Algorítmica

Capítulo 3. Algoritmos Greedy

Tema 9. Heurísticas Greedy

- Algoritmos greedy como heurísticas:

El Problema del Coloreo de un Grafo. El problema del Viajante de Comercio. El Problema de la Mochila. El Problema de la Asignación de Tareas

Heurísticas

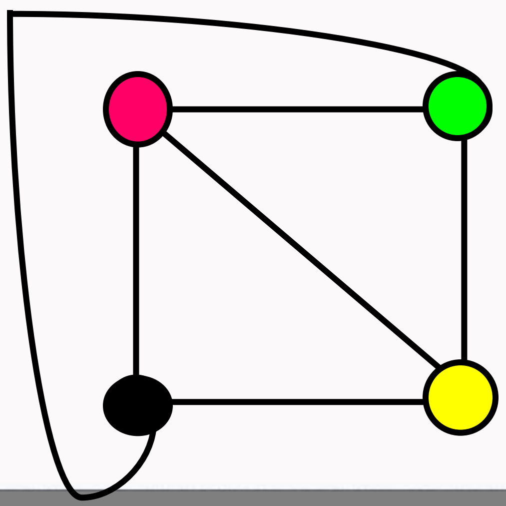
- Son procedimientos que, basados en la experiencia, proporcionan buenas soluciones a problemas concretos
  - Algoritmos Genéticos, Enfriamiento (Recocido) Simulado, Búsqueda Tabú,
  - Computación Evolutiva, GRASP (Greedy Randomized Adaptive Search Procedures), Búsqueda Dispersa, Colonias de Hormigas, Búsqueda por Entornos Variables, Búsqueda Local Guiada, Búsqueda Local Iterativa
  - Métodos Ruidosos, Aceptación de Umbrales, Algoritmos Meméticos, Redes de Neuronas, ...

Heuristícas Greedy

- Es mejor satisfacer que optimizar
- El tiempo efectivo que se tarda en resolver un problema es un factor clave
- Los algoritmos greedy son muy buenos como heurísticas
  - El Problema del Coloreo de un grafo
  - El Problema del Viajante de Comercio
  - El Problema de la Mochila
  - Problemas de Recubrimiento, de Rutas, ...
- Suelen usarse también para encontrar una primera solución (óptimo local)

El Problema del Coloreo de un Grafo

- Planteamiento
  - Dado un grafo plano G = (V, E), determinar el mínimo número de colores que se necesitan para colorear todos sus vértices, y que no haya dos de ellos adyacentes pintados con el mismo color
- Si el grafo no es plano puede requerir tantos colores como vértices haya
- Las aplicaciones son muchas
  - Representación de mapas
  - Diseño de páginas webs
  - Diseño de carreteras



El Problema del Coloreo de un Grafo

- El problema es NP y por ello se necesitan heurísticas para resolverlo
- El problema reúne todos los requisitos para ser resuelto con un algoritmo greedy
- Del esquema general greedy se deduce un algoritmo inmediatamente.
- Teorema de Appel-Hanke (1976): Un grafo plano requiere a lo sumo 4 colores para pintar sus nodos de modo que no haya vértices adyacentes con el mismo color

El Problema del Coloreo de un Grafo

- Suponemos que tenemos una paleta de colores (con mas colores que vértices)
- Elegimos un vértice no coloreado y un color. Pintamos ese vértice de ese color
- Lazo greedy: Seleccionamos un vértice no coloreado v. Si no es adyacente (por medio de una arista) a un vértice ya coloreado con el nuevo color, entonces coloreamos v con el nuevo color
- Se itera hasta pintar todos los vértices

Implementación del algoritmo

Funcion **COLOREO**

{COLOREO pone en NuevoColor los vertices de G que pueden tener el mismo color}

Begin

NuevoColor = ∅

Para cada vértice no coloreado v de G Hacer

Si v no es adyacente a ningún vértice en NuevoColor

Entonces

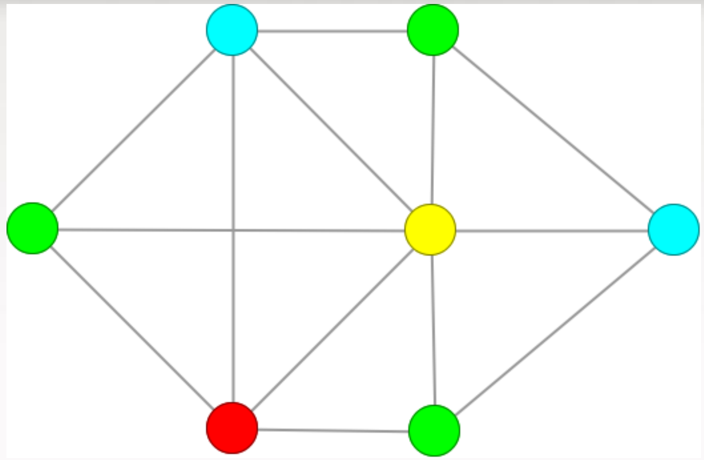
Marcar v como coloreado

Añadir v a NuevoColor

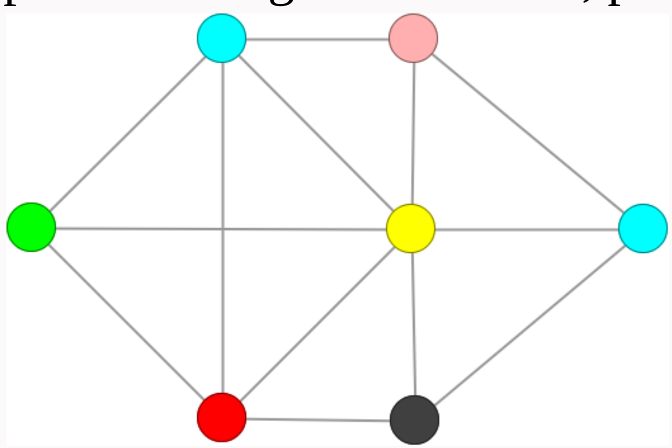
End

Se trata de un algoritmo que funciona en **O(n),** pero que no siempre da la solución óptima

