Branch and Bound

- Se recorre el árbol de estados de forma sistemática.
- Utiliza funciones de poda para eliminar ramas que no conducen a soluciones.
- En B&B se generan todos los hijos del e-nodo antes de que cualquier otro nodo vivo pase a ser el nuevo e-nodo.
- En backtracking tan pronto como se genera un nuevo hijo del e-nodo, este hijo pasa a ser el e-nodo.

Branch and Bound

- En backtracking los únicos nodos vivos son los que están en el camino de la raíz al e-nodo.
- En B&B puede haber más nodos vivos: se tienen que almacenar en alguna estructura de datos, llamada lista o contenedor de nodos vivos.
- En B&B el uso de cotas nos sirve, además de para podar, para determinar el orden de ramificación, seleccionando el nodo más prometedor como siguiente nodo a expandir.
- B&B se usa principalmente en problemas de optimización

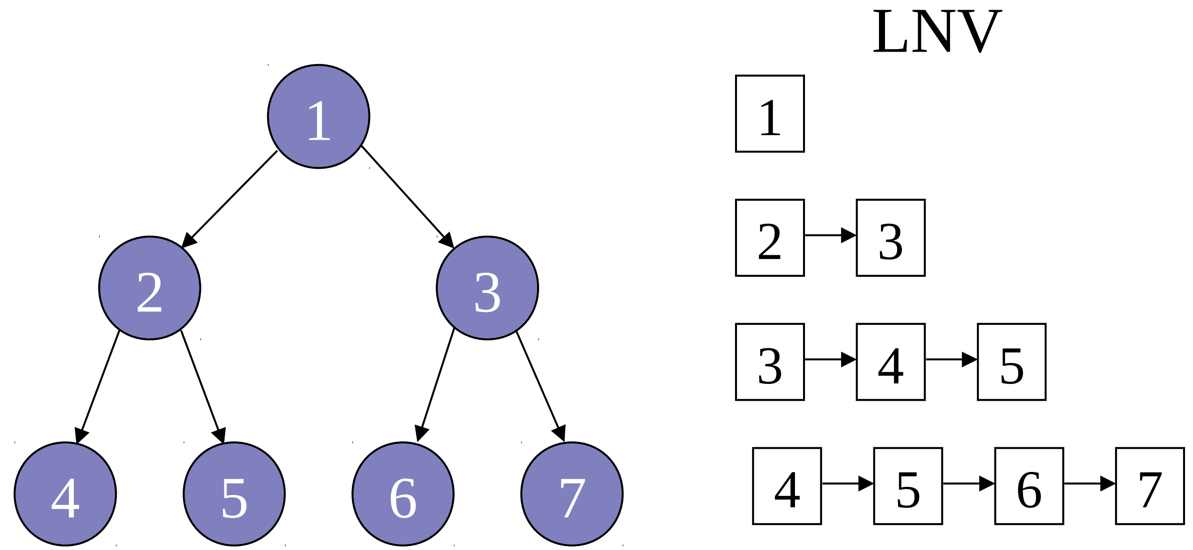
Esquema B&B

```
Algoritmo_B&B( )
{  contenedor<solucion> C;   solucion sol;
C.inserta(sol); // Raiz del arbol de estados
do {   sol=C.Selecciona_Siguiente_Nodo();
       if (sol.Factible()) {
           k = sol.Comp(); // Componente del e nodo;
           k++; // Siguiente componente
           for (sol.PrimerValorComp(k); sol.HayMasValores(k); sol.SigValComp(k))
               if (sol.Factible())  // Incluye las cotas
                    if (sol.NumComponentes()==sol.size())  //Es solucion?
                         sol.ActualizaSolucion();
                    else C.insert(sol);
       }
} while (!C.empty() )
}
```

¿Qué contenedor tenemos que utilizar?

