CENTRO UNIVERSITÁRIO MAURÍCIO DE NASSAU
CURSO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
MACHINE LEARNING

PROJETO I
TEMA: BUSCA TABU

LÍDER: - JOÃO MARIA BEZERRA JÚNIOR - Mat. Nº 01681641
PROGRAMADOR I: EXPEDITO FRAGOSO MENDONÇA - Mat. Nº 01695006
PROGRAMADOR II: - ROBERTO FAUSTO - Mat. Nº 01694640
REDATOR: - GUILHERME DALTRO DE OLIVEIRA - Mat. Nº 01713284
AUXILIAR: - Mat. Nº

Natal-RN
2025

LÍDER: - JOÃO MARIA BEZERRA JÚNIOR - Mat. Nº01681641
PROGRAMADOR I: EXPEDITO FRAGOSO MENDONÇA - Mat. Nº 01695006
PROGRAMADOR II: - ROBERTO FAUSTO - Mat. Nº 01694640
REDATOR: - GUILHERME DALTRO DE OLIVEIRA - Mat. Nº 01713284
AUXILIAR: - Mat. Nº

PROJETO I
TEMA: BUSCA TABU

Relatório presentado à disciplina de Machine Learning, correspondente à avaliação da 1ª Atividade Prática 2025.2 do curso de Análise e Desenvolvimento de Sistemas da Universidade Federal do Rio Grande do Norte, sob orientação do **Profº Eng. Esp. José Lindenberg de Andrade.**

Professor: Eng. Esp. José Lindenberg de Andrade.

Natal-RN
2025

RESUMO

Este relatório apresenta uma viso geral do algoritmo de Busca Tabu, uma meta-heurística de busca local projetada para resolver problemas de otimização combinatória. Discutimos seus componentes fundamentais, como a lista tabu e os critérios de aspiração, que permitem ao algoritmo escapar de ótimos locais e explorar o espaço de soluções de forma mais eficaz. O documento também inclui um pseudocódigo do algoritmo, exemplos práticos de sua aplicação, uma análise de suas vantagens e desvantagens e um apêndice com uma implementação completa em Python para o Problema do Caixeiro-Viajante.

Palavras-chave: Busca Tabu, Otimização Combinatória, Meta-heurística, Caixeiro-Viajante, Python.

Sumário

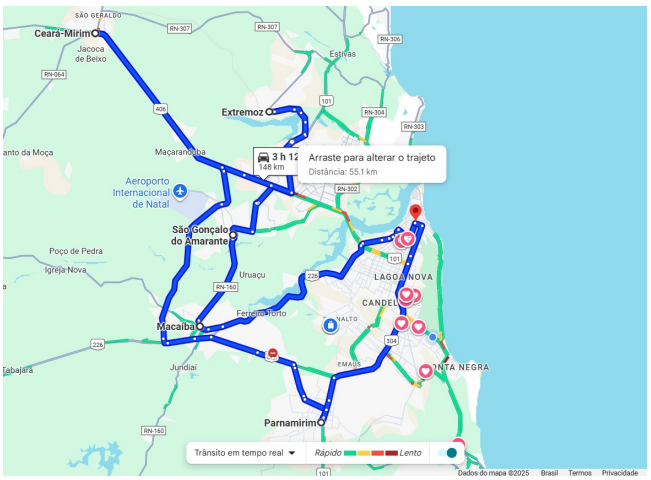

Figura 1: Caixeiro Viajante

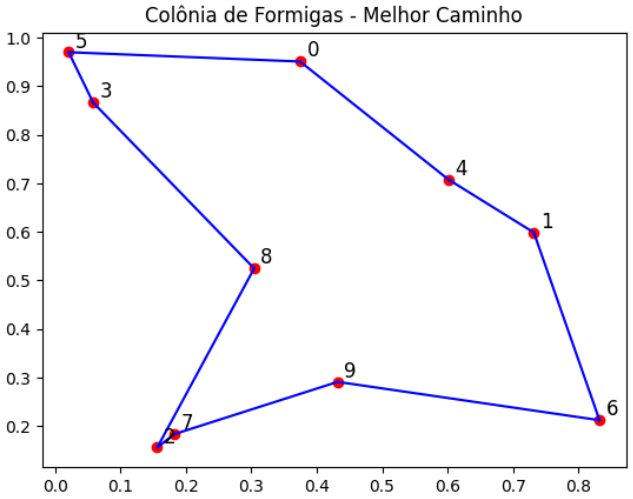

Figura 2: Colônia de Formigas

Lista de Figuras

1 Introdução

A **Busca Tabu** é uma metaheurística desenvolvida por Fred Glover em 1986, projetada para resolver problemas de otimização combinatória. Ela se destaca por utilizar memória adaptativa para evitar ciclos e escapar de mínimos locais, permitindo uma exploração mais eficaz do espaço de soluções.

