Actividad No. 3
Solución de problemas mediante Ascensión de Colinas

Adrian González Pardo

7 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 grafíca en R^2, R^3, hasta R^n por lo que podemos explorar su espacio mediante una heuristica la cual es generalmente el Ascensio 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 */
i=0
boolFound=false
2.- while i<mutationTotal(current_hiltop) && !boolFound{
  c1=mutation(current_hiltop,i) /* Muta de izquierda a deracha cada bit de la cadena */
  3.-if fitness(current_hiltop)<fitness(c1){ /* Verifica cual de las dos cadenas tiene mejor valor
    */
    current_hiltop=c1
    boolFound=true
  }
  i++;
}
4.- if(boolFound){ /* Verifica si existe un mejor fitness y si es asi va al paso 2*/
  goto 2
}else{ /* En caso contrario guarda la cadena y regresa a 1 */
  save(current_hiltop)
  goto 1
}
5.- return save /* Retorna la solucion */
```

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

```
1- current_hiltop=rand(String) /* Selecciona una cadena aleatoria */
i=0
boolFound=false
2.- while i<mutationTotal(current_hiltop){ /* Muto la cadena */
  c1=mutation(current_hiltop,i)
  if fitness(current_hiltop)<fitness(c1){ /* Verifico cual de las dos cadenas tiene mejor fitness
    */
    current_hiltop=c1
    boolFound=true
  }
  i++
}

3.- if boolFound{ /* Verifico si la colina que verifique inicialmente no encontro mejor fitness*/
  save(current_hiltop)
  goto 1
}

return save
```

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

```

1.- best_evaluated=random(String) /* Selecciona una cadena aleatoria */
i=0
3.-while(i<number_evaluated_lim){ /* Vuelve a paso 2 */
  2.- locus=random(best_evaluated) /* Mutacion random para locus que va de 0 a best_evaluated-1 */
  new_evaluated=mutation(best_evaluated,locus)
  if fitness(best_evaluated)<=fitness(new_evaluated)
    best_evaluated=new_evaluated
  i++
}
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 cuestios de los algoritmos heuristicos SAHC y NAHC en la busqueda de soluciones de un espacio bidimensional no encontraron la zona optima, el algoritmo RMHC lo encontro en un número menor de iteraciones con respecto a los algoritmos geneticos 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 modulos o toolbox de lenguajes de programación estos estan intimamente 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 calculo de un nuevo movimiento en el espacio de soluciones, por otro lado tambien este tipo de algoritmo esta 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 terminos del Aprendizaje de Maquina podemos encontrar desde regresión lineal, polinomial y logistica.

7. Aplicación de RMHC en distintos problemas

7.1. Knapsack Problem

Matemáticamente sabemos que el Knapsack Problem es un problema en el cual se busca una funcion $f(x_0, x_1, \cdots, x_n)$ la cual busca maximizar la utilidad de la solución al problema, es decir, $max\{f((x_0, x_1, \cdots, x_n)\}$ por lo que los valores iniciales de este problema son max_peso que es el peso maximo que soporta la mochila, *iterator* que es un

valor numerico el cual permite llevar el control del número de iteraciones en el algoritmo, por otro lado podemos llamar a nuestro estado inicial *objeto* el cual es seleccionado de nuestro conjunto o lista de objetos *objetos* los cuales al pasar por la otra función $g(x_i)$ el cual devuelve el peso del objeto en la mochila, $h(x_i)$ la cual devuelve el beneficio del objeto al ser incluido en la mochila, por otro lado para cada x_i corresponde al objeto que pertenece a la lista de objetos, entonces al momento de seleccionar un valor x_i aleatorio podemos guardarlo en una lista de soluciones $sol_knapsack$ el cual solo se le ingresara si en cada iteración el $objeto[x_i] > objeto[x_j]$ para asi no desperdiciar memoria en la implementación cuidando que si se busca agregar alguna de las dos opciones se debe considerar el limite en peso de la mochila, por lo tanto debemos evaluar que si el peso contenido de la mochila más el peso del objeto no sobrepasan al peso maximo de la mochila.

7.2. Travel Salesman Problem

En este problema podemos pensar en que nuestro espacio va a ser generado a traves de los vertices vecinos de nuestro nodo inicial seleccionado aleatoriamente, por ello la funcion a solucionar es la minimización de $f((x_0, x_1, \cdots, x_n)$ donde cada x_i es el nodo del grafo y con la existencia de que cada nodo tiene vecinos representados de la forma xi, j, con función $g(x_{i,j})$ donde esta devuelve el costo de trasladarse del nodo .ª.ªl nodo "b", por lo cual en este algoritmo podremos comparar primeramente el costo del nodo al que .ª"se compara con el costo de "b"por lo que después se añade aun conjunto de soluciones al final de que la iteración y la comparación fue realizada exitosamente.

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

Función:

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

Si bien sabemos que en esta función el encontrar minimos locales de cada x_i se ve cuando la función en ese $x_i = 0$ por lo cual esa es una ayuda pero el problema radica en el aumentar en D dimensiones, por lo cual la entrada es el especificar la D dimensiones en la función, después se determinara un numero maximo de iteraciones y de minimos locales, por lo cual seleccionaremos un x_i aleatoriamente y apartir de el afectaremos a un segundo x_j por la mutación aleatoria del primer valor comparando cual de esos dos valores es mejor evaluar debido al acceso más sencillo en memoria, después de ello asignaremos cualquier valor entre $[-10, 10]$ a las demas variables para que ahi podamos asignar un punto en todas las coordenadas y este lo añadiremos a la lista de soluciones $lista_espacio_D$ en la cual podremos mostrar al final la cantidad de valores minimos que obtengamos del programa