



Trabalho de Conclusão de Curso

Ajuste Automático de Parâmetros de Meta-heurísticas com Aprendizado de Máquina

Gabriel Luiz Leite Souza
glls@ic.ufal.br

Orientador:
Prof. Dr. Rian Gabriel Santos Pinheiro

Maceió, Maio de 2025

Gabriel Luiz Leite Souza

Ajuste Automático de Parâmetros de Meta-heurísticas com Aprendizado de Máquina

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação do Instituto de Computação da Universidade Federal de Alagoas.

Orientador:

Prof. Dr. Rian Gabriel Santos Pinheiro

Maceió, Maio de 2025

Catálogo na fonte
Universidade Federal de Alagoas
Biblioteca Central
Divisão de Tratamento Técnico

Bibliotecária: Helena Cristina Pimentel do Vale – CRB4/661

- S729a Souza, Gabriel Luiz Leite.
Ajuste automático de parâmetros de meta-heurísticas com aprendizado de máquina / Gabriel Luiz Leite Souza. – 2025.
36 f.: il.
- Orientador: Rian Gabriel Santos Pinheiro.
Monografia (Trabalho de Conclusão de Curso em Ciência da Computação) – Universidade Federal de Alagoas, Instituto de Computação. Maceió, 2025.
- Bibliografia: f. 34-36.
1. Problema do caixeiro viajante. 2. ACOTSP. 3. IRACE. 4. Otimização de parâmetros. 5. Aprendizado de máquina. 6. Inteligência artificial. I. Título.

CDU: 004.5

Agradecimentos

Gostaria de expressar minha profunda gratidão a todas as pessoas que contribuíram para a realização deste trabalho.

Aos meus pais e ao meu irmão, meu eterno agradecimento pelo amor, apoio incondicional e incentivo constante. Vocês são a minha força e inspiração, e este trabalho é dedicado a vocês.

Aos meus queridos colegas de curso, Ascânio Sávio, Fernanda Herculano, João Ayalla, João Falcão, Márcio Henrique, Michael Miller e Rodrigo Santos, meu sincero agradecimento. Vocês fizeram meu curso muito mais fácil e agradável com sua amizade, colaboração e companheirismo.

Ao professor Dr. Bruno Pimentel, meu agradecimento especial por me introduzir e guiar na fascinante área de inteligência artificial e machine learning. Sua orientação e conhecimento foram fundamentais para o desenvolvimento deste trabalho.

Ao meu orientador, professor Dr. Rian Pinheiro, minha imensa gratidão por toda a paciência, dedicação e conhecimento compartilhado durante o curso e a elaboração deste TCC. Sua orientação foi essencial para o sucesso deste trabalho.

Agradeço também a todos os professores e funcionários da instituição que, de alguma forma, contribuíram para a minha formação acadêmica e profissional.

Por fim, agradeço a Deus por todas as bênçãos e por me proporcionar a oportunidade de concluir esta etapa tão importante da minha vida.

Resumo

O Problema do Caixeiro Viajante (TSP) é um desafio clássico da otimização combinatória, com aplicações em logística, planejamento de rotas e gestão de redes. A eficácia de algoritmos meta-heurísticos, como o Ant Colony Optimization, depende criticamente da configuração adequada de parâmetros, um processo complexo e computacionalmente custoso devido à variabilidade entre instâncias do problema.

Este trabalho apresenta a `OptAI`, uma biblioteca em Python para otimização automatizada de parâmetros no TSP, como estudo de caso. A `OptAI` integra técnicas de Aprendizado de Máquina (ML) — como XGBoost, Random Forest e SVR — com frameworks de otimização consolidados, como IRACE e Optuna. Baseada na análise de logs históricos do IRACE, a biblioteca treina modelos de regressão para prever conjuntos de parâmetros personalizados para cada instância, utilizando características descritivas como número de cidades, densidade e estatísticas de distância.

Comparada ao IRACE, que retorna um único conjunto de parâmetros para todas as instâncias, a `OptAI` oferece configurações específicas, aproveitando a capacidade preditiva dos modelos de ML. Além disso, a biblioteca proporciona maior escalabilidade, evitando a exploração exaustiva do espaço de parâmetros, e permite integração modular com frameworks modernos para atualizações contínuas. Resultados iniciais validam sua eficácia, embora a dependência dos logs do IRACE ainda limite sua autonomia.

A `OptAI` representa uma contribuição relevante ao unir inteligência artificial e métodos clássicos, propondo uma abordagem híbrida de Aprendizado de Máquina e meta-heurísticas para problemas complexos de otimização combinatória.

Palavras-chave: Problema do caixeiro viajante, ACOTSP, IRACE, otimização de parâmetros, aprendizado de máquina, regressão, Optuna, TSPLIB95, XGBoost, Random Forest, inteligência artificial, bibliotecas Python, configuração automática de algoritmos

Abstract

The Traveling Salesman Problem (TSP) stands as a cornerstone challenge in combinatorial optimization, with widespread applications in logistics, route planning, and network management. The performance of metaheuristic algorithms, such as Ant Colony Optimization, hinges critically on precise parameter tuning — a task that is both intricate and computationally demanding due to the inherent variability across problem instances.

This work presents `OptAI`, a Python-based library designed for automated parameter optimization in the TSP, serving as a case study. `OptAI` seamlessly integrates advanced Machine Learning (ML) techniques — including `XGBoost`, `Random Forest`, and `SVR` — with robust optimization frameworks like `IRACE` and `Optuna`. By analyzing historical logs from `IRACE`, the library trains regression models to predict instance-specific parameter configurations, leveraging descriptive features such as the number of cities, density, and distance statistics.

In contrast to `IRACE`, which provides a single parameter set for all instances, `OptAI` delivers tailored configurations by harnessing the predictive capabilities of ML models. Furthermore, the library enhances scalability by circumventing exhaustive parameter space exploration and supports modular integration with modern frameworks for continuous updates. Preliminary results demonstrate its efficacy, although its reliance on `IRACE` logs currently limits its autonomy.

`OptAI` marks a significant advancement by bridging artificial intelligence with classical methods, offering a hybrid approach that combines Machine Learning and metaheuristics to tackle complex combinatorial optimization problems effectively.

Keywords: Traveling Salesman Problem, ACOTSP, `IRACE`, parameter optimization, machine learning, regression, `Optuna`, `TSPLIB95`, `XGBoost`, `Random Forest`, artificial intelligence, Python libraries, automatic algorithm configuration

Conteúdo

1	Introdução	1
1.1	Seleção de Parâmetros	1
1.2	Proposta e Implementação da Biblioteca OptAI	2
1.3	Objetivos	2
1.4	Estrutura do Trabalho	3
2	Fundamentação Teórica	4
2.1	Meta-heurísticas	4
2.2	Parâmetros e Tuning de Algoritmos	6
2.3	O ACOTSP e o Problema do Caixeiro Viajante (TSP)	8
2.4	Aprendizado de Máquina e Regressores	9
3	Trabalhos Relacionados	11
3.1	Configuração Automática de Algoritmos	11
3.2	Otimização de Hiperparâmetros com Abordagens Dinâmicas	12
3.3	Avaliação da importância de hiperparâmetros	12
3.4	Calibração dos parâmetros de Meta-heurísticas utilizando Aprendizado de Máquina	13
3.5	Relação com o Trabalho Atual	15
4	Metodologia	16
4.1	Seleção das Instâncias TSP	16
4.2	Escolha da Linguagem Python	17
4.3	Configuração Inicial com IRACE	17
4.4	Pré-processamento e Criação do Dataframe com Logs do IRACE e Dados Descritivos	17
4.5	Treinamento dos Regressores	18
5	OptAI	19
5.1	Arquitetura da OptAI	19
5.2	Execução Completa da OptAI: Visão Geral	20
5.3	Módulos da OptAI	22
5.4	Pré-processamento dos Logs do IRACE	22
5.5	Incremento do Dataframe com Dados Descritivos	23
5.6	Treinamento dos Regressores	23
5.7	Otimização com Optuna	24
5.8	Salvamento e Carregamento de Regressores Treinados	25
5.9	Retroalimentação e Incremento de Dados	25

5.10	Pontos Finais	25
6	Resultados e Discussão	27
6.1	Configuração do Ambiente de Teste	27
6.2	Parâmetros de Execução	27
6.3	Cenário de Treinamento e Teste	28
6.4	Resultados da Comparação	28
6.5	Análise dos Resultados	29
6.6	Discussão	30
7	Considerações Finais e Trabalhos Futuros	31
7.1	Considerações Finais	31
7.1.1	Desempenho da OptAI	32
7.1.2	Vantagens da OptAI	32
7.1.3	Limitações e Desafios	32
7.2	Trabalhos Futuros	32
	Referências bibliográficas	34

1

Introdução

A otimização combinatória é uma área da ciência da computação e da pesquisa operacional que lida com problemas de otimização, em que o objetivo é encontrar a melhor combinação de elementos que satisfaça certas restrições. Um dos problemas mais estudados nessa área é o Problema do Caixeiro Viajante (TSP) [Reinelt, 1994]. O TSP consiste em encontrar a rota mais curta possível que permite a um caixeiro viajante visitar um conjunto de cidades exatamente uma vez e retornar à cidade de origem. Devido à sua complexidade computacional, o TSP é classificado como um problema *NP-difícil*, o que significa que não se conhece uma solução eficiente para instâncias grandes do problema, e ainda não se sabe se tal solução existe ou não.

A dificuldade de encontrar boas soluções para problemas *NP-difíceis* reside no fato de que o tempo necessário para encontrar a solução ótima com os algoritmos exatos conhecidos atualmente cresce, no pior caso, de forma superpolinomial (como exponencial, fatorial ou outra complexidade maior que polinomial) à medida que o tamanho da instância aumenta. Para contornar essa dificuldade, são utilizadas *meta-heurísticas*, que são técnicas de otimização que exploram o espaço de soluções de maneira inteligente, buscando um equilíbrio entre exploração (explorar novas áreas do espaço de soluções) e intensificação (aprofundar-se em áreas promissoras do espaço de soluções). Essas *meta-heurísticas* não garantem encontrar a solução ótima, mas podem encontrar soluções de alta qualidade em um tempo razoável.

1.1 Seleção de Parâmetros

A eficácia das *meta-heurísticas* combinatória depende fortemente da configuração de seus parâmetros. A seleção adequada de parâmetros pode melhorar significativamente o desempenho do algoritmo, enquanto uma configuração inadequada pode levar a soluções subótimas. A configuração automática de parâmetros é, portanto, uma área crucial de pesquisa, visando encontrar os melhores parâmetros para um algoritmo dado em um conjunto de instâncias de problema.

A configuração (tuning) de parâmetros é um processo crítico para otimizar o desempenho de algoritmos, envolvendo a seleção de valores que controlam seu comportamento e eficácia. Esse processo é essencial para garantir que os algoritmos atinjam resultados de alta qualidade em diferentes cenários de execução. Ferramentas como o IRACE (*Iterated Racing for Automatic Algorithm Configuration*) [López-Ibáñez et al., 2016] são amplamente utilizadas para automatizar e aprimorar essa tarefa.

1.2 Proposta e Implementação da Biblioteca OptAI

Este trabalho apresenta a biblioteca OptAI, uma ferramenta desenvolvida em Python para otimização de parâmetros em problemas de otimização combinatória, com o Problema do Caixeiro Viajante (TSP) como estudo de caso. O IRACE, ferramenta empregada para a otimização automática de parâmetros, retorna um único conjunto de parâmetros para todas as instâncias analisadas. Embora eficiente em alguns contextos, esse enfoque pode não ser adequado para aplicações em que diferentes instâncias podem exibir comportamentos e características distintas. A utilização de um único conjunto de parâmetros pode resultar em soluções subótimas para instâncias específicas, limitando a adaptação do algoritmo a cenários variados.