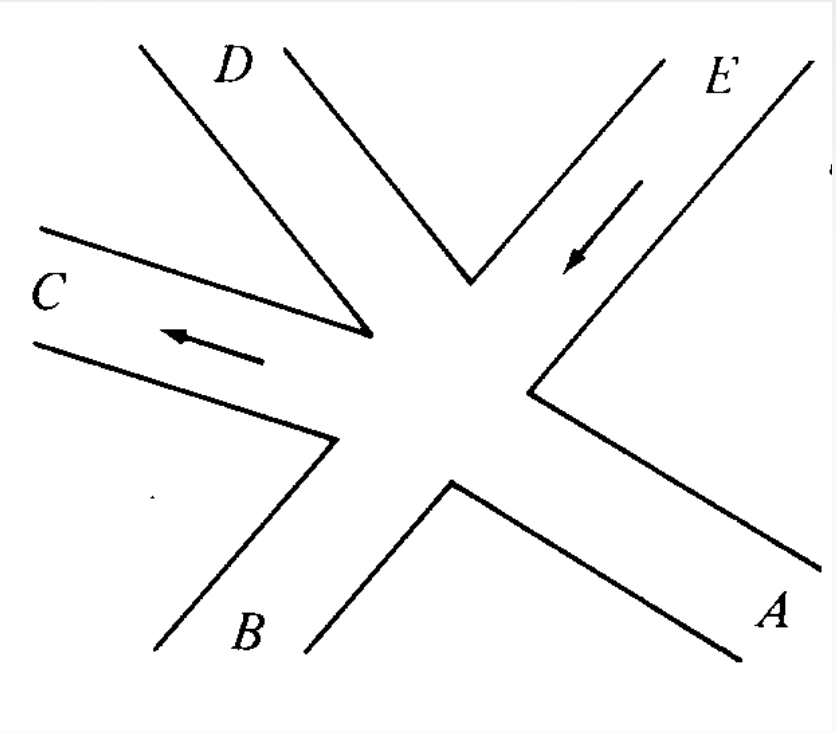
Ejemplo



El orden en el que se escogen los vértices para colorearlos puede ser decisivo: el algoritmo da la solución optimal en el grafo de arriba, pero no siempre es así



Ejemplo: Diseño de cruces de semáforos



• A la izquierda tenemos un cruce de calles que señalanlos sentidos de circulación.
• La falta de flechas, significa que podemos ir en las dos direcciones.
• Queremos diseñar un patrón de semáforos con el mínimo número de semáforos
• Suponemos un grafo cuyos vértices representan turnos, y cuyas aristas unen esos turnos que no pueden realizarse simultaneamente sin que haya colisiones
•El problema se convierte en uno de Coloreo de los Vértices de un Grafo

Las calles sin dirección, son en doble sentido. Por lo tanto tendríamos:

{AB, AC, AD, BA, BC, BD, DC, DA, DB, ED, EC, EA, EB}

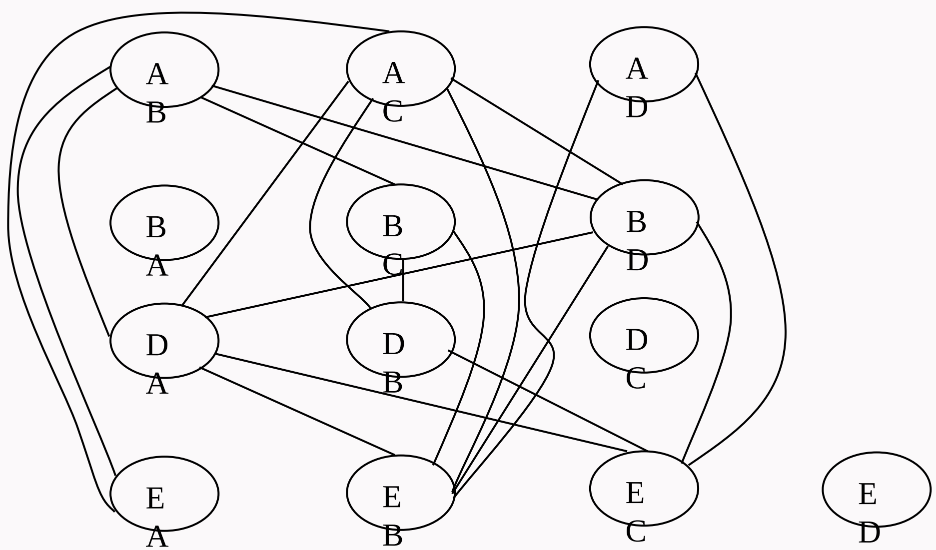
Ejemplo: Diseño de cruces de semáforos

- Los nodos representan giros, y dos nodos son adyacentes si sólo si éstos son incompatibles.
- Una heurística razonable para este problema es la siguiente:

