



Universidade Federal Da Fronteira Sul

INTELIGÊNCIA ARTIFICIAL

Acadêmico : Sebastien Lionel Lubin
Curso : Ciência da Computação

Busca por Têmpera Simulada

O que é ?

Um exemplo de aplicação da busca por têmpera simulada pode ser a solução do problema do caixeiro-viajante, que consiste em encontrar o menor caminho que passa por todas as cidades em um mapa. A ideia é encontrar um caminho aproximado para esse problema, já que encontrar a solução exata para grandes mapas é um problema NP difícil.

Busca por Temperatura Simulada

```
procedimento SA ( $f(\cdot)$ ,  $N(\cdot)$ ,  $\alpha$ ,  $SA_{max}$ ,  $T_0$ ,  $s$ )  
   $s^* \leftarrow s$       {Melhor solução obtida até então}  
  IterT  $\leftarrow 0$   {Número de iterações na temperatura  $T$ }  
   $T \leftarrow T_0$     {temperatura corrente}  
  enquanto ( $T > 0.0001$ )  
    enquanto (IterT  $< SA_{max}$ ) faça  
      IterT  $\leftarrow$  IterT + 1  
      Gerar um vizinho ( $s'$ ) aleatoriamente na vizinhança  $N^k(s)$   
       $\Delta = f(s') - f(s)$   
      se ( $\Delta < 0$ ) então  
         $s \leftarrow s'$   
        se ( $f(s') < f(s^*)$ ) então  $s^* \leftarrow s'$   
      senão  
        Tome  $x \in [0,1]$   
        se ( $x < e^{-\Delta/T}$ ) então  
           $s = s'$   
      fim-se  
    fim-enquanto  
     $T = T \times \alpha$   
    IterT = 0  
  fim-enquanto  
  retorne  $s^*$   
fim-procedimento
```

A implementação foi baseada nessa ideia

Busca por Têmpera Simulada

A classe BuscaLocalTemperaSimulada recebe como parâmetros uma vizinhança, uma solução ótima conhecida (para avaliação de desempenho), um parâmetro mandatório (para ajuste da temperatura) e uma solução inicial (opcional).

```
from AlgoritmoBusca import AlgoritmoBusca
import time
from Vizinhanca import Vizinhanca
from Vizinhanca2opt import Vizinhanca2opt
from Solucao import Solucao
import random
import math

class BuscaLocalTemperaSimulada(AlgoritmoBusca):
    def __init__(self, vizinhanca: Vizinhanca, solucao_otima, parametro_mandato, solucao: Solucao = None):
        super().__init__("BTS"+vizinhanca.nome, vizinhanca.distancias, solucao_otima)
        self.parametro_mandato = parametro_mandato
        self.vizinhanca = vizinhanca
        self.solucao = self.gerar_solucao_inicial_aleatoria() if solucao is None else solucao
```


Busca por Têmpera Simulada

O método buscar solução implementa o algoritmo de busca em si. Ele mantém uma lista de soluções visitadas e atualiza a solução atual a cada iteração, tentando encontrar uma solução melhor do que a atual através da exploração de vizinhos.

```
def buscar_solucao(self) -> list[Solucao]:
    solucao_list = [self.solucao] # Adiciona a solução atual à lista de soluções
    iteracao = self.solucao.iteracao + 1 # Incrementa a iteração
    melhor_qualidade = self.solucao.qualidade # Salva a melhor qualidade atual
    caminho_atual = self.solucao.ciclo[:] # Salva o caminho atual
    i_inicial, j_inicial = 0, 1 # Define os índices iniciais para a busca na vizinhança
    temperatura = self.solucao.custo * self.parametro_mandato # Define a temperatura inicial
    alpha = 0.95 # Define o parâmetro de resfriamento

    while time.time() < self.tempo_limite: # Enquanto o tempo limite não for atingido
        vizinho = self.vizinhanca.proximo_vizinho(self.solucao, i_inicial, j_inicial)
        iteracao += 1 # Incrementa a iteração
        delta = vizinho.custo - self.solucao.custo # Calcula a diferença de custo entre o vizinho e a solução atual
        if delta < 0: # Se o vizinho for melhor que a solução atual
            self.solucao = vizinho # Define o vizinho como a nova solução
            caminho_atual = self.solucao.ciclo[:] # Atualiza o caminho atual
            if self.solucao.custo < melhor_qualidade: # Se a nova solução for melhor que a atual
                melhor_qualidade = self.solucao.custo # Atualiza a melhor qualidade atual
            solucao_list.append(self.solucao) # Adiciona a nova solução à lista de soluções visitadas
        else: # Se o vizinho não for melhor que a solução atual
            if temperatura == 0: # Se a temperatura atingir o valor mínimo
                return [self.solucao] # Retorna a solução atual como a melhor solução encontrada
            temperatura *= alpha # Resfria a temperatura
```

Busca por Têmpera Simulada

O processo de resfriamento gradual é implementado através da atualização da temperatura em cada iteração. Se o algoritmo encontrar um vizinho com uma solução melhor do que a atual, ele faz a transição para o novo estado. Caso contrário, ele pode aceitar o vizinho com uma probabilidade baseada na temperatura e na diferença de custo entre a solução atual e a do vizinho. O algoritmo termina quando atinge o tempo limite. A lista de soluções visitadas é retornada como resultado da execução.

```
p_aceitacao = math.exp(-delta/temperatura) # Calcula a
if random.random() < p_aceitacao:
    self.solucao = vizinho # Aceita o vizinho como a n
    caminho_atual = self.solucao.ciclo[:]
    solucao_list.append(self.solucao) # Adiciona a solu

i_inicial = (i_inicial + 1) % (len(caminho_atual) - 1) #
if i_inicial == 0:
    j_inicial += 1 # Incrementa o índice de coluna para a

    if j_inicial == len(caminho_atual): # Se o índice de c
        j_inicial = 1 # Reinicia o índice de coluna para c

temperatura = temperatura * alpha # Atualiza a temperatura

return solucao_list # retorna a lista de
```

Busca por Têmpera Simulada

Resultado da amostra





Busca por Têmpera Simulada

Concluindo que, a busca de têmpera simulada pode ser uma abordagem útil e eficaz para resolver problemas de otimização não lineares e não convexos, mesmo quando a função objetivo é complexa e não há uma formulação matemática precisa disponível. No entanto, é importante reconhecer que, como qualquer algoritmo de otimização, a busca de têmpera simulada tem suas limitações e pode não ser a solução ideal em todos os cenários. É importante avaliar cuidadosamente as características do problema a ser resolvido antes de decidir se a busca de têmpera simulada é a abordagem mais adequada.