Surge, então, a hipótese de que técnicas de Aprendizado de Máquina (ML) poderiam permitir a seleção de parâmetros personalizados para cada instância, potencialmente melhorando a qualidade das soluções obtidas. A OptAI realiza uma série de etapas, incluindo a seleção de instâncias TSP, pré-processamento dos logs do IRACE, incremento do dataframe com dados descritivos das instâncias, treinamento de regressores e otimização com a biblioteca Optuna, visando retornar os melhores conjuntos de parâmetros para novas instâncias. Vale ressaltar que a abordagem da OptAI não se limita ao TSP, podendo ser aplicada a outros problemas de otimização combinatória. Detalhes mais profundos sobre o funcionamento da OptAI serão abordados nos capítulos de metodologia e ao longo do trabalho.

1.3 Objetivos

Os objetivos deste trabalho são:

1. Implementar uma biblioteca robusta para otimização de parâmetros em problemas de otimização combinatória, com foco inicial em meta-heurísticas como o Ant Colony Optimization (ACO).
2. Utilizar técnicas de Aprendizado de Máquina (ex.: XGBoost, Random Forest) para configurar parâmetros de forma inteligente e adaptável a cada instância do problema.
3. Proporcionar uma ferramenta de fácil utilização, permitindo que usuários executem algoritmos com mínimas modificações no código.

Além disso, os objetivos específicos incluem:

- Realizar um estudo de caso com o Problema do Caixeiro Viajante (TSP) para demonstrar a eficácia da `OptAI` em comparação ao `IRACE`.
- Comparar o desempenho da biblioteca com o `IRACE` em um conjunto diversificado de instâncias TSP.
- Validar a abordagem proposta por meio de experimentos e análises comparativas, destacando a capacidade preditiva dos modelos de regressão treinados.

1.4 Estrutura do Trabalho

O Capítulo 2 apresenta a fundamentação teórica, cobrindo meta-heurísticas, *tunning* de algoritmos, ACOTSP e Aprendizado de Máquina. O Capítulo 3 discute trabalhos relacionados, focando em configuração automática de algoritmos, otimização de hiperparâmetros e calibração de parâmetros com aprendizado de máquina. O Capítulo 4 detalha a metodologia, incluindo seleção de instâncias TSP, implementação da `OptAI`, configuração com `IRACE` e treinamento de modelos. O Capítulo 5 descreve a arquitetura da `OptAI`, sua execução, pré-processamento de dados, treinamento de regressores e otimização com `Optuna`. O Capítulo 6 apresenta os resultados, com configuração de testes, parâmetros, cenários e análise comparativa. Por fim, o Capítulo 7 traz as considerações finais, abordando desempenho, vantagens, limitações e trabalhos futuros.

2

Fundamentação Teórica

Este capítulo fundamenta teoricamente e tecnicamente a abordagem deste TCC. Inicia-se com a análise das meta-heurísticas, estratégias essenciais para resolver problemas complexos como o TSP, onde a busca exaustiva torna-se inviável devido à elevada complexidade. São destacadas técnicas como ACO, Simulated Annealing, Tabu Search e Algoritmos Genéticos, com ênfase em suas características e aplicações.

Na sequência, aborda-se o *tunning* de parâmetros, etapa fundamental para aprimorar o desempenho dos algoritmos. São apresentadas metodologias como Grid Search, Randomized Search e Bayesian Optimization, com destaque para o IRACE, ferramenta de configuração automática adotada como método de comparação.

O capítulo também detalha o ACOTSP, uma implementação baseada em ACO para o TSP, que funciona como estudo de caso e exemplo padrão na documentação do IRACE. Por fim, são discutidos os regressores de Aprendizado de Máquina utilizados para prever os melhores conjuntos de parâmetros, integrando teoria e prática na otimização de algoritmos para problemas complexos.

2.1 Meta-heurísticas

As meta-heurísticas são estratégias de otimização que buscam equilibrar exploração (explorar novas áreas do espaço de soluções) e intensificação (aprofundar-se em soluções promissoras) para resolver problemas complexos. Essas técnicas são fundamentais em problemas de otimização combinatória, onde o número de soluções possíveis cresce exponencialmente com o tamanho do problema, tornando a busca exaustiva inviável para instâncias de grande escala [Malik et al., 2021].

A eficácia das meta-heurísticas reside na sua capacidade de navegar em espaços de busca vastos sem se perder em soluções subótimas. Elas são especialmente úteis quando o problema

não apresenta estruturas matemáticas claras ou quando a busca exata é computacionalmente proibitiva. Entre as meta-heurísticas mais conhecidas, destacam-se:

- **Otimização por Colônias de Formigas (ACO)** [Dorigo, 1992]: Inspirada no comportamento de formigas, que utilizam feromônios para marcar rotas, o ACO modela a busca por soluções através de agentes artificiais que depositam e seguem rastros de feromônio. No contexto do Problema do Caixeiro Viajante (TSP), as “formigas” constroem rotas completas, e a quantidade de feromônio em cada aresta é proporcional à qualidade da solução encontrada. A fórmula de transição de probabilidade é dada por:

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{j \in \Lambda} \tau_{ij}^\alpha \cdot \eta_{ij}^\beta}, \quad (2.1)$$

onde p_{ij}^k é a probabilidade de uma formiga k mover-se do nó i para o nó j . Os termos τ_{ij} representam o nível de feromônio na aresta (i, j) , enquanto η_{ij} é a informação heurística (geralmente $1/d_{ij}$, com d_{ij} sendo a distância entre os nós). Os parâmetros α e β controlam a influência relativa do feromônio e da heurística, respectivamente, e Λ denota o conjunto de nós vizinhos acessíveis a partir de i .

- **Simulated Annealing** [Kirkpatrick et al., 1983]: Técnica inspirada no processo de resfriamento de metais, que permite a exploração de soluções subótimas inicialmente, gradativamente reduzindo a probabilidade de aceitar soluções piores. Essa abordagem evita o aprisionamento em mínimos locais, tornando-a útil para problemas com múltiplos ótimos.

```

Função simulated_annealing():
  candidato ←  $s_0$   (estado inicial)
   $T \leftarrow T_0$   (temperatura inicial)
  repita
    próximo ← vizinho_aleatório(candidato)
     $\Delta E \leftarrow \text{energia}(\text{próximo}) - \text{energia}(\text{candidato})$ 
    se  $\Delta E \leq 0$ :
      candidato ← próximo
    senão:
      aceita com probabilidade  $\exp(-\Delta E/T)$ 
       $T \leftarrow \text{próximaTemperatura}(T)$ 
  até  $T < T_{\text{final}}$ 
  Retorne candidato

```

- **Tabu Search** [Glover, 1986]: Utiliza memória para evitar a revisitação de soluções recentemente exploradas, promovendo a exploração de novas áreas do espaço de soluções.

A técnica mantém uma “lista de tabu” para restringir movimentos, garantindo diversidade na busca.

Função `tabu_search()` :

$s \leftarrow s_0$ (*solução inicial*)

$s^* \leftarrow s$ (*melhor solução obtida até então*)

$\text{Iter} \leftarrow 0$ (*contador de iterações*)

$\text{MelhorIter} \leftarrow 0$ (*última iteração que melhorou s^**)

$BT_{\max} \leftarrow$ número máximo de iterações sem melhoria

$T \leftarrow \emptyset$ (*lista tabu vazia*)

Inicialize função de aspiração $A(f(s))$

Enquanto $\text{Iter} - \text{MelhorIter} \leq BT_{\max}$:

$\text{Iter} \leftarrow \text{Iter} + 1$

Escolha $m \notin T$ ou $f(s') < A(f(s))$

$s' \leftarrow s \oplus m$ (*aplica movimento m*)

Atualize lista tabu T

$s \leftarrow s'$

Se $f(s) < f(s^*)$:

$s^* \leftarrow s$

$\text{MelhorIter} \leftarrow \text{Iter}$

Atualize função de aspiração $A(f(s))$

Retorne s^*

- **Algoritmos Genéticos** [Malik et al., 2021, Capítulo 5]: Inspirados na evolução natural, empregam operadores de seleção, cruzamento e mutação para evoluir uma população de soluções ao longo do tempo. Essa abordagem é eficaz para problemas com múltiplas soluções ótimas e espaços de busca complexos.

Para uma compreensão mais aprofundada, recomenda-se consultar os livros *Handbook of Metaheuristics* [Michel Gendreau, 2019], *Metaheuristics: From Design to Implementation* [Talbi, 2009] e *Metaheuristic and Evolutionary Computation: Algorithms and Applications* [Malik et al., 2021], que detalham os fundamentos teóricos e aplicações práticas dessas técnicas.

2.2 Parâmetros e Tuning de Algoritmos

A eficácia de algoritmos de otimização depende fortemente da escolha de parâmetros, que controlam seu comportamento e desempenho. O *tuning* de parâmetros envolve ajustar esses valores para otimizar o desempenho do algoritmo em um conjunto de instâncias de problema. Técnicas comuns incluem:

Grid Search [Liashchynskyi and Liashchynskyi, 2019]: Explora de forma sistemática todas as combinações de parâmetros em um espaço multidimensional pré-definido. É especialmente útil para problemas com poucos hiperparâmetros e estrutura clara (ex.: ajuste de modelos como SVM ou KNN). Sua principal limitação é a complexidade computacional exponencial: mesmo com 4 parâmetros e 5 opções cada, são necessárias $5^4 = 625$ iterações, tornando-o inviável para tarefas de alta dimensionalidade ou com restrições de tempo. Apesar disso, permanece como uma escolha confiável quando a exaustividade é prioritária.

Randomized Search [Bergstra and Bengio, 2012]: Diferentemente do Grid Search, amostra aleatoriamente combinações de parâmetros a partir de distribuições estatísticas (uniforme, normal, etc.), testando um número fixo de configurações. Essa abordagem evita desperdício de recursos computacionais em regiões irrelevantes e permite maior flexibilidade para parâmetros contínuos. Estudos demonstram que, com o mesmo orçamento de iterações, Randomized Search supera o Grid Search em cenários onde apenas uma minoria de parâmetros impacta significativamente o desempenho do modelo, especialmente em grandes conjuntos de dados.

Bayesian Optimization [Snoek et al., 2012]: Emprega modelos probabilísticos, como Processos Gaussianos ou TPE, para criar um modelo substituto que mapeia a relação entre parâmetros e métricas de desempenho. A busca é guiada por funções de aquisição (ex.: Expected Improvement) que equilibram exploração (busca em áreas incertas) e exploração (aprofundamento em regiões promissoras). Sua eficiência reduz drasticamente o número de iterações, tornando-a ideal para problemas de alto custo computacional (ex.: treinamento de redes neurais). No entanto, sua eficácia depende da precisão do modelo substituto e pode ser afetada por ruídos ou variações não capturadas.

Algoritmos Evolutivos [Talbi, 2009, Capítulo 3; Seção 3.2]: Baseiam-se em mecanismos de evolução natural, como seleção, mutação e recombinação, para otimizar parâmetros. Em cada geração, soluções candidatas são avaliadas, e as mais adequadas separam-se para gerar novas combinações. São resistentes a problemas multimodais, não convexos ou ruidosos, se adaptando dinamicamente a mudanças no espaço de busca. Exemplos incluem Algoritmos Genéticos (GA) e Colônia de Formigas (ACO). Apesar de sua capacidade de escapar de mínimos locais, sua convergência costuma ser mais lenta do que métodos gradientes, exigindo calibração cuidadosa de hiperparâmetros como taxa de mutação e tamanho da população.

