solução da população é associada ao ponto de referência mais próximo, com base na menor distância perpendicular a uma linha de referência que conecta a origem ao ponto.

Quando a população precisa ser reduzida, o NSGA-III emprega uma operação de preservação de nicho (*niche-preservation*). O algoritmo identifica os pontos de referência com o menor número de soluções associadas (os nichos menos populosos) e seleciona, prioritariamente, indivíduos desses nichos para compor a próxima geração. Esse mecanismo força o algoritmo a manter uma diversidade estruturada, cobrindo uniformemente as direções de preferência definidas pelos pontos de referência. Embora projetado para MaOPs, a inclusão do NSGA-III neste estudo biobjetivo visa investigar se sua estrutura rígida oferece vantagens de convergência ou distribuição em comparação com a diversidade dinâmica de algoritmos como o NSGA-II.

3.3.3 Algoritmos Baseados em Indicadores

Esta classe de algoritmos utiliza uma métrica de qualidade, como o hipervolume, para guiar diretamente o processo de busca.

SMS-EMOA (*S-Metric Selection Evolutionary Multi-Objective Algorithm*)

O *S-Metric Selection Evolutionary Multi-objective Algorithm* (SMS-EMOA), proposto por Beume et al. (2007), representa a classe dos algoritmos baseados em indicadores. Diferente de abordagens que utilizam a dominância de Pareto puramente para seleção, como o NSGA-II, o SMS-EMOA incorpora uma métrica de qualidade quantitativa diretamente no processo evolutivo: o Hipervolume (também conhecido como *S-Metric*). A premissa teórica é que, ao maximizar o Hipervolume dominado pela população, o algoritmo simultaneamente maximiza a convergência e a diversidade, pois esta métrica captura ambas as características em um único valor escalar.

Uma distinção arquitetural crucial do SMS-EMOA é sua operação em um modelo de Estado Estacionário (*Steady-State*). Em cada iteração, apenas um novo indivíduo é gerado e adicionado à população de tamanho N, criando um conjunto temporário de $N + 1$ soluções. Em seguida, um procedimento de eliminação remove o "pior" indivíduo, retornando a população ao seu tamanho original. Este ciclo de atualização incremental permite uma pressão seletiva mais fina e imediata.

O critério para decidir qual indivíduo eliminar é hierárquico. Primeiro, a população unida ($N + 1$) é classificada em fronteiras de não-dominância, conforme descrito na Seção 3.3.1. O indivíduo a ser eliminado deve pertencer, obrigatoriamente, à pior fronteira (F_k). Se esta fronteira contiver mais de uma solução, o critério de desempate é a Contribuição Exclusiva de Hipervolume (ΔS). Para cada solução $s \in F_k$, calcula-se o quanto o hipervolume total da fronteira diminuiria se s fosse removida:

$$\Delta S(s, F_k) = S(F_k) - S(F_k \setminus \{s\}) \tag{3.10}$$

Onde $S(\cdot)$ é a medida de volume da união dos hipercubos dominados pelo conjunto. O algoritmo elimina a solução que possui o menor ΔS, pois sua remoção causa a menor perda de Hipervolume, preservando a integridade da fronteira aproximada.

A aplicação do SMS-EMOA ao MOTSP visa testar a hipótese de que a maximização direta da métrica alvo (Hipervolume) produz uma distribuição de soluções superior à heurística de *Crowding Distance* do NSGA-II. Teoricamente, o SMS-EMOA deve gerar a fronteira com a melhor uniformidade possível. Contudo, seu "custo por geração" é mais alto, o que pode resultar em uma convergência mais lenta em termos de tempo real de execução (*Wall-clock time*) para instâncias muito grandes, onde o cálculo da área para centenas de pontos torna-se oneroso.

3.3.4 Algoritmos Celulares e de Enxame

Esta categoria engloba algoritmos inspirados em sistemas biológicos, como a organização de células ou o comportamento coletivo de enxames.

MOCell (*Multi-Objective Cellular Genetic Algorithm*)

O *Multi-Objective Cellular Genetic Algorithm* (MOCell), proposto por Nebro et al. (2009b), pertence à classe dos Algoritmos Genéticos Celulares (cGA - *Cellular Genetic Algorithms*). Diferente dos algoritmos como o NSGA-II e o SPEA2, onde qualquer indivíduo

pode cruzar com qualquer outro da população (acasalamento global), no MOCell a população é estruturada espacialmente. Os indivíduos são dispostos em um Grid Bidimensional Toroidal, e a característica definidora deste modelo é a descentralização: as interações genéticas (seleção e cruzamento) ocorrem exclusivamente dentro de uma Vizinhança Local. Este modelo simula o isolamento por distância observado na natureza, promovendo uma difusão lenta de genes pela população. Isso permite a manutenção de diversos nichos de soluções simultaneamente (exploração/diversidade) enquanto realiza uma busca intensiva localmente (intensificação/convergência).

O MOCell combina o modelo celular canônico com um arquivo externo de elite para guiar a busca multiobjetivo. A população P de tamanho N é estruturada espacialmente em um grid de dimensões $L \times L$ (onde $N = L^2$). Para cada indivíduo localizado na coordenada (x, y), define-se uma vizinhança V_{xy}. Neste trabalho, utilizou-se a Vizinhança de Moore, que inclui os 8 vizinhos mais próximos de um indivíduo no grid, além do próprio indivíduo, totalizando 9 indivíduos. Esta topologia toroidal garante que todos os indivíduos, incluindo os das bordas, possuam o mesmo número de vizinhos.

Além da estrutura celular, o MOCell mantém um Arquivo Externo que armazena as soluções não-dominadas encontradas durante a busca. Este arquivo desempenha duas funções primordiais: Elitismo, preservando as melhores soluções descobertas ao longo da execução do algoritmo, e *Feedback*, onde periodicamente soluções do arquivo substituem indivíduos aleatórios no grid da população principal. Este mecanismo injeta "sangue novo" de alta qualidade na população estruturada, acelerando a convergência global em direção à Fronteira de Pareto real e evitando a estagnação.

O algoritmo itera sobre cada célula do grid. Para processar o indivíduo corrente Ind_{atual} na posição (x, y), o MOCell segue um ciclo evolutivo local. Primeiramente, dois pais são selecionados exclusivamente dentre os vizinhos de (x, y) na vizinhança V_{xy}, tipicamente por torneio ou roleta, aplicada aos *fitness* dos vizinhos. Em seguida, operadores genéticos como o Crossover PMX e a Mutação Swap são aplicados aos pais para gerar um novo indivíduo descendente, Ind_{novo}. Após a avaliação dos objetivos de Ind_{novo}, o algoritmo decide se Ind_{novo} substituirá Ind_{atual} baseado no princípio de dominância de Pareto. Se Ind_{novo} domina Ind_{atual}, Ind_{atual} é substituído. Se Ind_{atual} domina Ind_{novo}, Ind_{novo} é descartado. Caso sejam não-dominados entre si (incomparáveis), a substituição ocorre se Ind_{novo} tiver uma menor densidade (*Crowding Distance*) no contexto da vizinhança ou do arquivo, um critério que ajuda a manter a diversidade. Finalmente, o algoritmo tenta inserir Ind_{novo} no arquivo externo. Se Ind_{novo} não for dominado por nenhuma solução no arquivo e o arquivo não estiver cheio, ele é adicionado. Se o arquivo estiver cheio, a *Crowding Distance* é utilizada para remover o indivíduo mais aglomerado e abrir espaço, ou o indivíduo dominado pelo Ind_{novo}.

Quanto aos Operadores e Arquivo, o problema modelado foi o `BiObjectiveTSP` (Permutação). Para o crossover, utilizou-se o PMX com probabilidade $P_c = 0.9$. A aplicação deste operador em um contexto celular é interessante teoricamente: como os pais são vizinhos no grid, eles tendem a ser geneticamente similares, fazendo com que o crossover atue mais como um refinamento local do que como uma recombinação global disruptiva. A mutação empregada foi a Swap Mutation com probabilidade $P_m = 1/N$. Para o arquivo externo, utilizou-se o `CrowdingDistanceArchive` com capacidade igual ao tamanho da população (100). Este componente é crucial para assegurar que a diversidade global seja mantida, compensando a forte pressão de convergência local da vizinhança.