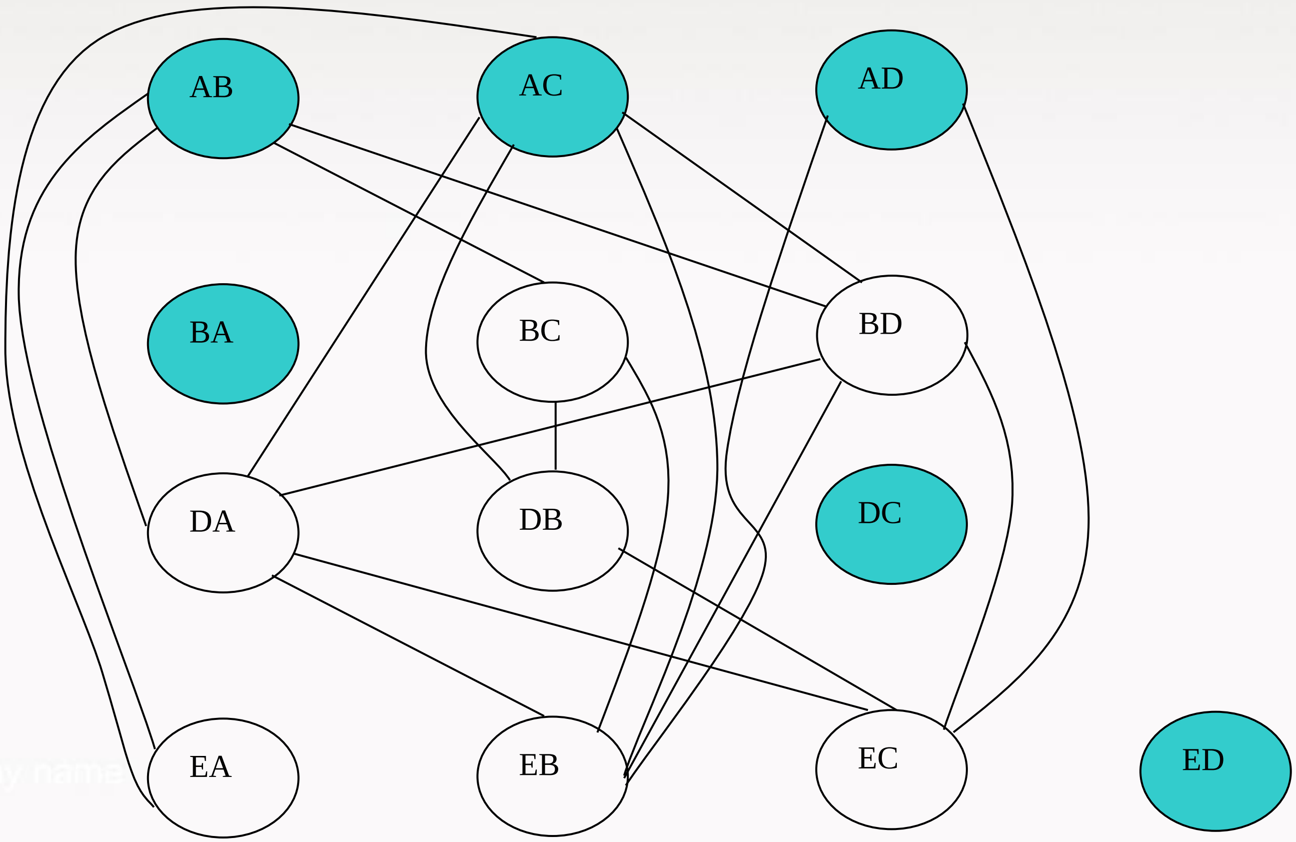
  - Comenzar coloreando todos los nodos que se pueda con un color, sin causar conflictos entre nodos adyacentes.

  - Si quedan nodos sin colorear, tomar un nuevo color y colorear tantos nodos no coloreados como se pueda, nuevamente sin causar conflictos.

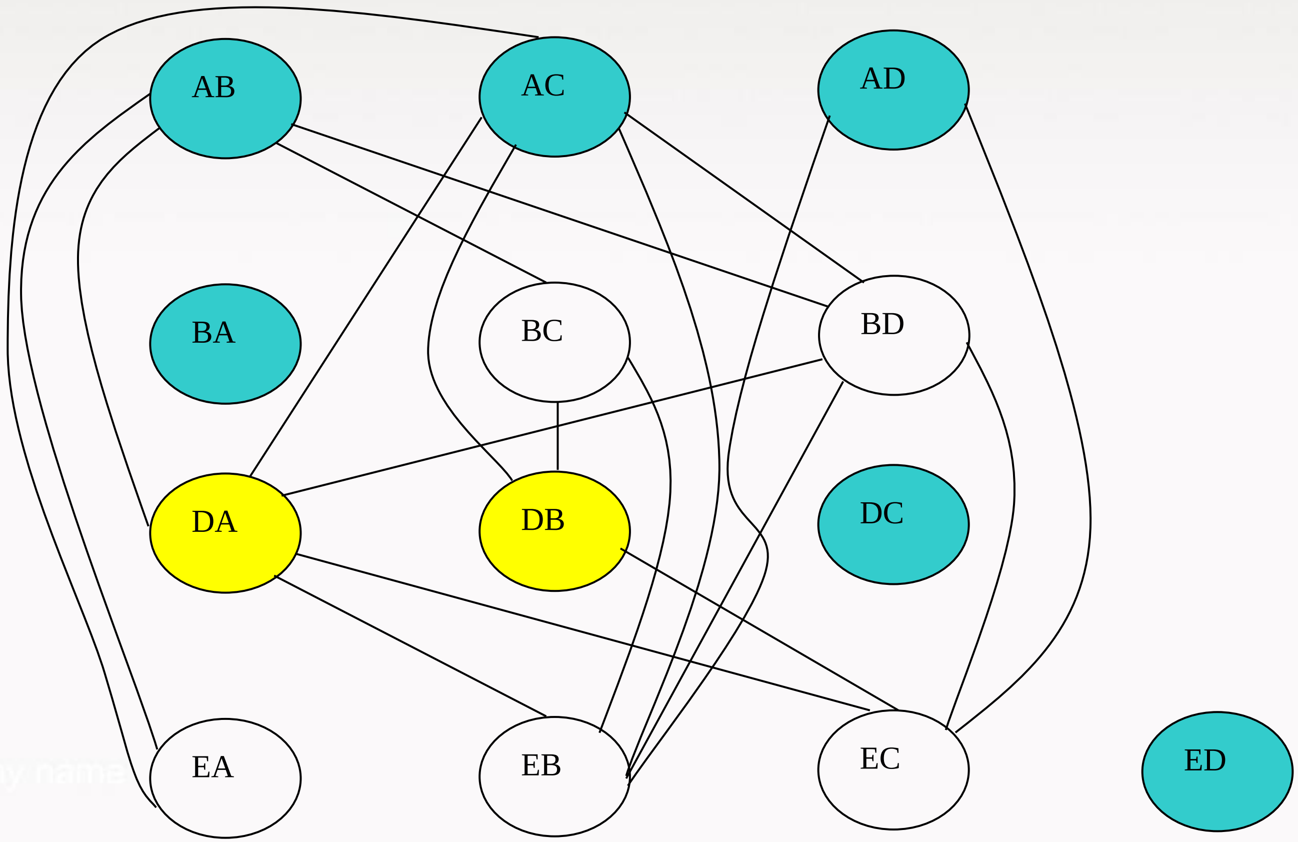
  - Continuar el proceso con nuevos colores hasta que ya no queden nodos sin colorear.



