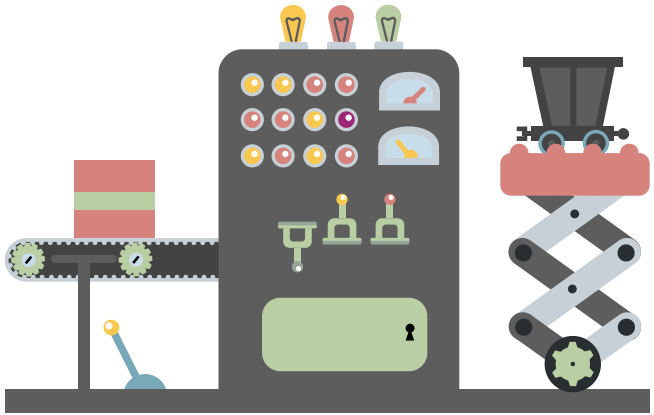


Recozimento Simulado (Simulated Annealing)

Pós-graduação em Ciência e Tecnologia da Computação

PCO 209 - Engenharia de Software Experimental

Breno de Oliveira Renó

Introdução

Recozimento Simulado

Recozimento Simulado (*Simulated Annealing*) é uma meta-heurística utilizada para otimização.

Essa técnica se fundamenta em uma analogia feita com a termodinâmica.

Introdução

Recozimento Simulado

Recozimento Simulado (*Simulated Annealing*) é uma meta-heurística utilizada para otimização.

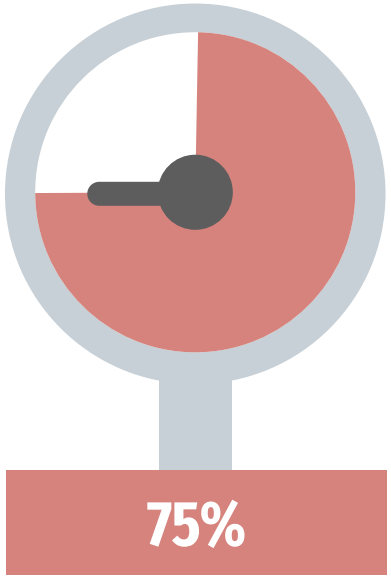
Essa técnica se fundamenta em uma analogia feita com a termodinâmica.

Na Termodinâmica

É um processo térmico utilizado, por exemplo, no aprimoramento de metais.

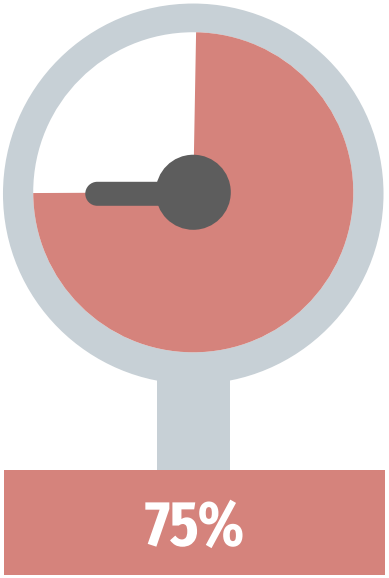
O material é aquecido até altas temperaturas, desta forma os átomos se movimentam livremente, depois o metal é resfriado gradativamente, assim os átomos se encaixam em uma posição melhor.

Introdução

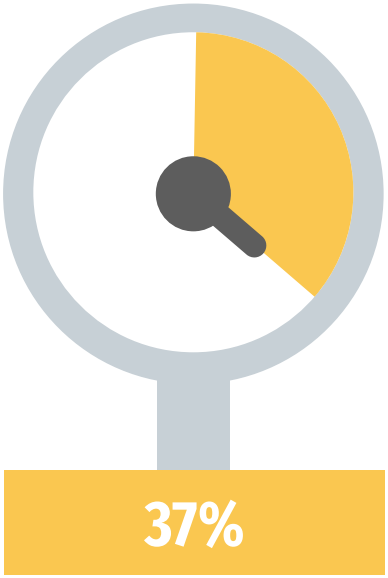


O mesmo princípio é usado no algoritmo, começando em temperaturas altas e resfriando lentamente. Inicialmente um ponto X é escolhido e a sua avaliação é feita.

Introdução

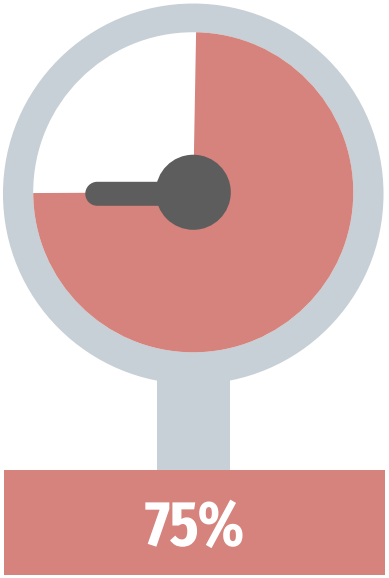


O mesmo princípio é usado no algoritmo, começando em temperaturas altas e resfriando lentamente. Inicialmente um ponto X é escolhido e a sua avaliação é feita.