A aplicação do MOCell ao MOTSP testa a hipótese de que a difusão controlada de informações genéticas pode evitar a convergência prematura para ótimos locais — um problema comum em algoritmos panmíticos (como NSGA-II) quando lidam com espaços de busca

combinatórios massivos (como o fatorial do TSP). No entanto, o custo de comunicação restrita é uma convergência global potencialmente mais lenta, o que pode ser uma desvantagem se o número de avaliações for limitado.

SMPSO (Speed-constrained Multi-objective PSO)

A Otimização por Enxame de Partículas (PSO), originalmente concebida para problemas contínuos, é uma meta-heurística inspirada no comportamento social de bandos. O *Speed-constrained Multi-objective Particle Swarm Optimization* (SMPSO), proposto por Nebro et al. (2009a), é uma de suas mais robustas adaptações para o domínio multiobjetivo. O algoritmo representa cada solução como uma "partícula" que se move no espaço de busca, possuindo um vetor de posição ($\vec{x}_i$) e um vetor de velocidade ($\vec{v}_i$). O movimento é influenciado por sua melhor posição histórica (*pBest*) e pela melhor posição encontrada por todo o enxame (*gBest*).

A cada iteração t, a velocidade e a posição de cada partícula i são atualizadas de acordo com as seguintes equações:

$$\vec{v}_i(t+1) = \chi \cdot [\vec{v}_i(t) + c_1 r_1 (p\vec{B}est_i - \vec{x}_i(t)) + c_2 r_2 (g\vec{B}est - \vec{x}_i(t))] \tag{3.11}$$

$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i(t+1) \tag{3.12}$$

Onde χ é um fator de constrição que amortece a velocidade, c_1 e c_2 são coeficientes de aceleração que ponderam a influência cognitiva (individual) e social (coletiva), e r_1, r_2 são valores aleatórios que introduzem estocasticidade na busca.

O SMPSO aprimora o PSO canônico com três mecanismos chave. Primeiro, ele gerencia um arquivo externo (um repositório de elite) que armazena as melhores soluções não-dominadas encontradas durante a busca. O líder global, $g\vec{B}est$, de cada partícula é selecionado a partir deste arquivo utilizando a métrica de *Crowding Distance*, o que promove uma boa distribuição dos guias. Segundo, para evitar a convergência prematura e a estagnação, o algoritmo aplica um operador de mutação polinomial (uma "turbulência") a uma fração das partículas a cada iteração. Por fim, sua principal inovação é um coeficiente de constrição de velocidade, que limita a magnitude do "salto" de uma partícula a cada passo, prevenindo a divergência e melhorando a capacidade de exploração fina em torno de boas soluções.

Uma vez que o PSO e o SMPSO operam nativamente em espaços de busca contínuos, sua aplicação ao MOTSP (um problema discreto de permutação) requer uma técnica de mapeamento. Para este trabalho, foi utilizada a representação de *Random Keys*, proposta por Bean (1994), que traduz a posição contínua de uma partícula em uma rota discreta. Nesta abordagem, a posição de cada partícula é um vetor de números reais, e a rota do TSP é obtida ordenando-se os índices desse vetor com base nos valores que ele contém. A sequência de índices resultante forma a permutação de cidades a ser avaliada. Embora esta técnica permita a aplicação direta de algoritmos contínuos a problemas discretos, ela possui uma desvantagem teórica: pequenas mudanças no espaço contínuo podem levar a grandes e imprevisíveis alterações na permutação resultante, o que pode dificultar a convergência suave do algoritmo.

MOACO (*Multi-Objective Ant Colony Optimization*)

A Otimização por Colônia de Formigas (*Ant Colony Optimization*, ACO), introduzida por Dorigo (1992), é uma meta-heurística inspirada no comportamento de forrageamento de formigas, que utilizam comunicação indireta (estigmergia) via feromônios para encontrar os

caminhos mais curtos. A sua extensão para problemas multiobjetivo, o MOACO, adapta esse conceito para encontrar um conjunto de soluções de compromisso na Fronteira de Pareto.

Diferente dos algoritmos evolutivos, que modificam soluções completas, o MOACO é um algoritmo construtivo: cada "formiga" constrói uma solução (rota) passo a passo. A decisão de qual cidade visitar em seguida é probabilística, guiada por duas fontes de informação: a trilha de feromônio (τ_{ij}), que representa a memória coletiva sobre a qualidade de uma aresta, e a informação heurística (η_{ij}), que indica a atratividade *a priori* de uma aresta (geralmente o inverso da distância).

A probabilidade de uma formiga se mover da cidade i para a j é dada pela Equação 3.13, onde α e β são parâmetros que ponderam a importância do feromônio e da heurística, respectivamente.

$$P_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}]^{\alpha} \cdot [\eta_{il}]^{\beta}} \quad \text{se } j \in \mathcal{N}_i^k, \text{ caso contrário } 0 \tag{3.13}$$

Onde $\mathcal{N}_i^k$ é o conjunto de cidades ainda não visitadas pela formiga k. Para o cenário biobjetivo, a informação heurística η_{ij} foi calculada como o inverso da soma das distâncias em ambos os objetivos, agregando a informação de custo em um único valor.

A implementação utilizada neste trabalho, inspirada na abordagem P-ACO (*Pareto-Based ACO*), de Doerner et al. (2004), e em implementações de referência como a de López-Ibáñez (2005), utiliza uma única matriz de feromônio para guiar a busca. O ciclo de aprendizado é baseado em um mecanismo de evaporação e depósito. A cada iteração, uma porcentagem do feromônio em todas as arestas evapora, simulando o esquecimento e evitando a estagnação. Em seguida, apenas as formigas que construíram as melhores soluções — aquelas que pertencem à Fronteira de Pareto da iteração atual — são autorizadas a depositar feromônio nas arestas de suas rotas. Esse mecanismo de atualização elitista reforça agressivamente os caminhos que levam a soluções de alta qualidade, guiando o enxame de forma eficaz em direção à fronteira ótima.

3.4 Metodologia Experimental

Para avaliar e comparar o desempenho dos algoritmos de forma justa, sistemática e reproduzível, foi definida uma metodologia experimental rigorosa. O objetivo é submeter cada algoritmo às mesmas condições de teste e medir seu desempenho através de um conjunto consistente de métricas.

A unidade fundamental do experimento consiste na execução de um algoritmo específico sobre uma instância de problema particular. Para garantir uma comparação equitativa, os parâmetros de execução foram padronizados para todos os algoritmos sempre que possível. Isso inclui o estabelecimento de um critério de parada comum, como um número máximo de avaliações da função objetivo, e um tamanho de população fixo, assegurando que todos os algoritmos disponham de uma quantidade comparável de recursos computacionais para realizar a busca.

A performance de cada algoritmo em uma dada execução foi quantificada por meio de três métricas principais:

1. **Eficiência Temporal (Tempo de Execução):** Mede o tempo computacional total necessário para que o algoritmo complete sua execução. Esta métrica reflete a complexidade e a sobrecarga operacional de cada abordagem.

2. **Eficiência Espacial (Consumo de Memória):** Corresponde ao pico de memória alocada pelo processo durante a execução. Essa métrica avalia a demanda do algoritmo por recursos de memória, um fator crucial para sua aplicabilidade em ambientes com capacidade limitada.

3. **Qualidade da Solução (Hipervolume):** Avalia a qualidade do conjunto de soluções não-dominadas (a fronteira de Pareto aproximada) retornado pelo algoritmo. O Hipervolume (HV) é uma métrica de qualidade abrangente que mede a área no espaço de objetivos dominada pelo conjunto de soluções, relativa a um ponto de referência. Um valor de HV maior indica um melhor desempenho combinado em termos de convergência (proximidade da fronteira ótima) e diversidade (espalhamento das soluções). Para garantir a comparabilidade, um ponto de referência consistente foi utilizado para cada dimensão de problema.

Considerando a natureza estocástica das meta-heurísticas, uma única execução não é representativa do desempenho geral de um algoritmo. Portanto, o protocolo experimental estipula que cada par algoritmo-instância seja executado de forma independente múltiplas vezes. Os dados brutos coletados de cada uma dessas execuções (tempo, memória e HV) são então agregados para análise estatística.