Neste trabalho, foi escolhido como principal ferramenta de estudo o IRACE (Iterated Racing for Automatic Algorithm Configuration), devido à sua relevância na configuração automática de algoritmos para problemas de otimização.

O IRACE (Iterated Racing for Automatic Algorithm Configuration) [López-Ibáñez et al., 2016] é uma ferramenta amplamente utilizada para a configuração automática de algoritmos. Ele emprega uma abordagem de “corridas iterativas”, onde diferentes configurações de parâmetros são testadas e as menos eficientes são eliminadas. Esse processo continua até que uma configuração ótima seja identificada. O IRACE é particularmente útil em problemas de otimização combinatória, como o TSP, onde a configuração adequada dos parâmetros pode levar a melhorias significativas no desempenho.

2.3 O ACOTSP e o Problema do Caixeiro Viajante (TSP)

O ACOTSP (Ant Colony Optimization for the Traveling Salesman Problem) é uma abordagem baseada em algoritmos inspirados no comportamento de formigas para resolver o TSP de forma aproximada. Neste trabalho, o TSP foi adotado como estudo de caso, enquanto o Irace foi selecionado como método de comparação. Vale ressaltar que a implementação ACOTSP é o exemplo padrão na documentação oficial do Irace. Os algoritmos implementados no ACOTSP são:

- **Ant System (AS):** O primeiro algoritmo proposto para o ACO, que baseia-se em regras simples de atualização de feromônio.
- **Elitist Ant System (EAS):** Reforça explicitamente a melhor solução encontrada até o momento, depositando feromônio adicional proporcional à qualidade dessa solução.
- **Rank-based Ant System (RAS):** Prioriza as melhores soluções ao atualizar o feromônio, aumentando a probabilidade de explorar caminhos promissores.
- **Max-Min Ant System (MMAS):** Limita os valores de feromônio entre máximos e mínimos, evitando a convergência prematura e melhorando a exploração.
- **Ant Colony System (ACS):** Uma versão otimizada do ACO, que utiliza regras de atualização de feromônio mais eficientes e estratégias de elitismo para acelerar a convergência.

O Problema do Caixeiro Viajante (TSP) é um dos mais emblemáticos problemas de otimização combinatória, com aplicações em logística, planejamento de rotas e ciência da computação. Sua formulação simples — encontrar a menor rota que visita todas as cidades exatamente uma vez — contrasta com sua complexidade computacional, classificada como *NP-difícil* [Reinelt, 1994].

A obra de Reinelt [1994] teve um impacto significativo no estudo do TSP, oferecendo uma exploração abrangente dos aspectos computacionais envolvidos. Seu livro *The Traveling Salesman: Computational Solutions for TSP Applications* aborda desde conceitos fundamentais de

teoria dos grafos até técnicas algorítmicas, como heurísticas de construção (vizinho mais próximo, inserção mais barata), heurísticas de melhoria local (2-opt, Lin-Kernighan), e métodos exatos baseados em programação linear e branch-and-cut.

Além disso, Reinelt desenvolveu a *TSPLIB* [Reinelt, 1991], uma biblioteca padrão de instâncias do TSP (simétricas e assimétricas) e problemas relacionados de roteamento de veículos. A *TSPLIB* se tornou um benchmark essencial para a comunidade científica, permitindo a comparação justa e replicável do desempenho de diferentes algoritmos. A distribuição online da *TSPLIB95* facilitou o acesso e o estudo das instâncias por pesquisadores em todo o mundo.

2.4 Aprendizado de Máquina e Regressores

O Aprendizado de Máquina (ML) é uma área da inteligência artificial que envolve o desenvolvimento de algoritmos capazes de aprender a partir de dados [Murphy, 2012]. Em vez de seguir regras explícitas, os modelos de ML aprendem padrões e relações nos dados para fazer previsões ou tomar decisões. No contexto da *OptAI*, o ML é utilizado para treinar regressores que preveem os melhores conjuntos de parâmetros para novas instâncias do TSP.

Regressores são modelos de ML utilizados para prever valores contínuos, como parâmetros de algoritmos. Eles podem ser categorizados em regressores simples e ensemble:

- **Regressores Simples:** Utilizam um único modelo para fazer previsões. São geralmente mais fáceis de interpretar e mais rápidos para treinar, mas podem não capturar padrões complexos nos dados tão bem quanto os métodos ensemble. Exemplos incluem:
 - **Decision Tree** [Quinlan, 1986]: Simples e interpretável, mas pode sofrer com overfitting em dados complexos.
 - **Support Vector Regressor (SVR)** [Basak et al., 2007]: Eficaz em capturar relações não lineares, mas sensível a parâmetros de configuração.
 - **Regressor Padrão do scikit-learn** [Pedregosa et al., 2018]: Baseado em algoritmos clássicos, como Regressão Linear, para comparação de desempenho.
- **Ensemble:** Combinam múltiplos modelos para melhorar a precisão das previsões. São geralmente mais precisos, mas podem ser mais complexos e computacionalmente intensivos. Exemplos incluem:
 - **XGBoost Regressor** [Chen and Guestrin, 2016]: Conhecido por sua capacidade de lidar com grandes volumes de dados e capturar interações complexas entre variáveis.
 - **Random Forest** [Louppe, 2015]: Robusto e menos propenso a overfitting, com boa performance em dados não lineares.

- **Gradient Boosting [He et al., 2019]**: Combina múltiplos modelos fracos para melhorar a precisão, sendo eficaz em dados heterogêneos.

A diversidade de modelos permite explorar diferentes abordagens para a previsão de parâmetros otimizados, adaptando-se a cenários específicos. A escolha do modelo ideal depende de fatores como a complexidade do problema, o tamanho dos dados e os recursos computacionais disponíveis.

Trabalhos Relacionados

Este capítulo apresenta uma revisão aprofundada da literatura relevante que fundamenta o desenvolvimento deste trabalho. Exploraremos as contribuições seminais e contemporâneas que moldaram o campo da otimização combinatória, com foco particular no Problema do Caixeiro Viajante (TSP), bem como em técnicas avançadas de meta-heurísticas, configuração automática de algoritmos, otimização de hiperparâmetros e a emergente área de Aprendizado de Máquina para a redução de problemas complexos, que são cruciais para o nosso estudo.

3.1 Configuração Automática de Algoritmos

Algoritmos de otimização frequentemente possuem um grande número de parâmetros que precisam ser ajustados cuidadosamente para maximizar seu desempenho em instâncias específicas ou classes de problemas. A tarefa de ajustar manualmente dezenas ou centenas de parâmetros para obter o melhor desempenho em uma classe específica de instâncias é um processo demorado, propenso a erros e difícil de replicar. Isso motivou o desenvolvimento de métodos para a configuração automática de algoritmos (Automatic Algorithm Configuration — AAC).

No contexto deste trabalho, optou-se por utilizar o IRACE (Iterated Racing for Automatic Algorithm Configuration) [López-Ibáñez et al., 2016] e a biblioteca Optuna [Akiba et al., 2019] para a otimização de parâmetros. A escolha do IRACE baseia-se em sua eficácia comprovada em cenários de configuração automática, especialmente para problemas com espaços de parâmetros complexos e instâncias de treinamento limitadas. Já a Optuna foi integrada para explorar estratégias de busca adaptativas e *pruning*, permitindo uma otimização mais eficiente e flexível. Essa combinação visa explorar diferentes abordagens para validar a robustez da OptAI em contextos variados.

Além do IRACE e da Optuna, outros métodos notáveis para a configuração automática de algoritmos incluem:

- **SMAC (Sequential Model-Based Algorithm Configuration)** [Hutter et al., 2011]: Utiliza modelos probabilísticos para prever o desempenho de configurações de parâmetros e seleciona as configurações mais promissoras para avaliação.
- **Hyperopt** [Bergstra et al., 2015]: Uma biblioteca de otimização que suporta várias estratégias de busca, incluindo Random Search, Tree of Parzen Estimators (TPE) e Adaptive TPE.
- **Bayesian Optimization** [Snoek et al., 2012]: Utiliza modelos bayesianos para otimizar hiperparâmetros, equilibrando exploração e intensificação de maneira eficiente.

Esses métodos são fundamentais para a configuração automática de algoritmos, permitindo a otimização eficiente e eficaz de parâmetros em diversos cenários. Em futuras etapas, é possível explorar outras abordagens, como SMAC ou Bayesian Optimization, para comparar resultados e aprimorar a flexibilidade da OptAI.

3.2 Otimização de Hiperparâmetros com Abordagens Dinâmicas

No campo de Aprendizado de Máquina, a otimização de hiperparâmetros (HPO) é uma subárea da AAC que se concentra especificamente em encontrar os melhores valores para os hiperparâmetros que controlam o processo de aprendizado de um modelo. Com a crescente complexidade dos modelos de Aprendizado de Máquina, especialmente redes neurais profundas, a HPO eficiente tornou-se uma etapa crítica.

O framework Optuna [Akiba et al., 2019] representa uma abordagem inovadora para HPO com sua filosofia “define-by-run”. Diferentemente de *frameworks* mais antigos que exigiam a definição estática e completa do espaço de busca de hiperparâmetros antes do início do processo, o Optuna permite que o espaço de busca seja construído dinamicamente durante a execução das avaliações. Essa flexibilidade é particularmente útil ao lidar com modelos complexos que possuem parâmetros condicionais (onde a relevância de um parâmetro depende do valor de outro) ou estruturas variáveis. Além disso, o Optuna integra mecanismos eficientes de amostragem de parâmetros e de poda (pruning) de experimentos não promissores, o que acelera significativamente o processo de HPO, tornando-o mais prático para aplicações em larga escala.

3.3 Avaliação da importância de hiperparâmetros

O artigo “An Efficient Approach for Assessing Hyperparameter Importance” de Hutter et al. [2014] introduz uma abordagem eficiente para avaliar a importância dos hiperparâmetros em

algoritmos de aprendizado de máquina e solucionadores de problemas combinatórios. A metodologia utiliza modelos de floresta aleatória ajustados a dados coletados por otimização bayesiana, combinada com um algoritmo de tempo linear para calcular marginais de previsões. Essa técnica é integrada a um framework de ANOVA funcional, que quantifica a importância de hiperparâmetros individuais e suas interações. Experimentos com frameworks de Aprendizado de Máquina e solucionadores combinatórios demonstraram que a maior parte da variação no desempenho é atribuída a poucos hiperparâmetros, fornecendo insights valiosos sobre a relação entre configurações de hiperparâmetros e resultados. A abordagem oferece uma base científica para designers de algoritmos compreenderem como escolhas de projeto afetam o desempenho geral, contribuindo para uma otimização mais eficaz e uma análise dos mecanismos subjacentes.

Essa metodologia foi posteriormente estendida por [van Rijn and Hutter \[2018\]](#) em seu trabalho “Hyperparameter Importance Across Datasets”, que aplica o mesmo framework de ANOVA funcional, mas sob uma perspectiva de Meta-aprendizado. Enquanto [Hutter et al. \[2014\]](#) foca na análise de importância de hiperparâmetros para um único conjunto de dados, [van Rijn and Hutter \[2018\]](#) generalizam os resultados para múltiplos conjuntos de dados, identificando padrões universais de importância e valores ideais de hiperparâmetros. Isso é alcançado por meio da análise de dados experimentais do OpenML, permitindo inferir priors (distribuições de probabilidade) para valores de hiperparâmetros que tendem a performar bem em diferentes cenários.

