## PRÁCTICA 2. Recocido Simulado

Alumno: Isaac André Canalizo Mendoza 201728726

### I Objetivos

General: Diseñar un algoritmo de recocido simulado para la resolución de dos problemas discretos y uno continuo.

Específico: Para cada uno de los problemas encontrar cuál es la configuración de Simulated Annealing que obtiene en promedio el mejor resultado.

### II Desarrollo

**Problemas continuos**

1. **My\_func4:**
   1. **Temperatura inicial:**

Se estableció con un valor de 100

* 1. **Función de descenso de temperatura:**

Se estableció como ((100)/(i+1)) ya que se obtenía un descenso controlado adecuado para la elección de vecindario y la función de aceptación mostraba resultados favorables.

* 1. **Función de aceptación:**

Metrópolis con un valor de k=10

* 1. **Configuración del vecindario:**

for i in range(100):

    temp = 100/(i+1)

    if(temp<10 and temp > 5):

        #El tamaño del vecindario es de 10

        x1, x2 = vecindario(5, x1prim), vecindario(5, x2prim)

    elif(temp<=5 and temp > 2.5):

        #Varia en 4 el tamaño del vecindario

        x1, x2 = vecindario(2, x1prim), vecindario(2, x2prim)

    elif(temp<=2.5 and temp > 1.5):

        #El tamaño del vecindario sigue siendo de 4 pero solo se modifica uno de los dos valores

        if( i%2 == 0):

            x1 = vecindario(2, x1prim)

        else:

            x2 =  vecindario(2, x2prim)

    elif(temp <= 1.5):

        #El tamaño de vecindario se reduce a 2 y se continua aplicando a un solo valor

        if( i%2 == 0):

            x1 = vecindario(1, x1prim)

        else:

            x2 =  vecindario(1, x2prim)

    elif(temp <= 1.3):

        #El tamaño del vecindario se reduce a 1

        if( i%2 == 0):

            x1 = vecindario(.5, x1prim)

        else:

            x2 =  vecindario(.5, x2prim)

    else:

        #Se generan aleatorios en el rango [-25, 25]

        x1, x2 = generateRand(), generateRand()

La función vecindario es la siguiente :

*def* vecindario(*rango*, *numero*):

    limInferior = numero - rango

    limSuperior = numero + rango

    if(limSuperior > 25.0):

        r = random.uniform(limInferior, 25.0)

    elif(limInferior < -25.0):

        r = random.uniform(-25.0, limSuperior)

    else:

        r = random.uniform(limInferior, limSuperior)

    return r

1. **My\_func5:**
   1. **Se siguió la misma configuración que en el problema anterior**

**Problema discreto**

1. **My\_graph1:**
   1. **Temperatura inicial: 500**
   2. **Función de descenso de temperatura: temp = 500/(i+1)**
   3. **Función de aceptación: metrópolis con k = 100**
   4. **Elección de vecindario**

**Configuración del vecindario:**

temp = 500/(i+1)

    if(temp<1):

        #Antes de llegar acá itera 500 veces

        if(temp>0.67):

            # Itera hasta 750

            if(i%2 == 0):

                r = random.randint(1,7)

                x = switch\_pair(r, comb)

            else:

                x = switch\_pair(8, comb, 8)

        elif(temp>0.51):

#Itera hasta 980

            r = random.randint(0,2)

            match r:

                case 0:

                        x = switch\_pair(6, comb)

                case 1:

                        x = switch\_pair(7, comb)

                case 2:

                    x = switch\_pair(8, comb, 4)

        elif(mejor > 5):

            #Itera las últimas 20 veces

            r = random.randint(0,18)

            x = switch\_pair(0, comb, r)

## Cambio de pares

*def* opc1(*lista*: *list*, *pos*:*int*):

        lista[pos], lista[pos+1] = lista[pos+1], lista[pos]

        return lista

############################## CAMBIO DE EXTREMOS ########################################

*def* opc2(*lista*):

        for i in range(5):

            lista[i], lista[19-i] = lista[19-i], lista[i]

        return lista

############################## CAMBIO DE CENTRO ########################################

*def* opc3(*lista*):

        for i in range(5):

            lista[5+i], lista[14-i] = lista[14-i], lista[5+i]

        return lista

############################### MEZCLA DE PRIMER Y TERCER CUARTO ##########################

*def* opc4(*lista*):

        for i in range(5):

            lista[i], lista[14-i] = lista[14-i], lista[i]

        return lista

############################### MEZCLA SEGUNDO Y CUARTO CUARTO ############################

*def* opc5(*lista*):

        for i in range(5):

            lista[5+i], lista[19-i] = lista[19-i], lista[5+i]

        return lista

############################### MEZCLA PRIMERA MITAD ############################

*def* opc6(*lista*):

    used = []

    con = 0

    while len(used) < 10:

        r = random.randint(0,9)

        if(r not in used):

            lista[con], lista[r] = lista[r], lista[con]

            used.append(r)

            con += 1

    return lista

############################### MEZCLA SEGUNDA MITAD ############################

*def* opc7(*lista*):

    used = []

    con = 5

    while len(used) < 10:

        r = random.randint(10,19)

        if(r not in used):

            lista[con], lista[r] = lista[r], lista[con]

            used.append(r)

            con += 1

    return lista

*def* opc8(*lista*, *numRand*):

    rand = []

    while len(rand) < numRand:

        r = random.randint(0,19)

        if(r not in rand):

            rand.append(r)

    lim = *int*(numRand/2)

    for i in range(lim):

        lista[rand[i]], lista[ rand[(numRand-1)-i] ] = lista[rand[(numRand-1)-i]], lista[rand[i]]

    return lista

### IV Resultados

Para el primer problema continuo obtiene un resultado promedio de: 0.464763 con 10 iteraciones

Tabla

Descripción generada automáticamente

Fig. 1.1 Captura de suma de resultados de diez corridas y su promedio

Texto

Descripción generada automáticamente

Fig. 1.2 Resultado de la última iteración

Para el segundo problema continuo obtiene un resultado promedio de: 0.565086 con 10 iteraciones

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Fig. 1.1 Captura de suma de resultados de diez corridas y su promedio

Texto

Descripción generada automáticamente

Fig. 1.2 Resultado de la última iteración

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Para el primer problema discreto obtiene un resultado promedio de: 4435.1 con 10 iteraciones