UNIVERSIDAD TORCUATO DI TELLA

Tecnología Digital V: Diseño de algoritmos
Licenciatura en Tecnología Digital
Primer Semestre, 2025

TP1 - Algoritmos exactos para el KP-01 con Grafo de Conflictos

---

Motivación

El Problema de la Mochila 0-1 (KP01) es un problema fundamental de optimización combinatoria con importantes aplicaciones prácticas en asignación de recursos, finanzas y logística. Modela escenarios de toma de decisiones en los que se debe seleccionar un conjunto de ítems, cada uno con un peso y un valor, para maximizar el valor total sin exceder una capacidad límite. Debido a su naturaleza NP-hard, el KP-01 es un problema de referencia para el desarrollo de técnicas de optimización exactas y heurísticas. Sus aplicaciones en el mundo real incluyen la carga de mercancías, la planificación presupuestaria y los problemas de corte de materiales (ver, e.g. [2]), lo que lo convierte en un problema clave en la investigación operativa y la informática [3].
En muchas situaciones prácticas, restricciones adicionales imponen conflictos entre los ítems, lo que significa que ciertos pares no pueden seleccionarse simultáneamente. Esto ocurre en aplicaciones como en la optimización de carteras financieras o selección de proyectos, donde regulaciones pueden prohibir la selección simultánea de ciertos activos. De manera similar, en la manufactura y logística, algunos componentes de máquinas o productos a ser enviados pueden ser incompatibles debido a restricciones de seguridad o funcionalidad. Estas consideraciones motivan el estudio del Problema de la Mochila 0-1 con un Grafo de Conflictos, donde las aristas del grafo representan restricciones de incompatibilidad entre ítems.

El Problema de la Mochila 0-1 con Grafo de Conflictos

El Problema de la Mochila 0-1 con Grafo de Conflictos (KP01wCG) extiende el problema clásico de la mochila (KP-01) al incorporar un conjunto de conflictos entre ítems. Formalmente, se tiene un conjunto de ítems $N = \{1, \ldots, n\}$, donde cada ítem $i \in N$ tiene un beneficio asociado $p_i > 0$ y un peso $w_i > 0$. Se dispone de una mochila con una capacidad máxima C, y el objetivo es seleccionar un subconjunto de ítems que maximice el beneficio total sin superar la capacidad C.
Además de esta restricción, se define un *grafo de conflictos* $G = (N, E)$ sobre el conjunto de ítems. Cada arista $(i, j) \in E$ representa un conflicto entre los ítems i y j, lo que significa que no pueden seleccionarse simultáneamente. Esta generalización modela diversos escenarios del mundo real en los que los ítems están limitados no solo por el peso, sino también por incompatibilidades mutuas.
Debido a su mayor complejidad, el KP01wCG requiere enfoques de solución especializados que van más allá de los utilizados para el problema clásico KP01. Algoritmos exactos de última generación para resolver este problema han sido propuestos recientemente en [1], donde se desarrolla un algoritmo *branch and bound* para manejar de manera eficiente las restricciones adicionales impuestas por el grafo de conflictos.

Objetivos

En este trabajo exploraremos distintos algoritmos para el KP01wCG, incluyendo comparaciones entre distintas implementaciones y lenguajes. En particular, buscamos responder las siguientes

preguntas implementando los algoritmos y realizando experimentos computacionales. Considerando algoritmos exactos:

- ¿Hasta qué tamaño de instancias podemos abordar para el KP01wCG, y en qué tiempos de cómputo?
- ¿Cómo afecta el lenguaje de programación en la performance del algoritmo?
- ¿Cómo afecta la implementación en la performance del algoritmo?
- Tomando como problema el KP-01 (sin conflictos): ¿Cómo se compara la performance de un algoritmo de backtracking con uno basado en programación dinámica?

Los datos

Para la realización del trabajo se contará con distintos conjuntos de instancias de características diferentes para el KP01wCG. El formato de los archivos de entrada es el siguiente:

1. **Primera línea**: Número total de ítems, denotado como n.
2. **Segunda línea**: Capacidad de la mochila, denotada como C.
3. **Tercera línea**: Una secuencia de n enteros $w_1, w_2, \ldots, w_n$, donde cada w_j representa el peso del ítem j.
4. **Cuarta línea**: Una secuencia de n enteros $p_1, p_2, \ldots, p_n$, donde cada p_j representa la ganancia asociada al ítem j.
5. **Quinta línea**: Un entero m, que indica la cantidad de pares de ítems en conflicto.
6. **Las siguientes m líneas**: Cada línea contiene dos enteros u y v ($0 \leq u, v < n$), indicando que los ítems u y v están en conflicto y no pueden ser seleccionados simultáneamente en la solución.