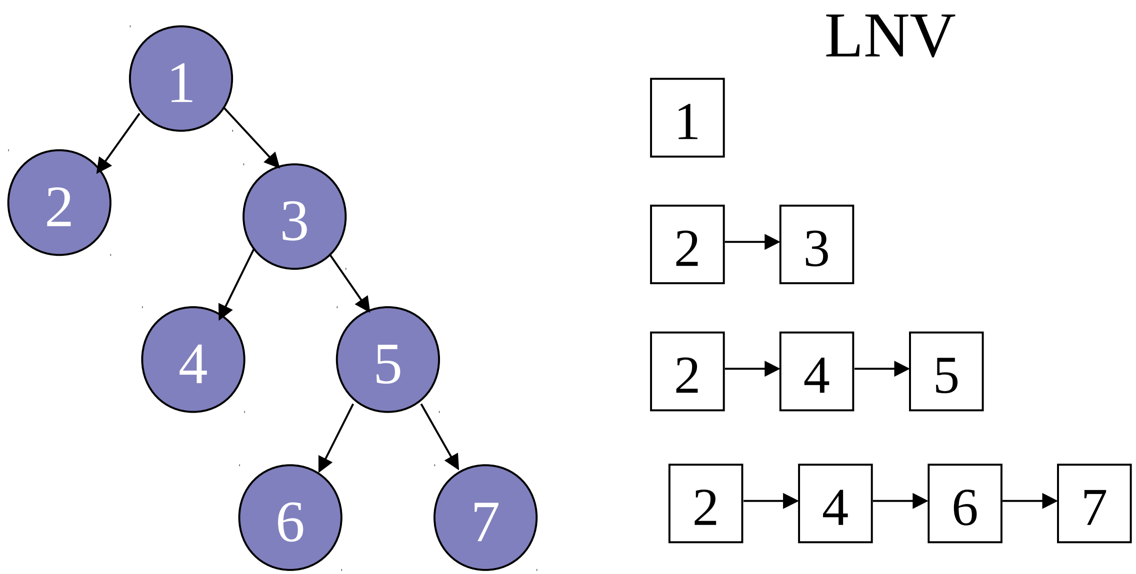
Qué contenedor tenemos que utilizar?

- El tipo de contenedor determina la estrategia o criterio de ramificación:
  - Cola para criterio FIFO
    - queue<solucion> C



Qué contenedor tenemos que utilizar?

- El tipo de contenedor determina la estrategia o criterio de ramificación:
  - Pila para criterio LIFO
    - stack<solucion> C



Qué contenedor tenemos que utilizar?

- Las estrategias FIFO y LIFO realizan una búsqueda sistemática “a ciegas”.
- Si dispusiéramos de una estimación del beneficio o coste óptimo que se puede conseguir a partir de un nodo,
- sería mejor buscar primero por los nodos con mejor valor estimado (mayor beneficio o menor costo)

Qué contenedor tenemos que utilizar?

- El tipo de contenedor determina la estrategia o criterio de ramificación:
  - Cola con prioridad para criterio LC (Least Cost)
    - priority_queue<solucion, comparacion<solucion> > C

    donde comparacion<solucion> es la función que nos permite ordenar los elementos en el Heap, depende de si el problema es maximizar o minimizar un determinado objetivo

Estrategia LC

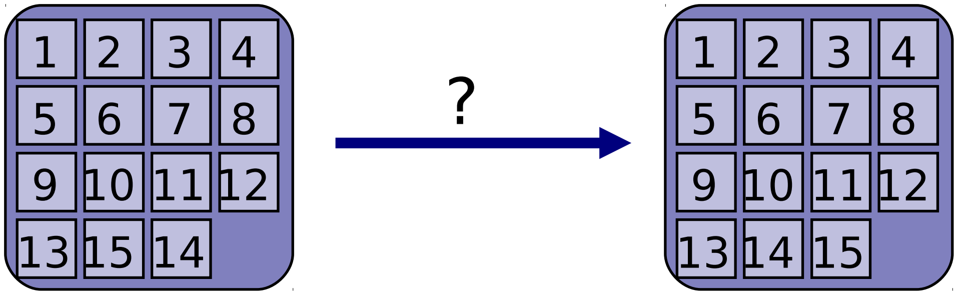
- En problemas de optimización, la estimación del beneficio o coste (lo prometedor que es un nodo) suele coincidir con la cota (superior o inferior, respect.) local.
- En problemas de otro tipo (p.e. juegos) se suele estimar el coste de hallar la solución a partir de cada nodo.