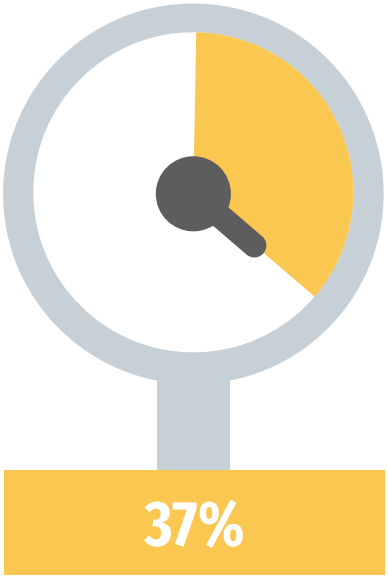


O algoritmo faz um movimento até um dos seus vizinhos X', e avalia esse novo ponto. Caso haja uma melhoria no resultado o algoritmo se move até ele e refaz o processo anterior.

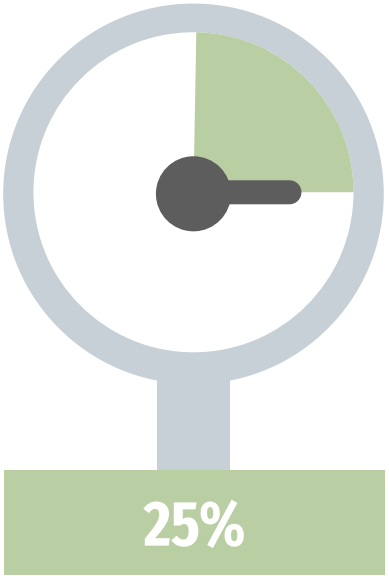
Introdução



O mesmo princípio é usado no algoritmo, começando em temperaturas altas e resfriando lentamente. Inicialmente um ponto X é escolhido e a sua avaliação é feita.



O algoritmo faz um movimento até um dos seus vizinhos X', e avalia esse novo ponto. Caso haja uma melhoria no resultado o algoritmo se move até ele e refaz o processo anterior.



Caso o ponto X' apresente um resultado inferior, o algoritmo pode se mover até ele se a probabilidade de ir para um ponto negativo for superior a um valor aleatório.

Probabilidade: $(p) = Exp^{(X' - X / T)}$

Probabilidade
Dada pela diferença entre o valor do candidato (X') menos o valor da solução atual (X), dividido pela temperatura (T). Nas primeiras iterações, o valor da temperatura T está elevado, e a probabilidade do algoritmo aceitar valores negativos é maior. Posteriormente, conforme são decorridas as iterações, essa temperatura diminui, e a probabilidade também.

Probabilidade: $(p) = Exp^{(X' - X / T)}$

Probabilidade

Dada pela diferença entre o valor do candidato (X') menos o valor da solução atual (X), dividido pela temperatura (T).

Nas primeiras iterações, o valor da temperatura T está elevado, e a probabilidade do algoritmo aceitar valores negativos é maior.

Posteriormente, conforme são decorridas as iterações, essa temperatura diminui, e a probabilidade também.

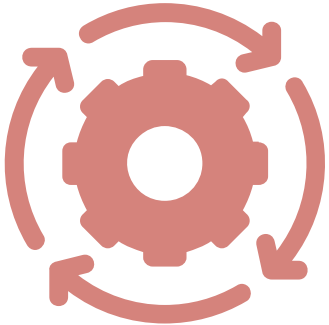
Comportamento

Esse formato exemplifica uma característica inicial bastante exploratória do algoritmo.

Ele consegue se movimentar livremente pelo espaço de busca de soluções, e conforme a temperatura vai se aproximando de 0, essa movimentação tende a diminuir, uma vez que o algoritmo passará a admitir apenas soluções positivas.

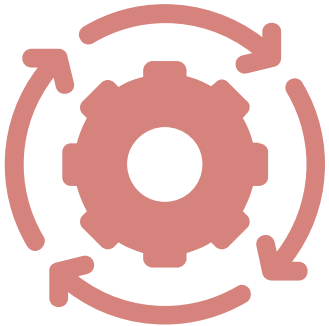
Código

```
# Breno de Oliveira Renó - brenooliveirareno@unifei.edu.br
# Aplicação do Recozimento Simulado no Problema do Caixeiro Viajante
# ------------------------------------------------------------------
# FONTE: Construído com base no código de Chncyhn
# Disponível em: https://github.com/chncyhn/simulated-annealing-tsp
# ------------------------------------------------------------------
```



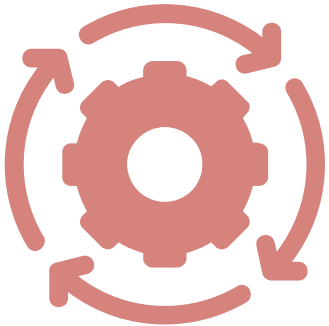
Código

```python
from SimulatedAnneling import RecozimentoSimulado

# Abre e formata os dados vindos do arquivo indicado
def abre_arquivo_coordenadas(arquivo):
    coordenadas = []
    with open(arquivo, "r") as f:
        for linha in f.readlines():
            linha = [float(x.replace("\n", "")) for x in linha.split(" ")]
            coordenadas.append(linha)
    return coordenadas


# Inicio do Código
coordenadas = abre_arquivo_coordenadas("berlin52.txt")

# Inicializa a Classe
rs_classe = RecozimentoSimulado(coordenadas)

# Executa as Funções
rs_classe.recozimento_simulado()
# Para ver a Rota Feita
rs_classe.apresenta_melhor_rota()
# Para ver Todas as Soluções Avaliadas
rs_classe.apresenta_todas_solucoes()
# Para ver as Melhorias
rs_classe.apresenta_melhorias()
```