A etapa final da metodologia consiste na consolidação desses dados, onde são calculadas estatísticas descritivas, como a média e o desvio padrão, para cada métrica. Essas estatísticas resumem o comportamento central e a variabilidade do desempenho de cada algoritmo, formando a base para a análise comparativa e as conclusões apresentadas no capítulo 5.

Capítulo 4

Implementação

Este capítulo descreve os detalhes da implementação desenvolvida para executar a análise comparativa dos algoritmos. O foco é apresentar a arquitetura do software, as ferramentas utilizadas e o fluxo de trabalho para a coleta e o processamento dos dados de desempenho. A implementação foi desenvolvida em **Python**, uma linguagem moderna e amplamente adotada, com um vasto ecossistema de bibliotecas para computação científica e análise de dados.

4.1 Arquitetura e Ferramentas

A arquitetura da solução foi construída sobre um conjunto de bibliotecas Python de código aberto, que forneceram a base para a experimentação com algoritmos de otimização multiobjetivo.

- **jmetalpy:** Versão em Python do conhecido *framework* Java jMetal por Benítez-Hidalgo et al. (2019). Foi a ferramenta central para este trabalho, fornecendo uma arquitetura robusta e implementações de alta qualidade para os algoritmos de otimização multiobjetivo (NSGA-II, SMPSO, MOEA/D, etc.). A principal vantagem do 'jmetalpy' é sua estrutura modular, que separa as responsabilidades entre a definição do problema, a lógica do algoritmo e a representação da solução. Isso permitiu que o foco do trabalho fosse a modelagem do MOTSP, enquanto se aproveitavam as implementações canônicas e validadas dos algoritmos, garantindo uma base de comparação justa e confiável.

- **Tracemalloc:** Biblioteca padrão do Python utilizada para a medição precisa do consumo de memória. Ela permite rastrear os blocos de memória alocados, fornecendo o uso de memória atual e o pico de alocação durante um trecho de código, sendo essencial para a análise de eficiência.

- **Pandas:** Biblioteca fundamental para a manipulação e análise de dados. Foi utilizada no pós-processamento para agregar os resultados brutos das execuções, calcular estatísticas descritivas (média, desvio padrão) e organizar os dados para a geração de tabelas.

- **Matplotlib:** Utilizada para a criação dos gráficos e visualizações apresentados no Capítulo de resultados, permitindo a plotagem das fronteiras de Pareto e dos gráficos comparativos de desempenho.

4.2 Ambiente Experimental

Todos os experimentos foram conduzidos em uma única máquina para garantir a consistência e a comparabilidade dos resultados de tempo e memória. As especificações de hardware e software do ambiente de teste foram as seguintes: Processador (CPU): Intel(R) Core(TM) i9-10900, com cache de 20MB e frequência de 2,80GHz (base) a 5,20GHz (turbo máximo). Memória RAM: 32 GB.

A utilização de um ambiente de execução fixo é crucial para eliminar variações de desempenho. É importante notar que, embora o processador possua múltiplos núcleos, todos os algoritmos foram executados de forma sequencial, utilizando apenas um núcleo, para garantir uma medição de tempo de processamento justa e comparável entre as diferentes abordagens. Cada algoritmo foi executado de forma isolada para que as medições de consumo de recursos fossem o mais precisas possível.

4.3 Estrutura do Experimento

O processo experimental foi orquestrado por meio de scripts Python individuais para cada algoritmo. Essa abordagem modular permite a configuração e execução isolada de cada experimento. A seguir, detalha-se a estrutura desses scripts.

Para garantir uma comparação justa, todos os algoritmos foram configurados com um tamanho de população de 100 indivíduos. O critério de parada foi baseado no número máximo de avaliações da função objetivo, sendo definido de duas formas:

- Para a maioria dos algoritmos (GDE3, NSGA-II, MOCell, MOEA/D, NSGA-III, SMPSO, SPEA2), o critério foi de $500 \times N$ avaliações, onde N é o número de cidades da instância. Isso resultou em 50.000 avaliações para 100 cidades, 100.000 para 200, 200.000 para 400 e 400.000 para 800 cidades.

- Para os algoritmos MOACO e SMS-EMOA, o critério foi de $100 \times N$ avaliações, resultando em 10.000 avaliações para 100 cidades, 20.000 para 200, 40.000 para 400 e 80.000 para 800 cidades.

Essa abordagem garante que todos os algoritmos disponham de um orçamento computacional proporcional à complexidade do problema, considerando as características de convergência de cada um.

4.3.1 Definição do Problema (MOTSP)

Para modelar o Problema do Caixeiro Viajante Multiobjetivo, foi criada uma classe ‘BiObjectiveTSP‘ que herda da classe base ‘PermutationProblem‘ do ‘jmetalpy‘. Essa herança é um ponto-chave da integração com o framework, pois permite “ensinar” aos algoritmos como o problema do TSP deve ser tratado. A classe ‘BiObjectiveTSP‘ é responsável por:

1. **Carregar as Instâncias:** Ler os dois arquivos de instância do problema (no formato TSPLIB) e construir as respectivas matrizes de distância, que representam os dois objetivos a serem minimizados.

2. **Avaliar uma Solução:** Implementar o método ‘evaluate()‘, que recebe uma solução (uma permutação de cidades) e calcula o custo total da rota para cada um dos dois objetivos, conforme as matrizes de distância.

O Código 4.1 ilustra o método de avaliação, que é o núcleo da definição do problema. Ao implementar este método, qualquer algoritmo do ‘jmetalpy‘ que opera sobre permutações pode ser aplicado diretamente para resolver o MOTSP, sem a necessidade de modificar a lógica interna do algoritmo.

Listing 4.1: Código do método ‘evaluate‘ da classe ‘BiObjectiveTSP‘.

```
def evaluate(self, solution: PermutationSolution) -> PermutationSolution:
    total_distance_1 = 0.0
    for i in range(self.number_of_variables() - 1):
        total_distance_1 += self.distance_matrix_1[solution.variables[i]][solution.variables[i+1]]
    total_distance_1 += self.distance_matrix_1[solution.variables[-1]][solution.variables[0]]

    total_distance_2 = 0.0
    for i in range(self.number_of_variables() - 1):
        total_distance_2 += self.distance_matrix_2[solution.variables[i]][solution.variables[i+1]]
    total_distance_2 += self.distance_matrix_2[solution.variables[-1]][solution.variables[0]]

    solution.objectives[0] = total_distance_1
    solution.objectives[1] = total_distance_2
    return solution
```

4.3.2 Execução e Coleta de Métricas

Para cada algoritmo, um script principal é responsável por configurar os parâmetros, executar o otimizador e coletar as métricas de desempenho. O ‘jmetalpy‘ facilita enormemente esse processo, permitindo que a configuração e execução de um algoritmo complexo como o NSGA-II sejam feitas em poucas linhas de código, como mostra o Código 4.2. A estrutura é padronizada: instancia-se o problema, instancia-se o algoritmo com seus respectivos parâmetros (tamanho da população, operadores de mutação e cruzamento, critério de parada) e, por fim, invoca-se o método ‘run()‘.

O fluxo de coleta de métricas é o seguinte:

1. **Medição de Memória:** A biblioteca ‘tracemalloc‘ foi iniciada antes da execução do algoritmo com ‘tracemalloc.start()‘.

2. **Medição de Tempo:** O tempo de início foi capturado com ‘time.time()‘ imediatamente antes da chamada do método ‘run()‘ do algoritmo.

3. **Execução:** O algoritmo foi executado.

4. **Coleta dos Resultados:** Ao final da execução, o tempo final foi capturado, e o pico de memória (`peak`) foi obtido através de `tracemalloc.get_traced_memory()`. O ‘tracemalloc‘ foi então finalizado. A diferença de tempo fornece o tempo de execução, e o pico de memória é convertido para Megabytes (MB) para a análise.

