Computó evolutivo

Análisis del algoritmo

Profesor: Carlos Ignacio Hérnandez Castellanos

El algoritmo que seleccioné fue el de recocido simulado.

1. Completitud

El algoritmo de recocido simulado, dada la idea de ampliar o disminuir la vecindad de un punto, que aplicado al problema de la mochila estaríamos hablando de que dada una solución inicial busca las combinaciones de los elementos que incluye y algunos de los que no para poder así generar una solución un poco mejor y como he dicho, inicia con una solución que quizás no es la más óptima pero cumple con los prerrequisitos del problema lo que nos indica que siempre nos da una solución aunque no sea la correcta

2. optimalidad

En este aspecto el algoritmo de recocido simulado al igual que algunos algoritmos de optimización por erísticas se aproximan a una solución sin embargo no siempre se alcanza el óptimo global debido a que existe un factor probabilístico, que en este caso es el poder partir en una solución que esté cercana a la solución más óptima de todas, y como la elección de la solución inicial es elegida aleatoriamente no podremos asegurar que siempre se llegue a la mejor solución

3. complejidad en tiempo

Para este análisis nos enfocaremos únicamente en los métodos que se encuentran en el archivo *recocido_simulado.py* dado que en otro solo encontraremos funciones como la lectura del archivo o de la impresión en terminal de los resultados y por ende solo aumentan en $\mathbf{O(n)}$ siendo n el tamaño de elementos en el archivo y en el arreglo resultante.
Dicho esto comencemos con la función *solucion_inicial*, en las líneas 19 y 20 observamos que son en tiempo constante, dado que solo son una asignación de variable y un cálculo respectivamente, después tenemos que se crea una lista por comprensión la cual requiere de un for que va de 0 a la longitud de elementos de la mochila por lo que podemos concluir que construir que la construcción de la lista índices es en tiempo lineal respecto al numero de elementos que podemos tomar o no. después nos encontramos con un while el cual tienen como condición ejecutarse mientras la capacidad de la mochila sea positiva, por lo que en el peor de los casos que sería en el que reduzcamos en una unidad la capacidad de la mochila y por tanto nos tomaríamos el mismo número de pasos que la capacidad de la mochila. Dentro del while nos encontramos con operaciones de tiempo constantes como operaciones aritméticas, comparaciones y asignaciones, así como la eliminación del último elemento de la lista, y la llamada a la función *cost_peso_actual* el cual nos devuelve la suma de los pesos y del valor dada una combinación de elementos seleccionados para la meter a la mochila y si observamos en el peor caso que es tener una combinación de elementos cuya cardinalidad sea igual a la capacidad, considerando ese caso observemos que en la primer iteración la lista está vacía y por tanto hicimos 0 pasos en contar la suma de los pesos de la combinación, en la segunda iteración tendremos un elemento y por ende tardaríamos un paso en obtener la suma de los pesos, en la tercera iteración tardaríamos 2 pasos, y así sucesivamente hasta llegar a la n+1 iteraciones donde tardaríamos n pasos para obtener la suma total de pesos en el cual se alcanzará la condición para terminar con el while y por ende con la ejecución de la función; por tanto podremos concluir que la complejidad total de la función es $\mathbf{O}(n^2)$ en el peor caso.
Continuamos con la función *simulacion* en la cual se ejecuta el algoritmo de recocido simulado, comenzamos con un par de asignaciones en tiempo constante, después el recorrido en tiempo lineal respecto al tamaño de la lista que almacena la solución inicial. Posteriormente tenemos un ciclo while el cual se detendrá hasta que la temperatura sea negativa ó ya no encontremos una solución mejor. Es un tanto complicado determinar de forma exacta cuantas iteraciones se necesitan para terminar así que continuemos con el resto del código y después regresemos para intentar aproximar u obtener el peor caso.
dentro del while tendremos una operación lineal para obtener la suma de los valores de la solución actual, posteriormente tendremos un for que se ejecuta tantas veces como le indiquemos a través de la variable iteraciones, dentro del for nos encontramos con la llamada a otra función llamada $genera_combinacion$ el cual realiza una búsqueda en el vecindario de nuestra solución inicial alternando los elementos al seleccionarlos o