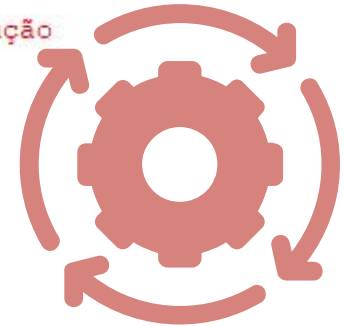
Código

```
# Definições Iniciais das Coordenadas e Tamanho do Espaço
self.coords = coordenadas
self.N = len(coordenadas)

# Define a Temperatura de Maneira Dinamica
self.T = math.sqrt(self.N)
# Define o Resfriamento e o critério de parada
self.taxa_resfriamento = 0.995
self.temperatura_de_parada = 1e-10

# Definição do Máximo de Iterações - Usado como Possível Critério de Parada
self.iteracao = 1
self.max_iteracoes = 1000000

# Definições Adicionais
self.vertices = [i for i in range(self.N)]
self.melhor_solucao = None
self.melhor_distancia = float("Inf")
self.lista_distancias = []
self.lista_todas_distancias = []
```



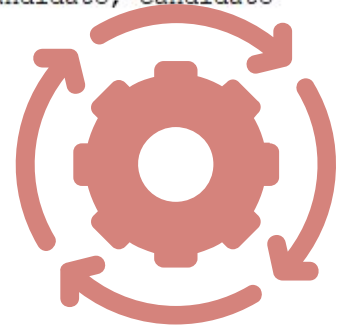
Código

```
def solucao_inicial(self):
    # Iniciando de um Ponto Aleatório
    vertice_att = random.choice(self.vertices)
    solucao_att = [vertice_att]
    # Inicia a Partir do Ponto Aleatório Escolhido
    vertices_livres = set(self.vertices)
    vertices_livres.remove(vertice_att)
    while vertices_livres:
        # Encontra o Vizinho Mais Próximo
        prox_v = min(vertices_livres, key=lambda x: self.dist(vertice_att, x))
        # Remove da Lista de Vertices Não Usados e Insere na Solução
        vertices_livres.remove(prox_v)
        solucao_att.append(prox_v)
        vertice_att = prox_v

    distancia_att = self.distancia_solucao(solucao_att)
    # Se for a Melhor Solução, Então Atualiza a Melhor Solução
    if distancia_att < self.melhor_distancia:
        self.melhor_distancia = distancia_att
        self.melhor_solucao = solucao_att

    self.lista_distancias.append(distancia_att)
    # Adiciona na Lista com Todas as Soluções Avaliadas
    self.lista_todas_distancias.append(distancia_att)
    return solucao_att, distancia_att
```



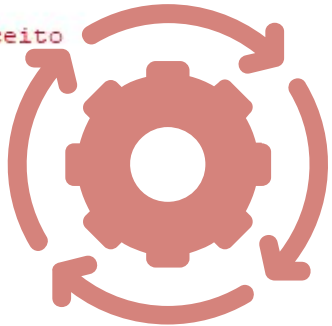
Código

```python
# Probabilidade: (p) = Exp(X'- X / T)
def probabilidade_aceitacao(self, distancia_candidato):
    return math.exp(-abs(distancia_candidato - self.distancia_atual) / self.T)

# Função que Verifica se o Candidato Será Aceito
def aceita_candidato(self, candidato):
    distancia_candidato = self.distancia_solucao(candidato)
    # Adiciona na Lista com Todas as Soluções Avaliadas
    self.lista_todas_distancias.append(distancia_candidato)
    # Aceita o Candidato se Ele for Melhor que o Melhor Atual
    if distancia_candidato < self.distancia_atual:
        self.distancia_atual, self.solucao_atual = distancia_candidato, candidato
        if distancia_candidato < self.melhor_distancia:
            self.melhor_distancia, self.melhor_solucao = distancia_candidato, candidato
    # Chama a Função Probabilistica se Ele for Pior que o Melhor Atual
    else:
        if random.random() < self.probabilidade_aceitacao(distancia_candidato):
            self.distancia_atual, self.solucao_atual = distancia_candidato, candidato
```

