

TSP Problem (Simulated Annealing)

Penadillo Lazares Wenses Johan



**UNIVERSIDAD
NACIONAL DE
INGENIERÍA**

6 de diciembre de 2021

Índice

Introducción

Simulated Annealing

Implementación en python 3

Introducción

Algoritmo de recocido simulado (SA), se presentó por primera vez como un algoritmo de búsqueda para problemas de optimización combinatoria , es un popular algoritmo meta-heurístico iterativo ampliamente utilizado para abordar problemas de optimización discretos y continuos. La característica clave del algoritmo SA radica en los medios para escapar de los óptimos locales al permitir movimientos de escalada para encontrar un óptimo global. Una de las principales carencias de SA es que tiene varios parámetros para ajustar y su rendimiento es sensible a los valores de esos parámetros de control.

Simulated Annealing

La idea fundamental es aceptar movimientos que den como resultado soluciones de peor calidad que la solución actual para escapar de los mínimos locales. La probabilidad de aceptar tal movimiento disminuye durante la búsqueda a través del parámetro de temperatura.

El algoritmo SA comienza con una solución inicial x , y la solución candidata y es generada posteriormente (ya sea al azar o usando alguna regla pre-especificada) a partir de la vecindad de x .

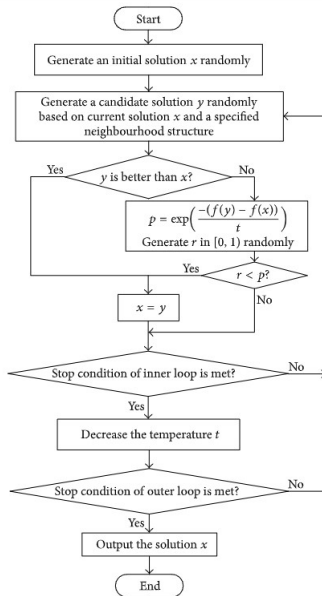
El criterio de aceptación de Metrópolis

Modela cómo un sistema termodinámico se mueve de un estado a otro en el que se minimiza la energía, se utiliza para decidir si se acepta y o no. La solución candidata y se acepta como la solución actual x en función de la probabilidad de aceptación:

$$p = \begin{cases} 1 & \text{si } f(y) \leq f(x) \\ e^{\frac{-(f(y)-f(x))}{t}} & \text{si } f(y) > f(x) \end{cases}$$

Donde t es el parámetro de temperatura.

Flujo de ejecución del algoritmo



Simulated Annealing

Para aplicar el algoritmo SA a un problema específico, se debe especificar la estructura del vecindario y el programa de enfriamiento.

El programa de enfriamiento geométrico, que puede describirse mediante la siguiente fórmula de actualización de temperatura:

$$t_{k+1} = \alpha t_k$$

Valores típicos para α están entre 0,8 y 0,99

Clase Coordenada

```
1 class Coordinate:
2     def __init__(self, x, y):
3         self.x = x
4         self.y = y
5
6     @staticmethod
7     def get_distance(a, b):
8         return numpy.sqrt(numpy.abs(a.x-b.x)+numpy.abs(a.y-b.y))
9
10    @staticmethod
11    def get_total_distance(coords):
12        dist = 0
13        for first, second in zip(coords[:-1], coords[1:]):
14            dist += Coordinate.get_distance(first, second)
15        dist += Coordinate.get_distance(coords[0], coords[-1])
16        return dist
```

```
1  # Crearemos las coordenadas
2  coords = []
3  for i in range(20):
4      coords.append(Coordinate(numpy.random.uniform()
5                               , numpy.random.uniform()))
6
7  # Simulated annealing algorithm
8  cost0 = Coordinate.get_total_distance(coords)
9  T = 30 # Temperatura inicial
10 factor = 0.99 # factor de decrecimiento
11
12 for i in range(1000):
13     for j in range(300):
14         # Generando una nueva solucion
15         r1, r2 = numpy.random.randint(0, len(coords), size=2)
16
17         temp = coords[r1]
18         coords[r1] = coords[r2]
19         coords[r2] = temp
20
21         # Calculando el nuevo costo
22         cost1 = Coordinate.get_total_distance(coords)
23
```

```
24     if cost1 < cost0:
25         cost0 = cost1
26     else:
27         x = numpy.random.uniform()
28         if x < numpy.exp((cost0-cost1)/T):
29             cost0 = cost1
30         else:
31             temp = coords[r1]
32             coords[r1] = coords[r2]
33             coords[r2] = temp
34 T = T*factor
```