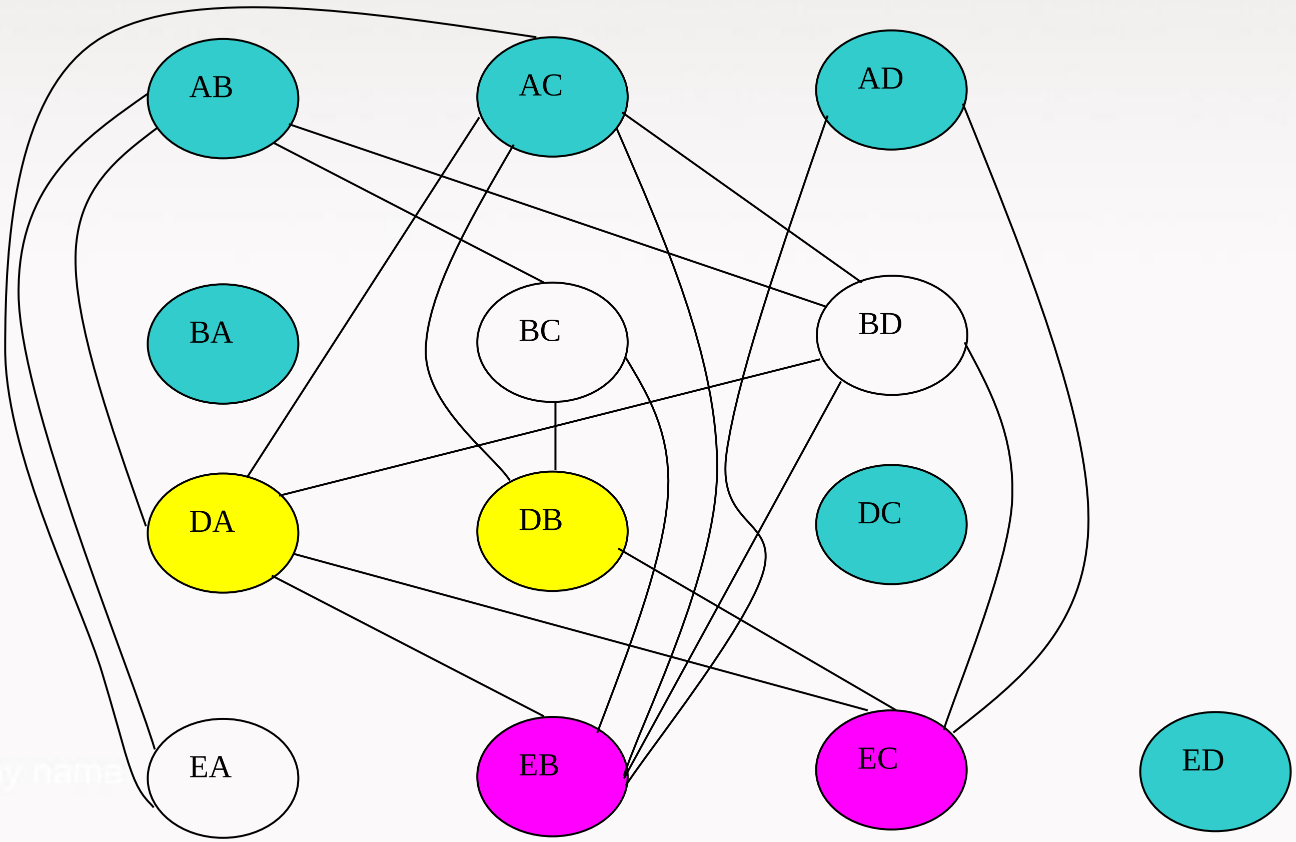
Ejemplo: Diseño de cruces de semáforos



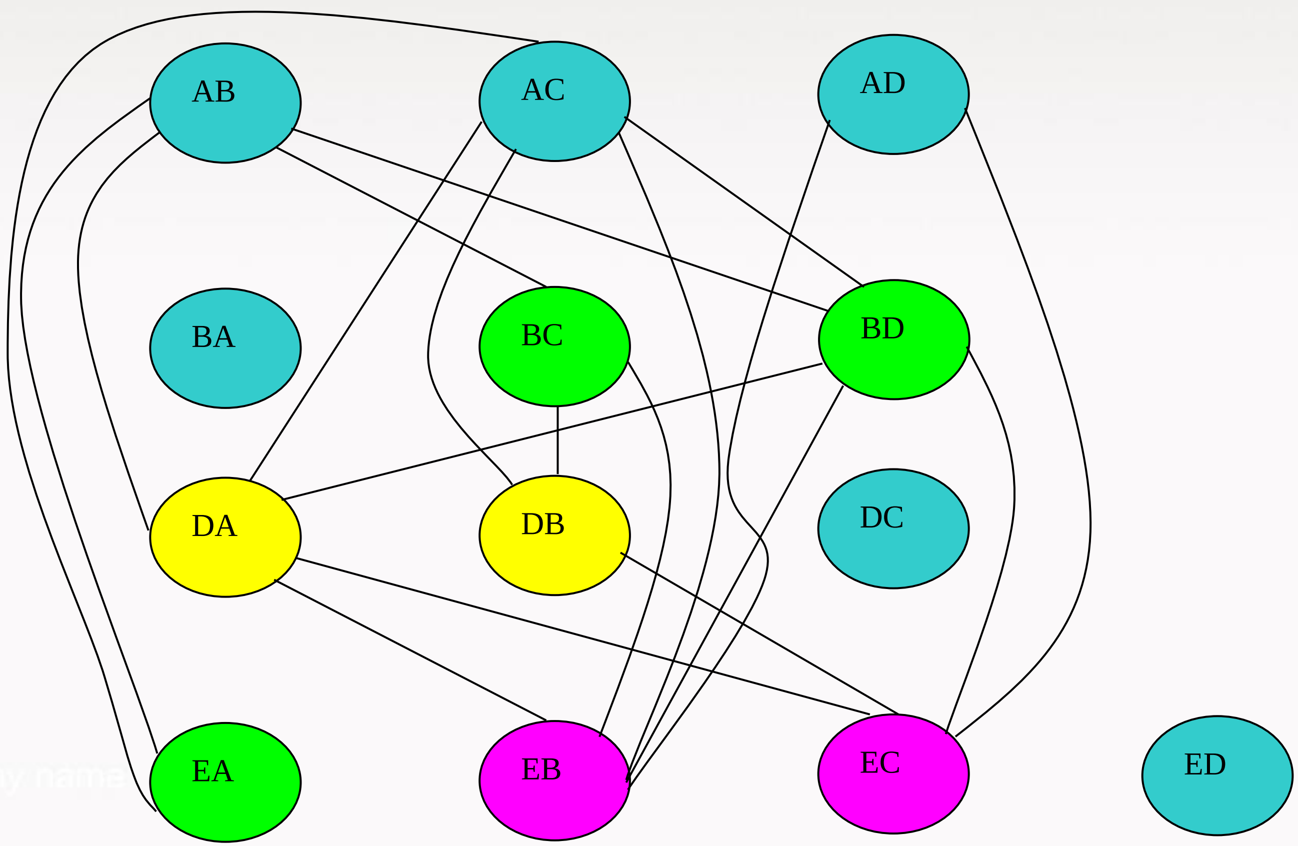
Ejemplo: Diseño de cruces de semáforos



Ejemplo: Diseño de cruces de semáforos



Ejemplo: Diseño de cruces de semáforos



El Problema del Viajante de Comercio

- Un viajante de comercio que reside en una ciudad, tiene que trazar una ruta que, partiendo de su ciudad, visite todas las ciudades a las que tiene que ir una y sólo una vez, volviendo al origen y con un recorrido mínimo
- Es un problema NP, no existen algoritmos en tiempo polinomial, aunque si los hay exactos que lo resuelven para grafos de tamaños "pequeños".
- Para grafos grandes, es necesario utilizar heurísticas, ya que el problema se hace intratable en el tiempo.
- El PVC es uno de los mas importantes en Algorítmica

El Problema del Viajante de Comercio

- Matemáticamente puede formularse como un problema de Programación Lineal con Números Enteros:

$$
\begin{aligned}
&\min \sum_{i=0}^{n} \sum_{j\neq i, j=0}^{n} c_{ij} x_{ij} && \\
&0 \leq x_{ij} \leq 1 && i, j = 0, \ldots n \\
&x_{ij} \text{ integer} && i, j = 0, \ldots n \\
&\sum_{i=0, i\neq j}^{n} x_{ij} = 1 && j = 0, \ldots, n \\
&\sum_{j=0, j\neq i}^{n} x_{ij} = 1 && i = 1, \ldots, n \\
&u_i - u_j + n x_{ij} \leq n - 1 && 1 \leq i \neq j \leq n.
\end{aligned}
$$

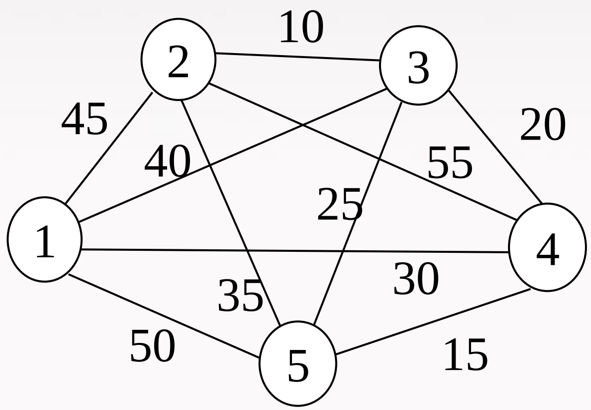