4.3.3 Implementação Customizada do MOACO

Uma contribuição central deste trabalho foi o desenvolvimento de uma implementação própria do MOACO (*Multi-Objective Ant Colony Optimization*), uma vez que o algoritmo não estava disponível no *framework* ‘jmetalpy‘. A nova implementação foi desenvolvida em Python, baseando-se nos conceitos do P-ACO (*Pareto-Based Ant Colony Optimization*) de Doerner et al. (2004) e, mais especificamente, na implementação de referência em linguagem C de López-Ibáñez

Listing 4.2: Código de exemplo de execução de um algoritmo e coleta de métricas.

```
# Configura o algoritmo (ex: NSGA-II)
algorithm = NSGAII(
   problem=problem,
   population_size=100,
   offspring_population_size=100,
   mutation=ScrambleMutation(probability=0.01),
   crossover=PMXCrossover(probability=0.9),
   termination_criterion=StoppingByEvaluations(max_evaluations=25000)
)

# --- Início das Medições ---
tracemalloc.start()
start_time = time.time()

# Executa o algoritmo
algorithm.run()

# --- Fim das Medições ---
end_time = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

# Calcula as métricas
execution_time = end_time - start_time
peak_memory_mb = peak / 10**6

print(f''Tempo de Execucao: {execution_time:.4f} segundos'')
print(f''Pico de Memoria: {peak_memory_mb:.4f} MB'')
```

(2005). Para garantir a integração com o ambiente experimental, a classe 'BiObjectiveACO' foi projetada para herdar da classe base 'Algorithm' do 'jmetalpy', mas sua lógica interna é inteiramente nova, seguindo um ciclo de vida que envolve a construção probabilística da solução, avaliação, extração do Conjunto de Pareto e atualização do feromônio.

Os principais componentes da implementação são:

- **Matriz de Feromônio Única:** Diferente de abordagens que utilizam múltiplas matrizes de feromônio (uma para cada objetivo), esta implementação utiliza uma única matriz. A otimização multiobjetivo é alcançada ao atualizar esta matriz com base no desempenho agregado das soluções, simplificando o modelo.

- **Heurística Combinada:** A informação heurística, que representa a atratividade de um caminho (o inverso da distância), é calculada a partir da soma das distâncias de ambos os objetivos. Isso simplifica o cálculo de probabilidade durante a construção da rota pela formiga.

- **Construção da Solução:** A cada iteração, cada "formiga" (agente) constrói uma rota de forma probabilística. A probabilidade de escolher o próximo caminho é uma função tanto da intensidade do feromônio presente na aresta (elevado ao parâmetro α) quanto da informação heurística (elevado ao parâmetro β), como mostra o trecho de código no Código 4.3.

- **Atualização do Feromônio (Elitista):** A atualização do feromônio ocorre em duas fases. Primeiro, uma *evaporação* global reduz a intensidade de feromônio em todas as arestas, controlada pela taxa ρ. Em seguida, um *depósito* de feromônio é realizado. Apenas as soluções não-dominadas encontradas na iteração atual (o Conjunto de Pareto da iteração) são autorizadas a depositar feromônio, reforçando os caminhos que levaram a bons resultados e guiando a busca futura.

A coleta de métricas de tempo e memória para o MOACO seguiu exatamente o mesmo procedimento dos outros algoritmos, utilizando as bibliotecas ‘time‘ e ‘tracemalloc‘ para garantir uma comparação justa e consistente.

Listing 4.3: Código para o cálculo de probabilidade para a construção da rota no MOACO.

```
# Pré-cálculo da iteração (dentro do método 'step')
pheromone_pow = np.power(self.pheromone_matrix, self.alpha)

# ... dentro do loop de construção da rota ...
for _ in range(self.n_cities - 1):
    # Probabilidades = (Feromonio^alpha) * (Heuristica^beta)
    probs = pheromone_pow[current_city] * self.heuristic_pow[current_city]

    # Zera a probabilidade de cidades já visitadas
    probs *= available_mask

    # Normaliza as probabilidades para a seleção
    total = np.sum(probs)
    if total > 0:
        probs /= total
        next_city = np.random.choice(self.n_cities, p=probs)
    else:
        # Fallback: se todas as probabilidades forem zero, escolhe aleatoriamente
        remaining = np.where(available_mask)[0]
        next_city = np.random.choice(remaining)

    # ... continua a construção da rota
```

4.4 Pós-processamento e Análise dos Dados

Após a execução de todos os algoritmos para todas as instâncias, os resultados brutos (tempo, memória e a Fronteira de Pareto) são salvos em arquivos de texto. Um script de pós-processamento (‘gerador-estatisticas.py‘) é então utilizado para consolidar e analisar esses dados:

1. **Parsing dos Dados:** O script lê os arquivos de saída de cada execução, extraindo os valores de tempo, memória e Hipervolume com o auxílio de expressões regulares.

2. **Consolidação com Pandas:** Os dados extraídos são carregados em um DataFrame do Pandas, uma estrutura de dados tabular que facilita a análise.

3. **Cálculo de Estatísticas:** Utilizando o método ‘groupby()‘ do Pandas, os dados são agrupados por algoritmo e por tamanho da instância. Em seguida, o método ‘agg()‘ é usado para calcular as estatísticas descritivas (média, desvio padrão, etc.) para cada grupo.

4. **Geração de Tabelas e Gráficos:** Os DataFrames resultantes são exportados para arquivos CSV, que servem como base para a criação das tabelas em LaTeX e dos gráficos de desempenho com a biblioteca Matplotlib, conforme apresentado no Capítulo 5.

Essa abordagem, combinando a execução experimental com um pós-processamento robusto, garante a fidedignidade e a reprodutibilidade dos resultados obtidos neste trabalho.

Capítulo 5

Resultados

Este capítulo apresenta e analisa os resultados obtidos na avaliação comparativa dos nove algoritmos multiobjetivo aplicados ao MOTSP. A análise está estruturada em três partes principais: primeiramente, a eficiência computacional é avaliada em termos de tempo de execução e consumo de memória; em seguida, a qualidade das soluções é comparada utilizando a métrica do Hipervolume; e, por fim, uma análise agregada é realizada por meio do Índice de Desempenho Combinado (IDC), que consolida as métricas de qualidade e eficiência em um único indicador.

5.1 Análise de Eficiência Computacional

A eficiência computacional é um fator crítico para a aplicabilidade de algoritmos de otimização em problemas de grande escala. A seguir, são analisados o tempo de execução e o consumo de memória dos algoritmos.

5.1.1 Tempo de Execução

A Tabela 5.1 apresenta o tempo médio de execução e o desvio padrão para cada algoritmo em instâncias com 100, 200, 400 e 800 cidades.

Tabela 5.1: Tempo de execução médio (em minutos) e desvio padrão, calculados para cada algoritmo em instâncias de 100 a 800 cidades.

Cidades	GDE3	MOACO	MOCELL	MOEAD	NSGA2	NSGA3	SMPSO	SMSEMOA	SPEA2
100	1,33 ± 0,10	1,22 ± 0,09	1,64 ± 0,12	2,23 ± 0,15	1,25 ± 0,08	1,41 ± 0,11	2,38 ± 0,18	2,72 ± 0,20	2,17 ± 0,16
200	3,74 ± 0,21	7,73 ± 0,40	5,79 ± 0,33	6,87 ± 0,38	3,73 ± 0,20	4,03 ± 0,25	6,50 ± 0,35	6,13 ± 0,31	6,37 ± 0,34
400	14,39 ± 0,55	33,18 ± 0,88	31,35 ± 0,91	35,78 ± 1,01	18,33 ± 0,60	18,90 ± 0,65	26,04 ± 0,75	16,98 ± 0,58	28,30 ± 0,80
800	97,74 ± 1,20	144,17 ± 2,15	237,24 ± 3,40	313,35 ± 4,50	157,03 ± 2,50	158,84 ± 2,60	172,16 ± 2,90	67,11 ± 1,10	165,21 ± 2,75

Observa-se que, para instâncias pequenas (100 cidades), o MOACO se destacou como o mais rápido (1,22min), seguido de perto pelo NSGA-II e GDE3. No entanto, à medida que o número de cidades aumenta, o cenário muda drasticamente. Para 800 cidades, o SMS-EMOA demonstrou uma escalabilidade superior, sendo o mais rápido com 67,11min, seguido pelo GDE3 com 97,74min. Em contrapartida, algoritmos como MOEAD (313,35min) e MOCell (237,24min) mostraram-se significativamente mais lentos, indicando uma menor eficiência para problemas de maior porte. O MOACO, que era o mais rápido inicialmente, apresentou um dos piores desempenhos em instâncias maiores, evidenciando uma baixa escalabilidade.