Ejemplo: el juego del 15

Samuel Loyd: El juego del 15 o “taken”.

Problema publicado en un periódico de Nueva York en 1878 y por cuya solución se ofrecieron 1000 dólares.

- El problema original:

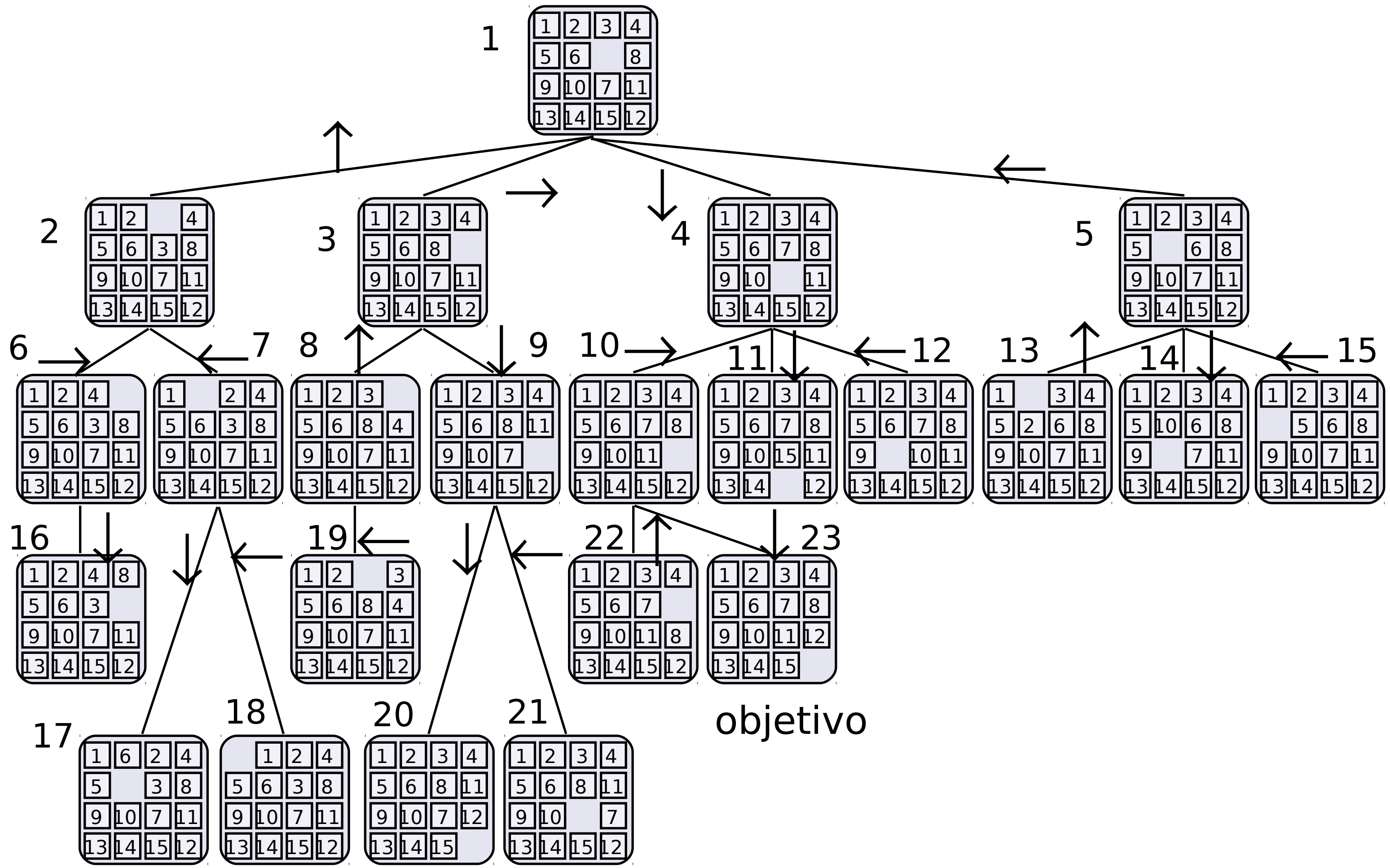


Problema de Lloyd — El objetivo

- Decisión: encontrar una secuencia de movimientos que lleve al objetivo
- Optimización: encontrar la secuencia de movimientos más corta

Ejemplo: el juego del 15

- Configuración: permutación de (1,2,...,16)
- Solución: secuencia de configuraciones que
  Empiezan en el estado inicial y acaban en el final.
  De cada configuración se puede pasar a la siguiente.
  No hay configuraciones repetidas



Ejemplo: el juego del 15

- **Problema muy difícil para backtracking**
  - **El árbol de búsqueda es potencialmente muy profundo (16! niveles), aunque puede haber soluciones muy cerca de la raíz.**

- **Se puede resolver con branch and bound (aunque hay métodos mejores): Algoritmo A***

- **Estimación del coste:**
  - número de fichas mal colocadas (puede engañar)