- La primera restricción asegura que desde cada origen 0,..., *n* se llegue a un destino, y la segunda asegura que desde cada ciudad 1,..., *n* se salga exactamente hacia una ciudad (ambas restricciones también implican que exista exactamente una salida desde la ciudad 0.)
- La última restricción obliga a que un solo camino cubra todas las ciudades y no dos o más caminos disjuntos cubran conjuntamente todas las ciudades.

El Problema del Viajante de Comercio

- En el contexto de la Algorítmica, suponemos un grafo no dirigido y completo G = (N, A) y L una matriz de distancias no negativas referida a G.
- Se quiere encontrar un **Circuito Hamiltoniano Minimal**.
- Este es un problema Greedy típico, que presenta las 6 condiciones para poder ser enfocado con un algoritmo greedy
- Destaca de esas 6 caracteristicas **la condición de factibilidad**:
  – que al seleccionar una arista no se formen ciclos,
  – que las aristas que se escojan cumplan la condición de no ser incidentes en tercera posición al nodo escogido

El Problema del Viajante de Comercio

- Consideremos el siguiente grafo

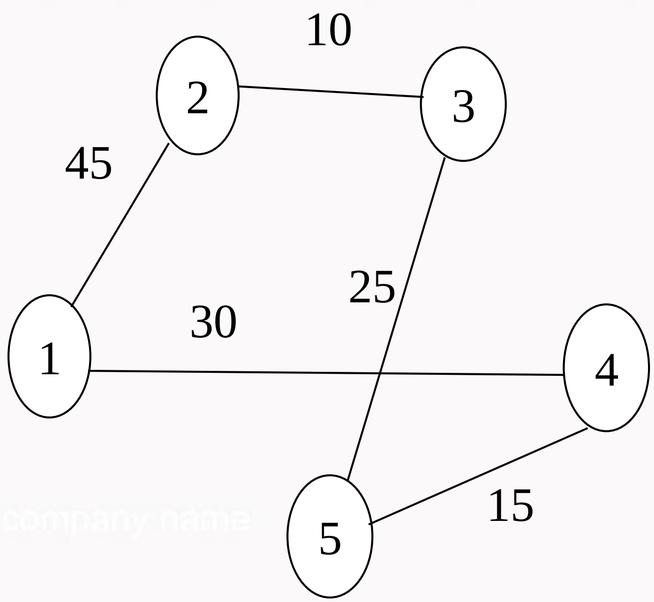


- Posibilidades:
  - Los nodos son los candidatos. Empezar en un nodo cualquiera y en cada paso moverse al nodo no visitado más próximo al último nodo seleccionado.
  - Las aristas son los candidatos. Hacer igual que en el Algoritmo de Kruskal, pero garantizando que se forme un ciclo.