no, el cual toma $n^2 + n$ pasos en el peor de los casos dado que recorre elemento por elemento la lista de la solución inicial y crea nuevas listas cuidando de no exceder la capacidad total de la mochila. Posteriormente nos encontramos con una operación constante y otra lineal (la tan mencionada suma de los valores) y posteriormente operaciones constantes. Observemos que dentro del while, en el peor caso, se llevan a cabo $n^2 + n + n + n + c$ operaciones, donde n es la longitud de la lista de la solución actual y c es una constante donde se incluyen las operaciones no mencionadas. De aquí tendremos que solo dentro del while tendremos una complejidad $\mathbf{O}(n^2)$ dado que en notación O-grande se desprecian los términos inferiores y las constantes. Ahora podemos darnos una idea de en que momento podría el while detenerse y es que obtengamos una mejor solución o tener una temperatura negativa, para la primer condición podemos ver que en el peor caso habremos recorrido todas las combinaciones de la solución actual y siendo la última la mejor de todas, si tomamos en cuenta que en el peor caso la solución actual tiene todos los elementos entonces nos tardaremos n pasos, para la segunda condición dado que se disminuye en 0.9 la temperatura entonces nos tardaremos $t-1$ pasos siendo t la temperatura inicial. Por tanto podemos concluir que en el pero de los casos nos tardaremos $n(n^2) = n^3$ pasos. Dado que la notación O-grande desprecia términos inferiores y constantes podremos decir que la complejidad total es $\mathbf{O}(n^3)$

4. complejidad en espacio

Para esto volveremos a analizar función por función del archivo $recocido_simulado$
Para la función $solucion_inicial$ requerimos de 4 constantes y una lista llamada índices del mismo tamaño que la totalidad de elementos así que aquí tendremos, en el peor caso, es **O(n)** donde n es la cantidad de elementos que podemos o no meter a la mochila.
En la función $recocido_algoritmo$ requerimos de dos listas, una que almacena la primer solución que como vimos es $O(n)$ y la que almacena la solución local, que en el peor caso contendrá todos los elementos, en el cual también tendremos una complejidad en espacio de $O(n)$.
En la función *simulacion* requerimos de una lista con la solución inicial y configuraciones el cual contiene todas aquellas combinaciones de elementos de la solución inicial, que en el peor caso, la solución inicial contiene todos los elementos recibidos, entonces para la lista configuraciones tendrá la siguiente estructura:

$$[[1], [1, 2], ..., [1, ..., n]]$$

que tendrá un total de $n * n - 1/2$ elementos lo que al final tendrá una complejidad $O(n^2)$ en espacio el cual es la misma complejidad en espacio siguiendo la idea de O-grande donde despreciamos términos inferiores y constantes

5. resultados

Para el archivo ks_4_0 el resultado fue:
— Valor — Peso —
————+————
— 10 — 5 —
— 4 — 3 —
Para el archivo ks_50_1 el resultado fue:
— Valor — Peso —
————+————
— 279 — 281 —
— 961 — 938 —
— 757 — 718 —
— 838 — 767 —
— 895 — 957 —
— 262 — 279 —
— 779 — 758 —
— 196 — 182 —

— 26 — 28 —
Para el archivo $ks_1 0000_0$ el resultado fue:
— Valor — Peso —
————————+————
— 160709 — 158671 —
— 140581 — 146236 —
— 81 — 81 —
— 45979 — 44522 —
— 3130 — 2985 —
— 250 — 256 —
— 117010 — 120061 —
— 67635 — 65208 —
— 103747 — 103340 —
— 125689 — 138805 —
— 164490 — 169518 —
— 22 — 21 —
— 2895 — 2695 —
— 2586 — 2439 —
— 1761 — 1635 —
— 4303 — 4378 —
— 22369 — 22649 —
— 15710 — 16499 —