MARP - Práctica 2

Diego Atance Sanz

24 de abril de 2020

Índice

1. Detalles de implementación **2**
1.1. Ficheros principales 2

2. Casos de prueba y análisis de la complejidad **2**
2.1. Casos de prueba 2
2.2. Análisis de la complejidad 2

3. Conclusiones **3**
3.1. Figuras 4

1. Detalles de implementación

Se ha implementado el problema de la mochila con dos soluciones diferentes:

- Solución trivial: *Backtracking* sin poda, solo apto para problemas de tamaño muy reducido.
- Solución eficiente: Problema de ramificación y poda, usando la poda que trata el problema considerando el caso donde los objetos son fraccionables y se puede aplicar programación voraz.

1.1. Ficheros principales

1. **custom_classes.h:** Contiene las clases necesarias para la correcta ejecución del algoritmo principal.
   - **Node:** Nodo de del árbol de exploración.
   - **MyPair:** Clase auxiliar con el único objetivo de permitir la ordenacion de los elementos en la cola de prioridad de máximos en función de su división (valor/peso).
2. **backpack.cpp** Archivo que implementa los siguientes algoritmos
   - **main:** Recibe por parámetro el número de elementos que contendrá la mochila y el modo de resolución. Posteriormente genera el conjunto de datos y ejecuta la búsqueda correspondiente.
   - **create_items:** Se encarga de generar el conjunto de datos sobre el que se trabajará posteriormente.
   - **print_sol:** Se encagarga, en caso de requerirse, de imprimir la solución obtenida por el algoritmo.
   - **calc_estimate:** Función de poda para el algoritmo de ramificación y poda. Trata el problema como vimos en el tema de búsqueda voraz, asumiendo que los objetos son fraccionables y llegan ordenados de mayor a menor relación valor/peso.
   - **backpack_goodopoda:** Algoritmo de ramificación y poda óptimo, recibe un conjunto de datos y calcula la solución del problema. Requiere que el conjunto de datos venga ordenado de mayor a menor respecto a su relación valor/peso.
   - **recursive_backpack:** Implementa la función recursiva en la que se basa la solución trivial.
   - **backpack_nopoda:** Configura la búsqueda no óptima y lanza la función recursiva.

2. Casos de prueba y análisis de la complejidad

2.1. Casos de prueba

Para la comprobación del algoritmo y su eficiencia, se ha optado por la implementación de un generador de casos de prueba que asigna pesos y valores a los N objetos que se soliciten para la mochila.
El peso máximo a introducir se ha decidido asignarlo siempre a un 45 % del total del peso de los objetos de la mochila para intentar así obtener unos resultado homogéneos y evitar que en algunas mochilas entrasen el total de los objetos y en otras se tuviese que devolver una mochila vacía.

2.2. Análisis de la complejidad

Se presentan dos casos a estudiar y comparar, ambos obtienen la solución óptima usando diferentes aproximaciones:

1. **Caso trivial** Con coste exponencial, la única poda usada para esta parte del programa se trata de observar si existen suficientes objetos sin seleccionar para la mochila los cuales permitan mejorar la solución ya encontrada. Esto conlleva la exploración de un número elevadísimo de nodos, pese a que de manera individual sea extremadamente rápida (avg. 300 - 600 nodos por ms), a mayor número de nodos, menos número de nodos por ms es posible explorar. La solución más voluminosa que se ha podido calcular usando este algoritmo es de 25 elementos en la mochila, conllevando la exploración de casi 28 millones de nodos, a 250 - 300 nodos por ms.

2. **Ramificación y poda** Se trata de la solución eficiente y explicada en las clases de teoría. Su función de poda aborda el problema como si se enfrentase al problema de la mochila de objetos fraccionables y calcula para cada nodo del arbol cuál es su objetivo prometedor. Este cálculo conlleva mucho más tiempo que el de la solución trivial , pero rebaja de manera muy notoria el número de nodos explorados. La solución de mayor volumen calculable por este algoritmo supera los 10000 elementos en la mochila, visitando tan sólo 6500 nodos y explorando aproximadamente 1 nodo por ms.