As informações obtidas por meio desses estudos foram essenciais para a definição dos grids de parâmetros da `OptAI`, bem como para sua aplicação por meio do método `RandomizedSearchCV`. Essa abordagem viabiliza uma busca mais eficiente e direcionada aos hiperparâmetros críticos, reduzindo custos computacionais e aumentando a probabilidade de identificar configurações ótimas. Melhorias futuras incluem a criação de uma meta-base de conhecimento, a ser implementada quando a biblioteca atingir maturidade e acumular informações suficientes.

3.4 Calibração dos parâmetros de Meta-heurísticas utilizando Aprendizado de Máquina

O artigo “Learnheuristics: Hybridizing Metaheuristics with Machine Learning for Optimization with Dynamic Inputs” de [Calvet et al. \[2017\]](#) propõe um framework híbrido para calibração automática de parâmetros em meta-heurísticas, integrando técnicas de Aprendizado de Máquina (ML) para resolver problemas de otimização em ambientes dinâmicos, onde os parâmetros do problema variam ao longo do tempo. A abordagem, denominada *learnheuristics*, utiliza modelos de ML treinados em dados históricos para identificar configurações promissoras de parâmetros, reduzindo o tempo computacional e melhorando a adaptação a cenários dinâmicos. Essa filosofia é central para a biblioteca `OptAI`, que adota a mesma lógica: integrar métodos clássicos de otimização com ferramentas de ML para calibrar parâmetros de forma eficiente.

Um estudo posterior, “Hybrid Metaheuristics with Machine Learning for Dynamic Optimization Problems” de [Karimi-Mamaghan et al. \[2022\]](#), complementa essa linha de pesquisa ao apresentar uma taxonomia detalhada das formas de integração entre ML e meta-heurísticas, com destaque para aplicações na calibração de parâmetros. O artigo explora métodos como previsão de padrões em configurações eficazes, seleção de operadores adaptativos e geração de soluções iniciais informadas. Além disso, discute desafios como a incerteza nos dados de entrada e a adaptação a mudanças não estacionárias, validando a eficácia dessas técnicas em problemas práticos, como o roteamento de veículos dinâmico (VRP) e gestão de estoque. Esses *insights* reforçam a relevância da OptAI, que aplica princípios semelhantes à calibração de parâmetros em problemas combinatórios.

Na OptAI, a calibração de parâmetros é implementada de forma prática, utilizando o TSP como exemplo ilustrativo. Embora o TSP seja o problema alvo, a biblioteca é modular e aplicável a qualquer problema de otimização combinatória. A OptAI coleta dados históricos de execução via IRACE, armazenando-os em uma base de conhecimento estruturada. Esses dados são pré-processados e usados para treinar regressores (como XGBoost, Random Forest e SVR), que aprendem relações entre características das instâncias (ex.: número de arestas, estatísticas de dados) e configurações de parâmetros eficazes. Essa abordagem reflete a filosofia das *learnheuristics*, onde modelos de ML preveem configurações otimizadas, evitando buscas exaustivas e direcionando a meta-heurística para regiões promissoras do espaço de soluções.

Uma diferença notável é que a OptAI foca em calibração estática de parâmetros com adaptação a novas instâncias, enquanto o framework de [Calvet et al. \[2017\]](#) é projetado para ajuste contínuo em ambientes dinâmicos (ex.: mudanças em tempo real). No entanto, ambos compartilham a lógica de usar dados históricos para reduzir a complexidade computacional. Na OptAI, isso é concretizado por meio de grids de parâmetros (Fast, Stock e Slow) e da biblioteca Optuna, que seleciona configurações ótimas com base nas previsões dos regressores. Essa estratégia é análoga ao uso de ML para inicialização ou filtragem de soluções nas *learnheuristics*, garantindo que a busca seja mais direcionada e eficiente. Além disso, a OptAI retorna configurações personalizadas para cada instância do TSP, ajustando os parâmetros conforme as características únicas de cada problema.

O artigo de [Karimi-Mamaghan et al. \[2022\]](#) destaca a importância de técnicas de agrupamento (ex.: k-means, SOM) para lidar com a incerteza estatística nos dados de entrada, segmentando instâncias em classes com distribuições distintas. Embora a OptAI ainda não explore esse aspecto, ele sugere uma direção futura para aprimorar a robustez da calibração em cenários com variabilidade estocástica. Por fim, os resultados experimentais da OptAI, onde o XGBoost Regressor se destacou na previsão de configurações de parâmetros, validam a eficácia da integração entre ML e meta-heurísticas. Isso reforça a premissa dos artigos de [Calvet et al. \[2017\]](#) e [Karimi-Mamaghan et al. \[2022\]](#): a hibridização dessas técnicas é uma solução robusta para calibrar parâmetros em problemas complexos, seja em ambientes dinâmicos ou estáticos com alta variabilidade entre instâncias.

3.5 Relação com o Trabalho Atual

Este trabalho se insere na linha de pesquisa que explora o uso de Aprendizado de Máquina para recomendar configurações de parâmetros em algoritmos de otimização, com o Problema do Caixeiro Viajante (TSP) como estudo de caso. Ao contrário de abordagens que priorizam a redução da complexidade dos problemas em si [Reinelt, 1994], a OptAI se concentra na personalização de parâmetros de meta-heurísticas com base em características específicas das instâncias do TSP. A biblioteca combina técnicas modernas de otimização automática, como IRACE e Optuna, com modelos de regressão (XGBoost, Random Forest, SVR), para prever conjuntos de parâmetros otimizados para novas instâncias.

A OptAI se diferencia por três aspectos principais. Primeiro, sua abordagem de personalização por instância contrasta com métodos tradicionais, como os propostos por Calvet et al. [2017], que geralmente utilizam configurações genéricas ou ajustes manuais. A biblioteca gera configurações especializadas para cada instância do TSP, aproveitando regressores treinados em uma base histórica de logs do IRACE. Esses modelos identificam relações entre características das instâncias (número de arestas e vértices) e dados estatísticos (desvio padrão, mínimos e máximos), permitindo adaptação precisa a cada problema. Segundo, a OptAI integra ferramentas modernas, combinando frameworks consolidados (IRACE para coleta de dados e Optuna para otimização) com técnicas avançadas de Aprendizado de Máquina. Terceiro, a biblioteca prioriza escalabilidade e generalização, sendo modular e adaptável a outros problemas de otimização combinatória, diferentemente de trabalhos como Karimi-Mamaghan et al. [2022], que se limitam a aplicações específicas.

Além disso, a OptAI valoriza a diversidade dos dados de treinamento, armazenando logs de execução em uma base de conhecimento estruturada. Essa estratégia permite generalização a cenários diversos, evitando viés para instâncias similares.



Metodologia

Este capítulo descreve a metodologia adotada para o desenvolvimento e avaliação da biblioteca `OptAI`, focando na seleção de instâncias do Problema do Caixeiro Viajante (TSP), na escolha da linguagem de programação, na configuração inicial de parâmetros, no pré-processamento de dados e no treinamento de modelos de regressão. A Figura 4.1 ilustra a arquitetura geral da metodologia proposta, destacando as etapas principais.

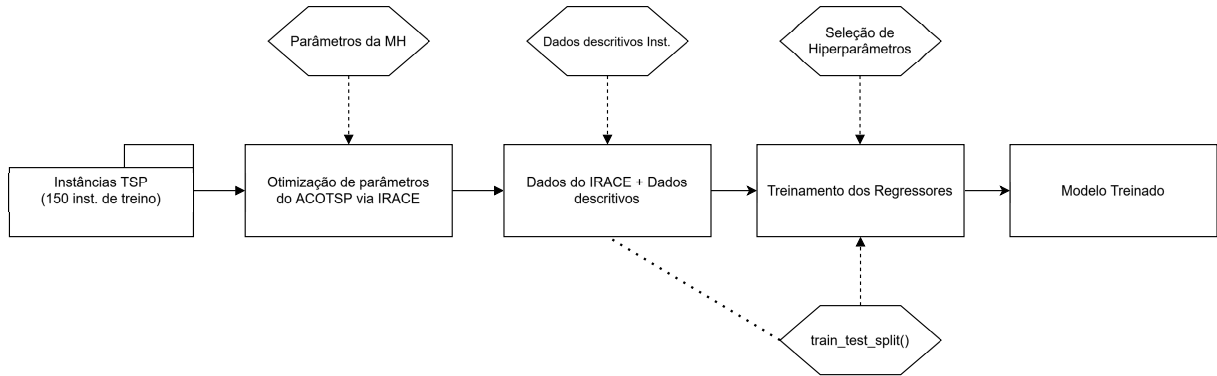


Figura 4.1: Arquitetura da Metodologia

4.1 Seleção das Instâncias TSP

Para este trabalho, foram selecionadas 200 instâncias do Problema do Caixeiro Viajante (TSP), sendo 150 utilizadas para treinamento e 50 para teste e validação. Essas instâncias foram obtidas a partir da biblioteca TSPLIB95 [Reinelt, 1991], que é amplamente utilizada na comunidade de pesquisa para *benchmarking* de algoritmos de otimização. A escolha dessas instâncias visa garantir uma diversidade de cenários, variando em tamanho e complexidade, para avaliar a robustez e a eficácia da abordagem proposta.

4.2 Escolha da Linguagem Python

A escolha da linguagem Python para o desenvolvimento da biblioteca OptAI é fundamentada em várias vantagens que essa linguagem oferece. Python é amplamente reconhecido por sua simplicidade e legibilidade, o que facilita o desenvolvimento e a manutenção do código. A sintaxe clara e intuitiva de Python permite que desenvolvedores, independentemente do nível de experiência, possam entender e modificar o código com relativa facilidade.

Além disso, Python possui uma vasta comunidade de desenvolvedores e um rico ecossistema de bibliotecas e *frameworks*, especialmente na área de Aprendizado de Máquina e Otimização. Bibliotecas como `scikit-learn` [Pedregosa et al., 2018], `XGBoost` [Chen and Guestrin, 2016], `Optuna` [Akiba et al., 2019] e `TSPLIB95` [Reinelt, 1991] fornecem ferramentas poderosas e bem documentadas que aceleram o processo de desenvolvimento e permitem a implementação de algoritmos complexos de maneira eficiente.

A interoperabilidade de Python com outras linguagens e ferramentas também é um ponto forte, permitindo a integração com sistemas existentes e a expansão das funcionalidades da biblioteca OptAI.

4.3 Configuração Inicial com IRACE

A configuração inicial dos parâmetros para o ACOTSP foi realizada utilizando o IRACE (Iterated Racing for Automatic Algorithm Configuration) [López-Ibáñez et al., 2016]. O IRACE é uma ferramenta que emprega uma abordagem de corrida iterativa para encontrar os melhores parâmetros para um algoritmo dado, otimizando seu desempenho em um conjunto de instâncias de problema. Neste trabalho, o IRACE foi utilizado para rodar o binário do ACOTSP com um cenário de parâmetros predefinido, gerando logs de execução que contêm informações detalhadas sobre os parâmetros utilizados e o desempenho obtido.

4.4 Pré-processamento e Criação do Dataframe com Logs do IRACE e Dados Descritivos

Os logs de execução do IRACE foram submetidos a um pré-processamento visando gerar uma base de dados estruturada. Esse processo incluiu a extração de informações-chave, como os parâmetros configurados no comando de execução do IRACE, seus valores associados, o desempenho alcançado e a instância TSP correspondente. Esses dados foram estruturados em um dataframe inicial, servindo de base para as etapas posteriores de análise e treinamento.

