

Actividad No. 4 Corrección

Solución de problemas mediante Ascensión de Colinas

Laboratorio 1

Adrian González Pardo

13 de octubre de 2020

1. Ventajas y desventajas de la Ascensión de Colinas

Inicialmente el espacio de soluciones de un problema puede ser representado en una grafica en R^2 , R^3 , hasta R^n por lo que podemos explorar su espacio mediante una heuristica 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 minimo local sin que sea el minimo del espacio
Permite moverse en el espacio de diferentes formas, permitiendo que encuentre un minimo	Dependiendo de la implementación del algoritmo puede que que llegue al minimo o no, pero al final arroja 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

2. Pseudocódigo del Algoritmo Steepest Ascent Hill Climbing (Ascenso Más Empinado Escalada de Colinas)(SAHC)

```
1- current_hiltop=rand(String) /* Selecciona una cadena aleatoria */
2 i=0
3 boolFound=false
4 2.- while i<mutationTotal(current_hiltop) && !boolFound{
5     c1=mutation(current_hiltop,i) /* Muta de izquierda a deracha cada bit de la cadena */
6     3.-if fitness(current_hiltop)<fitness(c1){ /* Verifica cual de las dos cadenas tiene mejor valor
7         */
8         current_hiltop=c1
9         boolFound=true
10    }
11    i++;
12 }
13 4.- if(boolFound){ /* Verifica si existe un mejor fitness y si es asi va al paso 2*/
14     goto 2
15 }else{ /* En caso contrario guarda la cadena y regresa a 1 */
16     save(current_hiltop)
17     goto 1
18 }
19 5.- return save /* Retorna la solucion */
```

3. Next-Ascent Hill-Climbing (Próxima Ascenso Escalado)(NAHC)

```

1 1- current_hiltop=rand(String) /* Selecciona una cadena aleatoria */
2 i=0
3 boolFound=false
4 2.- while i<mutationTotal(current_hiltop){ /* Muto la cadena */
5     c1=mutation(current_hiltop,i)
6     if fitness(current_hiltop)<fitness(c1){ /* Verifico cual de las dos cadenas tiene mejor fitness
7         */
8         current_hiltop=c1
9         boolFound=true
10    }
11    i++
12 }
13 3.- if boolFound{ /* Verifico si la colina que verifique inicialmente no encontro mejor fitness*/
14     save(current_hiltop)
15     goto 1
16 }
17
18 return save

```

4. Random-Mutation Hill-Climbing (Ascenso Por Mutación Aleatoria)(RMHC)

```

1
2 1.- best_evaluated=random(String) /* Selecciona una cadena aleatoria */
3 /* Ejemplo una cadena de bits
4     5 productos ()
5     10100 -> Verificar que sea valia con respecto al peso maximo
6     1.1 Calcular f_best <- Calculo de beneficio
7
8
9 */
10 i=0
11 3.-while(i<number_evaluated_lim){ /* Vuelve a paso 2 */
12     2.- locus=random(best_evaluated) /* Mutacion random para locus que va de 0 a best_evaluated-1
13         escoger 1 bit */
14     new_evaluated=mutation(best_evaluated,locus) /* si x cambia a ~x*/
15     /* Verifico si new_evaluated es valido */
16     if fitness(best_evaluated)<=fitness(new_evaluated)
17         best_evaluated=new_evaluated
18     i++
19 }
20 4.- return best_evaluated /* Retorna la cadena aleatoria mejor evaluada*/

```

5. Resultados de Forrest y Mitchell

Si bien en sus conclusiones nos podemos dar cuenta en cuestiones de los algoritmos heurísticos SAHC y NAHC en la búsqueda de soluciones de un espacio bidimensional no encontraron la zona óptima, el algoritmo RMHC lo encontró en un número menor de iteraciones con respecto a los algoritmos genéticos que se implementaron en la investigación, esto pasa en relación a que el algoritmo RMHC explora el espacio de soluciones en este caso digamos o aproximemos en grafos es más sencillo tomar la decisión sobre que nodo iterar sin realizar tantas comparaciones.

6. Aplicaciones de Algoritmos de Ascenso de Colinas

Si bien en muchas ocasiones estos algoritmos se ven como una caja negra a la hora de ser utilizados en muchos módulos o toolbox de lenguajes de programación estos están íntimamente aplicados en el cálculo de gradiente descendente de N variables para optimizar un aprendizaje supervisado o no supervisado, donde sobre cada iteración se realiza el cálculo de un nuevo movimiento en el espacio de soluciones, por otro lado también este tipo de algoritmo está implementado en algunas aplicaciones en las que se realiza la planificación de rutas de transporte e incluso en sistemas de reconocimiento de objetos 3D, una vez más retomando los términos del Aprendizaje de Máquina podemos encontrar desde regresión lineal, polinomial y logística.