2 Fundamentos da Busca Tabu

A busca tabu parte de uma soluço inicial e realiza movimentos locais para gerar novas soluções. Para evitar revisitar soluções já exploradas, utiliza uma **lista tabu**, que armazena movimentos ou atributos de soluções proibidas temporariamente.

Componentes principais

- **Movimentos vizinhos:** pequenas modificações na soluço atual.
- **Lista tabu:** estrutura de memória que impede reversões recentes.
- **Critério de aspiração:** permite ignorar a lista tabu se a nova soluço for suficientemente boa.
- **Função objetivo:** avalia a qualidade de cada solução.

3 Problemas Clássicos de Otimização Combinatória

3.1 Problema do Caixeiro Viajante (TSP)

O TSP consiste em encontrar o menor caminho possível que permita a um vendedor visitar uma lista de cidades exatamente uma vez e retornar à cidade de origem.

- **Entrada:** Conjunto de cidades e distâncias entre elas.
- **Objetivo:** Minimizar a distância total percorrida.
- **Aplicação da Busca Tabu:** A busca tabu evita ciclos e melhora rotas por meio de trocas de arestas e permutações de cidades.

3.2 Problema da Mochila (Knapsack Problem)

Neste problema, o objetivo é selecionar um subconjunto de itens com valores e pesos, de modo a maximizar o valor total sem ultrapassar a capacidade da mochila.

- **Entrada:** Lista de itens com peso e valor, e capacidade da mochila.
- **Objetivo:** Maximizar o valor total dos itens escolhidos.
- **Aplicação da Busca Tabu:** A busca tabu pode explorar combinações de itens, evitando soluções repetidas e melhorando a seleção com base em valor/peso.

3.3 Problema de Roteamento de Veículos (VRP)

O VRP é uma generalização do TSP, onde múltiplos veículos devem atender a uma série de clientes com demandas específicas, partindo e retornando a um depósito central.

- **Entrada:** Localização dos clientes, demandas, número de veículos e capacidade.
- **Objetivo:** Minimizar o custo total de transporte.
- **Aplicação da Busca Tabu:** A busca tabu pode otimizar rotas múltiplas, respeitando restrições de capacidade e tempo, com trocas entre rotas e clientes.

3.4 Otimização Combinatória

A otimização combinatória trata de encontrar a melhor soluço entre um conjunto finito (mas possivelmente muito grande) de soluções possíveis.

- **Características:** Espaço de busca discreto, múltiplas restrições, soluções inteiras.
- **Aplicação da Busca Tabu:** A busca tabu é eficaz em problemas NP-difíceis, como programação inteira, alocação de recursos e escalonamento.

4 Vantagens e Desvantagens da Busca Tabu

Vantagens

- Capacidade de escapar de mínimos locais.
- Flexibilidade para diferentes tipos de problemas.
- Facilidade de hibridização com outras heurísticas.

Desvantagens

- Sensibilidade à parametrização (tamanho da lista tabu, critérios de aspiração).
- Custo computacional elevado em problemas muito grandes.

5 Conclusão

A busca tabu é uma ferramenta poderosa e versátil para resolver problemas complexos de otimização combinatória. Sua capacidade de incorporar memória e adaptar-se ao histórico da busca permite superar limitações de métodos locais tradicionais, sendo especialmente útil em problemas como TSP, mochila, VRP e outros desafios combinatórios.

6 CÓDIGO

6.1 Código de Exemplo: Busca Tabu para o PCV em Python

O código a seguir apresenta uma implementação prática do algoritmo de Busca Tabu para resolver uma instância do Problema do Caixeiro-Viajante com cidades da região metropolitana de Natal/RN.

