Cómputo Evolutivo
Tarea 2

Adrián García Pérez
315131224

1 Representación Binaria para Números Reales

a) Describe brevemente el algoritmo de representación binaria para mapear (codificar y decodificar) números naturales. Menciona cuántos números son representables con m bits.

El procedimiento para pasar de un número natural a binario consiste en la división iterativa del mismo, guardando los residuos correspondientes en la cabeza de una lista o pila. Esos dígitos serán su representación binaria.

Algorithm 1: Natural to binary

Input: Integer n
Output: List: *bin_list*

```
1 bin_list ← [ ]
2 while n > 0 do
3 |   res = n mod 2
  |   /* Add res to the start of the list                    */
4 |   prepend(bin_list, rem)
5 |   n = n/2
6 return bin_list
```

Y el procedimiento para pasar de un número binario a natural consiste en sumar iterativamente las potencias de 2 correspondientes a cada posición del arreglo de bits, de derecha a izquierda.

Algorithm 2: Binary to natural

Output: Integer n
Input: List: *bin_list*

```
1 n ← 0
2 length ← |bin_list|
3 range ← [0, 1, ..., length]
4 for i in range do
5 |   if bin_list[i] == 1 then
6 |   |   tmp ← pow(2, length − i − 1)
7 |   |   n ← n + tmp
8 return n
```

Así, si tenemos m bits disponibles, sabemos que podemos representar hasta 2^m diferentes números naturales.

Ambos algoritmos se encuentran implementados en el archivo `src/p1/BinaryUtils.py`.

b) Describe en qué consiste la representación de números con códigos de Gray. Menciona las ventajas o desventajas en comparación con la representación binaria tradicional.

El código de Gray es una representación de los números utilizando los dígitos 0 y 1, como el binario. Sin embargo, la diferencia con este último es que no sigue el mismo patrón, si no que en codificación Gray, si tenemos dos números consecutivos n y m, sus representaciones en gray, n' y m', difieren en un solo dígito en toda la cadena de bits.

Su motivación fue debido a cuando los sistemas de cómputo representan números. En el caso del número 3 y 4 (011, 100 respectivamente en binario), las tres entradas de bits difieren, provocando que, si buscamos representar dichos números con switches, los tres switches tengan que cambiar su estado. Esto puede llevar a un problema de amgigüedad, debido a que no sabemos si los switches cambiaron al mismo tiempo o no, pudiendo arrojar lecturas erróneas.

Así, la principal ventaja de la codificación Gray es que, debido a la diferencia de un solo bit en números consecutivos, podemos aprovechar esta propiedad en diferentes áreas, como desarrollo de circuitos booleanos, algoritmos genéticos, correción de errores, etc.

c) Si se requiere una representación uniform de números reales en el intervalo $[a, b]$, ¿cómo se generaliza la representación binaria (o los códigos de Gray) para mapear números reales? ¿Cuál es la máxima precisión?

En primer lugar, necesitamos que, dado el intervalo $[a, b]$ con $b < a$, saber la cantidad de números que podemos representar en ese intervalo, dada una cantidad de m bits.

Así, la siguiente expresión nos indica la separación entre los números reales que vamos a poder representar, la precisión:

$$\Delta = \left\lfloor \frac{b-a}{2^m - 1} \right\rfloor,$$

se "reparten" $b - a$ números entre $2^m - 1$ bits. Esto nos indica cual es la "separación" Δ en decimales entre los números que vamos a representar.

Así, si buscamos saber la representación binaria de un número real r según nuestra codificación, resta obtener la diferencia $a - r$, dividirla por nuestra separación Δ y pasar ese número a su representación binaria. Si nuestra cantidad de bits m es mayor a los bits necesarios para representar el entero, rellenamos el arreglo con ceros.

La siguiente expresión, dadas la cota superior, inferior y la delta deseada, nos indica la cantidad de bits necesarios para llegar a la precisión especificada por la delta.

$$m = \log_2 \left(\frac{b-a}{\Delta} + 1 \right)$$

El algoritmo para obtener la representación en Gray, consiste en tomar la representación binaria y crear nuestro arreglo de Gray. Después, agregar el primer bit en binario a la lista de Gray, y el resto de dígitos serán el resultado de hacer un `exclusive or` con el bit actual y siguiente del arreglo binario, es decir:

Algorithm 3: Binary to Gray

Input: List *bin_list*
Output: List: *gray_list*

```
1 gray_list ← [bin_list[0]]
2 length ← |bin_list|
3 range ← [1, 2, ..., length]
4 for i in range do
5     append(bin_list[i − 1] ⊕ bin_list[i])
6 return gray_list
```

El algoritmo se encuentra implementado en el archivo `src/p1/BinaryUtils.py`.

2 Búsqueda Local por Escalada

a) Implementa un algoritmo para generar soluciones aleatorias de números reales representados con códigos de gray, utilizando m bits.

Algorithm 4: Random Gray Solution

Input: Double a, Double b, Integer d, Integer m
Output: Vector: *numbers*
/* *numbers* `will have the output` */
/* *tmp* `is filled with` d `random numbers in range` $[a, b]$ */
1 *numbers* ← [ ]
2 *tmp_vec* ← random_numbers(a, b, d)
3 **for** *rand_num in tmp_vec* **do**
4 | *gray_number* ← real_to_gray(*rand_num*, m)
5 | *numbers*.append(*gray_number*)
6 return *numbers*

El algoritmo es el mismo para obtener soluciones aleatorias en binario, salvo el cambio en la línea 4 utilizando una función `real_to_bin`.