7. Aplicación de RMHC en distintos problemas con su estructura de datos

7.1. Knapsack Problem

$$f(x) = \sum_{i=1}^N h(g(x_i))$$

Donde:

x_i es un objeto que puede almacenarse en la mochila

$g(x)$ es una funcion booleana la cual nos dice si x_i esta en la solucion

$h(x)$ es una funcion la cual devuelve el valor de beneficio por el objeto x_i

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

$$p(x) = \sum_{i=1}^N m(g(x_i)) \leq \text{Peso_maximo}$$

Donde:

$m(x)$ es una funcion la cual devuelve el peso que añade a la mochila por cada objeto x_i

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

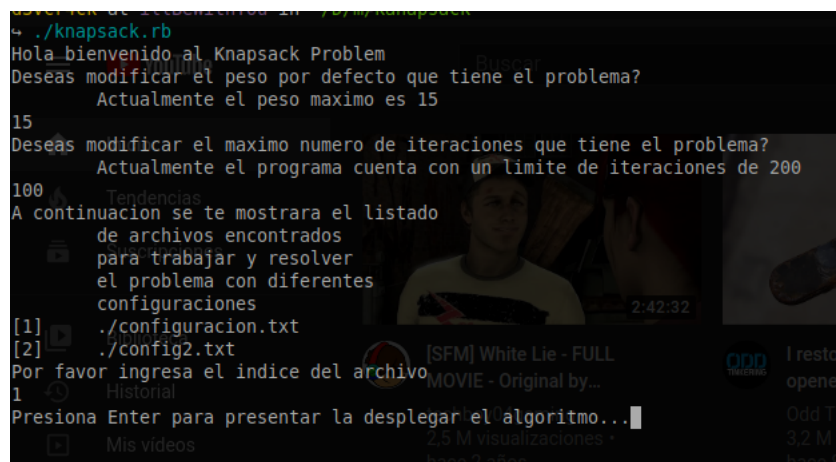
Los estados iniciales de este problema son:

- El conjunto de objetos que pueden pertenecer a la mochila
- Una solución inicial la cual es valida para el peso maximo de la solucion
- Segun el algoritmo de RMHC el best_solution y new_best es una cadena de bits las cuales si estan en 1 significara que el objeto en la mochila de acuerdo a la posición de cada 1 y 0
- f_best y f_new son los valores de beneficio que pueden ser agregados a la nueva solucion de acuerdo a la modificación de objetos en el paso de mutación.

En el diseño de la solución se solicita los siguientes datos:

- Peso máximo de la mochila
- Limite de iteraciones
- Indice de la configuración del archivo el cual contiene la estructura de (peso,beneficio)

Ejemplo:



```
4 ./knapsack.rb
Hola bienvenido al Knapsack Problem
Deseas modificar el peso por defecto que tiene el problema?
    Actualmente el peso maximo es 15
15
Deseas modificar el maximo numero de iteraciones que tiene el problema?
    Actualmente el programa cuenta con un limite de iteraciones de 200
100
A continuacion se te mostrara el listado
de archivos encontrados
para trabajar y resolver
el problema con diferentes
configuraciones
[1] ./configuracion.txt
[2] ./config2.txt
Por favor ingresa el indice del archivo
1
Presiona Enter para presentar la desplegar el algoritmo...
```

Figura 1 Primera parte del programa

```
Muestra de objetos de la mochila
Objeto 1: [ ] Peso 1, Beneficio 45
Objeto 2: [ ] Peso 3, Beneficio 26
Objeto 3: [ ] Peso 4, Beneficio 1
Objeto 4: [ ] Peso 4, Beneficio 10
Objeto 5: [ ] Peso 4, Beneficio 30
Objeto 6: [ ] Peso 2, Beneficio 30
Objeto 7: [ ] Peso 1, Beneficio 40
Solucion inicial
Objeto 1: [✓] Peso 1, Beneficio 45
Objeto 2: [✓] Peso 3, Beneficio 26
Peso total: 4
Con beneficio: 71
Solucion encontrada
Objeto 1: [✓] Peso 1, Beneficio 45
Objeto 2: [✓] Peso 3, Beneficio 26
Objeto 3: [✓] Peso 4, Beneficio 1
Peso total: 8
Con beneficio: 72
En la iteracion 5
```

Figura 2 Parte de la ejecución a partir de la implementación

7.2. Travel Salesman Problem

La función que buscamos es la minimización de costo total del problema representado como:

$$f(x) = \sum_{i=1}^N g(h(x_i))$$

Donde:

x_i es una arista de un vertice v_j que va a v_k

$h(x)$ es una funcion que considera o no el trayecto v_j a v_k de x_i

$g(x)$ es una funcion que devuelve el valor de costo de x_i

Los estados de este problema son los siguientes:

- Una solución random valida de recorrido v_i hasta v_j de tal forma que entre desos dos existen al menos N vertices que sumados en sus costos nos dan una respuesta.
- Limite de iteraciones
- Indice de la configuración de formato ($vertice_i$, $costo$, $vertice_j$)

7.3. Obtención de minimos de la función f(x)

Función:

$$f(x) = \sum_{i=1}^D x_i^2 \quad \text{con} \quad x_i \in [-10, 10]$$

Para el calculo de minimos es necesario buscar inicialmente una solución random de M puntos en los cuales no se garantiza que en algun x_i sea igual a cero, por lo que es necesario apartir de esta respuesta se puede comenzar a modificar para que exista un valor aproximado de M puntos minimos de acuerdo al maximo numero de iteraciones.

Los estados de este problema son:

- Solución random de puntos (x_1, x_2, \dots, x_D) de tal forma en que no todos los puntos son minimos, por lo cual consideraremos un mínimo de la función cuando algún $x_i = 0$.
- Limite de iteraciones
- Lista de puntos a la solución random

$$I_{Im} = gImu^5 z(V - EIm)$$