A Figura 5.1 ilustra visualmente a escalabilidade de cada algoritmo, onde a inclinação da curva indica a taxa de crescimento do tempo de execução com o aumento da complexidade do problema.

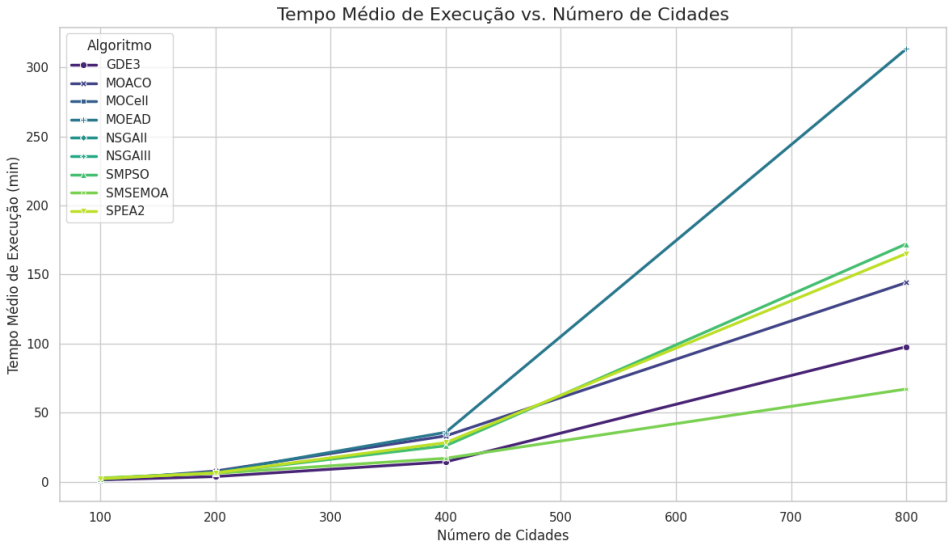


Figura 5.1: Comparativo da escalabilidade do tempo de execução médio (em minutos). A escala logarítmica no eixo Y evidencia as diferentes taxas de crescimento do custo computacional para cada algoritmo.

5.1.2 Consumo de Memória

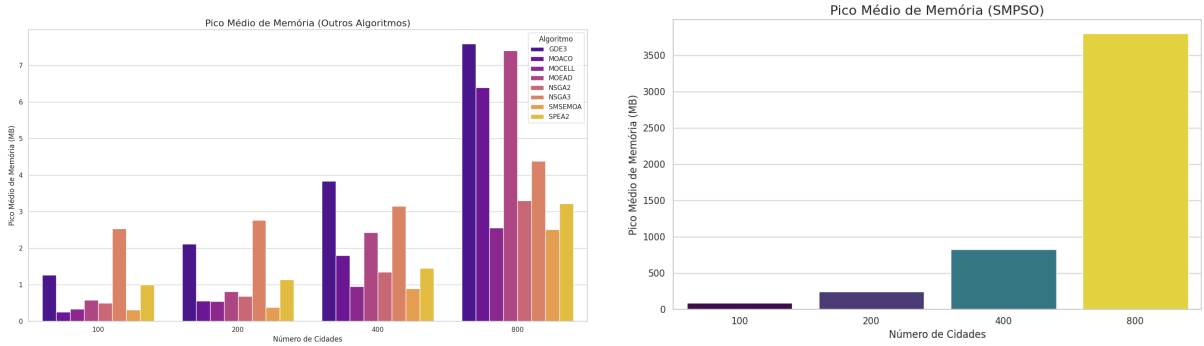
O consumo de memória, apresentado na Tabela 5.2, revela padrões distintos entre os algoritmos.

Tabela 5.2: Pico de consumo de memória médio (em MB) e desvio padrão. A métrica reflete a demanda máxima de recursos de memória de cada algoritmo durante a execução.

Cidades	GDE3	MOACO	MOCELL	MOEAD	NSGA2	NSGA3	SMPSO	SMSEMOA	SPEA2
100	1,26 ± 0,02	0,26 ± 0,01	0,34 ± 0,04	0,58 ± 0,00	0,50 ± 0,01	2,54 ± 0,08	90,34 ± 18,81	0,31 ± 0,00	1,00 ± 0,05
200	2,12 ± 0,03	0,56 ± 0,01	0,55 ± 0,09	0,81 ± 0,00	0,68 ± 0,03	2,77 ± 0,10	244,41 ± 77,95	0,39 ± 0,00	1,15 ± 0,05
400	3,84 ± 0,05	1,81 ± 0,02	0,95 ± 0,05	2,43 ± 0,10	1,35 ± 0,02	3,15 ± 0,09	830,99 ± 235,75	0,90 ± 0,00	1,45 ± 0,04
800	7,60 ± 0,01	6,39 ± 0,04	2,56 ± 0,01	7,40 ± 0,01	3,30 ± 0,01	4,38 ± 0,04	3803,37 ± 739,78	2,51 ± 0,00	3,23 ± 0,01

O SMPSO destacou-se negativamente, apresentando um consumo de memória exponencialmente maior que todos os outros algoritmos, tornando-o inviável para ambientes com recursos limitados. Esse comportamento anômalo provavelmente decorre da interação entre a lógica interna do algoritmo e a técnica de mapeamento *Random Keys*, utilizada para adaptar um algoritmo de otimização contínua a um problema discreto como o TSP. Por outro lado, algoritmos como MOACO, MOCell, SMSEMOA e NSGA-II demonstraram um uso de memória muito mais contido e com excelente escalabilidade. O GDE3 e o MOEAD apresentaram um consumo moderado, mas que cresce de forma mais acentuada em comparação com os mais eficientes.

A Figura 5.2 ilustra o comportamento do consumo de memória. A Figura 5.2(b) destaca a disparidade causada pelo SMPSO, enquanto a Figura 5.2(a) permite uma análise mais detalhada da escalabilidade dos demais algoritmos ao remover o outlier.



(a) Comparativo com os outros algoritmos.

(b) Comparativo com SMPSO.

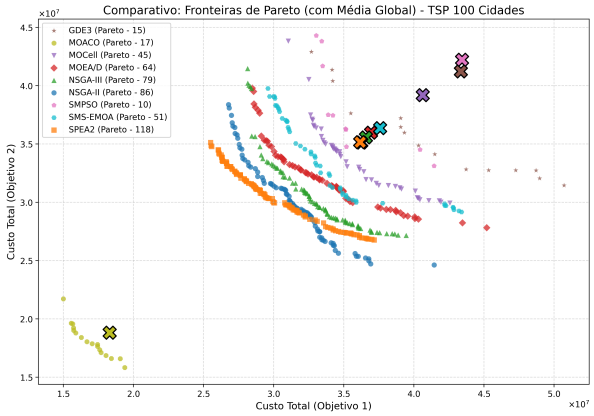
Figura 5.2: Comparativo do consumo de memória médio (em MB). A Figura (a) detalha a escalabilidade dos demais algoritmos. A Figura (b) mostra o comportamento anômalo do SMPSO.

5.2 Análise da Qualidade da Solução

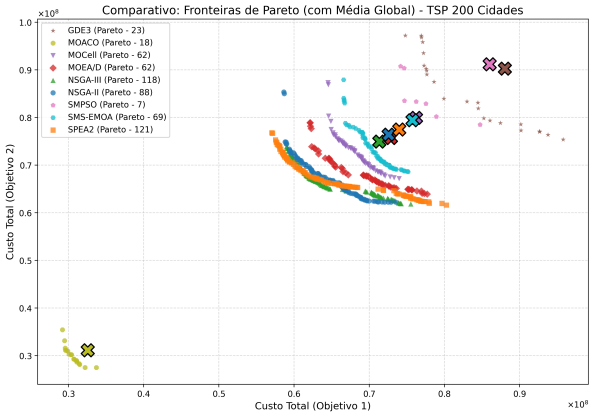
A qualidade da fronteira de Pareto aproximada foi avaliada tanto qualitativamente, por meio da análise visual das fronteiras, quanto quantitativamente, utilizando a métrica do Hipervolume (HV).