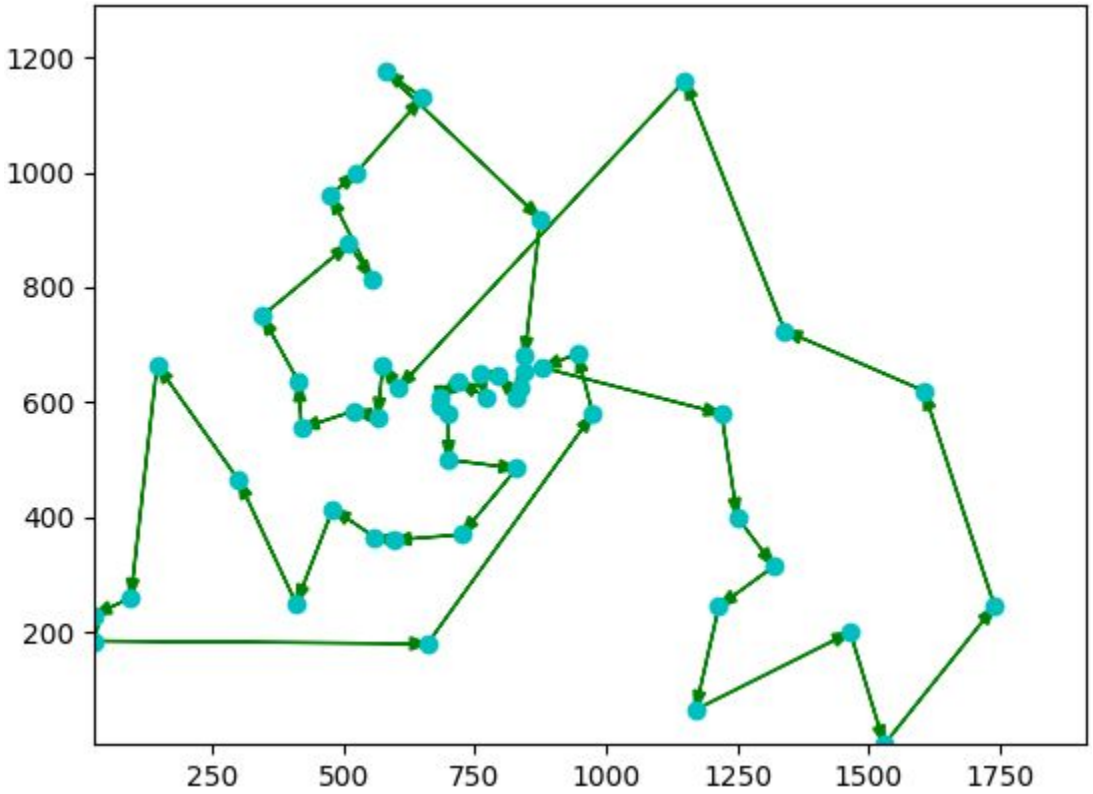


Código

```
# Função de Execução do Recozimento Simulado
def recozimento_simulado(self):
    # Inicializa a Função com a Solução Inicial
    self.solucao_atual, self.distancia_atual = self.solucao_inicial()
    print("Distancia Inicial: ", self.distancia_atual)
    # Para ver o Caminho da Solução Inicial
    self.apresenta_melhor_rota()
    print("------------------------")
    print("Aplicando Recozimento...")
    print("------------------------")

    # Enquanto os Critérios de Parada Não Forem Alcançados
    while self.T >= self.temperatura_de_parada and self.iteracao < self.max_iteracoes:
        # Define um Novo Candidato
        candidato = list(self.solucao_atual)
        l = random.randint(2, self.N - 1)
        i = random.randint(0, self.N - l)
        candidato[i: (i + l)] = reversed(candidato[i: (i + l)])
        # Chama a Função Para Verificar se o Candidato Deve ser Aceito
        self.aceita_candidato(candidato)
        # Aplica a Taxa de Resfriamento
        #print("Temperatura: ", self.T)
        self.T *= self.taxa_resfriamento
        self.iteracao += 1
        # Adiciona na Lista de Resultados
        self.lista_distancias.append(self.distancia_atual)

    print("Melhor Distancia Obtida: ", self.melhor_distancia)
    self.apresenta_melhoria_porcentagem()
```

