Algoritmos de vuelta atrás

Algoritmia y optimización
Grado en Ingeniería en Inteligencia Artificial

Vuelta atrás

Introducción

- Hay determinados problemas cuya única solución es **enumerar todas las posibles soluciones** y guardar la mejor.
- Este enfoque es óptimo **por definición** pero lleva a complejidades asintóticas **exponenciales**.
- La **vuelta atrás** es un estrategia para enumerar todas las posibles soluciones, adecuada para añadir **mejoras prácticas a la eficiencia**.

El problema de la mochila (general)

Vuelta atrás

El problema de la mochila

Instancia

Valores $(v_1, v_2, \ldots, v_n)$

Pesos $(w_1, w_2, \ldots, w_n)$

Peso máximo W

Problema (versión general)

$$\arg \max_{\mathbf{x} \in \{0,1\}^n} \sum_{i=1}^{n} v_i x_i$$

s.t. $\sum_{i=1}^{n} w_i x_i \leq W$

$\mathbf{v}, \mathbf{w} \in \mathbb{R}$

Vuelta atrás

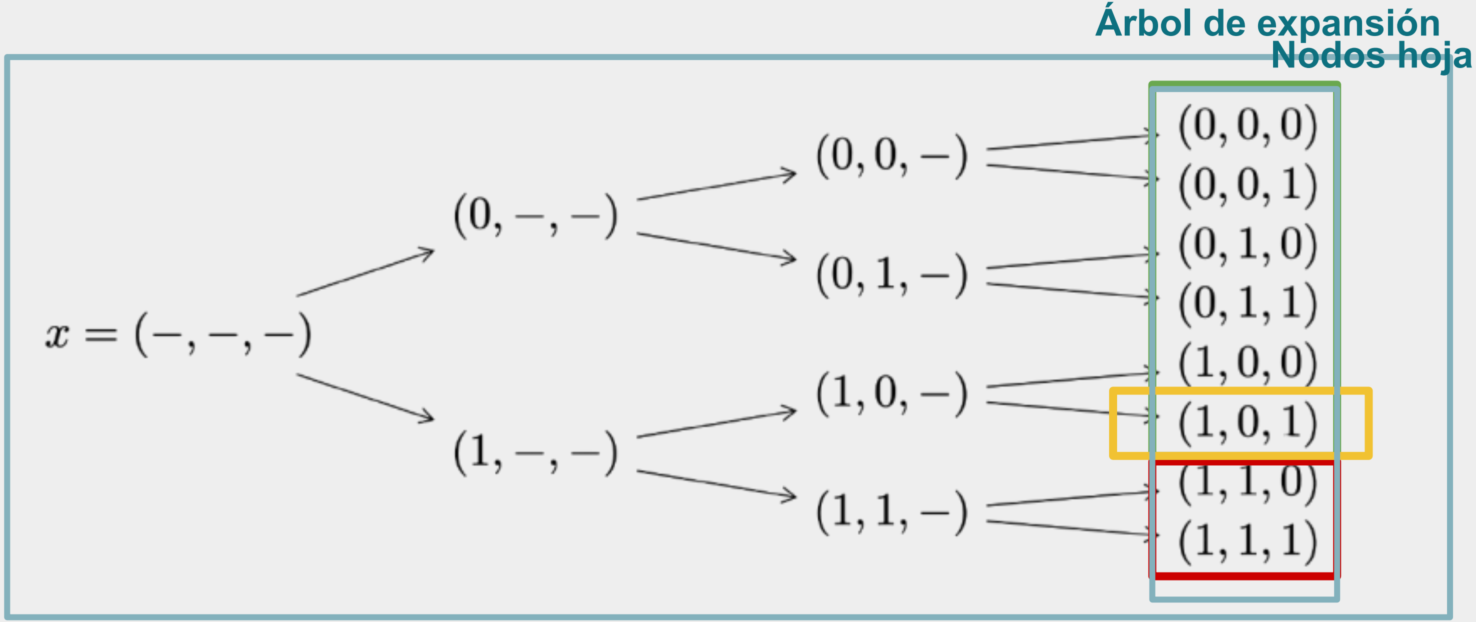
El problema de la mochila (general)

- Posibles soluciones:
    - **Programación dinámica**: infinitos subproblemas.
    - **Estrategia voraz**: no es óptima.
- **Enumerar** todas las posibles formas de llenar la mochila:
    - Soluciones que **cumplen con la restricciones** (soluciones factibles)
    - Solución que **maximiza el valor** (solución óptima)

Vuelta atrás

El problema de la mochila (general)

Asumiendo v = {6, 5, 1}, w = {25, 10, 5} y W = 30.



Vuelta atrás

El problema de la mochila (general)

- **Vuelta atrás**: forma sistemática de generar todas las posibles configuraciones de la mochila.
  - **Codificamos** la solución en una tupla (vector binario).
  - **Versión recursiva**: cada expansión del árbol es una llamada recursiva considerando una opción en un índice concreto de la tupla.
    - Cuando la llamada *vuelve*, se considera la siguiente opción.
    - Una vez agotadas las opciones, se *vuelve atrás* al elemento anterior del vector solución.