A criação de um dataframe robusto a partir dos logs do IRACE e dos dados descritivos das instâncias TSP é um passo crucial na metodologia da OptAI. Os logs do IRACE contêm informações valiosas sobre os parâmetros utilizados e o desempenho obtido em cada execução.

Ao pré-processar esses logs e organizá-los em um dataframe, é possível estruturar os dados de maneira que facilite a análise e o treinamento de modelos de Aprendizado de Máquina.

A inclusão de dados descritivos das instâncias TSP — como número de cidades, arestas, densidade, grau médio e comprimento médio do caminho — enriquece o dataframe com informações contextuais que facilitam a identificação de padrões e relações entre os parâmetros e o desempenho do algoritmo. Esses dados oferecem uma visão abrangente das características das instâncias TSP, possibilitando uma análise mais aprofundada e a detecção de fatores que impactam o desempenho do ACOTSP.

A estruturação dos dados em um dataframe também facilita a manipulação e a visualização dos dados, permitindo a realização de análises exploratórias e a identificação de padrões e anomalias. Além disso, a utilização de um dataframe padronizado facilita a integração com outras ferramentas de análise de dados e Aprendizado de Máquina, promovendo a interoperabilidade e a expansão das funcionalidades da OptAI.

4.5 Treinamento dos Regressores

A partir de um dataframe robusto contendo parâmetros, métricas de desempenho e atributos descritivos das instâncias, foi iniciado o treinamento de seis modelos de regressão distintos: XGBoost Regressor, Random Forest, Decision Tree, Support Vector Regressor (SVR), Gradient Boosting e o modelo de baseline do scikit-learn. Para cada modelo, configuraram-se três grids de parâmetros — Fast, Stock e Slow — com variações controladas de tamanho e complexidade, ajustadas às dimensões da base de dados e à capacidade computacional disponível. A otimização dos hiperparâmetros foi realizada por meio do Randomized Search CV [Bergstra and Bengio, 2012], método que amostra aleatoriamente combinações de parâmetros, maximizando a eficiência do treinamento. Apesar de todos os modelos terem sido avaliados, o XGBoost foi selecionado como referência devido ao desempenho superior e à estabilidade observada nas métricas de validação.

5

OptAI

Este capítulo detalha a arquitetura e o funcionamento da biblioteca OptAI, projetada para otimizar parâmetros em algoritmos de resolução do Problema do Caixeiro Viajante (TSP). A estrutura modular da OptAI integra ferramentas como IRACE, Optuna e modelos de regressão, proporcionando uma abordagem automatizada e eficiente para a configuração de meta-heurísticas.

5.1 Arquitetura da OptAI

A OptAI é estruturada em módulos que interagem entre si para realizar a otimização de parâmetros. A arquitetura geral da biblioteca pode ser descrita pelos diagramas a seguir:

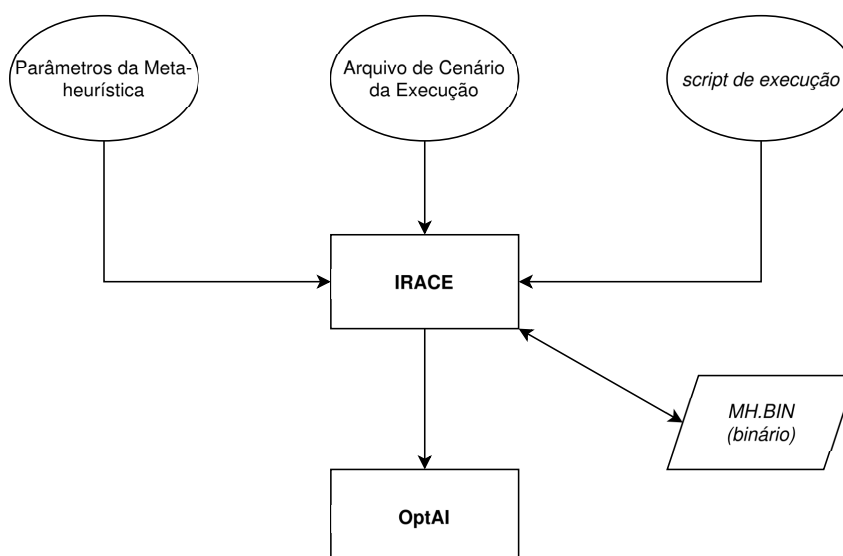


Figura 5.1: Processo de otimização realizado pelo IRACE

A Figura 5.1 mostra o processo de otimização do IRACE, que envolve a recepção de parâmetros da meta-heurística, do cenário de execução, do script e do binário associado. Após o término das execuções conduzidas pelo IRACE, inicia-se a fase do OptAI.

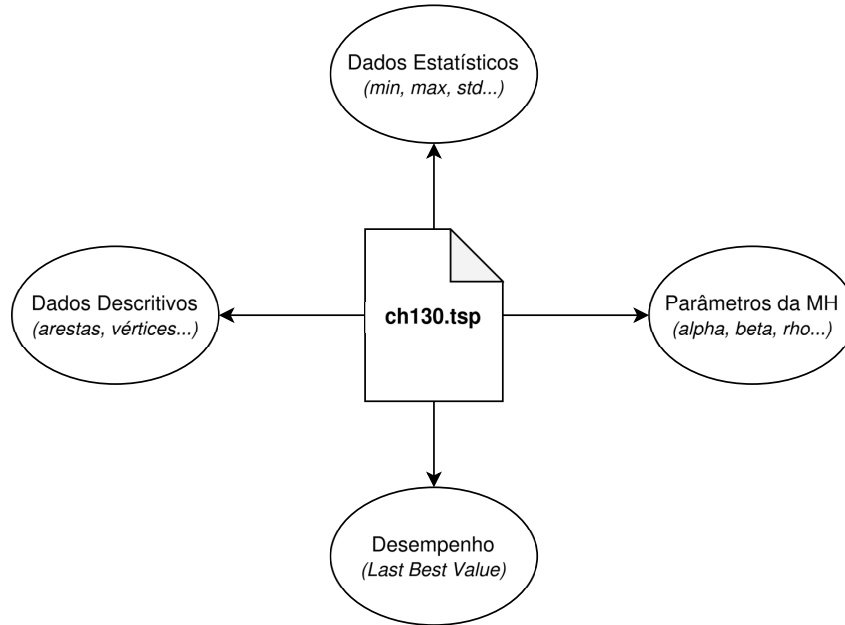


Figura 5.2: Dados das Instâncias

A Figura 5.2 apresenta os dados de uma instância do dataframe de treinamento do regressor. Esse conjunto inclui informações de múltiplas instâncias, bem como variações dos mesmos casos com parâmetros distintos, resultando em desempenhos diferentes. Os dados compreendem: i) características descritivas do arquivo “.tsp” (arestas, vértices, densidade, etc.); ii) estatísticas da instância (mínimos, máximos, médias, desvio padrão, etc.); iii) valores dos parâmetros da meta-heurística aplicada (alpha, beta, rho, ants, etc.); e iv) o desempenho alcançado ao executar a instância com os parâmetros da meta-heurística.

A Figura 5.3 ilustra os processos do OptAI, incluindo o pré-processamento dos logs do IRACE, o treinamento dos regressores e o retorno dos conjuntos de parâmetros novos para as novas instâncias.

5.2 Execução Completa da OptAI: Visão Geral

A seguir, apresenta-se a visão geral da execução completa da OptAI, abrangendo desde a configuração do IRACE até a comparação final dos resultados.

A Figura 5.1 mostra o fluxo inicial da execução da OptAI, incluindo a configuração do IRACE (cenário, instâncias de treino, binário da meta-heurística, tempo limite, etc.) e sua execução com duração limitada a 1h45. Após a finalização, são coletados os logs de execução do IRACE e das instâncias “.tsp” de treino. Com a obtenção dos parâmetros ótimos do IRACE, inicia-se o pré-processamento:

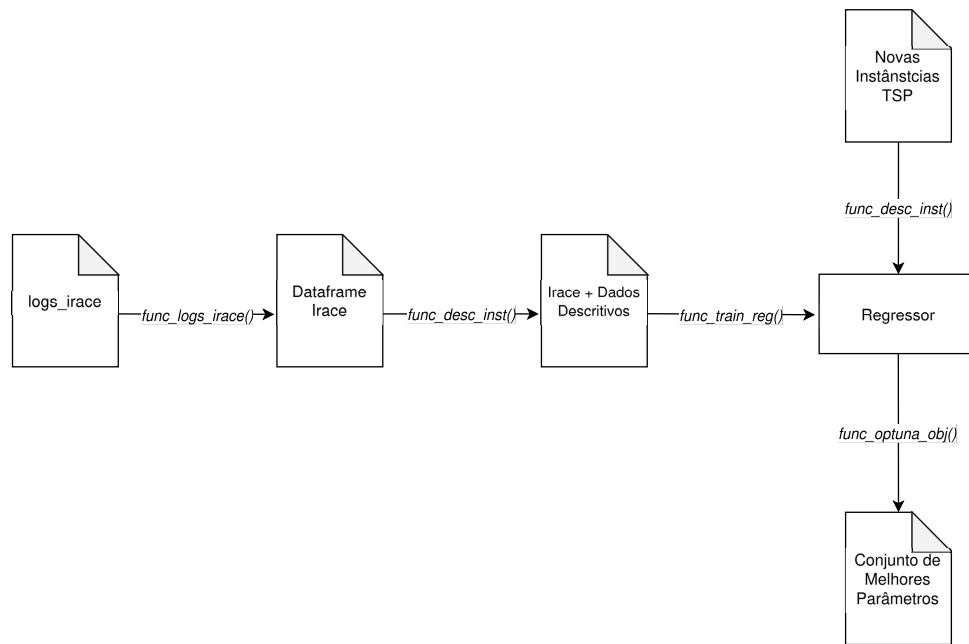


Figura 5.3: Arquitetura da OptAI

1. Identificação dos comandos, instâncias e desempenhos registrados;
2. Seleção das instâncias de treino e extração de dados descritivos (arestas, vértices, densidade) e estatísticos (mínimos, máximos, médias, desvio padrão).

Essas informações são consolidadas em um dataframe, servindo como base para o treinamento dos regressores.

O treinamento dos modelos é realizado com três configurações de grid de parâmetros (fast, stock, slow), sendo stock a padrão. Os dados são divididos em 80% para treino e 20% para teste, com seleção do melhor grid via `RandomizedSearchCV`. As métricas de validação incluem:

- **MSE (Mean Squared Error):** mede o erro médio quadrático;
- **MAE (Mean Absolute Error):** mede o valor absoluto médio do erro;
- **MAPE (Mean Absolute Percentage Error):** expressa o erro em percentual.

Para obter mais detalhes sobre as métricas de validação, consulte [Murphy \[2012\]](#).

Com o regressor treinado, inicia-se a recomendação de parâmetros via `Optuna`, configurando a meta-heurística, tipo de estudo (minimização para o TSP), número de *trials*, tempo máximo por *trial* e *pruner*. Para as 50 instâncias de teste, a `Optuna` identifica o conjunto de parâmetros ótimo. Destaca-se que os dados de teste não são utilizados durante o treinamento do regressor. A `Optuna` baseia-se exclusivamente no modelo treinado, empregando características descritivas das instâncias de teste (número de cidades, densidade, etc.) para sugerir parâmetros ideais, reproduzindo um cenário de generalização realista. Essa abordagem assegura que o desempenho reportado represente a capacidade do modelo de se adaptar a novas instâncias,

evitando contaminação dos dados de teste no processo de aprendizado. Por fim, os resultados da Optuna são confrontados com os parâmetros otimizados pelo IRACE (detalhes no Capítulo 6).