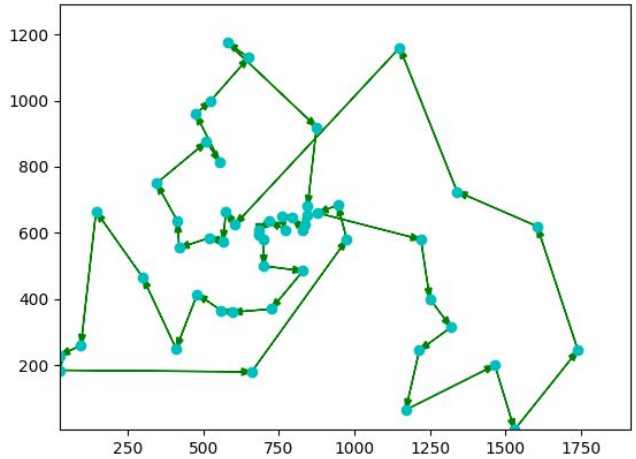


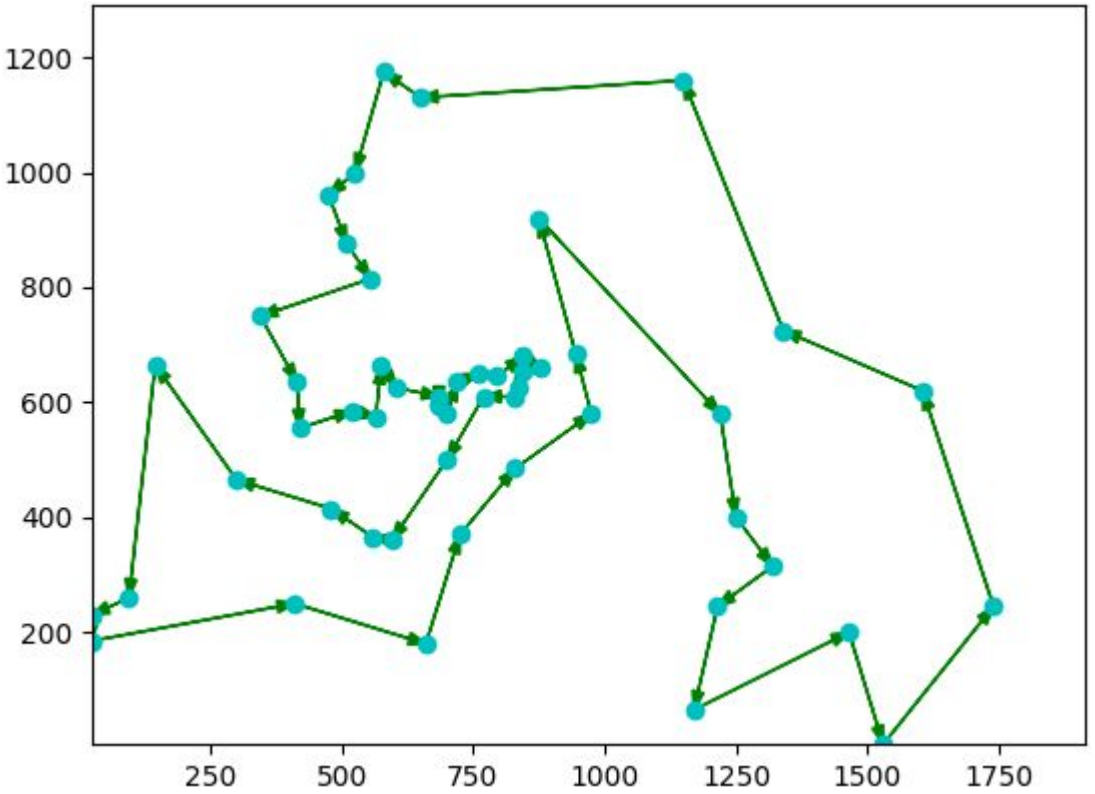
Resultados



berlin52.txt

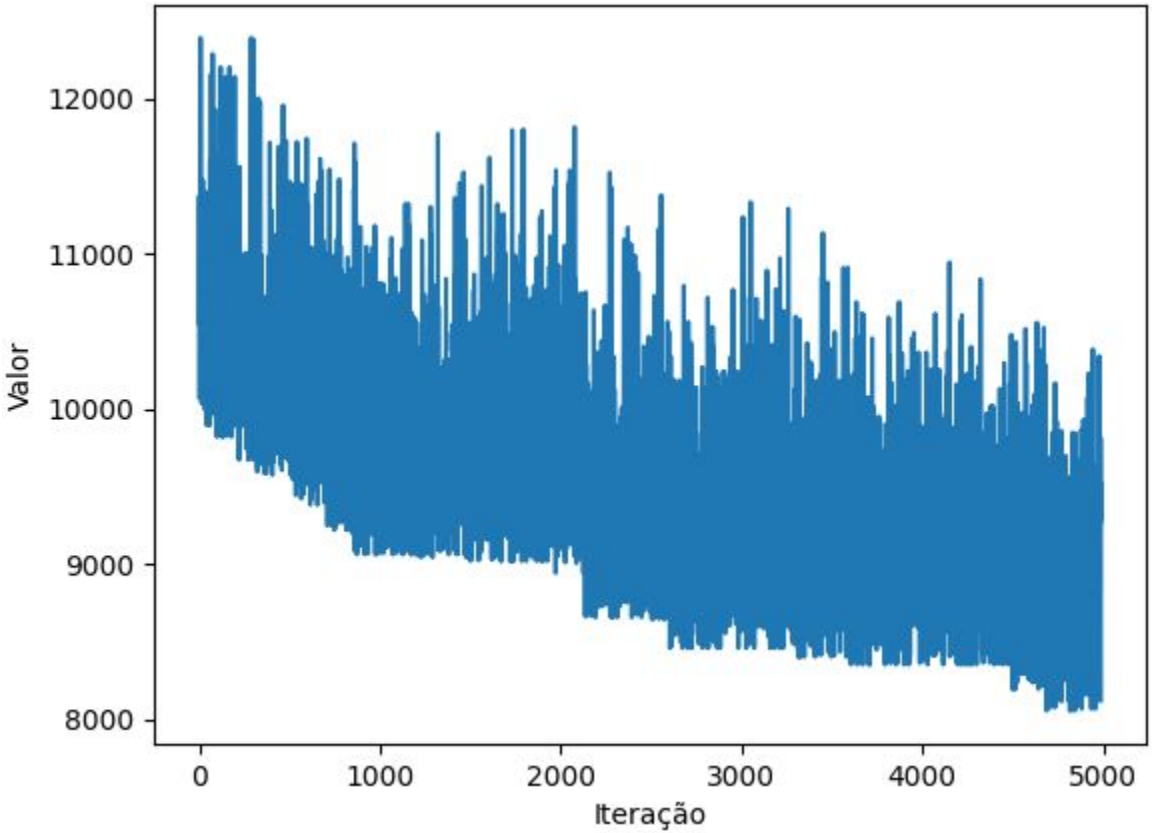
Resultados





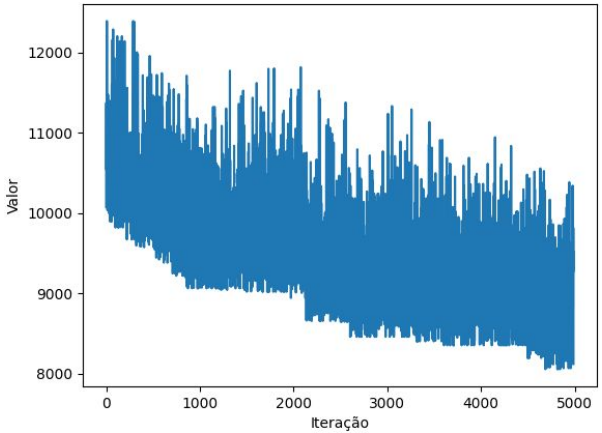
berlin52.txt

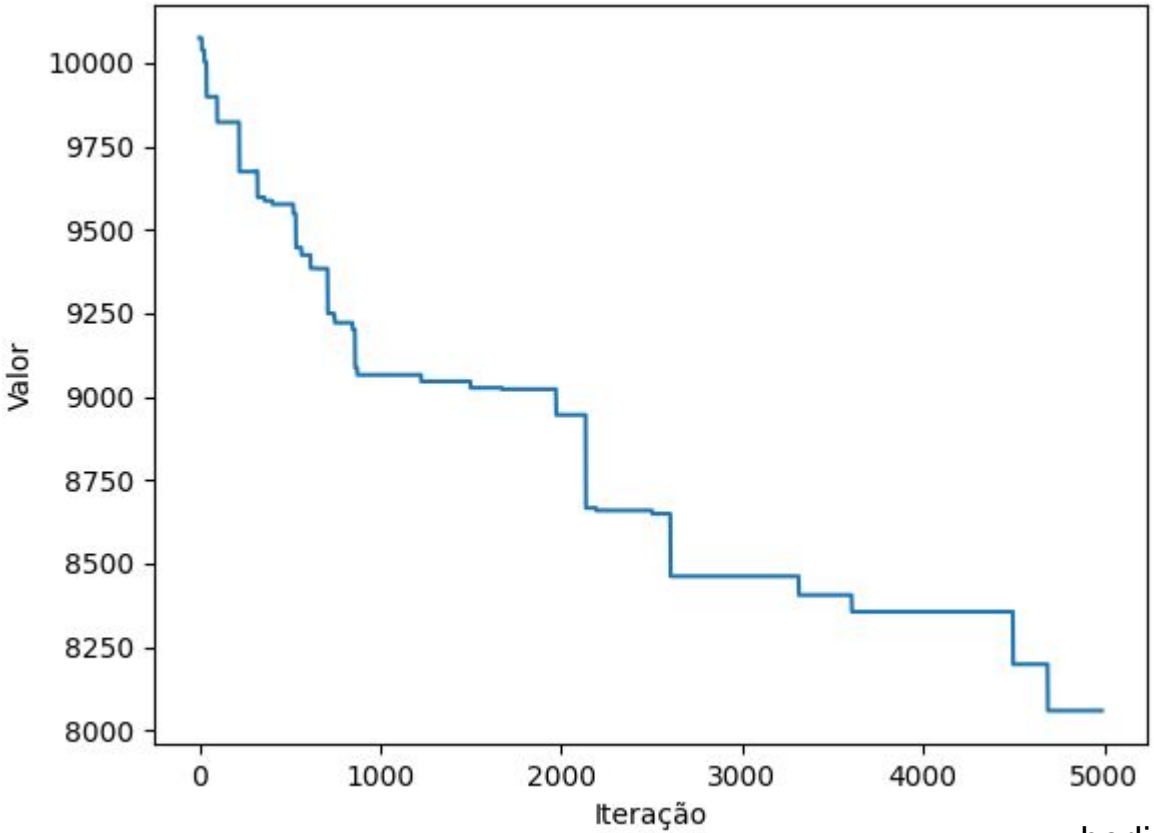
Resultados



berlin52.txt

Resultados

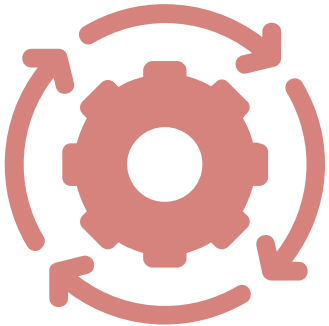




berlin52.txt

Resultados

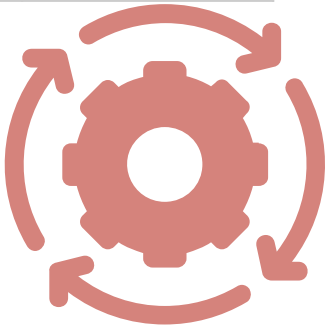
Bases Utilizadas
1. berlin52
2. kroA100
3. kroA150
4. kroA200
5. kroB100
6. kroB150
7. kroB200
8. kroC100
9. linhp318
10. ts225



Resultados

berlin52		
Solução Inicial	Solução Final	Melhoria %
10075.51	8056.69	20,04
8182.19	7839.34	4,19
9575.16	8685.76	9,29
9194.13	7943.64	13,6
9575.16	8067.65	15,74
Média		12,57

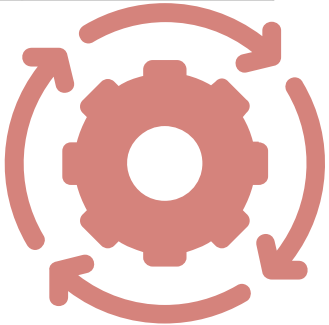
kroA100		
Solução Inicial	Solução Final	Melhoria %
28506.82	24024.79	15,72
28322.51	24734.02	12,67
27385.45	23568.25	13,94
28246.98	26240.16	7,1
26209.39	24035.90	8,29
Média		11,54



