INSTITUTO POLITÉCNICO NACIONAL

Centro de Investigación en Computación

ASIGNATURA:

Metaheurísticas

Actividad #16:

Guía Taller Laboratorio No.7

"Solución de problemas mediante Recocido Simulado"

PROFESORA:

Dra. Yenny Villuendas Rey

PRESENTA:

Juan René Hernández Sánchez

Adriana Montserrat García Carrillo

Centro de Investigación en Computación
Instituto Politécnico Nacional

1. Introducción

Muchos problemas de ingeniería, planificación y fabricación pueden ser modelados como minimizar o maximizar una función de coste sobre un conjunto finito de variables discretas. Esta clase de problemas, llamados de optimización combinatoria, ha recibido mucha atención en las dos últimas décadas y se han conseguido importantes logros en su análisis. Uno de estos logros es la separación de esta clase en dos subclases. La primera contiene los problemas que pueden resolverse de forma eficiente, es decir, los problemas para los que se conocen algoritmos que resuelven cada instancia de forma óptima en tiempo polinómico. La segunda subclase contiene los problemas que son notoriamente difíciles, denominados formalmente NP-duros.

Para un algoritmo NP-duro se cree que no existe ningún algoritmo que resuelva cada instancia en tiempo polinómico. En consecuencia, hay instancias que requieren un tiempo súper polinomial o exponencial para ser resueltas de forma óptima. Está claro que los problemas difíciles deben tratarse en la práctica, lo cual puede hacerse mediante dos tipos de algoritmos: de optimización, que encuentran soluciones óptimas posiblemente utilizando grandes cantidades de tiempo de cómputo, o algoritmos heurísticos que encuentran soluciones aproximadas en pequeñas cantidades de tiempo de cálculo.

Los algoritmos de búsqueda local son del tipo heurísticos. Constituyen un enfoque general muy utilizado para los problemas de optimización combinatoria. Suelen ser instancias de varios esquemas de búsqueda general, pero todas tienen la misma característica de una función de vecindad subyacente, que se utiliza para guiar la búsqueda de una buena solución.

El recocido simulado, es uno de los algoritmos de búsqueda local más conocidos, ya que tiene un rendimiento bastante bueno y es ampliamente aplicable.

En la física de la materia condensada, el recocido simulado se conoce como un proceso térmico para obtener estados de baja energía de un sólido en un baño de calor. El proceso consiste en los siguientes dos pasos [1]:

- aumentar la temperatura del baño de calor hasta un valor máximo en el que el sólido se funde
- disminuir cuidadosamente la temperatura del baño de calor hasta que las partículas se organicen en el estado básico del sólido.

A continuación, se presentará una explicación del recocido simulado, sus características, aplicaciones e implementación

2. Desarrollo

Asignatura: Metaheurísticas
Actividad No.16
Guía Taller Laboratorio No.7
Título: Solución de problemas mediante Recocido Simulado

Contenido:

- Métodos heurísticos de solución de problemas.
- Recocido Simulado.

Objetivo: Modelar problemas clásicos de búsqueda mediante el uso de algoritmos de Recocido Simulado, para la solución de problemas de la profesión.

1. Realice la modelación matemática necesaria para la solución, mediante SA, de la obtención de mínimos de la función $f(x) = \sum_{i=1}^{D} x_i^2,\ con\ -10 \leq\ x_i\ \leq 10$.

-Recuerde que la modelación matemática incluye: definición de los estados inicial y final, definición del test objetivo, y definición de las acciones posibles (operadores).

- **Estado inicial:** arreglo de D números x
  s.a.
  $x = \{x_i \mid -10 \leq\ x_i \leq 10,\ 1 \leq i \leq D\}$.

- **Estado final:** arreglo de D números x'
  s.a.
  $\min\limits_{-10 \leq x_i \leq 10} \{ \sum_{i=1}^{D} x_i^2 \}$

- **Test objetivo:** $f(x) = \sum_{i=1}^{D} x_i^2,\ con\ -10 \leq\ x_i \leq 10.$

- **Acciones posibles (operadores):**
  $g: x \rightarrow x'$ **s.a.** $\exists\ x_i \neq\ x_i',\ -10 \leq\ x_i' \leq 10,\ 1 \leq i \leq D.$

2. Realice una corrida manual del algoritmo de Recocido Simulado sobre el problema anterior. Defina para ello un esquema de recocido de su preferencia.

```
#Definición de parámetros
interacionMax = 3     #número de vecinos
a = 0.3               #valor de alfa
temp = 30° C
tempMax = temp
tempMin = 10° C
K = 1
```

#Se establece el arreglo inicial x
x = [8, -4, 1, 3, -7]

#Se calcula la función de costo
edo_anterior = $8^2 + -4^2 + 1^2 + 3^2 + -7^2$ = 64 + 16 + 1 + 9 + 49 = 139

#Evaluación vecino 1 de 3
sucesor = [8, -4, 2, 3, -7]
edo_nuevo = $8^2 + -4^2 + 2^2 + 3^2 + -7^2$ = 64 + 16 + 4 + 9 + 49 = 142

