Maestría en Ciencias de la Computación

Asignatura: **Metaheurísticas** martes, 9 de marzo de 2021

**Actividad No.3**

**Guía Clase Práctica No.1**

**Título**: Solución de problemas mediante Ascensión de Colinas

**Alumnos:**

* Rojas Zepeda Luis Eduardo.
* Ruiz Fonseca Emmanuel Alejandro.

**Contenido:**

▪ Métodos heurísticos de solución de problemas.

▪ Ascensión de Colinas.

▪ Ascensión de Colinas con mutación aleatoria

**Objetivo:**

Modelar problemas clásicos de búsqueda mediante el uso de algoritmos de Ascensión de Colinas, para la solución de problemas de la profesión.

**Qué Estudiar**

Métodos heurísticos. Algoritmo de Ascensión de Colinas. Operadores. Ventajas y Desventajas. Aplicaciones.

**Cómo Estudiar**

1. Enuncie las ventajas y desventajas de la Ascensión de Colinas

Inicialmente el espacio de soluciones de un problema puede ser representado en una gráfica en R2, R3, hasta Rn por lo que podemos explorar su espacio mediante una heurística la cual es generalmente el Ascenso de Colinas el cual proporciona las siguientes ventajas y desventajas:

|  |  |
| --- | --- |
| **Ventajas** | **Desventajas** |
| Permite realizar un número menor de iteraciones con respecto a la exploración del espacio en modalidad de fuerza bruta | El resultado puede quedarse en un mínimo o máximo local sin que este sea el mínimo o máximo global pudiendo situarse muy lejos de la solución satisfactoria |
| Permite moverse en el espacio de diferentes formas, permitiendo que encuentre un mínimo | Dependiendo de la implementación del algoritmo este puede caer en una meseta generando una caminata aleatoria sin llegar a una solución |
| Dependiendo de la implementación puede que tenga un número distinto de iteraciones | En caso de ser un espacio muy grande puede que la implementación, salto o movimiento entre el espacio demore demasiado |
| Es un bucle que se mueve continuamente en dirección del valor creciente | La posibilidad de encontrar una solución en un espacio de búsqueda reducido podría llevar a encontrar cretas cuyos laterales sean demasiado inclinados. |
| Es un algoritmo que no mantiene un árbol de búsqueda, por lo que no necesita registrar el estado de un valor | La búsqueda podría llegar a oscilar de un lateral a otro con pocos avances, a menos que existan operadores que faciliten el movimiento a la parte superior de la cresta. |
| Este puede escoger de forma aleatoria un sucesor si existe más de una mejor solución |  |

2. Detalle el pseudocódigo del algoritmo Steepest-ascent hill-climbing (SAHC)

Descripción del algoritmo:

1. Escogemos una cadena aleatoriamente. Llamamos a esta cadena como la mejor solución.

2. Sistemáticamente mutamos cada bit en la cadena de izquierda a derecha, guardando la función objetivo de las cadenas resultantes.

3. Si alguna de las cadenas resultantes da un incremento en la función objetivo, entonces guardamos como la mejor solución a la cadena resultante dada por el más alto incremento de la función objetivo

4. Si no hay incremento de la función objetivo, entonces guardamos la actual mejor solución y volvemos al paso 1. De lo contrario, vamos al paso 2 con la nueva mejor solución.

5. Cuando se haya alcanzado el número establecido de iteraciones de la función, se devuelve la mejor solución obtenida.

Pseudocódigo del algoritmo:

1) Establecemos el número **n** de Iteraciones.

2) Generamos una cadena aleatoriamente.

**3)** **Inicio.**

**4) For i =1 to n.**

5) De forma sistemática mutamos a cada individuo de la cadena de izquierda a derecha.

6) Se evalúa la FO de la cadena resultante.

7) **Si** alguno de los resultados incrementa la FO.

8) Se establece la cadena como la mejor solución.

9) **Si** **no** hay incremento en la FO

10) Se guarda la cadena actual como la mejor solución.

11) Generamos nuevamente una cadena aleatoriamente.

**12) End if.**

**13) End for.**

14) Cuando el número de evaluaciones sea alcanzado se retorna la mejor solución.

15) Fin.

Ejemplo práctico.