Resultados

kroA150		
Solução Inicial	Solução Final	Melhoria %
34496.59	31658.31	8,23
32891.63	30233.07	8,08
33059.61	31753.43	3,95
34426.69	30960.07	10,07
35935.29	32671.17	9,08
Média		7,88

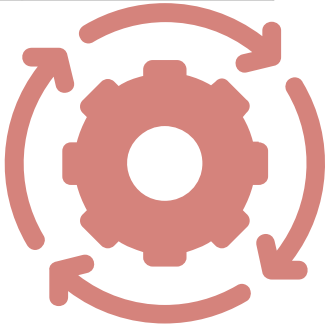
kroA200		
Solução Inicial	Solução Final	Melhoria %
36656.81	35152.93	4,1
36119.34	35069.67	2,91
41992.65	38871.68	7,43
37437.40	35677.92	4,7
37508.71	35875.45	4,35
Média		4,69



Resultados

kroB100		
Solução Inicial	Solução Final	Melhoria %
27490.92	25032.20	8,94
26338.32	25044.08	4,91
28557.48	24881.02	12,87
28557.48	23480.15	17,78
28682.20	24976.35	12,92
Média		11,48

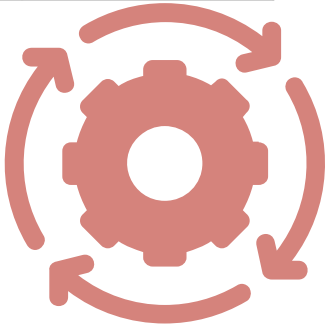
kroB150		
Solução Inicial	Solução Final	Melhoria %
35884.06	31558.59	12,05
33297.81	30518.08	8,35
32610.84	29937.72	8,2
36334.02	31068.07	14,49
33427.22	30315.07	9,31
Média		10,48



Resultados

kroB200		
Solução Inicial	Solução Final	Melhoria %
36791.90	35283.08	4,1
37485.37	35998.16	3,97
36613.86	35334.99	3,49
35848.23	34095.72	4,89
36642.94	34947.70	4,63
Média		4,21

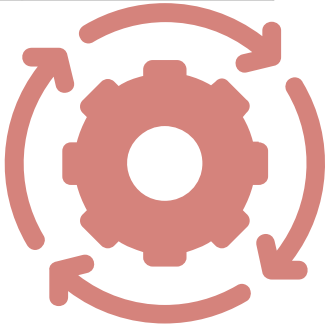
kroC100		
Solução Inicial	Solução Final	Melhoria %
29346.61	24616.96	16,12
26556.17	23696.95	10,77
28045.48	24220.60	13,64
28494.90	22521.17	20,96
26700.57	24268.44	9,11
Média		14,12



Resultados

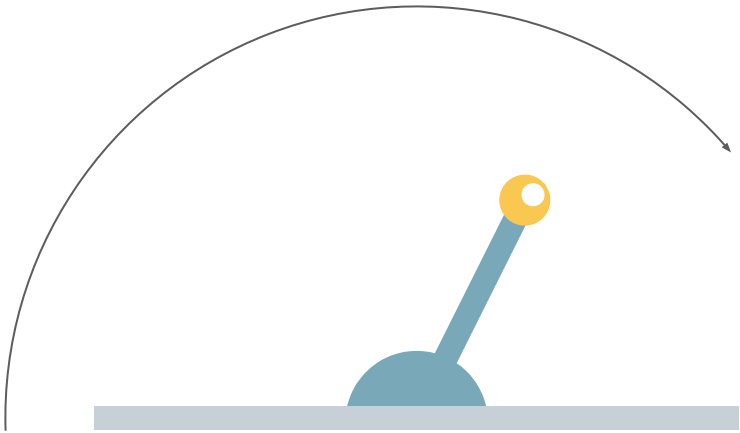
linhp318		
Solução Inicial	Solução Final	Melhoria %
52404.82	51397.14	1,92
51204.84	50205.39	1,95
51445.43	51013.95	0,84
52437.05	51727.97	1,35
53573.50	51965.19	3,01
Média		1,81

ts225		
Solução Inicial	Solução Final	Melhoria %
155727.92	145457.01	6,6
150597.76	141078.90	6,32
145103.31	140976.01	2,84
151186.19	145612.62	3,69
148740.26	144041.68	3,16
Média		4,52



Discussão

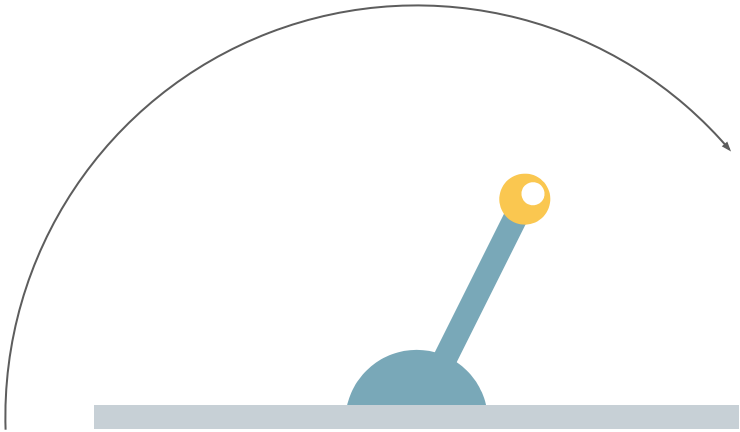
Comparações		
Base	ILS	SA
berlin52	7.45 %	12,57 %
kroA100	6,16 %	11,54 %
kroA150	0,54 %	7,88 %
kroA200	4,73 %	4,69 %
kroB100	2,38 %	11,48 %
kroB150	9,09 %	10,48 %
kroB200	0 %	4,21 %
kroC100	6,01 %	14,12 %
linhp318	3,57 %	1,81 %
ts225	1,12 %	4,52 %