La implementación se encuentra en el archivo `src/p1/Solution.py`, utilizando algunas funciones definidas en `src/p1/BinaryUtils.py`.

b) Implementa una función para obtener la vecindad considerando el intercambio de un bit.

Algorithm 5: Random Neighbourhood

Input: List: *bin_list*
Output: List: *random_neighbours*
1 *random_neighbours* ← [[ ]]
2 *length* ← $|$*bin_list*$|$
3 *range* ← $[1, 2, ..., length]$
4 **for** i *in range* **do**
5 | $n = [1 - bit$ if $pos == i$ else bit for (pos, bit) in enumerate(*solution*)$]$
6 | *random_neighbours*$[i] \leftarrow n$
7 return *random_neighbours*

La implementación se encuentra en el archivo `src/p1/BinaryUtils.py`.

c) Implementa la búsqueda por escalada con cualquiera de las 3 estrategias vistas en clase.

Algorithm 6: Hill Climbing

Input: Double a, Double b, Integer d, Integer m
Output: List: *random_neighbours*
1 *sol* ← RandomGraySolution(a, b, d, m)
2 *best_eval* ← eval(*sol*)
3 *improvement* ← $True$
4 **while** *improvement* **do**
5 | *improvement* ← $False$
6 | *neighbourhood* ← RandomNeighbourhood(*sol*)
7 | *rand_solution* ← select_random(*neighbourhood*)
8 | *rand_eval* ← eval(*rand_solution*)
9 | **if** *rand_eval* $<$ *best*$_e$*val* **then**
10 | | *best_eval* ← *rand_eval*
11 | | *sol* ← *rand_solution*
12 | | *improvement* ← $True$

d) Prueba tu implementación utilizando las 6 funciones de optimización continua para la tarea 1.

Función	Promedio de iteraciones	Codificación	Promedio	Mejor valor	Peor valor
Sphere	105.6	Gray	10.451	0.0488	43.031
Ackley	78.9	Gray	20.231	20.0001	20.978
Griewank	161.7	Gray	26.560	0.611	208.136
TenthPower	119	Gray	1199950.584	2.71E-08	11131900.871
Rastrigin	106.2	Gray	32.799	18.615	72.824
Rosenbrock	131	Gray	283.911	0.988	1061.449

Las ejecuciones se encuentran en el archivo `ejecuciones/Funciones.xls`.

3 Búsqueda por Escalada Estocástica para TSP

- Implementa el algoritmo de Búsqueda por Escalada Estocástica para el problema TSP.

Algorithm 7: Stochastic Hill Climbing

Input: TSPInstance: *instance*, Integer: *max_iterations*, Integer: *temp*
Output: TSPSolution: *best_solution*, Double: *best_eval*

```
1  iterations ← 0
2  current_solution ← random_solution(instance)
3  current_eval ← eval(current_solution)
4  best_eval ← ∞
5  best_solution ← current_solution
6  while iterations < max_iterations do
7      neighbourhood ← gen_neighbourhood(current_solution)
8      rand_neighbour ← select_random(neighbourhood)
9      rand_eval ← eval(rand_neighbour)
10     if rand_eval < current_eval then
11         current_solution ← rand_neighbour
12         if rand_eval < best_eval then
13             best_eval ← rand_eval
14             best_solution ← rand_neighbour
15     else
16         acceptance_probability ← exp(−(rand_eval − current_eval)/temp)
17         rand_prob ← random_float()
18         if rand_prob < acceptance_probability then
19             current_solution ← rand_neighbour
20     iterations ← iterations + 1
21 return best_solution, best_eval
```

La implementación se encuentra en el archivo `src/p2/main.py`, utilizando las implementaciones de `src/p2/Instance.py` y `src/p2/Solution.py`.

4 Prueba de la Implementación

4 Prueba la implementación considerando las 3 instancias de prueba del TSP utilizadas en la tarea 1

Instancia	Temperatura	Mejor valor	Promedio	Peor valor
pbl395.tsp	5	18671.863	19210.009	19481.842
pbl395.tsp	100	18307.857	19090.321	19877.921
pbl395.tsp	10000	18349.353	18887.254	19324.232
xqf131.tsp	5	3991.798	4288.800	4558.210
xqf131.tsp	100	3981.006	4216.777	4582.975
xqf131.tsp	10000	3767.293	4182.232	4402.255
xqg237.tsp	5	11490.922	11805.955	12293.413
xqg237.tsp	100	11448.066	11738.093	12006.897
xqg237.tsp	10000	11153.6939	11514.095	11838.446

¿Qué efecto tiene el valor de T en la búsqueda por escalada estocástica?

El valor de la temperatura es muy importante en la ejecución del algoritmo, ya que entre más alto sea, más probable será el algoritmo salte escogiendo soluciones peores a la actual. Sin embargo, esto puede favorecer en casos donde estemos atrapados en mínimos locales, pudiéndonos llevar a nuevos mínimos que pueden ser mejores, pero también puede llevarnos a otros lugares con peores resultados.

¿Hay alguna relación entre el valor de T, el tamaño de la instancia y los resultados obtenidos en cada caso?

En general, se pudo ver una ligera mejoría en el valor promedio de las instancias cuando usábamos una temperatura mayor. Tal vez podría ser mejor si analizamos cada instancia al cambiar el valor de la temperatura, porque a mi parecer, cada una debe tener un valor que le beneficie o le empeore.

5 Conclusiones

Me parece que la solución de esta tarea resulto ser más complicada en algunos aspectos a la anterior, aunque así mismo los resultados fueron mejores.

Me hubiera gustado analizar con más detenimiento el comportamiento de las ejecuciones al variar el número de bits para las representaciones, así como analizar los resultados de ambos ejercicios de manera gráfica.