/\* Selecciona una cadena aleatoria \*/

current\_hiltop=rand(String)

i=0

boolFound=false

while i<mutationTotal(current\_hiltop) && !boolFound{

/\* Muta de izquierda a derecha cada bit de la cadena \*/

c1=mutation(current\_hiltop,i)

/\* Verifica cual de las dos cadenas tiene mejor valor\*/

if fitness(current\_hiltop)<fitness(c1){

current\_hiltop=c1

boolFound=true

}

i++;

}

/\* Verifica si existe un mejor fitness y si es asi va al paso 2\*/

if(boolFound){

goto 2

}else{ /\* En caso contrario guarda la cadena y regresa a 1 \*/

save(current\_hiltop)

goto 1

}

return save /\* Retorna la solución \*/

3. Detalle el pseudocódigo del algoritmo Next-ascent hill-climbing (NAHC)

Descripción del algoritmo:

1. Genera una cadena de forma aleatoria. Nombramos la cadena como la mejor solución.

2. Mutamos cada bit de la cadena de izquierda a derecha, evaluamos el resultado de la función objetivo. Si se encuentra un incremento en la función objetivo, entonces se establece como mejor solución la cadena actual sin evaluar ningún bit de la solución original. Con la nueva mejor solución encontrada hasta el momento continuamos mutando la cadena en la última posición conocida donde la función objetivo presentó un incremento.

3. Si la función objetivo no supera a la mejor solución, guardamos la mejor solución y repetimos el paso 1.

4. Cuando el número de evaluaciones sea alcanzado. Regresamos la mejor solución encontrada.

Pseudocódigo del algoritmo:

1) Establecemos el número **n** de Iteraciones.

2) Generamos una cadena aleatoriamente.

3) Establecemos la constante de mutación.

**4)** **Inicio.**

**5)** **For i =1 to n.**

6) Escogemos un individuo de izquierda a derecha.

7) Aplicamos la mutación por cada bit y evaluamos la función objetivo.

**8) Si** al mutar un bit incrementa la FO

9) Se guarda la solución actual como mejor solución.

10) Se continúa mutando desde el último bit evaluado.

**11) Si no** hay incremento en la FO

12) Se establece la cadena como la mejor solución.

13) Generamos nuevamente una cadena aleatoriamente.

**14) End if.**

**15) End for.**

16) Cuando el número de evaluaciones sea alcanzado se retorna la mejor solución.

**17)** **Fin.**

4. Detalle el pseudocódigo del algoritmo Random mutation hill-climbing (RMHC)

Descripción del algoritmo:

1. Genera una cadena de forma aleatoria. Nombramos la cadena como la mejor evaluada.

2. Escogemos un índice aleatorio para mutación. Si la mutación es igual o mayor que la mejor solución evaluada, se establece como la solución evaluada a la cadena resultante.

3. Regresar al paso 2

4. Cuando el número de evaluaciones sea alcanzado. Regresamos la mejor solución encontrada.

Pseudocódigo del algoritmo:

1) Establecemos el número **n** de Iteraciones.

2) Generamos una cadena aleatoriamente.

3) Establecemos la constante de mutación.

**4)** **Inicio.**

**5)** **For i =1 to n.**

6) Escogemos un individuo de forma aleatoria y se aplica la mutación.

7) Se evalúa el individuo seleccionado.

8) **Si** el individuo => Mejor solución.

9) Se actualiza la mejor solución.

10) **End if.**

**11)**  **End for.**

12) Cuando el número de evaluaciones sea alcanzado se retorna la mejor solución.

**13)** **Fin.**

5. Mencione aplicaciones de los algoritmos de Ascensión de Colinas

· El problema de las 8 Reinas

· Juego 8 Puzzle

· Ajedrez.

· Backgammon

· Problemas de optimización

· Recorrido de grafos

· Algoritmos genéticos

· Recorridos simulados.