Discussão

Comparações		
Base	ILS	SA
berlin52	7.45 %	12,57 %
kroA100	6,16 %	11,54 %
kroA150	0,54 %	7,88 %
kroA200	4,73 %	4,69 %
kroB100	2,38 %	11,48 %
kroB150	9,09 %	10,48 %
kroB200	0 %	4,21 %
kroC100	6,01 %	14,12 %
linhp318	3,57 %	1,81 %
ts225	1,12 %	4,52 %

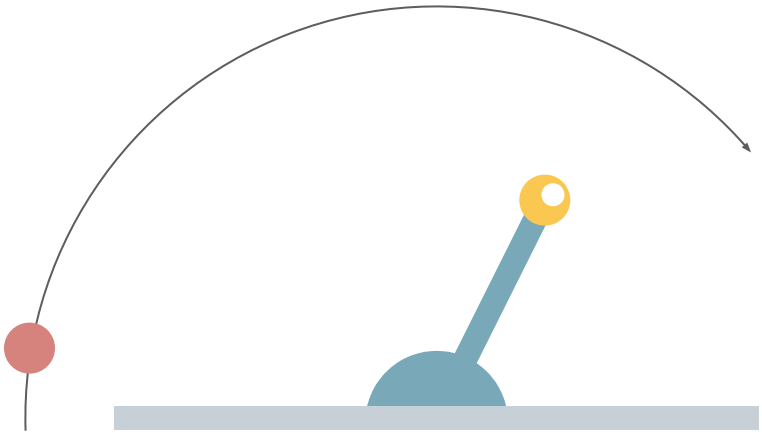
Foi Melhor	
ILS	2 Vezes
SA	8 Vezes



Discussão

O problema do caixeiro viajante é um assunto estudado a muitos anos, e uma solução perfeita ainda não existe.

1.0



Discussão

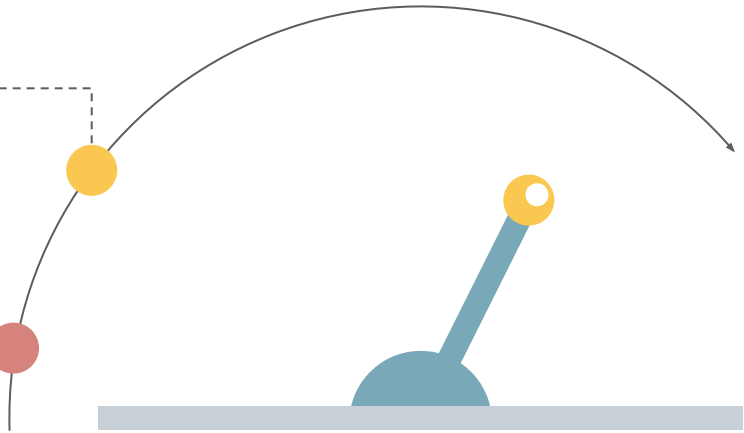
A Pesquisa Local Iterada permite "aprender" com o mínimo local encontrado com uma busca local.

É possível alcançar resultados interessantes através de um algoritmo relativamente simples.

2.0

O problema do caixeiro viajante é um assunto estudado a muitos anos, e uma solução perfeita ainda não existe.

1.0



Discussão

A aplicação do Recozimento Simulado sobre uma solução inicial também se mostra como uma opção simples, porém poderosa.

O algoritmo se inicia com uma solução inicial inteligente e a otimiza, melhorando os resultados de maneira considerável.

3.0

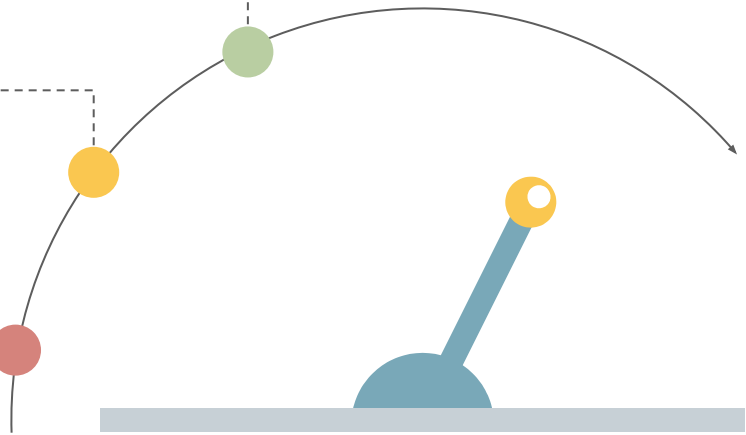
A Pesquisa Local Iterada permite "aprender" com o mínimo local encontrado com uma busca local.

É possível alcançar resultados interessantes através de um algoritmo relativamente simples.

2.0

O problema do caixeiro viajante é um assunto estudado a muitos anos, e uma solução perfeita ainda não existe.

1.0



Obrigado!

brenooliveirareno@unifei.edu.br
@brenooreno
+55 35 9 98177836