

Facultad de Ingeniería de Sistemas e Informática
E.A.P de Ingeniería de Sistemas e Informática

TRABAJO GRUPAL

GRUPO N°3

Curso: Investigación Operativa
Docente: Cubas Becerra, Richard Javier

Integrantes:

Aldana Chipana, Mauricio	[22200164]
Castillo Reupo, John	[22200117]
Escribas Alan, Daniel	[22200057]
Espíritu Unsihuay, Erika Milagros	[22200170]
Valdiviezo Goicochea, Wisner	[22200217]

Lima, Perú
2024

Contents

Contents	2
1 Introducción al Templado Simulado	3
2 Funcionamiento del Algoritmo	3
3 Parámetros del Algoritmo	3
4 Ventajas y Limitaciones	3
4.1 Ventajas	3
4.2 Limitaciones	4
5 Caso Aplicativo	4

1 Introducción al Templado Simulado

El templado simulado es un algoritmo de optimización inspirado en el proceso de enfriamiento de metales. Este método es utilizado para encontrar soluciones aproximadas a problemas de optimización complejos.

2 Funcionamiento del Algoritmo

El algoritmo de templado simulado se basa en la analogía con el enfriamiento de metales. A continuación se presenta un modelo matemático que describe su funcionamiento:

$$P(E) = \exp\left(-\frac{\Delta E}{kT}\right) \quad (1)$$

Donde:

- $P(E)$ es la probabilidad de aceptar una solución peor.
- ΔE es el cambio en la energía (o costo) de la solución.
- k es una constante de Boltzmann.
- T es la temperatura.

El algoritmo comienza con una temperatura alta que se va reduciendo gradualmente. En cada iteración, se generan soluciones vecinas y se aceptan o rechazan basándose en la probabilidad $P(E)$.

3 Parámetros del Algoritmo

Los parámetros clave del algoritmo de templado simulado incluyen:

- Temperatura inicial (T_0)
- Tasa de enfriamiento (α)
- Número de iteraciones por temperatura

Estos parámetros deben ser ajustados cuidadosamente para obtener un buen rendimiento del algoritmo.

4 Ventajas y Limitaciones

4.1 Ventajas

- Capacidad para escapar de óptimos locales.
- Flexibilidad para ser aplicado a una amplia variedad de problemas.

4.2 Limitaciones

- Sensibilidad a la elección de parámetros.
- Puede ser computacionalmente costoso para problemas muy grandes.

5 Caso Aplicativo

Para ilustrar el uso del templado simulado, consideremos un problema de optimización en la investigación operativa, como la optimización de rutas de transporte. El objetivo es encontrar rutas eficientes que minimicen el costo total de transporte, considerando restricciones como la capacidad de los vehículos y las ventanas de tiempo de entrega.

El problema de optimización de rutas de transporte (Vehicle Routing Problem, VRP) es un problema clásico en la investigación operativa. En este problema, un conjunto de vehículos debe recoger y entregar bienes a un conjunto de clientes, minimizando el costo total de transporte. Las restricciones incluyen la capacidad de los vehículos y las ventanas de tiempo en las que las entregas deben realizarse.

El algoritmo de templado simulado puede ser utilizado para encontrar soluciones aproximadas a este problema. A continuación, se presenta una implementación en Python del algoritmo de templado simulado para resolver el VRP.

```
1 import math
2 import random
3
4 # Funcion de costo (distancia total de la ruta)
5 def calculate_cost(route, distance_matrix):
6     cost = 0
7     for i in range(len(route) - 1):
8         cost += distance_matrix[route[i]][route[i + 1]]
9     cost += distance_matrix[route[-1]][route[0]] # Regresar al
10         punto de inicio
11     return cost
12
13 # Generar una solucion vecina
14 def generate_neighbor(route):
15     new_route = route[:]
16     i, j = random.sample(range(len(route)), 2)
17     new_route[i], new_route[j] = new_route[j], new_route[i]
18     return new_route
19
20 # Algoritmo de templado simulado
21 def simulated_annealing(distance_matrix, initial_temp, cooling_rate,
22     num_iterations):
23     current_route = list(range(len(distance_matrix)))
24     random.shuffle(current_route)
25     current_cost = calculate_cost(current_route, distance_matrix)
26     best_route = current_route[:]
27     best_cost = current_cost
28     temperature = initial_temp
29
30     for _ in range(num_iterations):
```

```

29     new_route = generate_neighbor(current_route)
30     new_cost = calculate_cost(new_route, distance_matrix)
31     delta_cost = new_cost - current_cost
32
33     if delta_cost < 0 or random.random() < math.exp(-delta_cost
34         / temperature):
35         current_route = new_route
36         current_cost = new_cost
37
38         if current_cost < best_cost:
39             best_route = current_route
40             best_cost = current_cost
41
42     temperature *= cooling_rate
43
44     return best_route, best_cost
45
46 # Ejemplo de uso
47 distance_matrix = [
48     [0, 2, 9, 10],
49     [1, 0, 6, 4],
50     [15, 7, 0, 8],
51     [6, 3, 12, 0]
52 ]
53
54 initial_temp = 1000
55 cooling_rate = 0.99
56 num_iterations = 10000
57
58 best_route, best_cost = simulated_annealing(distance_matrix,
59     initial_temp, cooling_rate, num_iterations)
60 print("Mejor ruta:", best_route)
61 print("Costo de la mejor ruta:", best_cost)

```

Este código proporciona una base para implementar el algoritmo de templado simulado en Python. Puedes adaptarlo y expandirlo según tus necesidades específicas.