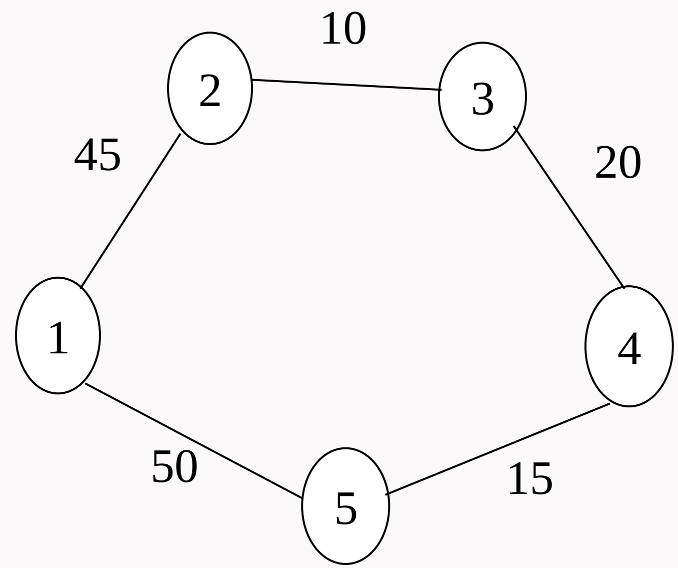
El Problema del Viajante de Comercio

- Solución con la primera heurística
- Solución empezando en el nodo 1

Solución empezando en el nodo 5



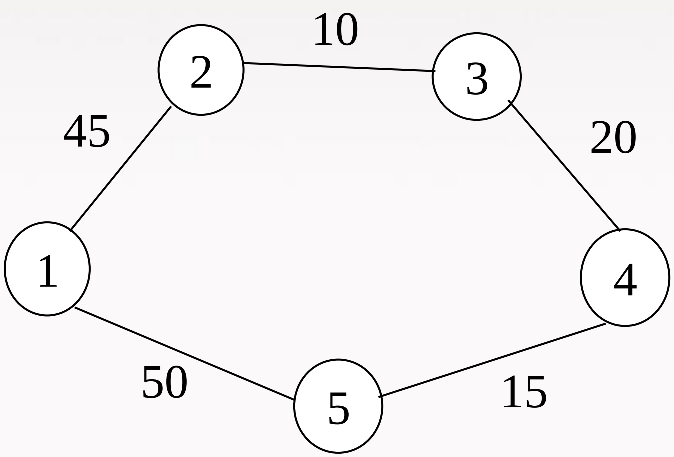
Solución: (1, 4, 5 ,3, 2), **125**



Solución: (5,4,3, 2,1), **140**

El Problema del Viajante de Comercio

- Solución con la segunda heurística



- Solución: ((2, 3), (4, 5), (3, 4), (1, 2), (1, 5))

Coste = 10+15+20+45+50 = 140

- En todos los casos la eficiencia es la del algoritmo de ordenación que se use

El problema de la Mochila Fraccional

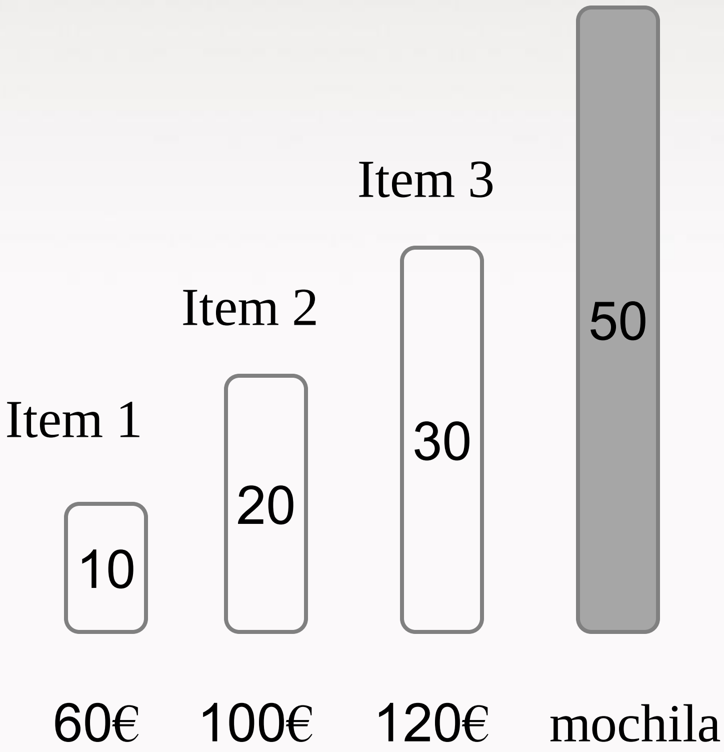
- Tenemos n objetos y una mochila. El objeto i tiene un peso w_i y la mochila tiene una capacidad M.
- Si metemos en la mochila la fraccion x_i, $0 \leq x_i \leq 1$, del objeto i, generamos un beneficio de valor $p_i x_i$
- El objetivo es rellenar la mochila de tal manera que se maximice el beneficio que produce el peso total de los objetos que se transportan, con la limitación de la capacidad de valor M

$$\text{maximizar} \quad \sum_{1 \leq i \leq n} p_i x_i$$

$$\text{sujeto a} \quad \sum_{1 \leq i \leq n} w_i x_i \leq M$$

con $0 \leq x_i \leq 1$, $1 \leq i \leq n$

Ejemplo



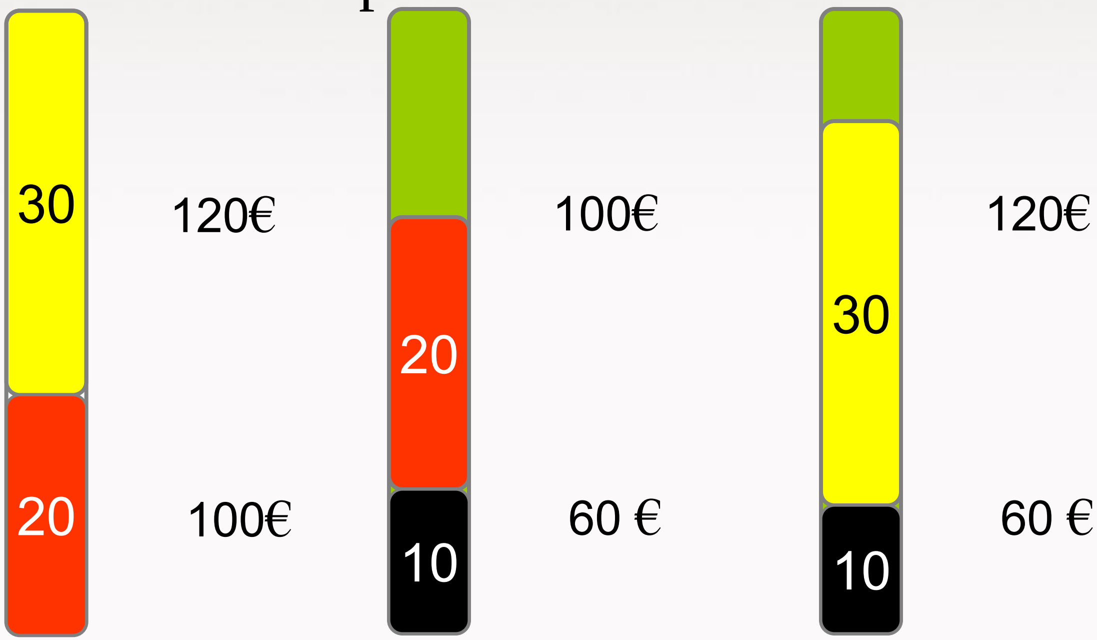
- Es un claro problema de tipo greedy
- Sus aplicaciones son inumerables
- Es un banco de pruebas algorítmico
- La técnica greedy produce soluciones optimales para este tipo de problemas
- También hay problemas de mochila booleanos (0 ó 1).