5.3 Módulos da OptAI

A arquitetura da OptAI é dividida em módulos principais, cada um desempenhando uma função específica no processo de otimização de parâmetros, conforme resumido a seguir:

- **Módulo de Pré-processamento:** Processa os logs gerados pelo IRACE e cria o dataframe inicial, extraíndo informações relevantes como parâmetros utilizados e desempenho obtido.
- **Módulo de Incremento de Dados:** Adiciona dados descritivos das instâncias TSP ao dataframe, utilizando a biblioteca TSPLIB95 para extrair atributos como número de cidades, densidade, grau médio, e dados estatísticos.
- **Módulo de Treinamento:** Treina os regressores utilizando diferentes grids de parâmetros (Fast, Stock e Slow) para seis regressores: XGBoost, Random Forest, Decision Tree, SVR, Gradient Boosting e o regressor padrão do `scikit-learn`.
- **Módulo de Otimização:** Utiliza a biblioteca Optuna para otimizar ainda mais os parâmetros dos regressores treinados, permitindo ajustes finos e melhorando o desempenho na previsão de parâmetros para novas instâncias TSP.
- **Módulo de Persistência:** Permite salvar e carregar regressores treinados utilizando a biblioteca joblib, economizando tempo e recursos computacionais.
- **Módulo de Retroalimentação:** Atualiza continuamente o dataframe com novos dados de execução, enriquecendo a base de dados e melhorando a precisão dos modelos de previsão ao longo do tempo.

Os pseudocódigos apresentados a seguir detalham o conteúdo dos módulos.

5.4 Pré-processamento dos Logs do IRACE

A OptAI inicia o processo de otimização pré-processando os logs gerados pelo IRACE. Esses logs contêm informações detalhadas sobre os parâmetros utilizados e o desempenho obtido em cada execução. O pré-processamento envolve a extração de informações relevantes dos logs e a organização dessas informações em um dataframe. O pseudocódigo a seguir ilustra o processo de pré-processamento:

Função `pré_processar_logs(logs)` :

Inicialize um dataframe vazio

Para cada log em logs:

Extraia parâmetros utilizados

Extraia desempenho obtido

Extraia instância TSP correspondente

Adicione uma nova linha ao dataframe com os dados extraídos

Retorne dataframe

Este módulo é crucial para transformar os dados brutos dos logs em uma estrutura organizada que pode ser utilizada para treinamento de modelos. A Figura 5.3 ilustra a arquitetura da OptAI, destacando a integração deste módulo no fluxo de trabalho.

5.5 Incremento do Dataframe com Dados Descritivos

Após o pré-processamento inicial, a OptAI incrementa o dataframe com dados descritivos das instâncias TSP. Utilizando a biblioteca TSPLIB95, são extraídos atributos como número de cidades, número de arestas, densidade, grau médio, comprimento médio do caminho, e diversos dados estatísticos. Esses dados descritivos fornecem um contexto adicional que pode ajudar os modelos a capturar padrões e relações entre os parâmetros e o desempenho do algoritmo. O pseudocódigo a seguir demonstra essa funcionalidade:

Função `incrementar_dataframe(dataframe, instâncias)`:

Para cada instância em instâncias:

Carregue a instância utilizando TSPLIB95

Extraia atributos descritivos (número de cidades, arestas, etc.)

Calcule dados estatísticos (média, desvio padrão, etc.)

Adicione os atributos descritivos ao dataframe correspondente

Retorne dataframe

5.6 Treinamento dos Regressores

A partir do dataframe final, composto pelos logs do IRACE (contendo parâmetros configurados e desempenho obtido) e pelos dados descritivos das instâncias TSP (como número de cidades, densidade, grau médio e comprimento médio do caminho) [Reinelt, 1991], foi realizado o treinamento dos modelos de regressão. A OptAI implementa seis regressores diferentes: XGBoost Regressor, Random Forest, Decision Tree, Support Vector Regressor (SVR), Gradient Boosting e o regressor padrão do `scikit-learn` [Pedregosa et al., 2018]. Apesar de todos os modelos serem implementados, o XGBoost foi adotado como modelo inicial padrão devido ao

seu desempenho superior nos testes realizados, equilibrando precisão e eficiência computacional. Antes do treinamento, os dados foram divididos em conjuntos de treino e teste utilizando a função `train_test_split()` do `scikit-learn`, com 80% dos dados destinados ao treinamento (`X_train`, `y_train`) e os 20% restantes para avaliação (`X_test`, `y_test`). Para cada regressor, são definidos três tipos de grid de parâmetros (Fast, Stock e Slow), e o *tunning* de parâmetros é realizado utilizando o Randomized Search CV [Bergstra and Bengio, 2012]. O pseudocódigo a seguir ilustra o processo de treinamento:

Função `treinar_regressores(dataframe)`:

Divida o dataframe em características (X) e alvo (y)

Divida X e y em `X_train`, `X_test`, `y_train`, `y_test` com `test_size=0.2`

Inicialize uma lista vazia para armazenar os regressores treinados

Para cada regressor em [XGB, RF, DT, SVR, GB, SKL]:

Defina grids de parâmetros [Fast, Stock, Slow]

Realize Randomized Search CV no conjunto de treino (`X_train`, `y_train`)

Treine o regressor com os melhores parâmetros encontrados

Adicione o regressor treinado à lista

Retorne lista de regressores treinados

A escolha do grid de parâmetros (Fast, Stock, Slow) depende do tamanho da base de dados e dos recursos computacionais disponíveis. O grid Fast é ideal para cenários com limitações de tempo e recursos, enquanto o grid Slow é mais adequado para cenários onde a precisão é prioritária e há maior disponibilidade de recursos.

5.7 Otimização com Optuna

Após o treinamento dos regressores, a OptAI utiliza a biblioteca Optuna para otimizar ainda mais os parâmetros dos modelos. O Optuna permite ajustar finamente os regressores treinados, melhorando seu desempenho na previsão de conjuntos de parâmetros para novas instâncias TSP. O pseudocódigo a seguir demonstra a otimização com Optuna:

Função `otimizar_com_optuna(regressor, dataframe)`:

Defina espaço de busca dos parâmetros

Inicialize estudo Optuna

Para cada iteração:

Proponha novos parâmetros utilizando Optuna

Avalie desempenho do regressor no dataframe com os novos parâmetros

Atualize estudo Optuna com o desempenho avaliado

Retorne melhores parâmetros encontrados

A Optuna é uma ferramenta poderosa para otimização hiperparamétrica, permitindo explorar um amplo espaço de parâmetros de maneira eficiente.

5.8 Salvamento e Carregamento de Regressores Treinados

A OptAI permite salvar regressores treinados e carregá-los posteriormente, economizando tempo e recursos computacionais. Essa funcionalidade é útil para reutilizar modelos treinados em diferentes cenários ou para evitar retrabalho. O pseudocódigo a seguir ilustra o salvamento e carregamento de regressores:

Função `salvar_regressor(regressor, caminho):`

Utilize a biblioteca `joblib` para salvar o regressor no caminho especificado

Função `carregar_regressor(caminho):`

Utilize a biblioteca `joblib` para carregar o regressor do caminho especificado

Retorne regressor

A biblioteca `joblib` é utilizada para realizar o salvamento e carregamento dos modelos de maneira eficiente.

5.9 Retroalimentação e Incremento de Dados

A OptAI permite a retroalimentação e o incremento de dados, onde novos dados de execução podem ser incorporados ao dataframe existente. Isso enriquece a base de dados e melhora a precisão dos modelos de previsão ao longo do tempo. O pseudocódigo a seguir demonstra essa funcionalidade:

Função `retroalimentar_dados(novo_log):`

Pré-processe `novo_log`

Incrementar dataframe com `novo_log`

Atualize regressores com o dataframe incrementado

A capacidade de retroalimentação contínua permite que a OptAI se adapte a novos cenários e melhore seu desempenho ao longo do tempo.

5.10 Pontos Finais

A biblioteca OptAI proporciona uma abordagem eficiente e intuitiva para a configuração automática de algoritmos no problema do Caixeiro Viajante (TSP). Inclui funcionalidades como

pré-processamento de dados, treinamento de regressores, otimização com Optuna, salvamento e carregamento de modelos, além de retroalimentação de dados, tornando-a uma ferramenta poderosa para pesquisadores e profissionais em otimização e machine learning. Além de facilitar a experimentação com múltiplas variações de algoritmos ACO, a OptAI permite identificar a abordagem mais adequada para cada cenário, oferecendo uma solução robusta e adaptativa para a otimização de parâmetros no TSP.

6

Resultados e Discussão

Este capítulo apresenta e discute os resultados obtidos ao comparar a OptAI com o IRACE no problema do Caixeiro Viajante (TSP). A análise dos resultados é fundamental para avaliar a eficácia da OptAI e identificar suas vantagens e limitações em relação ao IRACE, que é uma ferramenta amplamente utilizada para a configuração automática de algoritmos.

6.1 Configuração do Ambiente de Teste

Para garantir a reprodutibilidade e a precisão dos experimentos, é essencial descrever o ambiente de teste utilizado. A máquina utilizada para gerar os resultados é composta por um processador AMD Ryzen 5 5600X, 16 GB de memória RAM, uma GPU Nvidia RTX 4070 e o sistema operacional Linux Mint 22 Cinnamon 6.2.9. É importante ressaltar que a GPU não foi utilizada para treinar os regressores ou em qualquer parte do processo, garantindo que a comparação seja baseada exclusivamente no desempenho da CPU.

6.2 Parâmetros de Execução

O IRACE foi executado com um tempo máximo de 1h45, definido para garantir a realização de iterações suficientes na identificação de conjuntos de parâmetros eficazes. Adicionalmente, o treinamento do regressor da OptAI e a obtenção dos parâmetros para as instâncias de teste demandaram 8 minutos. A duração reduzida destaca a eficiência da OptAI na otimização paramétrica.

6.3 Cenário de Treinamento e Teste

Para criar o cenário de treinamento e teste, foram selecionadas 200 instâncias TSP. Dessas, 150 foram utilizadas para treinamento e 50 para teste. O IRACE utiliza uma técnica de otimização que automaticamente reduz o espaço de instâncias de treinamento para diminuir o custo computacional. Como utilizamos o IRACE em sua configuração padrão de uso, não alteramos o parâmetro para utilizar todas as instâncias possíveis. O IRACE utilizou, em média, entre 15 a 17 instâncias das 150 disponíveis para treinamento. Isso é bastante desfavorável para a OptAI, pois ela se beneficia de dados de instâncias variadas, tornando a comparação da OptAI com o IRACE bem desafiadora.

Para reduzir a variabilidade entre execuções e garantir a robustez dos resultados, cada instância foi executada 5 vezes. A média dos resultados dessas execuções foi utilizada para a análise, minimizando o impacto de variações aleatórias no desempenho.