(Ver Figuras 1 y 2)

3. Conclusiones

Como se puede observar fácilmente en las gráficas aportadas, la complejidad (y capacidad de cálculo) de los algoritmos no es comparable. En el caso del algoritmo trivial, la complejidad es claramente exponencial, creciendo de forma exagerada a partir de los 20 elementos y resultando incalculable.
En el caso del algoritmo de ramificación y poda eficiente, podemos observar una complejidad lineal para los datos analizados. En un segundo análisis mas en profundidad, adjunto en el apartado de "extras", se aprecia una tendencia exponencial en la complejidad del problema, sin llegar a suponer un problema hasta alcanzar los 2000 elementos en la mochila. (Ver Figura 3).
Gracias a estos datos y teniendo en wcuenta que los casos de prueba han sido ejecutados en un portátil de trabajo personal (que pese a estar bajo una carga mínima no aporta mucha potencia de cálculo) podemos confirmar la eficiencia absoluta de las funciones de poda propuestas y observar la gran diferencia en tiempos y capacidades de cómputo que estas respesentan.

3.1. Figuras

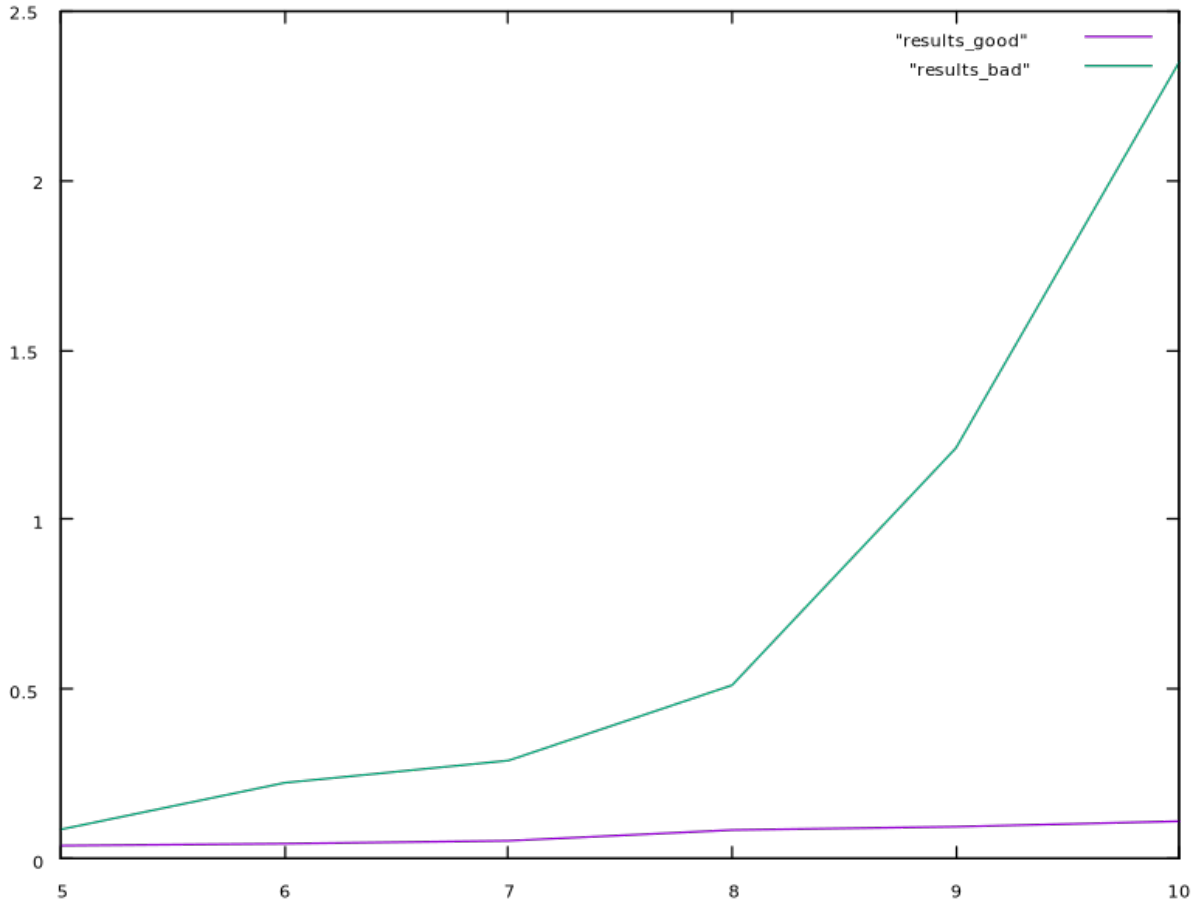

Figura 1: Comparativa entre los tiempos de ejecución para los casos de prueba entre 5 y 10 elementos de la mochila. Y - ms / X - Nodos