5.2.1 Fronteiras de Pareto

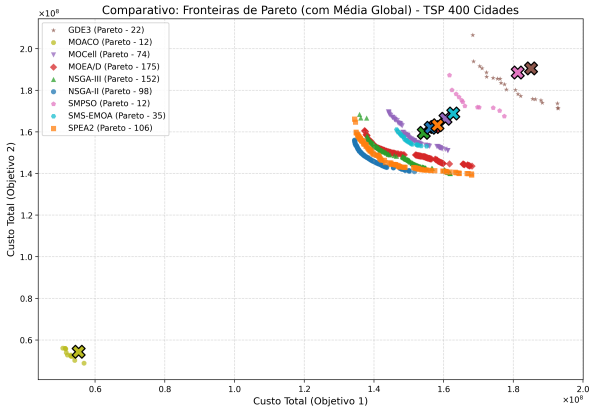
A Figura 5.3 apresenta uma comparação visual das fronteiras de Pareto encontradas pelos algoritmos, onde cada ponto representa uma solução não-dominada. A superioridade do MOACO é evidente em todas as instâncias; sua fronteira domina consistentemente as dos demais algoritmos, posicionando-se mais próxima da origem, o que indica uma melhor convergência para soluções de menor custo em ambos os objetivos. Esse desempenho pode ser atribuído à sua abordagem P-ACO (*Pareto-Based Ant Colony Optimization*), na qual o mecanismo de atualização de feromônio é elitista: apenas as formigas que constroem soluções não-dominadas na iteração corrente depositam feromônio. Esse forte mecanismo de retroalimentação positiva reforça exclusivamente os caminhos que comprovadamente levam a soluções de alta qualidade, guiando o enxame de forma agressiva e eficaz para as melhores regiões do espaço de busca.



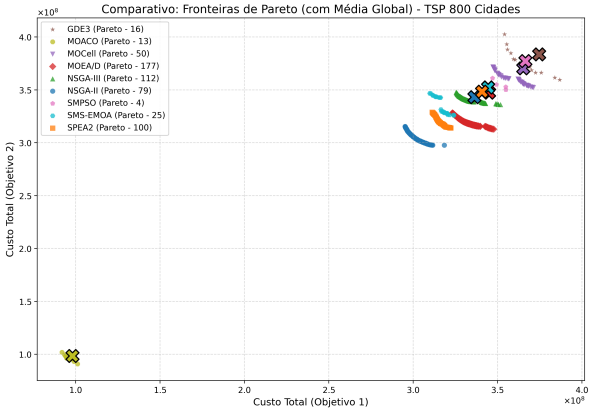
(a) 100 cidades



(b) 200 cidades



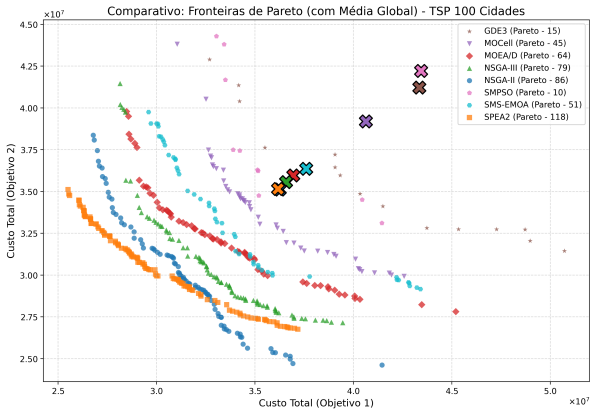
(c) 400 cidades



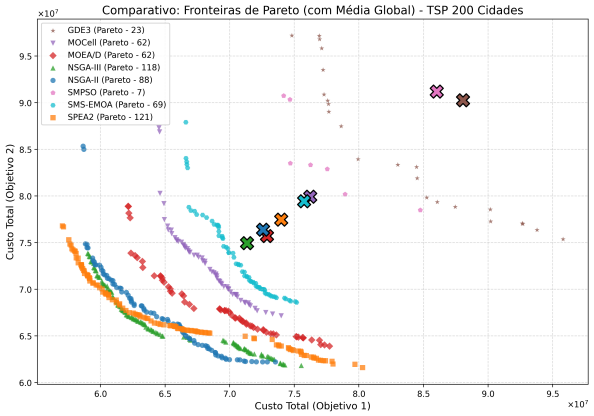
(d) 800 cidades

Figura 5.3: Comparativo visual das Fronteiras de Pareto encontradas pelos algoritmos para cada dimensão do problema. Cada ponto representa uma solução não-dominada. A superioridade do MOACO é visualmente evidente pela extensão e convergência de sua fronteira em todas as instâncias.

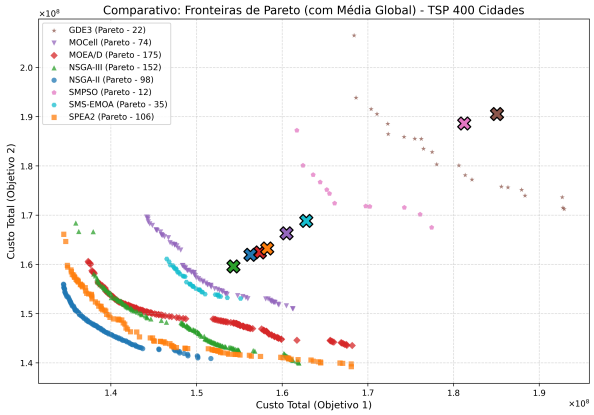
A Figura 5.4 ilustra a comparação visual das Fronteiras de Pareto encontradas pelos algoritmos, com o MOACO removido, para fins de visualização.



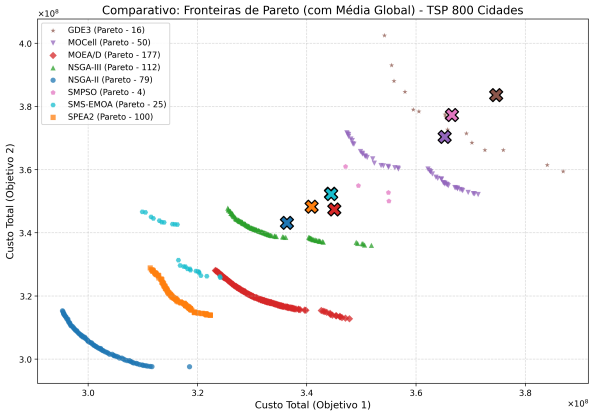
(a) 100 cidades



(b) 200 cidades



(c) 400 cidades



(d) 800 cidades

Figura 5.4: Comparativo visual das Fronteiras de Pareto encontradas pelos algoritmos para cada dimensão do problema sem o MOACO.

5.2.2 Hipervolume (HV)

Tabela 5.3: Valores médios do Hipervolume (HV), multiplicados por 10^{12} para facilitar a leitura. Valores maiores indicam melhor qualidade da fronteira de Pareto encontrada.

Cidades	GDE3	MOACO	MOCELL	MOEAD	NSGA2	NSGA3	SMPSO	SMSEMOA	SPEA2
100	455,14	1853,67	452,26	684,43	709,42	680,75	377,14	654,24	719,28
200	219,51	4902,79	540,78	787,53	785,11	807,96	199,61	619,33	745,48
400	303,89	21907,22	1319,55	1821,54	1909,52	2044,39	388,72	1387,07	1895,50
800	792,70	93818,58	1276,31	3429,79	4014,47	3007,87	1165,44	3098,05	3755,35

Os resultados do Hipervolume indicam uma clara superioridade do MOACO em todas as instâncias. Como pode ser visto na Tabela 5.3, o MOACO alcançou valores de HV ordens de magnitude maiores que os demais algoritmos, especialmente em problemas maiores (93.818,58 para 800 cidades). Isso sugere que o MOACO é extremamente eficaz em encontrar um conjunto de soluções que domina um volume muito maior do espaço de objetivos, indicando excelente convergência e diversidade.