  - suma, para cada ficha, de la distancia a la posición donde le tocaría estar

Esquema B&B LC

```
Algoritmo_B&B( )
{  priority_queue<solucion> apo;   solucion sol; fin=false;
apo.insert(sol); // Raiz del arbol de estados
do {   sol=apo.top();
       if (sol.Factible()) {
          k = sol.Comp(); // Componente del e nodo;
          k++; // Siguiente componente
          for (sol.PrimerValorComp(k); sol.HayMasValores(k); sol.SigValComp(k))
              if (sol.Factible())  // Incluye las cotas
                 if (sol.NumComponentes()==sol.size())  //Es solucion?
                      sol.ActualizaSolucion();
                 else apo.insert(sol);
       } else fin=true;
} while ( (!apo.empty()) && (!fin))
}
```

Ejemplo Mochila 0/1

- Supongamos $n = 4$, $M = 16$, y los siguientes objetos:

i	b_i	p_i	b_i / p_i
1	\$40	2	\$20
2	\$30	5	\$6
3	\$50	10	\$5
4	\$10	5	\$2

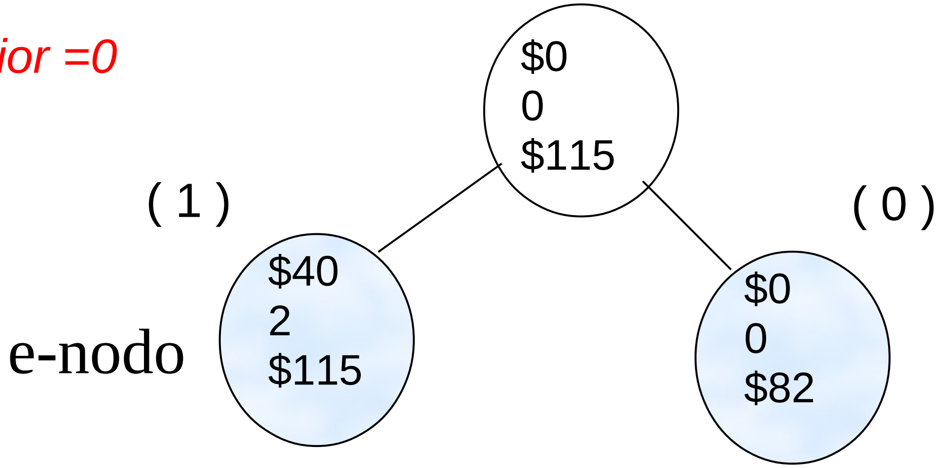
- Para el cálculo de las cotas locales utilizamos el greedy NO-0/1

ganancia
peso
estimador

Ejemplo

Cota Inferior =0

X – No Factible
O – No optimal



Objeto 1 [$40, 2]