Notar que al no considerar las incompatibilidades, cada una de las instancias es a su vez válida para el KP-01. A modo de ejemplo, consideremos el siguiente archivo:

```
5
50
10 20 30 40 50
60 100 120 200 150
3
0 2
1 4
3 4
```

- $n = 5$, hay 5 ítems.
- $C = 50$, la capacidad de la mochila es 50.
- Pesos: $w = (10, 20, 30, 40, 50)$.

- Beneficios: $p = (60, 100, 120, 200, 150)$.
- $m = 3$, hay 3 pares de conflictos:
  - Ítems 0 y 2 están en conflicto.
  - Ítems 1 y 4 están en conflicto.
  - Ítems 3 y 4 están en conflicto.

El trabajo

El trabajo práctico debe ser realizado en grupos de exactamente 3 personas. La resolución del trabajo se compone de varias etapas, incluyendo el diseño de los algoritmos, programación, experimentación, así como también el reporte detallado de la evaluación de los métodos implementados, los resultados obtenidos y la discusión correspondiente. Se pide:

1. **Algoritmos.**
   (a) KP01wCG. Se pide implementar los siguientes algoritmos, usando el lenguaje `C++`.
   - **Fuerza bruta.** Proponer un algoritmo de fuerza bruta (BF) para el KP01wCG.
   - **Backtracking.** Proponer un algoritmo de backtracking (BT) para el KP01wCG, implementando al menos una poda por factibilidad y una por optimalidad.

   En ambos casos, incluir en el informe el pseudocódigo, una descripción de las estructuras de datos utilizadas y la complejidad de los distintos componentes.

   (b) KP-01. Notar que en el KP01wCG generaliza al KP-01 (¿Cómo?), y por lo tanto es posible reutilizar el backtracking implementado para el KP01wCG definiendo convenientemente la instancia descartando los conflictos.
   - Implementar el algoritmo de Programación Dinámica (DP) para el KP-01 en `C++`.
   - Re-implementar el algoritmo de BT y de DP en `Python`. En este punto, es posible usar la asistencia de AI (chatGPT, Claude, etc.). Incluir en el informe una breve descripción respecto a qué herramienta se utilizó, cómo fue utilizada y el link correspondiente a la conversación.

2. **Experimentación y discusión.** Realizar una experimentación exhaustiva usando (al menos) los datasets provistos. Algunas ideas de experimentación para abordar las preguntas formuladas originalmente:
   - Evaluar tiempo de cómputo y calidad de solución para BF y BT en instancias de KP01wCG, en `C++`
   - Comparar los tiempos de ejecución del BT implementado en `C++` con la correspondiente implementación en `Python` en instancias de KP01wCG.
   - Comparar las implementaciones del BT y DP para ambos lenguajes, en instancias del KP-01.

   En base a los resultados, establecer comparaciones entre los distintos algoritmos e implementaciones a fin de responder la siguiente pregunta: ¿cuál sería la sugerencia del grupo respecto al algoritmo, implementación y lenguaje de programación?

3. **Diseño e implementación.** Como parte del trabajo se entrega un conjunto limitado de clases que representan las entidades principales involucradas en el desarrollo (una instancia, algoritmos, solución, etc.), que ustedes pueden expandir y modificar según crean conveniente. Aplicando conocimientos de materias previas, la entrega debe tener un diseño acorde tomando como punto de partida las clases provistas, agregando aquellas que considere necesarias. Implementar una clase `Grafo` para representar el grafo de conflicto, que ofrezca los métodos necesarios para poder implementar los algoritmos pedidos y justificando la decisión respecto a la representanción interna. Incluir de ser posible casos de test para al menos 2 clases.
4. **Algoritmos, informe, presentación de resultados y entrega del código.** El modelo, la descripción de los algoritmos, los operadores, la justificación de lsa decisiones de diseño, el testing realizado, la presentación de resultados, instrucciones de compilación y ejecución.

Modalidad de entrega

Se pide presentar el modelo y la experimentación en un informe de máximo 15 páginas que contenga:

- introducción al problema y la decisión,
- descripción de los algoritmos propuestos e implementados, según corresponda,
- consideraciones generales respecto a la implementación del modelo, incluyendo dificultades que hayan encontrado,
- resumen de resultados obtenidos en la experimentación,
- conclusiones, posibles mejoras y observaciones adicionales que consideren pertinentes.

Junto con el informe debe entregarse el código con la implementación del modelo. El mismo debe ser entendible, incluyendo comentarios que faciliten su corrección y ejecución.

Fechas de entrega

Formato Electrónico: **10 de abril de 2025, 23:59 hs**, enviando el trabajo (informe + código) vía el campus virtual.
Importante: El horario es estricto. Las entregas recibidas después de la hora indicada serán consideradas re-entrega.

References

[1] Andrea Bettinelli, Valentina Cacchiani, and Enrico Malaguti. A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS Journal on Computing*, 29(3):457–473, 2017.

[2] Subodha Kumar, Milind Dawande, and Vijay Mookerjee. Optimal scheduling and placement of internet banner advertisements. *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1571–1584, 2007.

[3] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.