6.4 Resultados da Comparação

A Tabela 6.1 apresenta os resultados detalhados da comparação entre a OptAI e o IRACE para os algoritmos AS, MMAS, RAS, EAS e ACS. A tabela mostra a contagem de vitórias por modelo e média do gap para cada abordagem, sendo o gap:

$$\text{Gap} = \frac{\text{OptAI} - \text{IRACE}}{\text{IRACE}}. \quad (6.1)$$

Tabela 6.1: Contagem de Vitórias e Estatísticas do Gap por Algoritmo

Algoritmo	Vitórias IRACE	Vitórias OptAI	Empates	Gap Médio
AS	24	20	6	0.00318
MMAS	2	16	32	-0.00216
RAS	6	11	33	-0.00061
EAS	5	18	27	-0.00075
ACS	39	5	6	0.00903
Total	76	70	104	—

A Tabela 6.2 resume os resultados comparativos entre a OptAI e o IRACE em diferentes algoritmos aplicados a várias instâncias. Cada linha da tabela corresponde a uma instância específica, identificada na primeira coluna. As colunas subsequentes mostram os resultados para cada algoritmo (MMAS, AS, EAS, RAS, ACS), com os valores obtidos pelo IRACE e pela OptAI, respectivamente. Além disso, a tabela inclui uma coluna de “Gap” para cada algoritmo, que representa a diferença de desempenho entre a OptAI e o IRACE. Um valor negativo no gap

indica que a OptAI superou o IRACE naquela instância, enquanto um valor positivo sugere que o IRACE teve um desempenho melhor. Essa análise permite avaliar a eficácia e a competitividade da OptAI em relação ao IRACE em diferentes cenários de otimização de algoritmos.

Tabela 6.2: Resultados Comparativos por Instância

Instancia	MMAS			AS			EAS			RAS			ACS		
	IRACE	OptAI	Gap	IRACE	OptAI	Gap	IRACE	OptAI	Gap	IRACE	OptAI	Gap	IRACE	OptAI	Gap
ali535	210926	206170	-0.02255	205495	204890	-0.00294	210926	210926	0.00000	210926	210926	0.00000	203756	205167	0.00692
bck2217	7075	7075	0.00000	7046	7119	0.01036	7075	7075	0.00000	7119	7075	-0.00618	6895	6950	0.00798
beg3293	10409	10409	0.00000	10236	10409	0.01690	10409	10409	0.00000	10409	10409	0.00000	9981	9981	0.00000
bva2144	6614	6614	0.00000	6574	6545	-0.00441	6614	6614	0.00000	6614	6614	0.00000	6402	6475	0.01140
d1655	65200	65200	0.00000	64347	65200	0.01325	65200	65200	0.00000	65200	65200	0.00000	63231	63448	0.00344
d198	16790	16790	0.00000	15813	16246	0.02738	15940	16693	0.04723	15876	15873	-0.00019	15798	16823	0.06488
dbj2924	10591	10568	-0.00217	10552	10610	0.00549	10591	10591	0.00000	10610	10604	-0.00056	10333	10458	0.01210
djb2036	6591	6575	-0.00243	6457	6630	0.02679	6591	6591	0.00000	6630	6630	0.00000	6282	6255	-0.00430
dkc3938	13301	13267	-0.00256	13088	13301	0.01627	13301	13301	0.00000	13301	13301	0.00000	12789	12861	0.00563
dlb3694	11511	11481	-0.00261	11528	11476	-0.00451	11510	11502	-0.00069	11528	11528	0.00000	11197	11269	0.00643
fjs3649	9663	9663	0.00000	9663	9663	0.00000	9663	9663	0.00000	9663	9663	0.00000	9507	9566	0.00621
fqn5087	14663	14658	-0.00034	14663	14663	0.00000	14663	14658	-0.00034	14663	14658	-0.00034	13431	13526	0.00707
lim963	2920	2912	-0.00274	2872	2876	0.00139	2920	2897	-0.00788	2920	2920	0.00000	2827	2843	0.00566
lin105	14597	14597	0.00000	14597	14597	0.00000	14492	14379	-0.00780	14597	14597	0.00000	14379	14379	0.00000
lin318	43343	43343	0.00000	42311	42460	0.00352	42879	43343	0.01082	42836	43343	0.01184	42344	42290	-0.00128
lu980	11702	11702	0.00000	11605	11644	0.00336	11702	11645	-0.00487	11702	11702	0.00000	11432	11454	0.00192
mul979	90550	90051	-0.00551	89362	91341	0.02214	90668	91880	0.01337	91261	91000	-0.00286	87844	88228	0.00437
p654	37599	36149	-0.03856	35715	35252	-0.01296	37599	36149	-0.03856	37752	37752	0.00000	34646	34672	0.00075
pds2566	8086	8138	0.00643	7960	8138	0.02236	8086	8086	0.00000	8138	8138	0.00000	7834	7841	0.00089
portgen-2000-1	33502637	33502637	0.00000	33502637	33430001	-0.00216	33502637	33502637	0.00000	33502637	33502637	0.00000	32875121	33475687	0.01827
portgen-2000-108	33591322	33591322	0.00000	33670172	33572556	-0.00289	33591322	33591322	0.00000	34115960	34115960	0.00000	33009004	33440443	0.01307
portgen-2000-207	33909012	33909012	0.00000	33782889	33894481	0.00330	33909012	33899212	-0.00029	33909012	33909012	0.00000	33199765	33490275	0.00875
portgen-2000-228	34155805	34126478	-0.00086	34024259	34155805	0.00386	34155805	34155805	0.00000	34155805	34155805	0.00000	33349262	34014827	0.01996
portgen-2000-3	33544826	33544826	0.00000	33544826	33484615	-0.00179	33544826	33544826	0.00000	33544826	33544826	0.00000	32806301	33071191	0.00815
portgen-2000-4	33582471	33582471	0.00000	33624138	33591330	-0.00097	33582471	33582471	0.00000	33624138	33624138	0.00000	32887654	33222637	0.01023
portgen-2000-419	33820533	33820533	0.00000	33533861	33583939	0.00089	33820533	33613378	-0.00613	33820533	33820533	0.00000	32929679	33500395	0.01733
portgen-2000-539	33405953	33405953	0.00000	33367379	33314373	-0.00158	33405953	33405953	0.00000	33405953	33405953	0.00000	32647915	33045545	0.01218
portgen-2000-6	33938635	33805835	-0.00391	33938635	33575956	-0.01068	33938635	33805835	-0.00391	33938635	33938635	0.00000	33106287	33289918	0.00555
portgen-2000-605	33255151	33255151	0.00000	33230175	33182847	-0.00142	33255151	33158583	-0.00290	33255151	33255151	0.00000	32552726	32801428	0.00764
portgen-2000-626	33607143	33398954	-0.00619	33458024	33366751	-0.00272	33398954	33398954	0.00000	33612995	33612995	0.00000	32635560	32956038	0.00982
portgen-2000-937	33666900	33666900	0.00000	33690100	33579944	-0.00327	33666900	33666900	0.00000	33758868	33758868	0.00000	33183669	33606200	0.01273
pr107	44347	44347	0.00000	44303	44303	0.00000	44347	44303	-0.00099	44347	44347	0.00000	44303	44303	0.00000
pr152	78638	78638	0.00000	73818	74064	0.00333	74531	74149	-0.00513	74274	75648	0.01850	74358	74298	-0.00081
pr226	81382	81382	0.00000	80908	80451	-0.00564	81126	80688	-0.00540	80451	80455	0.00005	81386	81386	0.00000
pr2392	397004	397004	0.00000	393435	395056	0.00412	397004	397004	0.00000	397004	397004	0.00000	387759	383181	-0.01181
rat195	2347	2347	0.00000	2340	2339	-0.00042	2347	2347	0.00000	2339	2347	0.00342	2336	2347	0.00471
rat99	1276	1276	0.00000	1276	1276	0.00000	1211	1211	0.00000	1232	1212	-0.01623	1211	1276	0.05367
rbv1583	5621	5596	-0.00445	5587	5606	0.00340	5621	5621	0.00000	5621	5621	0.00000	5445	5483	0.00698
rbx711	3215	3215	0.00000	3191	3176	-0.00470	3215	3215	0.00000	3215	3193	-0.00684	3130	3191	0.01949
rl1323	282413	282413	0.00000	276448	277862	0.00511	282413	282413	0.00000	282413	282413	0.00000	271455	277009	0.02046
rl5915	608704	597662	-0.01814	608704	608704	0.00000	608704	597662	-0.01814	608704	608704	0.00000	583029	582877	-0.00026
st70	681	681	0.00000	681	676	-0.00734	680	676	-0.00588	675	680	0.00741	675	675	0.00000
tsp225	3992	3992	0.00000	3992	3952	-0.01002	3956	3943	-0.00330	3990	3992	0.00050	3930	3992	0.01573
u1432	162331	162331	0.00000	159281	159247	-0.00021	162331	162331	0.00000	162331	159931	-0.01485	155985	158230	0.01442
u2152	68150	68150	0.00000	66821	68150	0.01988	68150	68787	0.00935	68150	68150	0.00000	64861	65513	0.01005
u2319	246693	243020	-0.01489	241542	241873	0.00137	246693	243020	-0.01489	246693	246693	0.00000	237999	238844	0.00355
vm1084	244241	244241	0.00000	246230	244721	-0.00612	244241	244241	0.00000	246431	246431	0.00000	239722	240442	0.00300
xqg237	1047	1047	0.00000	1025	1031	0.00585	1027	1038	0.01071	1040	1036	-0.00385	1029	1029	0.00000
xql662	2669	2669	0.00000	2570	2588	0.00700	2669	2669	0.00000	2614	2597	-0.00650	2523	2581	0.02299
xva2993	8932	8915	-0.00190	8906	9072	0.01863	8932	8915	-0.00190	9072	8948	-0.01367	8686	8737	0.00587
Total	2	16	-0.00216	24	20	0.00318	5	18	-0.0075	6	11	-0.0062	39	5	0.00903

6.5 Análise dos Resultados

A análise dos resultados mostra que a OptAI pode superar o IRACE em certos cenários, apesar de ter acesso a menos dados de instâncias. No algoritmo AS, o IRACE venceu 24 instâncias, a OptAI, 21, e houve empate em 6 casos. Embora o IRACE tenha obtido mais vitórias, a OptAI apresentou desempenho competitivo, considerando que utilizou dados de apenas 15 a 17 instâncias.

As métricas de gap indicam que a OptAI teve um desempenho médio com 0.32% de desvio, contra 0.09% do IRACE. Isso sugere que, com mais dados e melhorias, a OptAI pode se tornar uma ferramenta mais robusta para a configuração automática de algoritmos no TSP.

Nos algoritmos MMAS e RAS, a OptAI obteve melhores resultados, vencendo ou empa-

tando na maioria das instâncias. Isso demonstra que a OptAI pode superar o IRACE em cenários onde o ajuste fino de parâmetros é crítico.

No algoritmo ACS, o IRACE venceu 39 instâncias, enquanto a OptAI venceu 6 e empatou em 6. Isso indica que, para alguns algoritmos, o IRACE mantém vantagem. Em contrapartida, a OptAI se mostrou competitiva no algoritmo EAS, vencendo 18 instâncias e empatando em 27.

6.6 Discussão

Os resultados obtidos neste estudo demonstram que a OptAI é uma ferramenta promissora para a configuração automática de algoritmos no TSP. A OptAI não só facilita a experimentação com diferentes algoritmos ACO, mas também permite a identificação do algoritmo mais adequado para cada cenário específico, proporcionando uma solução robusta e adaptável para a otimização de parâmetros.

A OptAI se beneficia de uma maior diversidade de dados de instâncias, o que é crucial para melhorar seu desempenho. Em trabalhos futuros, pode-se explorar a implementação de estratégias que permitam à OptAI utilizar mais instâncias de treinamento, mesmo que de forma reduzida, para garantir uma boa diversidade de dados.