Objeto 2 [$30, 5]

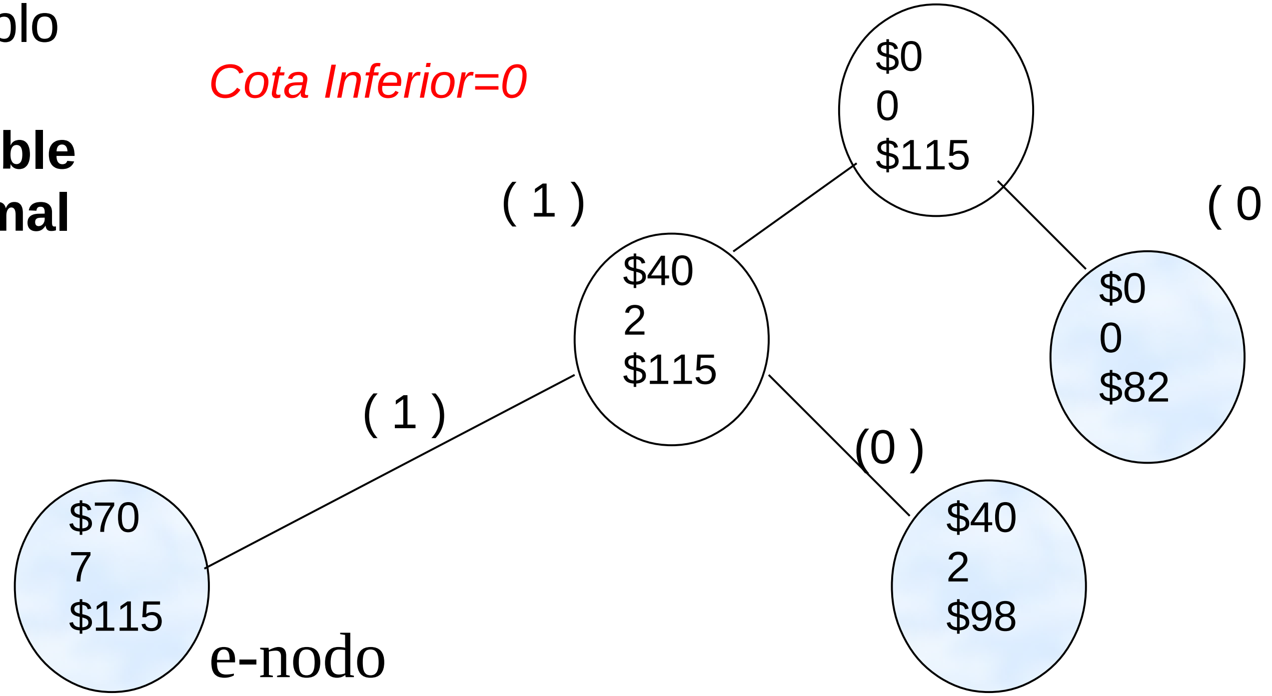
Objeto 3 [$50, 10]

Objeto 4 [$10, 5]

ganancia
peso
estimador

Ejemplo

X – No Factible
O – No optimal



Objeto 1 [$40, 2]

Objeto 2 [$30, 5]

Objeto 3 [$50, 10]

Objeto 4 [$10, 5]