#Calculo delta = edo_nuevo – edo_anterior
delta = 142 - 139 = 3 #Delta>0 y se está minimizando entonces se hace una 2da evaluación
random = 0.5
var = exp(3/1*30) = 1.1 #exp(-delta/K*temp)
#como 0.5 < 1.1 se acepta la solución; random<exp(-delta/K*temp)
edo_anterior = 142
x = [8, -4, 2, 3, -7]

#Evaluación vecino 2 de 3
sucesor = [5, -4, 2, 3, -7]
edo_nuevo = $5^2 + -4^2 + 2^2 + 3^2 + -7^2$ = 25 + 16 + 4 + 9 + 49 = 103

#Calculo delta = edo_nuevo – edo_anterior
delta = 103 – 142 = -39 #Delta < 0 entonces edo_anterior = edo_nuevo
edo_anterior = 103
x = [5, -4, 2, 3, -7]

#Evaluación vecino 3 de 3
sucesor = [5, -4, 2, 3, 6]
edo_nuevo = $5^2 + -4^2 + 2^2 + 3^2 + 6^2$ = 25 + 16 + 4 + 9 + 36 = 90

#Calculo delta = edo_nuevo – edo_anterior
delta = 90 – 103 = -13 #Delta < 0 entonces edo_anterior = edo_nuevo
edo_anterior = 90
x = [5, -4, 2, 3, 6]

#Disminución de la temp
temp = a * temp = 0.3 * 30 = 9 #Como temp es menor a 10° C se termina el algoritmo

3. Para el problema planteado, proponga las estructuras de datos necesarias para su implementación

- listas

4. Diseñe la interfaz de usuario para la solución del problema planteado mediante Recocido Simulado.

```python
import copy
import random
import math

def costo(funcion):
    valmax = 0
    for i in funcion:
        valmax += i**2
    return valmax

def vecindad(funcion,n):
    new_funcion = funcion.copy()
    locus = random.randint(0, n-1)
    new_funcion[locus] = random.uniform(-10, 10)
    return new_funcion

n = input('Ingresa el número de elementos: ')
n = int(n)

funcion_x = []
for i in range(n):
    funcion_x.append(random.uniform(-10, 10))

temp_min = 0.01
temp_max = 300.0
vecinos = n - 1  # n es número de elementos en el arreglo
alfa = 0.8
K = 1.0

edo_anterior = costo(funcion_x)

print("\n")
print("Valores Iniciales: ", funcion_x)
print("costo igual a: ", edo_anterior)
print("\n")

temp = temp_max

while temp >= temp_min:
    vecinos_revisados = 0
    while vecinos_revisados < vecinos:
        sucesor = vecindad(funcion_x,n)
```

```python
        edo_nuevo = costo(sucesor)
        delta = edo_nuevo - edo_anterior
        if delta > 0:
            if random.random() >= math.exp((-delta)/(K*temp)):
                # TODO eliminar el sucesor
                sucesor = []
                edo_nuevo = 0
            else:
                edo_anterior = edo_nuevo
                funcion_x = sucesor
        else:
            edo_anterior = edo_nuevo
            funcion_x = sucesor
        vecinos_revisados = vecinos_revisados + 1
    temp = temp * alfa


print("\n")
print("Valores Finales: ", funcion_x)
print("costo igual a: ", edo_anterior)
print("\n")
```

5. Ejecute la solución del problema planteado mediante Recocido simulado, utilizando para ello las estructuras de datos y la interfaz gráfica diseñadas.

```
Valores Iniciales:  [-4.546446972938297, 4.49609046087849, 7.228700274377022, 9.703745727097445, -6.445255418664448, -0.7517493404022346, -7.718648163597408, -4.9
53088832352681, 7.383733760877526, -3.1371620688095714]
costo igual a:  377.8801713377839




Valores Finales:  [-0.0009664585701152362, 0.07492606967783999, 0.03595837308391836, -0.01765432074379092, -0.450080760011911, 0.5531426767044696, 0.0755216551370
2924, -0.04900726198925298, -0.006360649401910834, 0.19484076408354412]
costo igual a:  0.5618676542517811
```

3. Conclusiones

Al realizar la corrida manual del algoritmo SA, se pudo apreciar que los algoritmos de búsqueda local pueden ser muy útiles si estamos interesados en el estado de la solución, pero no en el camino hacia ese objetivo. Estos algoritmos operan sólo en el estado actual y se mueven a estados vecinos. Al permitir un ascenso ocasional en el proceso de búsqueda, se puede escapar de la trampa de los mínimos locales, pero también existe la posibilidad de pasar óptimos globales después de alcanzarlos.

Se puede aplicar el SA para generar una solución a los problemas de optimización combinatoria asumiendo una analogía entre ellos y los sistemas físicos de muchas partículas con las siguientes equivalencias:

- Las soluciones del problema son equivalentes a los estados de un sistema físico.
- El costo de una solución es equivalente a la "energía" de un estado

Por último, es importante mencionar que los algoritmos de búsqueda local tienen dos ventajas fundamentales: usan muy poca memoria y pueden encontrar soluciones razonables en espacios de estados grandes o infinitos (continuos).

4. Referencias

[1] Burke & Kendall. Search Metodologies – 2005. Capítulo 7