Listing 1: Implementacao em Python da Busca Tabu para o PCV.

```
import math
import random

# --- 1. Definicao do Problema: Cidades e Distancias ---

# Dicionario de cidades com suas coordenadas (x, y)
cidades = {
    "Natal": (10, 20),
    "Parnamirim": (12, 15),
    "Macaiba": (8, 22),
    "Sao Goncalo": (9, 25),
    "Extremoz": (15, 30),
    "Ceara-Mirim": (18, 35)
}

# Lista de nomes das cidades para facil acesso
lista_cidades = list(cidades.keys())

# Funcao para calcular a distancia euclidiana entre duas cidades
def calcular_distancia(cidade1, cidade2):
    """Calcula a distancia euclidiana entre dois pontos (cidades)."""
    return math.sqrt((cidades[cidade1][0] - cidades[cidade2][0])**2 +
                     (cidades[cidade1][1] - cidades[cidade2][1])**2)

# Funcao para calcular o custo total de uma rota
def calcular_custo_rota(rota):
    """Calcula o custo total (distancia) de uma rota."""
    custo_total = 0
    for i in range(len(rota) - 1):
        custo_total += calcular_distancia(rota[i], rota[i+1])
    # Adiciona a distancia de volta    cidade inicial
    custo_total += calcular_distancia(rota[-1], rota[0])
    return custo_total
```

```
# --- 2. Implementacao da Busca Tabu ---

def busca_tabu(cidades_lista, max_iteracoes, tamanho_lista_tabu):
    """
    Implementa o algoritmo de Busca Tabu para
    o Problema do Caixeiro Viajante.
    """
    # a. Gera uma solucao inicial aleatoria
    rota_atual = random.sample(cidades_lista, len(cidades_lista))
    melhor_rota = rota_atual
    melhor_custo = calcular_custo_rota(melhor_rota)

    lista_tabu = []

    print(f"Rota Inicial: {' -> '.join(rota_atual)}")
    print(f"Custo Inicial: {melhor_custo:.2f}\n")

    for i in range(max_iteracoes):
        melhor_vizinho = None
        melhor_custo_vizinho = float('inf')
        movimento_do_melhor_vizinho = None

        # b. Explora a vizinhanca (usando movimentos
        de troca de 2 cidades)
        for j in range(len(rota_atual)):
            for k in range(j + 1, len(rota_atual)):
                vizinho = rota_atual[:]
                # Realiza a troca (movimento)
                vizinho[j], vizinho[k] = vizinho[k], vizinho[j]

                # Movimento candidato
                movimento_candidato =
                tuple(sorted((rota_atual[j], rota_atual[k])))

                # c. Verifica se o movimento nao esta na lista tabu
                if movimento_candidato not in lista_tabu:
                    custo_vizinho = calcular_custo_rota(vizinho)

                 if custo_vizinho < melhor_custo_vizinho:
```

```
                    melhor_vizinho = vizinho
                    melhor_custo_vizinho = custo_vizinho
                    movimento_do_melhor_vizinho =
                    movimento_candidato

        # d. Move para o melhor vizinho encontrado
        if melhor_vizinho:
            rota_atual = melhor_vizinho

            # Atualiza a lista tabu: adiciona o novo
            movimento e remove o mais antigo
            lista_tabu.append(movimento_do_melhor_vizinho)
            if len(lista_tabu) > tamanho_lista_tabu:
                lista_tabu.pop(0)

            # e. Atualiza a melhor solucao encontrada
            ate agora
            if melhor_custo_vizinho < melhor_custo:
                melhor_rota = melhor_vizinho
                melhor_custo = melhor_custo_vizinho

                print(f"Iteracao {i+1}: Nova melhor rota!
                Custo: {melhor_custo:.2f}")

    return melhor_rota, melhor_custo

# --- 3. Execucao e Resultados ---

if _name_ == "_main_":
    # Parametros do algoritmo
    MAX_ITERACOES = 200
    TAMANHO_LISTA_TABU = 10

    # Executa a Busca Tabu
    rota_final, custo_final = busca_tabu(lista_cidades,
    MAX_ITERACOES, TAMANHO_LISTA_TABU)

    # Imprime os resultados finais
    print("\n--- Resultado Final ---")
    print(f"Melhor Rota Encontrada: {' ->
```

```
'.join(rota_final)} -> {rota_final[0]}")
print(f"Menor Custo (Distancia): {custo_final:.2f}")
```

7 CONCLUSO DO CÓDIGO

A execução deste projeto demonstrou a eficácia do algoritmo meta-heurístico de Busca Tabu na resolução do Problema do Caixeiro Viajante. Através da implementação em Python, o algoritmo explorou sistematicamente o espaço de soluções e determinou com sucesso uma rota otimizada, resultando no caminho

'Parnamirim → Ceará-Mirim → Extremoz → So Gonçalo → Macaíba → Natal → Parnamirim' com um custo final de 45.90. Este resultado valida a Busca Tabu como uma ferramenta robusta para problemas de otimização combinatória, cumprindo o objetivo central desta avaliação prática de aplicar conceitos teóricos em um cenário real e complexo.

Rota Inicial: Natal -> Parnamirim -> Ceará-Mirim -> Extremoz -> Macaíba -> São Gonçalo Custo Inicial: 50.99

Iteração 1: Nova melhor rota encontrada! Custo: 48.70

Iteração 163: Nova melhor rota encontrada! Custo: 45.90

— Resultado Final — Melhor Rota Encontrada: Natal -> Macaíba -> São Gonçalo -> Extremoz -> Ceará-Mirim -> Parnamirim -> Natal Menor Custo (Distância): 45.90