ganancia
peso
estimador

Ejemplo

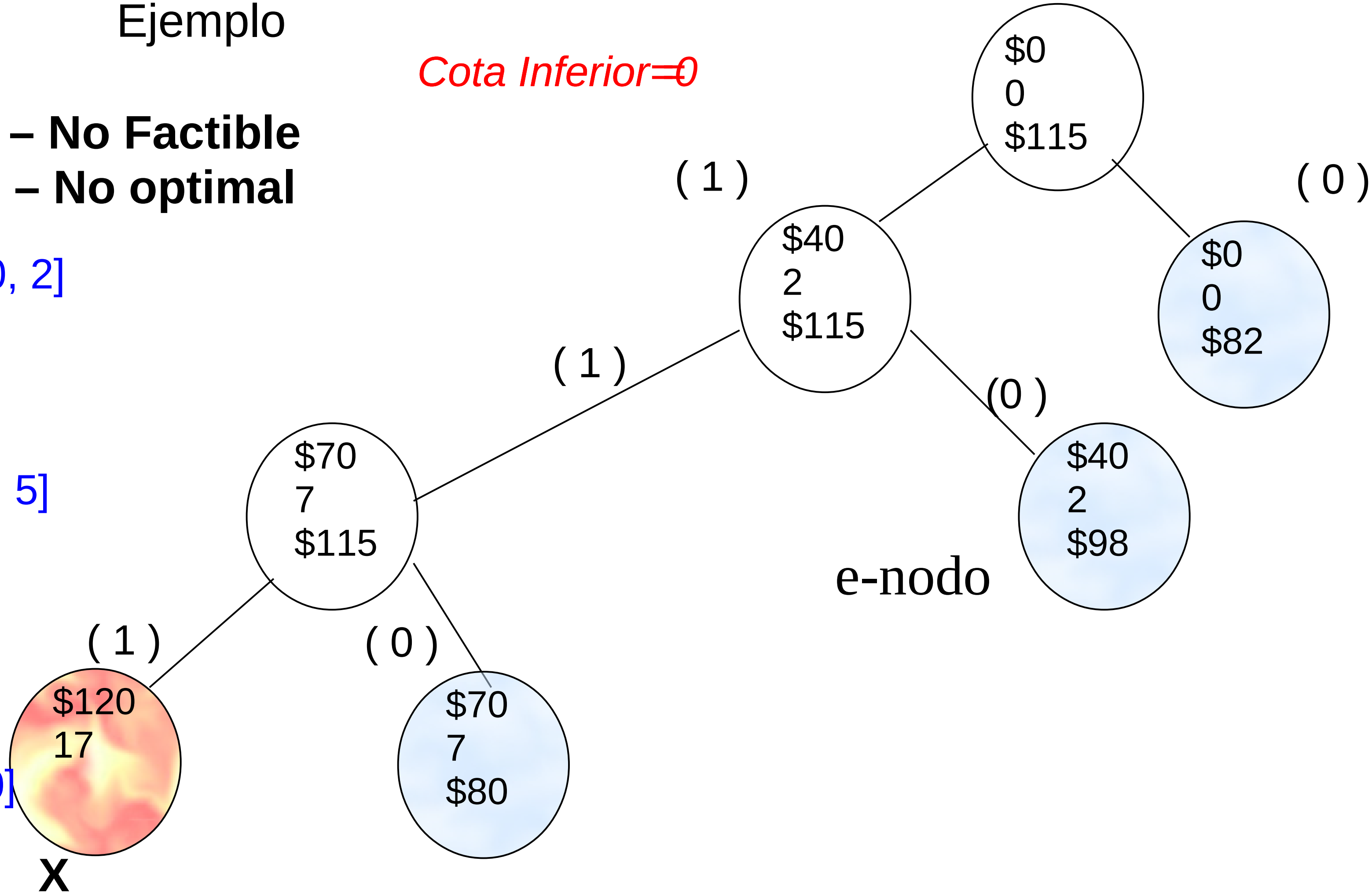
X – No Factible
O – No optimal

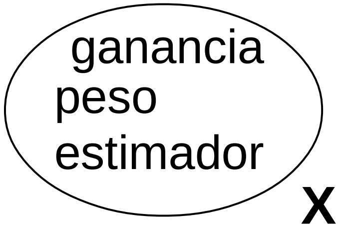
Objeto 1 [$40, 2]

Objeto 2 [$30, 5]

Objeto 3 [$50, 10]

Objeto 4 [$10, 5]