Vuelta atrás

El problema de la mochila (general): enumeración

Asumimos v, w, W accesibles globalmente (no cambian)

Llamada inicial: v_atras(0,|x|=n, best=0)

```
función v_atras(i, x, best):
    si i = n
        si peso(x,w) <= W:
            best = max(best, valor(x,v))

    si no
        x[i] = 0
        v_atras(i+1, x, best)
        x[i] = 1
        v_atras(i+1, x, best)
```

peso y *valor* tienen coste lineal: ¿podemos mejorar?

Vuelta atrás

El problema de la mochila (general): aprovechando cálculos

Llamada inicial: v_atras(0,|x|=n, best=0, 0, 0)

```
función v_atras(i, x, best, v_acc, w_acc):
    si i = n
        si w_acc <= W:
            best = max(best, v_acc)

    si no
        x[i] = 0
        v_atras(i+1, x, best, v_acc, w_acc)
        x[i] = 1
        v_atras(i+1, x, best, v_acc + v[i], w_acc + w[i])
```

¿Debemos esperar a una hoja para mirar la restricción de peso?

Vuelta atrás

El problema de la mochila (general): podando

Llamada inicial: v_atras(0,|x|=n, best=0, 0, 0)

```
función v_atras(i, x, best, v_acc, w_acc):
    si w_acc <= W:
        si i = n
            best = max(best, v_acc)

        si no
            x[i] = 0
            v_atras(i+1, x, best, v_acc, w_acc)
            x[i] = 1
            v_atras(i+1, x, best, v_acc + v[i], w_acc +
    w[i])
```

Vuelta atrás

Cotas optimistas

- Definimos como **solución parcial prometedora** aquella que **podría** mejorar al mejor valor obtenido hasta el momento.
- Interesa **podar** cualquier solución parcial **no prometedora**.
- ¿Podemos saber si una solución parcial es **prometedora**?
  - Asumiendo que todos los objetos restantes van a caber.
  - Asumiendo que todos los objetos restantes se pueden fraccionar.
- A estas estimaciones se les llama **cota optimista**.

Vuelta atrás

Cotas optimistas

- **Relajar las restricciones** del problema para obtener un cálculo optimista desde una solución parcial.
    - **Restricciones muy relajadas**: cota demasiado optimista, menos podas.
    - **Restricciones demasiado estrictas**: podrían podar soluciones prometedoras (*no es cota optimista*).

Vuelta atrás

El problema de la mochila (general): poda optimista

Llamada inicial: v_atras(0,|x|=n, best=0, 0, 0)

```
función v_atras(i, x, best, v_acc, w_acc):
    si w_acc <= W && es_prometedora(i, x, best):
        si i = n
            best = max(best, v_acc)

        si no
            x[i] = 0
            v_atras(i+1, x, best, v_acc, w_acc)
            x[i] = 1
            v_atras(i+1, x, best, v_acc + v[i], w_acc +
    w[i])
```

Vuelta atrás

Solución inicial

- No podemos **podar** hasta hasta tener una primera solución.
- ¿Podemos adelantarnos? → Utilizando una cota pesimista.
  - Una **cota pesimista** es una solución (sub)óptima de un problema.
  - Es importante que la solución sea **posible** (cumpla los requisitos del problema) y **eficiente** (para que sea útil).
  - Podemos utilizar un **algoritmo voraz**.

Vuelta atrás

El problema de la mochila (general): solución voraz

Llamada inicial: v_atras(0,|x|=n, best=voraz(v,w,W), 0, 0)

```
función v_atras(i, x, best, v_acc, w_acc):
    si w_acc <= W && es_prometedora(i, x, best):
        si i = n
            best = max(best, v_acc)

        si no
            x[i] = 0
            v_atras(i+1, x, best, v_acc, w_acc)
            x[i] = 1
            v_atras(i+1, x, best, v_acc + v[i], w_acc +
    w[i])
```

Vuelta atrás

El problema de la mochila (general): uso de cota pesimista

Llamada inicial: v_atras(0,|x|=n, best=voraz(v,w,W), 0, 0)

```
función v_atras(i, x, best, v_acc, w_acc):
    best = max(best, v_acc + voraz(i, v, w, W))

    si w_acc <= W && es_prometedora(i, x, best):
        si i = n
            best = max(best, v_acc)
        si no
            x[i] = 0
            v_atras(i+1, x, best, v_acc, w_acc)
            x[i] = 1
            v_atras(i+1, x, best, v_acc + v[i], w_acc +
    w[i])
```

Vuelta atrás

Resumen mejoras prácticas

Promedio del número de llamadas para 100 instancias aleatorias del problema de la mochila con 100 objetos.

Básico	Optimista	Inicio voraz	Pesimista
2,5e+30	4491	277	253

Esquema general

Vuelta atrás

Esquema general

- **Vuelta atrás**: forma sistemática de generar todas las soluciones.
- Para el elemento i-ésimo del vector solución:
  - Considerar una de las posibles opciones y continuar con el siguiente recursivamente.
  - Cuando la solución *vuelve*, considerar la siguiente opción.
  - Una vez agotadas las opciones, se *vuelve atrás* al elemento anterior del vector solución.