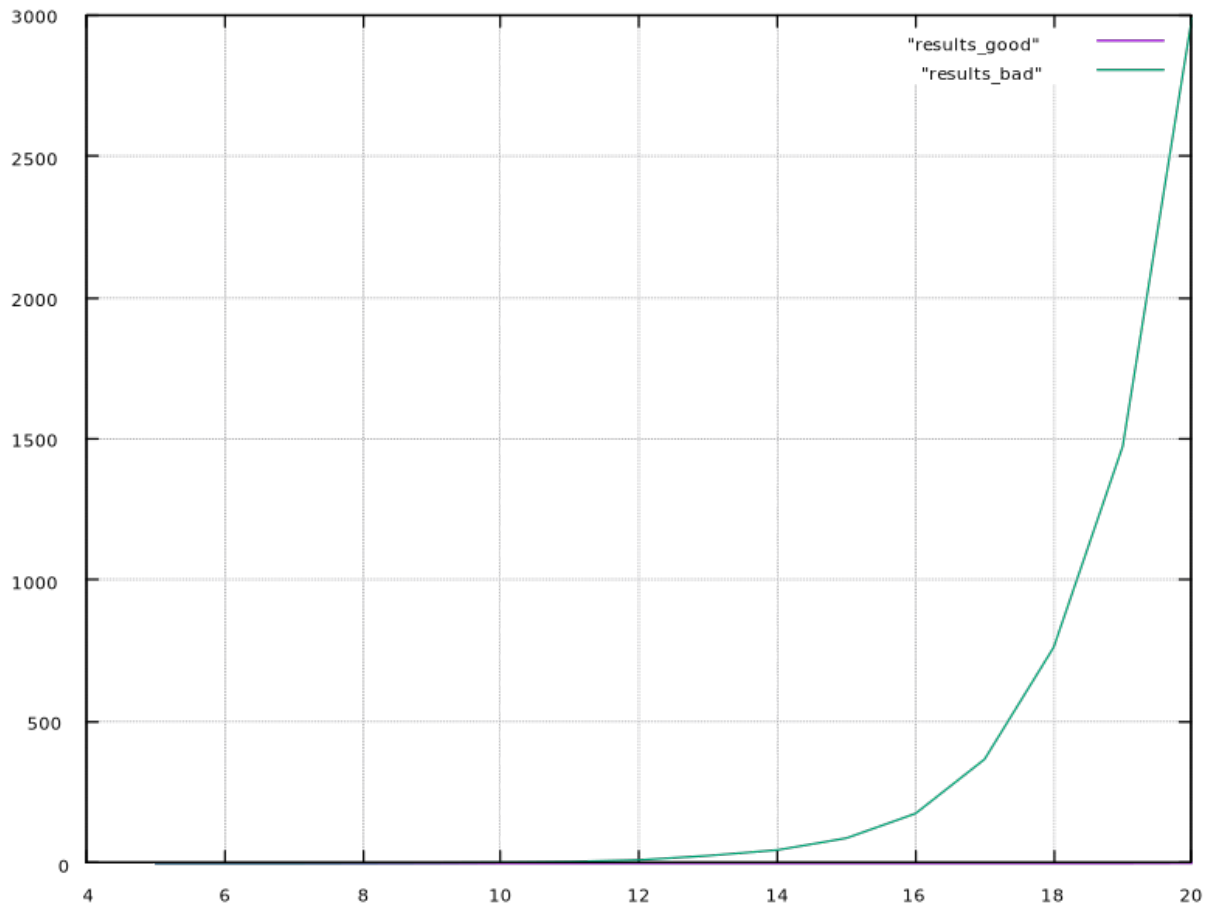

Figura 2: Comparativa entre los tiempos de ejecución para los casos de prueba entre 5 y 20 elementos de la mochila. Y - ms / X - Nodos

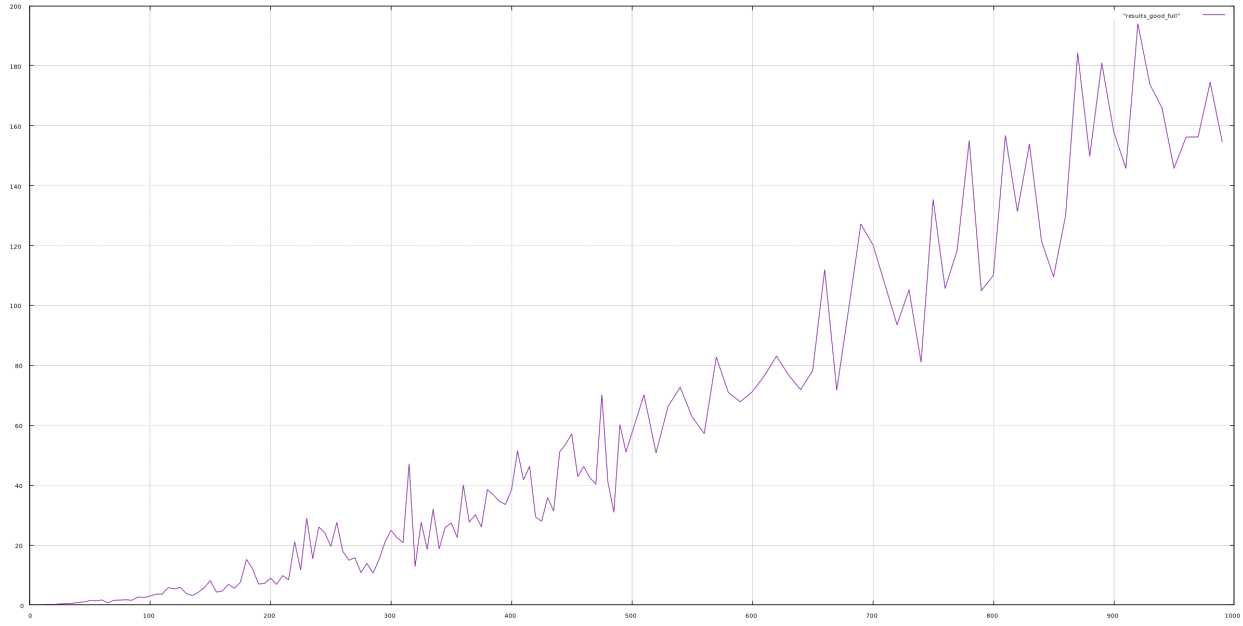

Figura 3: Tiempos de ejecución para los casos de prueba entre 5 y 1000 elementos de la mochila. Y - ms / X - Nodos

DIEGO ATANCE SANZ
ABRIL 2020
Ult. modificación 24 de Mayo del 2020
Creado con \LaTeX{}