Mochila 0/1

Óptimo

Los menos pesados

Los mas caros



Total = 220 € = 160 € = 180 €

¿Como seleccionamos los items?

Mochila fraccional

- Supongamos 5 objetos de pesos y precios dados por la tabla, la capacidad de la mochila es 100.

Precio (euros)	20	30	65	40
Peso (kilos)	10	20	30	40

- **Método 1 elegir primero el menos pesado**
  - Peso total = 10 + 20 + 30 + 40 = 100
  - Costo total = 20 + 30 + 65 + 40 = 155
- **Método 2 elegir primero el mas caro**
  - Peso Total = 30 + 50 + 20 = 100
  - Costo Total = 65 + 60 **+** 20 **=** 145

Mochila fraccional

- Método 3 elegir primero el que tenga mayor valor por unidad de peso (razón costo/ peso)

Precio (euros)	20	30	65	40	60
Peso (Kilos)	10	20	30	40	50
Precio/ Peso	2	1,5	2,1	1	1,2

  – Peso Total = 30 + 10 + 20 + 40 = 100

  – Costo Total = 65 + 20 + 30 + 48 = 163

Otro ejemplo de Mochila Fraccional

- Supongamos el siguiente caso de problema de la mochila: $n = 3$, $M = 20$, $(p_1,p_2,p_3) = (25,24,15)$ y $(w_1,w_2,w_3) = (18,15,10)$

(x_1,x_2,x_3)	$\sum w_i x_i$	$\sum p_i x_i$
1) (1/2,1/3,1/4)	16.5	24.25
2) (1,2/15,0)	20	28.2
3) (0,2/3,1)	20	31
4) (0,1,1/2)	20	31.5

Mochila fraccional

- Tomando los items en orden de mayor valor por unidad de peso, se obtiene una solucion optimal

{60/10, 100/20, 120/30}

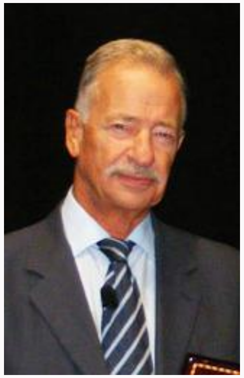
$\frac{20}{30}$ 80 €

20 100€

10 60 €

Total = 240 €

Silvano Martello



Paolo Toth

Solución Greedy

- Definimos la densidad del objeto A_i por p_i/w_i.
- Se usan objetos de tan alta densidad como sea posible, es decir, los seleccionaremos en orden decreciente de densidad.
- Si es posible se coge todo lo que se pueda de A_i, pero si no se rellena el espacio disponible de la mochila con una fraccion del objeto en curso, hasta completar la capacidad, y se desprecia el resto.
- Primero, se ordenan los objetos por densidad no creciente, es decir,

$$p_i/w_i \geq p_{i+1}/w_{i+1} \text{ para } 1 \leq i < n.$$

- Entonces se actúa de la siguiente manera

PseudoCodigo

```
Procedimiento MOCHILA_GREEDY(P,W,M,X,n)
//P(1:n) y W(1:n) contienen los costos y pesos respectivos de los n objetos
    ordenados como P(I)/W(I) ≥  P(I+1)/W(I+1). M es la capacidad de la
    mochila y X(1:n) es el vector solution//
real P(1:n), W(1:n), X(1:n), M, cr;
integer I,n;
    x = 0; //inicializa la solucion en cero //
    cr = M;  // cr = capacidad restante de la mochila //
    Para i = 1 hasta n Hacer
            Si W(i) > cr Entonces exit endif
            X(I) = 1;
            cr =  c - W(i);
    repetir
            Si I ≤ n Entonces X(I) = cr/W(I) endif
End MOCHILA_GREEDY
```

Demostración de la corrección

- Vamos a demostrar que el algoritmo siempre encuentra la solucion optimal del problema
- Sea $p_1/w_1 \geq p_2/w_2 \geq \ldots \geq p_n/w_n$
- Sea $X = (x_1, x_2, \ldots, x_n)$ la solución generada por MOCHILA_GREEDY, para una capacidad M
- Sea $Y = (y_1, y_2, \ldots, y_n)$ una solución factible cualquiera
- Queremos demostrar que

$$\sum_{i=1}^{n} (x_i - y_i) p_i \geq 0$$

Demostración de la corrección

1	2										n
1	1	1	1	1	1	1	1	1	1	1	1

- Si todos los x_i son 1, la solución es claramente optimal (es la única solución). En otro caso, sea k el menor número tal que $x_k < 1$.

1	2				k						n
1	1	..	..	1	x_k	0	0	..	..	0	0

Demostración de la corrección

1	2				k						n
1	1	..	..	1	x_k	0	0	..	..	0	0

$$\sum_{i=1}^{n}(x_i-y_i)p_i = $$

1

$$\sum_{i=1}^{k-1}(x_i-y_i)w_i\frac{p_i}{w_i}$$

2

$$+(x_k-y_k)w_k\frac{p_k}{w_k}$$

3

$$+\sum_{i=k+1}^{n}(x_i-y_i)w_i\frac{p_i}{w_i}$$

Demostración de la corrección

- Consideremos cada uno de esos bloques

$$\sum_{i=1}^{k-1}(x_i - y_i)w_i \frac{p_i}{w_i} \geq \sum_{i=1}^{k-1}(x_i - y_i)w_i \frac{p_k}{w_k}$$

$$(x_k - y_k)w_k \frac{p_k}{w_k} = (x_k - y_k)w_k \frac{p_k}{w_k}$$

$$\sum_{i=k+1}^{n}(x_i - y_i)w_i \frac{p_i}{w_i} \geq \sum_{i=k+1}^{n}(x_i - y_i)w_i \frac{p_k}{w_k}$$