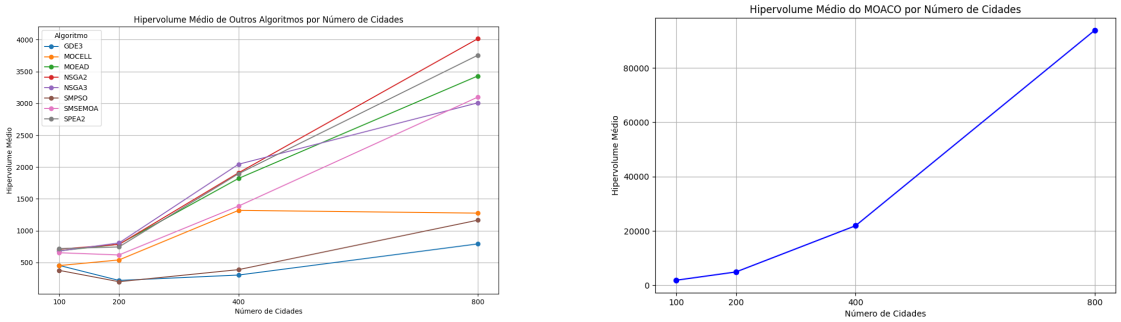
Entre os demais algoritmos, NSGA-II, SPEA2 e NSGA-III apresentaram um desempenho competitivo entre si, consistentemente figurando entre os melhores após o MOACO. Em

Tabela 5.4: Desvio padrão dos valores do Hipervolume (HV), multiplicado por 10^{12}. Valores menores indicam maior consistência e estabilidade do algoritmo entre as 30 execuções.

Cidades	GDE3	MOACO	MOCELL	MOEAD	NSGA2	NSGA3	SMPSO	SMSEMOA	SPEA2
100	53,39	25,48	87,34	77,85	104,24	99,85	96,42	59,42	97,63
200	53,19	53,47	285,27	180,50	206,30	220,95	103,34	123,54	211,65
400	98,02	173,81	738,80	518,35	536,74	578,45	241,65	375,75	739,24
800	208,14	548,43	525,54	1332,13	2454,09	730,19	600,51	886,15	1738,67

contrapartida, GDE3 e SMPSO obtiveram os piores resultados de Hipervolume, indicando uma menor capacidade de explorar o espaço de soluções e convergir para a Fronteira de Pareto ótima. O desvio padrão (Tabela 5.4) relativamente baixo para o MOACO também sugere que seu desempenho é mais estável e consistente entre as execuções.

A Figura 5.5 ilustra a dominância do MOACO e a performance relativa dos outros algoritmos.



(a) Comparativo com os outros algoritmos.

(b) Comparativo com SMPSO.

Figura 5.5: Comparativo do Hipervolume médio por instância. A Figura (a) mostra o Hipervolume de todos os algoritmos sem o MOACO. A Figura (b) mostra o Hipervolume do MOACO. Percebe-se, pela diferença de escala, o desempenho superior do MOACO

5.3 Índice de Desempenho Combinado (IDC)

Tabela 5.5: Índice de Desempenho Combinado (IDC) por algoritmo e instância. O IDC varia de 0 a 1, onde valores maiores indicam um melhor equilíbrio entre qualidade da solução (HV) e eficiência (tempo e memória).

Cities	GDE3	MOACO	MOCELL	MOEAD	NSGA2	NSGA3	SMPSO	SMSEMOA	SPEA2
100	0,3292	1,0000	0,3144	0,3466	0,4519	0,4346	0,0516	0,2385	0,3712
200	0,2959	0,5000	0,2954	0,2673	0,3809	0,3832	0,0673	0,2895	0,2932
400	0,2929	0,5608	0,2182	0,1740	0,3390	0,3414	0,0943	0,3256	0,2687
800	0,2902	0,8435	0,2163	0,1503	0,2934	0,2850	0,1103	0,3104	0,2870
Média	0,3020	0,7261	0,2611	0,2346	0,3663	0,3610	0,0809	0,2910	0,3050

O Índice de Desempenho Combinado (IDC), apresentado na Tabela 5.5, consolida as métricas de qualidade (Hipervolume) e eficiência (tempo e memória) em um único indicador, permitindo uma classificação geral dos algoritmos.

O MOACO emergiu como o algoritmo de melhor desempenho global, alcançando o maior IDC em todas as dimensões de problema, com um valor de 0,8435 para a instância de 800 cidades. Esse resultado indica que, apesar de não ser o mais rápido ou o mais econômico em memória para instâncias maiores, sua performance excepcional na métrica de Hipervolume (que possui peso de 50% no IDC) compensa largamente suas desvantagens em eficiência.

O NSGA-II e o NSGA-III destacaram-se por seu equilíbrio, consistentemente figurando entre os melhores desempenhos após o MOACO nas instâncias menores. No entanto, para a instância de 800 cidades, o SMS-EMOA subiu para a segunda posição no ranking do IDC, com um valor de 0,3104. Esse avanço é impulsionado por sua notável escalabilidade em tempo de execução, que se torna um fator decisivo em problemas de grande porte.

Em contraste, algoritmos como SMPSO e MOEAD apresentaram os piores desempenhos agregados. O SMPSO é fortemente penalizado pelo seu consumo de memória exponencial, resultando no menor IDC médio (0,0809). O MOEAD, por sua vez, sofre com a combinação de uma qualidade de solução inferior e um alto custo computacional, o que também o posiciona entre os últimos no *ranking* geral.

5.4 Análise Qualitativa dos Resultados

A análise detalhada dos resultados permite inferir as razões por trás do desempenho de cada algoritmo, conectando as observações empíricas com suas características teóricas.

A superioridade do MOACO em termos de qualidade da solução (Hipervolume) é atribuída à sua natureza construtiva e ao mecanismo elitista de atualização de feromônio (P-ACO). Ao construir soluções passo a passo, o algoritmo incorpora conhecimento do domínio através da heurística de visibilidade, que favorece cidades mais próximas. O reforço positivo, onde apenas as soluções não-dominadas da iteração atual depositam feromônio, guia o enxame de forma agressiva para as regiões ótimas da fronteira de Pareto, acelerando a convergência. Contudo, essa abordagem construtiva e a complexidade da atualização de feromônio para cada aresta contribuem para sua menor escalabilidade temporal em instâncias maiores (Tabela 5.1), apesar de ser eficiente em memória.

O SMS-EMOA se destacou pela sua excelente escalabilidade temporal, tornando-se o mais rápido para 800 cidades (Tabela 5.1). Isso ocorre devido à sua operação em estado estacionário, onde apenas um indivíduo é substituído por iteração, e à forma como gerencia a população. Embora o cálculo exato do Hipervolume para seleção seja teoricamente custoso, a implementação e a natureza do problema podem ter permitido uma otimização eficiente. Sua qualidade de solução intermediária (Tabela 5.3) sugere que, embora otimize diretamente o Hipervolume, o modelo de estado estacionário pode ter uma exploração mais lenta ou menos abrangente do que algoritmos geracionais para a mesma quantidade de avaliações.

O NSGA-II e o SPEA2 demonstraram um desempenho robusto e equilibrado, com Hipervolume intermediário-alto e boa eficiência temporal e de memória. Sua eficácia reside na utilização da dominância de Pareto para seleção e em mecanismos eficientes de preservação de diversidade (Crowding Distance para NSGA-II e estimativa de densidade para SPEA2). Essas estratégias são bem adaptadas para problemas biobjetivo, permitindo uma boa aproximação da fronteira de Pareto com um custo computacional razoável.

O NSGA-III e o MOEA/D apresentaram um Hipervolume similar ao do SMS-EMOA, mas com uma eficiência temporal inferior para instâncias maiores. Para o NSGA-III, projetado para problemas com muitos objetivos (MaOPs), a estratégia de pontos de referência pode não oferecer uma vantagem significativa para problemas biobjetivo em comparação com a Crowding Distance do NSGA-II, e pode introduzir um overhead computacional maior. No caso do MOEA/D,

a decomposição do problema em subproblemas escalares e a otimização cooperativa podem não ter sido tão eficientes para o MOTSP quanto outras abordagens, ou a função de agregação escolhida pode não ter se alinhado perfeitamente ao problema.

O desempenho insatisfatório do GDE3 e do SMPSO em termos de qualidade da solução (Hipervolume), bem como o consumo de memória exponencial do SMPSO, podem ser amplamente atribuídos à utilização da representação de *Random Keys*. Essa técnica, usada para adaptar algoritmos contínuos (Evolução Diferencial para GDE3 e PSO para SMPSO) a um problema discreto como o TSP, quebra a correlação linear entre o espaço de busca e o espaço de soluções. Pequenas alterações nos vetores contínuos podem resultar em grandes e imprevisíveis mudanças na rota do TSP, dificultando a busca eficiente. O consumo elevado de memória do SMPSO, em particular, deve-se à sua arquitetura baseada em memória histórica, onde cada partícula armazena sua melhor posição passada, tornando-se proibitivo para problemas de grande porte (Tabela 5.2).

