Heurisctic Gradient Descent

Penadillo Lazares Wenses Johan

UNIVERSIDAD NACIONAL DE INGENIERÍA

26 de noviembre de 2021

Índice

Gradient Descent

Este método consiste en minimizar una función, escogiendo un punto inicial, una dirección y un tamaño de paso. Para así cambiar el punto inicial y llegar al mínimo.

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.

Explicación del problema

El problema de TSP es uno de los problemas de optimización combinatoria más famosos. Pertenece a la clase de problemas de optimización NP-hard. Esto significa que no se conoce ningún algoritmo de tiempo polinomial que garantice su solución óptima global.

Sea un vendedor que tiene que visitar *n* ciudades. El problema del TSP consiste en encontrar el recorrido más corto por todas las ciudades de manera que ninguna ciudad sea visitada dos veces y el vendedor regrese a la ciudad de partida al final del recorrido.

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.

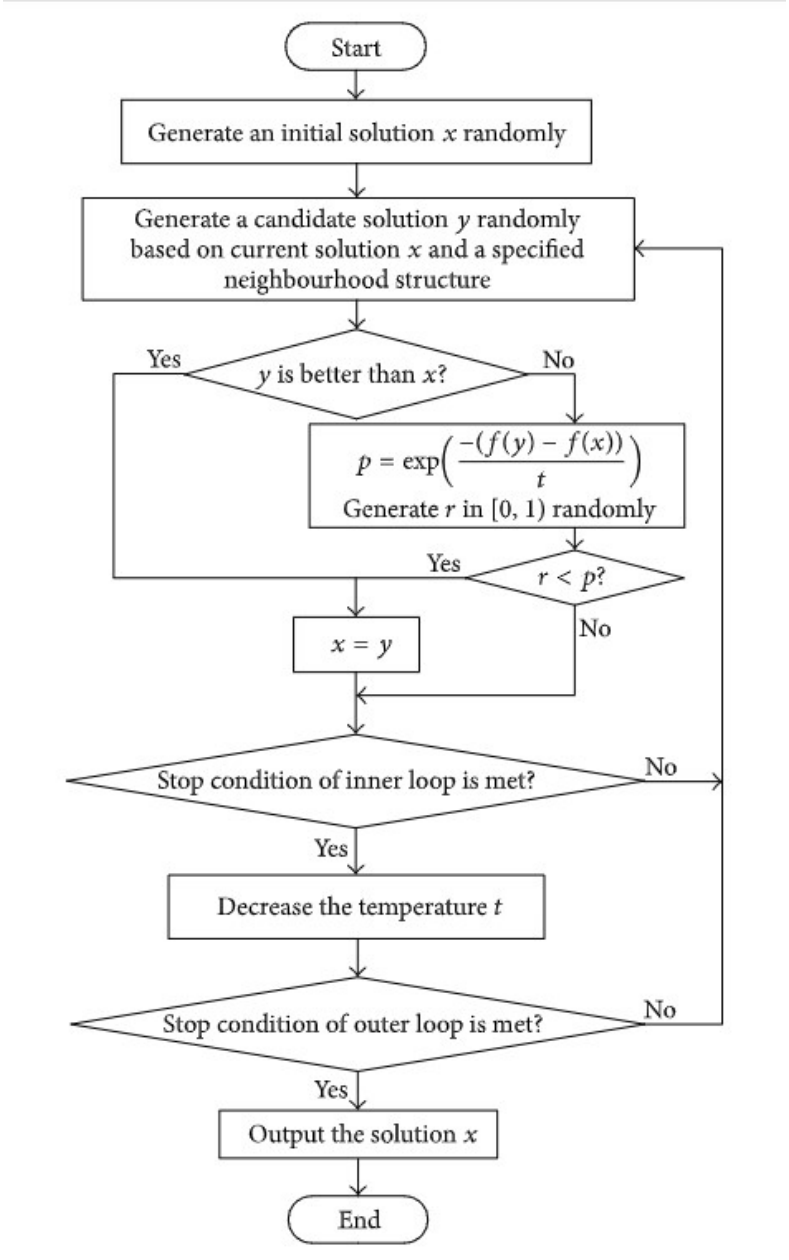
El criterio de aceptación de Metrópolis

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:

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

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

```
class Coordinate:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    @staticmethod
    def get_distance(a, b):
        return numpy.sqrt(numpy.abs(a.x-b.x)+numpy.abs(a.y-b.y))

    @staticmethod
    def get_total_distance(coords):
        dist = 0
        for first, second in zip(coords[:-1], coords[1:]):
            dist += Coordinate.get_distance(first, second)
        dist += Coordinate.get_distance(coords[0], coords[-1])
        return dist
```

```
# Crearemos las coordenadas
coords = []
for i in range(20):
    coords.append(Coordinate(numpy.random.uniform()
        , numpy.random.uniform()))

# Simulated annealing algorithm
cost0 = Coordinate.get_total_distance(coords)
T = 30 # Temperatura inicial
factor = 0.99 # factor de decrecimiento

for i in range(1000):
    for j in range(300):
        # Generando una nueva solucion
        r1, r2 = numpy.random.randint(0, len(coords), size=2)

        temp = coords[r1]
        coords[r1] = coords[r2]
        coords[r2] = temp

        # Calculando el nuevo costo
        cost1 = Coordinate.get_total_distance(coords)

```

```
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
```