Considerações Finais e Trabalhos Futuros

Este capítulo apresenta uma discussão extensa sobre os resultados obtidos ao longo do desenvolvimento da biblioteca `OptAI` e explora as possibilidades para trabalhos futuros. A `OptAI` foi desenvolvida com o objetivo de otimizar a configuração de parâmetros para algoritmos de otimização. O TSP foi escolhido como estudo de caso devido à sua relevância e complexidade, sendo um problema clássico e amplamente estudado na área de otimização combinatória.

7.1 Considerações Finais

Ao longo deste trabalho, foram abordados diversos aspectos relacionados à otimização de parâmetros para algoritmos ACOTSP. A `OptAI` foi desenvolvida para superar algumas das limitações do `IRACE`, especialmente em cenários onde a diversidade de dados de instâncias é crucial para a otimização. Os resultados obtidos demonstram que a `OptAI` tem o potencial de se tornar uma ferramenta poderosa para a configuração automática de algoritmos no TSP.

Além disso, a escolha de Python como linguagem de programação para o desenvolvimento da `OptAI` oferece várias vantagens significativas. Python é amplamente reconhecido por sua simplicidade e legibilidade, o que facilita o desenvolvimento e a manutenção do código. A vasta comunidade de desenvolvedores e a rica ecossistema de bibliotecas e frameworks, especialmente na área de Aprendizado de Máquina e Otimização, permitem a implementação de algoritmos complexos de maneira eficiente.

O uso de Aprendizado de Máquina na `OptAI` permite que a ferramenta aprenda com os dados de execução e melhore continuamente suas previsões de parâmetros. Isso é particularmente útil em problemas de otimização combinatória, como o TSP, onde a configuração adequada dos parâmetros pode levar a melhorias significativas no desempenho do algoritmo. A capacidade de ajuste fino dos parâmetros dos regressores e a utilização de técnicas avançadas de otimização, como o `Optuna`, contribuem para a robustez e a precisão das previsões da `OptAI`.

7.1.1 Desempenho da OptAI

A OptAI mostrou ser competitiva em relação ao IRACE, mesmo com a desvantagem de ter acesso a menos dados de instâncias. Em alguns algoritmos, como o MMAS e o EAS, a OptAI conseguiu vencer ou empatar na maioria das instâncias, demonstrando sua capacidade de superar o IRACE em certos cenários. Especificamente, a OptAI venceu em 16 instâncias do MMAS e empatou em 33, enquanto no EAS venceu em 18 instâncias e empatou em 27. Isso indica que a OptAI pode se beneficiar de um ajuste fino de parâmetros mais robusto.

No entanto, em algoritmos como o ACS, o IRACE ainda mantém uma vantagem significativa, vencendo em 39 instâncias contra 6 da OptAI, com 6 empates. Isso sugere que há espaço para melhorias na OptAI, especialmente na forma como os dados de instâncias são utilizados e na otimização dos parâmetros dos regressores.

7.1.2 Vantagens da OptAI

Uma das principais vantagens da OptAI é a sua capacidade de se beneficiar de uma maior diversidade de dados de instâncias. Isso é crucial para melhorar o desempenho dos regressores e, conseqüentemente, a precisão das previsões de parâmetros para novas instâncias. Além disso, a OptAI permite a experimentação com diferentes algoritmos ACO e a identificação do algoritmo mais adequado para cada cenário específico, proporcionando uma solução robusta e adaptável para a otimização de parâmetros.

7.1.3 Limitações e Desafios

Embora a OptAI tenha mostrado resultados promissores, ainda enfrenta algumas limitações e desafios. A principal limitação é a dependência dos dados de instâncias gerados pelo IRACE. Como o IRACE utiliza uma técnica de otimização que automaticamente reduz o espaço de instâncias de treinamento, a OptAI acaba tendo acesso a menos dados do que o ideal. Isso torna a comparação da OptAI com o IRACE bastante desafiador.

Além disso, a OptAI ainda não explora o uso de GPUs para o treinamento dos regressores e a otimização com o Optuna. A utilização de GPUs poderia melhorar significativamente o desempenho da OptAI, permitindo a realização de mais operações em menos tempo.

7.2 Trabalhos Futuros

Com base nos resultados obtidos e nas limitações identificadas, diversas possibilidades para trabalhos futuros podem ser exploradas para melhorar ainda mais a OptAI.

Independência do IRACE: Em trabalhos futuros, pode ser proposta a implementação de uma parte da OptAI que não utilize ou dependa do IRACE, implementando estratégias para

que a biblioteca desempenhe tão bem quanto ou supere o IRACE, utilizando somente implementações próprias da biblioteca. Isso envolveria o desenvolvimento de estratégias próprias para a geração de dados de instâncias, permitindo que a OptAI se beneficie de uma maior diversidade de dados. Essas estratégias poderiam incluir a geração de instâncias sintéticas ou a utilização de técnicas de aumento de dados para criar variações das instâncias existentes.

Utilização de GPUs: Outra área promissora é a utilização de GPUs para o treinamento dos regressores e a otimização com o Optuna. A implementação de suporte a GPUs poderia acelerar significativamente os processos de treinamento e otimização, permitindo que a OptAI realize mais operações em menos tempo. Isso seria especialmente útil em cenários onde o tempo de execução é uma limitação crítica.

Maior Diversidade de Instâncias: A utilização de mais instâncias para o treinamento dos regressores é outra área que merece atenção. Embora utilizar todas as instâncias disponíveis seja computacionalmente inviável, a implementação de técnicas de redução de instâncias que mantenham uma boa diversidade de dados poderia melhorar o desempenho da OptAI. Essas técnicas poderiam incluir algoritmos de seleção de instâncias baseados em diversidade ou métodos de amostragem estratificada.

Otimização de Parâmetros: A otimização dos parâmetros dos regressores é outra área que pode ser explorada em trabalhos futuros. A implementação de técnicas de otimização mais avançadas, como algoritmos evolutivos, poderia melhorar ainda mais a precisão das previsões de parâmetros da OptAI. Além disso, a exploração de diferentes estratégias de otimização de grid de parâmetros, utilizando bibliotecas como Hyperopt ou até mesmo o Optuna para otimizar os hiperparâmetros dos regressores, também poderia ser uma área promissora para pesquisa futura.

Estruturação de uma Meta-Base de Conhecimento: Uma melhoria futura inclui a criação de uma meta-base de conhecimento, que armazenaria informações sobre configurações ótimas, padrões de desempenho e relações entre instâncias e algoritmos. Essa base seria implementada quando a biblioteca atingir maturidade e acumular dados suficientes, servindo como um repositório para guiar decisões de configuração e reduzir a necessidade de treinamento contínuo.

Referências bibliográficas

- Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019. URL <https://arxiv.org/abs/1907.10902>.
- Debasish Basak, Srimanta Pal, and Dipak Patranabis. Support vector regression. *Neural Information Processing – Letters and Reviews*, 11, 11 2007.
- James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305, February 2012. ISSN 1532-4435.
- James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David Cox. Hyperopt: A python library for model selection and hyperparameter optimization. *Computational Science Discovery*, 8:014008, 07 2015. DOI [10.1088/1749-4699/8/1/014008](https://doi.org/10.1088/1749-4699/8/1/014008).
- Laura Calvet, J  sica de Armas, David Masip, and Angel A. Juan. Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics*, 15(1):261–280, 2017. DOI [doi:10.1515/math-2017-0029](https://doi.org/10.1515/math-2017-0029). URL <https://doi.org/10.1515/math-2017-0029>.
- Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016. URL <http://arxiv.org/abs/1603.02754>.
- Marco Dorigo. *Ant colony optimization: a new meta-heuristic*, *Optimization, Learning and Natural Algorithms (in Italian)*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.
- Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers Operations Research*, 13(5):533–549, 1986. ISSN 0305-0548. DOI [https://doi.org/10.1016/0305-0548\(86\)90048-1](https://doi.org/10.1016/0305-0548(86)90048-1). URL <https://www.sciencedirect.com/science/article/pii/0305054886900481>. Applications of Integer Programming.
- Zhiyuan He, Danchen Lin, Thomas Lau, and Mike Wu. Gradient boosting machine: A survey, 2019. URL <https://arxiv.org/abs/1908.06951>.

- Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, LION'05, page 507–523, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 9783642255656. DOI [10.1007/978-3-642-25566-3_40](https://doi.org/10.1007/978-3-642-25566-3_40). URL https://doi.org/10.1007/978-3-642-25566-3_40.
- Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. An efficient approach for assessing hyperparameter importance. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 754–762, Beijing, China, 22–24 Jun 2014. PMLR. URL <https://proceedings.mlr.press/v32/hutter14.html>.
- Maryam Karimi-Mamaghan, Mehrdad Mohammadi, Patrick Meyer, Amir Mohammad Karimi-Mamaghan, and El-Ghazali Talbi. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research*, 296(2):393–422, 2022. ISSN 0377-2217. DOI <https://doi.org/10.1016/j.ejor.2021.04.032>. URL <https://www.sciencedirect.com/science/article/pii/S0377221721003623>.
- S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. DOI [10.1126/science.220.4598.671](https://doi.org/10.1126/science.220.4598.671). URL <https://www.science.org/doi/abs/10.1126/science.220.4598.671>.
- Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: A big comparison for nas, 2019. URL <https://arxiv.org/abs/1912.06059>.
- Gilles Louppe. Understanding random forests: From theory to practice, 2015. URL <https://arxiv.org/abs/1407.7502>.
- Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. ISSN 2214-7160. DOI <https://doi.org/10.1016/j.orp.2016.09.002>. URL <https://www.sciencedirect.com/science/article/pii/S2214716015300270>.
- H. Malik, A. Iqbal, P. Joshi, S. Agrawal, and F.I. Bakhsh. *Metaheuristic and Evolutionary Computation: Algorithms and Applications*. Springer Singapore, Imprint: Springer, 2021. ISBN 9789811575723. URL <https://books.google.com.br/books?id=6sR9zwEACAAJ>.
- Jean-Yves Potvin Michel Gendreau. *Handbook of Metaheuristics*. International Series in Operations Research Management Science 272. Springer International Publishing, 3rd ed. edition, 2019. ISBN 978-3-319-91085-7;978-3-319-91086-4.

- Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning. The MIT Press, 1 edition, 2012. ISBN 0262018020,9780262018029.
- Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python, 2018. URL <https://arxiv.org/abs/1201.0490>.
- J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986. ISSN 0885-6125. DOI [10.1023/A:1022643204877](https://doi.org/10.1023/A:1022643204877). URL <https://doi.org/10.1023/A:1022643204877>.
- Gerhard Reinelt. TspLib—a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991. DOI [10.1287/ijoc.3.4.376](https://doi.org/10.1287/ijoc.3.4.376). URL <https://doi.org/10.1287/ijoc.3.4.376>.
- Gerhard Reinelt. *The traveling salesman: computational solutions for TSP applications*. Springer-Verlag, Berlin, Heidelberg, 1994. ISBN 3540583343.
- Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012. URL <https://arxiv.org/abs/1206.2944>.
- El-Ghazali Talbi. *Metaheuristics: from design to implementation*. Wiley Series on Parallel and Distributed Computing. Wiley, new edition, 2009. ISBN 9780470278581,0470278587.
- Jan N. van Rijn and Frank Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2367–2376, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. DOI [10.1145/3219819.3220058](https://doi.org/10.1145/3219819.3220058). URL <https://doi.org/10.1145/3219819.3220058>.