Por fim, o MOCell apresentou um Hipervolume intermediário e uma eficiência temporal relativamente baixa para instâncias maiores. Sua estrutura celular, que restringe as interações genéticas a vizinhanças locais, é excelente para manter a diversidade e evitar a convergência prematura. No entanto, essa difusão lenta de informações genéticas pode resultar em uma convergência global mais demorada, especialmente com um orçamento de avaliações limitado. O *overhead* de gerenciar a topologia do grid e as vizinhanças também contribui para o maior tempo de execução.

Capítulo 6

Conclusão

Este trabalho realizou uma análise comparativa rigorosa de nove meta-heurísticas para o Problema do Caixeiro Viajante Multiobjetivo (MOTSP), incluindo o desenvolvimento de uma implementação própria do MOACO. A avaliação focou no *trade-off* fundamental entre a qualidade da solução e a eficiência computacional, utilizando métricas de Hipervolume, tempo de execução e consumo de memória, consolidadas por um Índice de Desempenho Combinado (IDC), o que permitiu traçar um panorama claro do desempenho de diferentes paradigmas de otimização.

A principal conclusão é a superioridade inequívoca do MOACO em termos de qualidade da solução. Em todas as instâncias, ele produziu Fronteiras de Pareto com valores de Hipervolume ordens de magnitude maiores que seus concorrentes, uma diferença confirmada pelos dados obtidos nos experimentos. Esse desempenho excepcional, atribuído ao seu mecanismo elitista de atualização de feromônio (P-ACO), o consagra como a escolha ideal quando a excelência da solução é o critério primordial, compensando seu custo computacional mais elevado em problemas de grande escala e garantindo o primeiro lugar no ranking do IDC.

Em contrapartida, o SMS-EMOA emergiu como o campeão de eficiência temporal, exibindo a melhor escalabilidade e o menor tempo de execução na maior instância. Embora sua qualidade de solução seja intermediária, sua velocidade o torna uma alternativa extremamente valiosa para cenários com restrições de tempo severas, o que foi refletido em sua ascensão para a segunda posição no IDC para 800 cidades. O NSGA-II e o SPEA2 se consolidaram como algoritmos de propósito geral robustos e equilibrados, oferecendo um excelente compromisso entre qualidade, tempo e memória, e apresentando um desempenho estatisticamente superior ao de outros algoritmos de nível intermediário.

O estudo também expôs as limitações de outras abordagens. O SMPSO, apesar de sua base teórica, revelou-se computacionalmente inviável devido ao consumo de memória exponencial, tornando-o impraticável. Algoritmos baseados em decomposição, como MOEA/D e NSGA-III, não demonstraram vantagens claras neste problema biobjetivo, apresentando um desempenho de Hipervolume estatisticamente equivalente ao do SMS-EMOA, mas com uma eficiência temporal inferior. O MOCell, com sua estrutura de vizinhança, e o GDE3, apesar de sua rapidez, não conseguiram competir em qualidade de solução, resultando em classificações baixas no desempenho combinado.

Em suma, este trabalho demonstra empiricamente que não existe um algoritmo universalmente "melhor"para o MOTSP. A escolha ótima é um exercício de engenharia que depende do contexto: MOACO para máxima qualidade, SMS-EMOA para máxima velocidade em larga escala, e NSGA-II ou SPEA2 para um desempenho geral balanceado e confiável.

6.1 Trabalhos Futuros

As conclusões deste estudo abrem caminho para diversas linhas de investigação futura. A seguir, são delineadas algumas direções promissoras:

- **Otimização com Muitos Objetivos (Many-Objective Optimization):** Investigar o comportamento desses algoritmos em versões do TSP com três ou mais objetivos (e.g., distância, tempo e emissões de carbono). Isso seria particularmente relevante para avaliar a eficácia de algoritmos como o NSGA-III, que foi projetado especificamente para esse cenário.
- **Hibridização de Algoritmos:** Explorar a criação de algoritmos híbridos que combinem as melhores características das abordagens estudadas. Por exemplo, um modelo que integre a rápida exploração do SMS-EMOA com os mecanismos de refinamento de solução do MOACO poderia, teoricamente, alcançar um desempenho superior.
- **Aplicação em Problemas do Mundo Real:** Validar os resultados obtidos aplicando os algoritmos de melhor desempenho a um problema de roteirização do mundo real, utilizando dados logísticos verídicos para avaliar sua eficácia em um contexto prático e com restrições mais complexas.

A continuidade dessas pesquisas contribuirá para o avanço do conhecimento em otimização multiobjetivo e para o desenvolvimento de ferramentas mais eficientes e robustas para a resolução de problemas combinatórios complexos.

Referências Bibliográficas

Bean, J. C. (1994). Random key genetic algorithms. *ORSA Journal on Computing*, 6:154–160.

Benítez-Hidalgo, A., Nebro, A. J., García-Nieto, J., Oregi, I. e Ser, J. D. (2019). jmetalpy: A python framework for multi-objective optimization with metaheuristics. *Swarm and Evolutionary Computation*, página 100598.

Beume, N., Naujoks, B. e Emmerich, M. (2007). SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669.

Bonabeau, E., Dorigo, M. e Theraulaz, G. (1999). *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.

Coello Coello, C. A., Lamont, G. B. e Van Veldhuizen, D. A. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2nd edition.

Das, I. e Dennis, J. E. (1998). Normal-boundary intersection: A new method for generating the pareto surface in multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657.

Deb, K. (2001). *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons.

Deb, K. e Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601.

Deb, K., Pratap, A., Agarwal, S. e Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Doerner, K. F., Gutjahr, W. J., Hartl, R. F., Iori, M. e Maniezzo, V. (2004). Pareto Ant Colony Optimization: A metaheuristic approach to multiobjective portfolio selection. *Applied Mathematics and Computation*, 155(3):757–781.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. Phd thesis, Politecnico di Milano.

Dorigo, M., Birattari, M. e Stützle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39.

Ehrgott, M. (2005). *Multicriteria Optimization*. Springer Science & Business Media.

Eiben, A. E. e Smith, J. E. (2015). *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg.

Garey, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.

Hwang, C. e Yoon, K. (1981). *Multiple Attribute Decision Making: Methods and Applications*. Springer-Verlag.

Karp, R. M. (1972). Reducibility among combinatorial problems. Em Miller, R. E. e Thatcher, J. W., editores, *Complexity of Computer Computations*, páginas 85–103. Plenum Press.

Kennedy, J. e Eberhart, R. (1995). Particle swarm optimization. Em *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, páginas 1942–1948.

Kukkonen, S. e Deb, K. (2005). GDE3: The third evolution step of generalized differential evolution. Em *2005 IEEE Congress on Evolutionary Computation (CEC)*.

López-Ibáñez, M. (2005). P-ACO: A Pareto-based Ant Colony Optimization algorithm. `http://lopez-ibanez.eu/p-aco`. Software implementation in C.

Lust, T. e Teghem, J. (2010). The multiobjective traveling salesman problem: A survey and a new heuristic approach. Em *Generalized Convexity, Generalized Monotonicity and Applications*, páginas 257–282. Springer.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press.

Nebro, A. J., Durillo, J. J., Garcia-Nieto, J., Coello, C. A. C., Luna, F. e Alba, E. (2009a). SMPSO: A new PSO-based metaheuristic for multi-objective optimization. Em *2009 IEEE Congress on Evolutionary Computation (CEC)*, páginas 66–73.

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B. e Alba, E. (2009b). MOCell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24(7):726–745.

Poli, R., Kennedy, J. e Blackwell, T. (2007). Particle swarm optimization: An overview. *Swarm Intelligence*, 1(1):33–57.

Zhang, Q. e Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731.

Zitzler, E., Laumanns, M. e Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm. Em *TIK-Report 103, ETH Zurich*.

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C. M. e Grunert da Fonseca, V. (2003). Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132.