6. Realice la modelación matemática necesaria para la solución, mediante RMHC, de los problemas siguientes:

a. Problema de la mochila (*Knapsack problem*)

Donde:

es un objeto que puede almacenar en la mochila

es una función booleana la cual nos dice si está en la solución

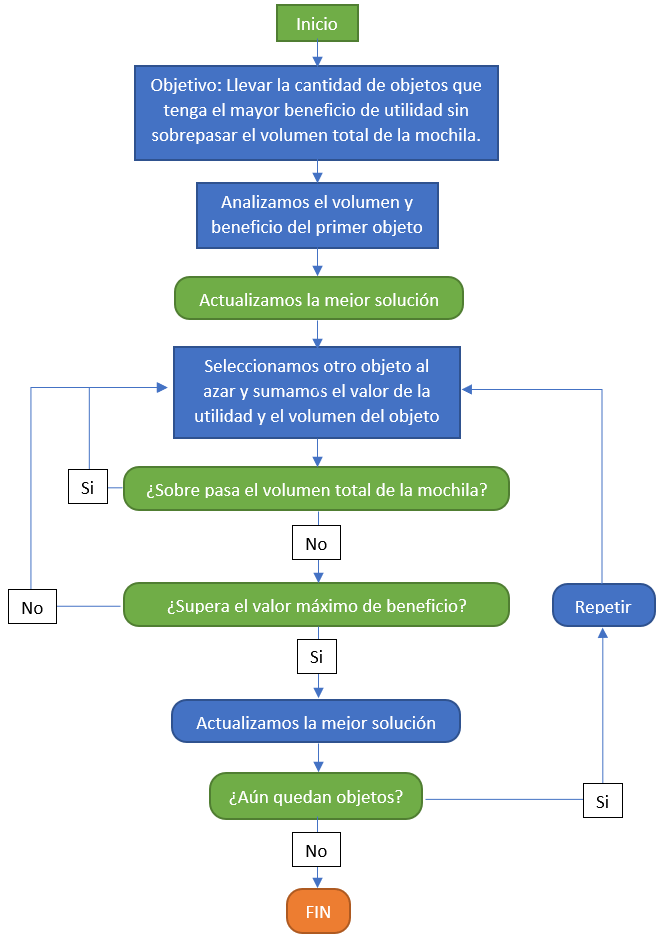
es una función la cual devuelve el valor de beneficio por el objeto

De esta forma igual se cubre una segunda función la cual es:

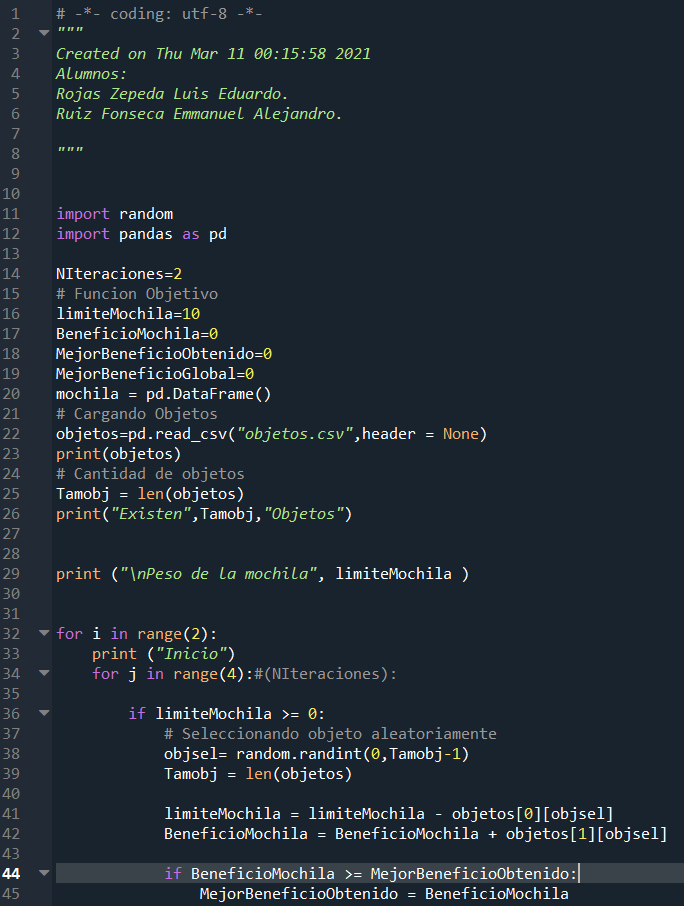
Donde:

es una función la cual devuelve el peso que añade a la mochila por cada objeto

De tal forma que la suma de pesos para resolver este problema debe de ser menor o igual al peso máximo que tiene la mochila de capacidad.



7. Dado el Problema de la Mochila, proponga las estructuras de datos necesarias para su implementación.



8. Diseñe la interfaz de usuario para la solución del problema planteado.