Demostración de la correccion

$$\sum_{i=1}^{n}(x_i - y_i)p_i \geq \sum_{i=1}^{n}(x_i - y_i)w_i \frac{p_k}{w_k}$$

$$\frac{p_k}{w_k}\sum_{i=1}^{n}(x_i - y_i)w_i$$

$= W$

$\sum_i x_i w_i = M$ por hipótesis, pero $\sum_i y_i w_i \leq M$

Por tanto, como siempre $W > 0$, entonces

$$\sum_{i=1}^{n}(x_i - y_i)p_i \geq 0$$

La asignación de tareas

- Supongamos que disponemos de n trabajadores y n tareas. Sea $b_{ij} > 0$ el coste de asignarle el trabajo j al trabajador i. Una asignación de tareas puede ser expresada como una asignación de los valores 0 ó 1 a las variables x_{ij}, donde
  - $x_{ij} = 0$ significa que al trabajador i no le han asignado la tarea j, y
  - $x_{ij} = 1$ indica que sí.
- Una asignación válida es aquella en la que a cada trabajador sólo le corresponde una tarea y cada tarea está asignada a un trabajador.

La asignación de tareas

- Dada una asignación válida, definimos el coste de dicha asignación como:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij} b_{ij}$$

- Diremos que una asignación es óptima si es de mínimo coste.
- Cara a diseñar un algoritmo greedy para resolver este problema podemos pensar en dos estrategias distintas: asignar cada trabajador la mejor tarea posible, o bien asignar cada tarea al mejor trabajador disponible.
- Sin embargo, ninguna de las dos estrategias tiene por qué encontrar siempre soluciones óptimas. ¿Es alguna mejor que la otra?

La asignación de tareas

- Lamentablemente, el algoritmo greedy no funciona para todos los casos como pone de manifiesto la siguiente matriz:

		1	2	3
	1	16	20	18
Trabajador	2	11	15	17
	3	17	1	20

- Para ella, el algoritmo produce una matriz de asignaciones en donde los “unos” están en las posiciones (1,1), (2,2) y (3,3), esto es, asigna la tarea *i* al trabajador *i* (*i* = 1, 2 ,3), con un valor de la asignación de 51 (= 16 + 15 + 20).
- Sin embargo la asignación óptima se consigue con los “unos” en posiciones (1,3), (2,1) y (3,2), esto es, asigna la tarea 3 al trabajador 1, la 1 al trabajador 2 y la tarea 2 al trabajador 3, con un valor de la asignación de 30 (= 18 + 11 + 1).

Algoritmo Húngaro

- Para resolver este problema hay un algoritmo exacto:
- Algoritmo Húngaro
  - Nos referimos a él en el segundo capitulo cuando hablamos de la necesidad de diseñar nuevos algoritmos
  - Haremos un ejercicio para ver como funciona, y que se conozca su existencia.

Ejemplo

Cada uno de cuatro laboratorios, A,B,C y D tienen que ser equipados con uno de cuatro equipos informáticos. El costo de instalación de cada equipo en cada laboratorio lo da la tabla. Queremos encontrar la asignación menos costosa.

	1	2	3	4
A	48	48	50	44
B	56	60	60	68
C	96	94	90	85
D	42	44	54	46

Algoritmo Húngaro:

- Etapa 1:
  - Encontrar el elemento de menor valor en cada fila de la matriz n x n de costos.
  - Construir una nueva matriz restando a cada costo el menor costo de su fila.
  - En esta nueva matriz, encontrar el menor costo de cada columna.
  - Construir una nueva matriz (llamada de ***costos reducidos***) restando a cada costo el menor costo de su columna.

Algoritmo Húngaro:

- Etapa 2:
  - Rayar el minimo numero de lineas (horizontal y/o vertical) que se necesiten para tachar todos los ceros de la matriz de costos reducidos.
  - Si se necesitan n lineas para tachar todos los ceros, hemos encontrado una solución optimal en esos ceros tachados.
  - Si tenemos menos de n lineas para tachar todos los ceros, ir a la etapa 3.

Algoritmo Húngaro

- Etapa 3:
  - Encontrar el menor elemento no cero (llamar a su valor k) en la matriz de costos reducidos que no este tachado por alguna linea de las pintadas en la etapa 2.
  - Restar *k* a cada elemento no tachado en la matriz de costos reducidos y sumar k a cada elemento de la matriz de costos reducidos que este tachado por dos lineas.

  - Volver a la Etapa 2.

Ejemplo

Cada uno de cuatro laboratorios, A,B,C y D tienen que ser equipados con uno de cuatro equipos informáticos. El costo de instalación de cada equipo en cada laboratorio lo da la tabla. Queremos encontrar la asignación menos costosa.

	1	2	3	4
A	48	48	50	44
B	56	60	60	68
C	96	94	90	85
D	42	44	54	46

Aplicación del método húngaro

Restamos 44 en la fila 1

	1	2	3	4
A	4	4	6	0
B	56	60	60	68
C	96	94	90	85
D	42	44	54	46

Aplicación del método húngaro

Restamos 56 en a fila 2, 85 en la fila 3 y 42 en la fila 4

	1	2	3	4
A	4	4	6	0
B	0	4	4	12
C	11	9	5	0
D	0	2	12	4

Aplicación del método húngaro

Restamos 2 en la columna 2 y 4 de la columna 3

Hay una solución factible En las celdas con ceros?

	1	2	3	4
A	4	2	2	0
B	0	2	0	12
C	11	7	1	0
D	0	0	8	4

$\boxed{3 \neq 4}$

Aplicación del método húngaro

Minimo elemento no tachado

	1	2	3	4
A	4	2	2	0
B	0	2	0	12
C	11	7	1	0
D	0	0	8	4

Aplicación del método húngaro

	1	2	3	4
A	4-1=3	2-1=1	2-1=1	0
B	0	2	0	12+1=13
C	11-1=10	7-1=6	1-1=0	0
D	0	0	8	4+1=5

Aplicación del método húngaro

	1	2	3	4
A	3	1	1	0
B	0	2	0	13
C	10	6	0	0
D	0	0	8	5

Ahora es posible la asignación optimal
usando las celdas con ceros

Aplicación del método húngaro

	1	2	3	4
A	3	1	1	X 0
B	X 0	2	0	13
C	10	6	0 X	0
D	0	X 0	8	5