Vuelta atrás

Esquema general

Llamada inicial: v_atras(|x|=n, 0, n, –)

```
función v_atras(x, i, n, best):
    si i = n
        si factible(x):
            best = mejor(best, valor(x))

    si no
        para cada o en opciones(i)
            x[i] = o
            v_atras(x, i+1, n, best)
```

Vuelta atrás

Esquema general

- **Vuelta atrás**: forma sistemática de generar todas las soluciones.
- ¿Podemos hacerlo mejor?
    - Si una solución parcial no es prometedora, se "poda".
    - ¿Podemos adelantar si una solución es prometedora? **Cota optimista**
    - ¿Podemos empezar a podar sin ninguna hoja? **Solución inicial voraz**
    - ¿Podemos actualizar la "mejor solución" antes de una hoja? **Cota pesimista**

Vuelta atrás

Esquema general

Llamada inicial: vuelta_atras([], 0, n, voraz())

```
función v_atras(x, i, n, best):
    si factible(x):
        si i = n
            best = mejor(best, valor(x))
        si no
            best = mejor(best, pesimista(x))
            si optimista(x) > best:
                para cada o en opciones(i)
                    x[i] = o
                    v_atras(x, i+1, n, best)
```

Vuelta atrás

Puntos claves

- **Formulación:** cómo codificar un vector solución.
- **Ahorrar cálculos:** reutilizar algunos cálculos relacionados con el valor de la solución o sus restricciones.
- **Cota optimista**: relajar las restricciones del problema para calcular rápidamente si la solución parcial es prometedora.
- **Cota pesimista**: calcular rápidamente una posible solución a partir de una solución parcial (o inicial), asegurando que sea factible.

Vuelta atrás

Consideraciones

- La vuelta atrás hace un recorrido en **profundidad**.
    - Existen variantes que exploran por **prioridad** (*ramificación y poda*).
- La versión recursiva se puede convertir a una **versión iterativa**.
- Las mejoras prácticas no reducen la complejidad asintótica sino únicamente empírica: **dependiente del problema y la instancia**.

El viajante de comercio

El viajante de comercio

Introducción

- El **problema del viajante de comercio** (*Travelling Salesman Problem*, o TSP):
  - Imaginemos a un vendedor que debe visitar una lista de ciudades exactamente una vez y regresar al punto de partida.
  - Su objetivo es encontrar la ruta más corta posible para minimizar el tiempo y el coste de viaje.

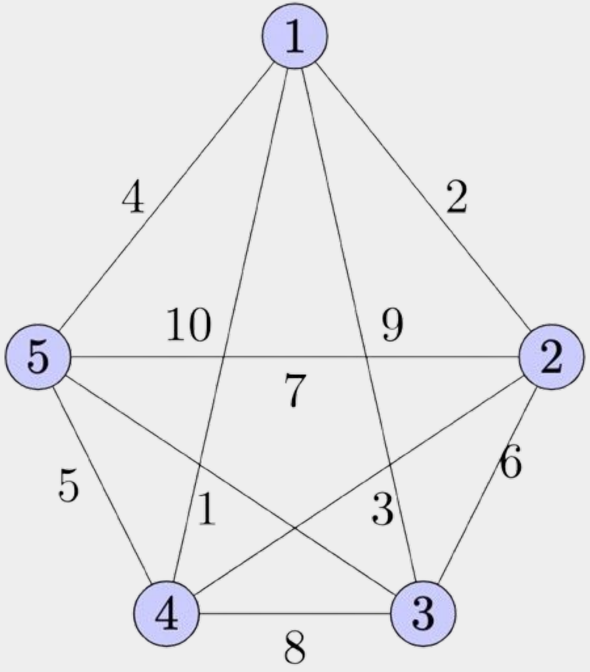
El viajante de comercio

Introducción

- Dado un grafo ponderado **g = (V, E)** con pesos no negativos, el problema es encontrar un ciclo hamiltoniano de mínimo coste.
  - Un **ciclo hamiltoniano** es un recorrido en el grafo que recorre todos los vértices sólo una vez y regresa al de partida.
  - El **coste** de un ciclo viene dado por la **suma de los pesos de las aristas** que lo componen.
  - Es posible que **no haya arista** entre dos nodos**.**

El viajante de comercio

Ejemplo



Ciclo de coste mínimo:

1 → 2 → 4 → 5 → 3 → 1

Coste:

2 + 3 + 5 + 1 + 9 = 20

El viajante de comercio

Formulación

- Antes de implementar, hay que considerar:
  - Vector solución
  - Bucle de expansión
  - Reusar cálculos
  - Cota optimista
  - Cota pesimista

El viajante de comercio

Formulación

- Antes de implementar, considerar:
    - Vector solución: x = |V| donde x[i] indica el nodo i-ésimo a visitar o x = [] e ir añadiendo.
    - Bucle de expansión: mantener un visitados booleano para evitar repetir
    - Reusar cálculos: llevar el coste del camino en la llamada
    - Cota optimista: Kruskal !!
    - Cota pesimista: voraz:
        - Voraz