Ejemplo

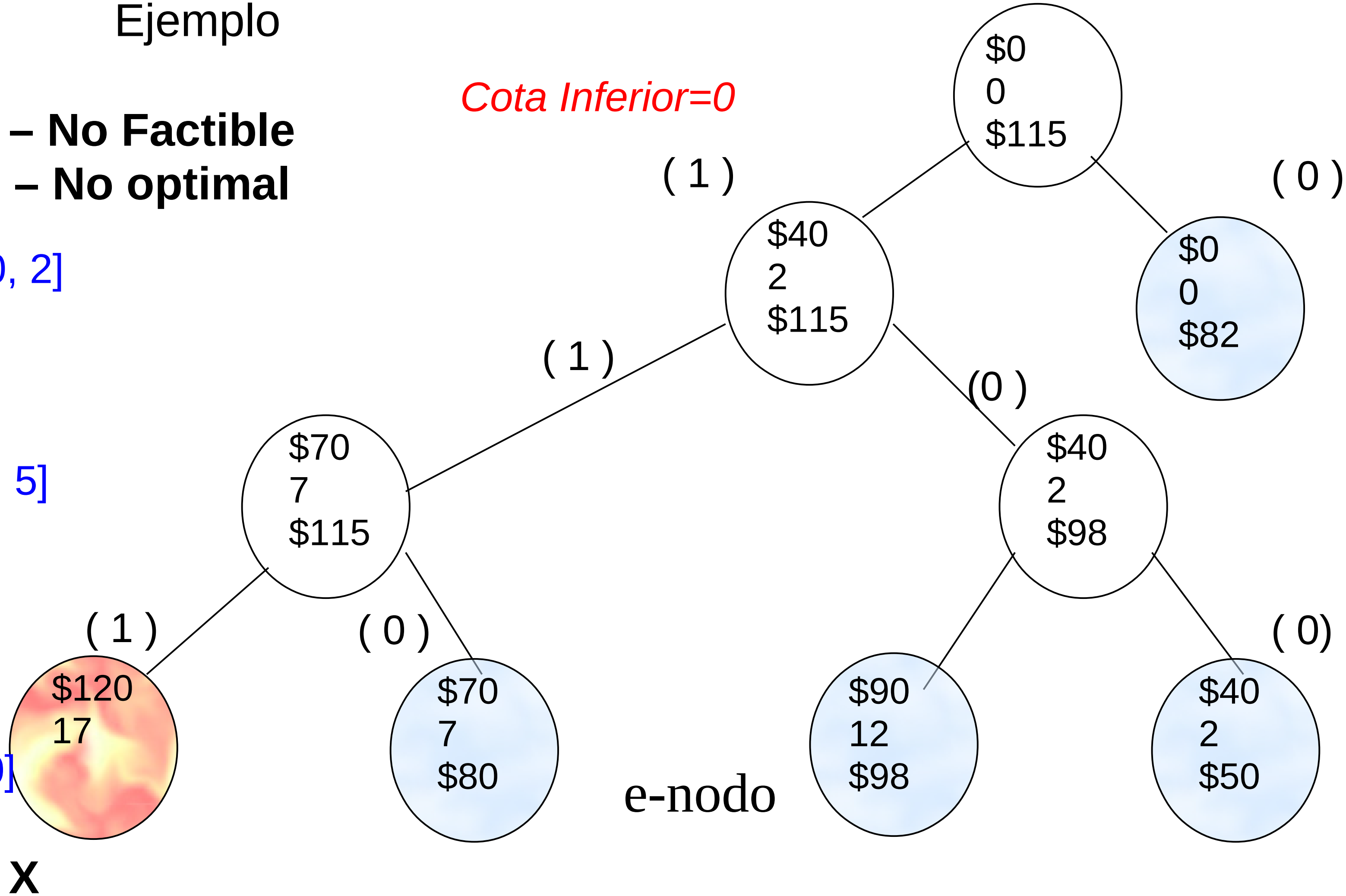
X – No Factible
O – No optimal

Objeto 1 [$40, 2]

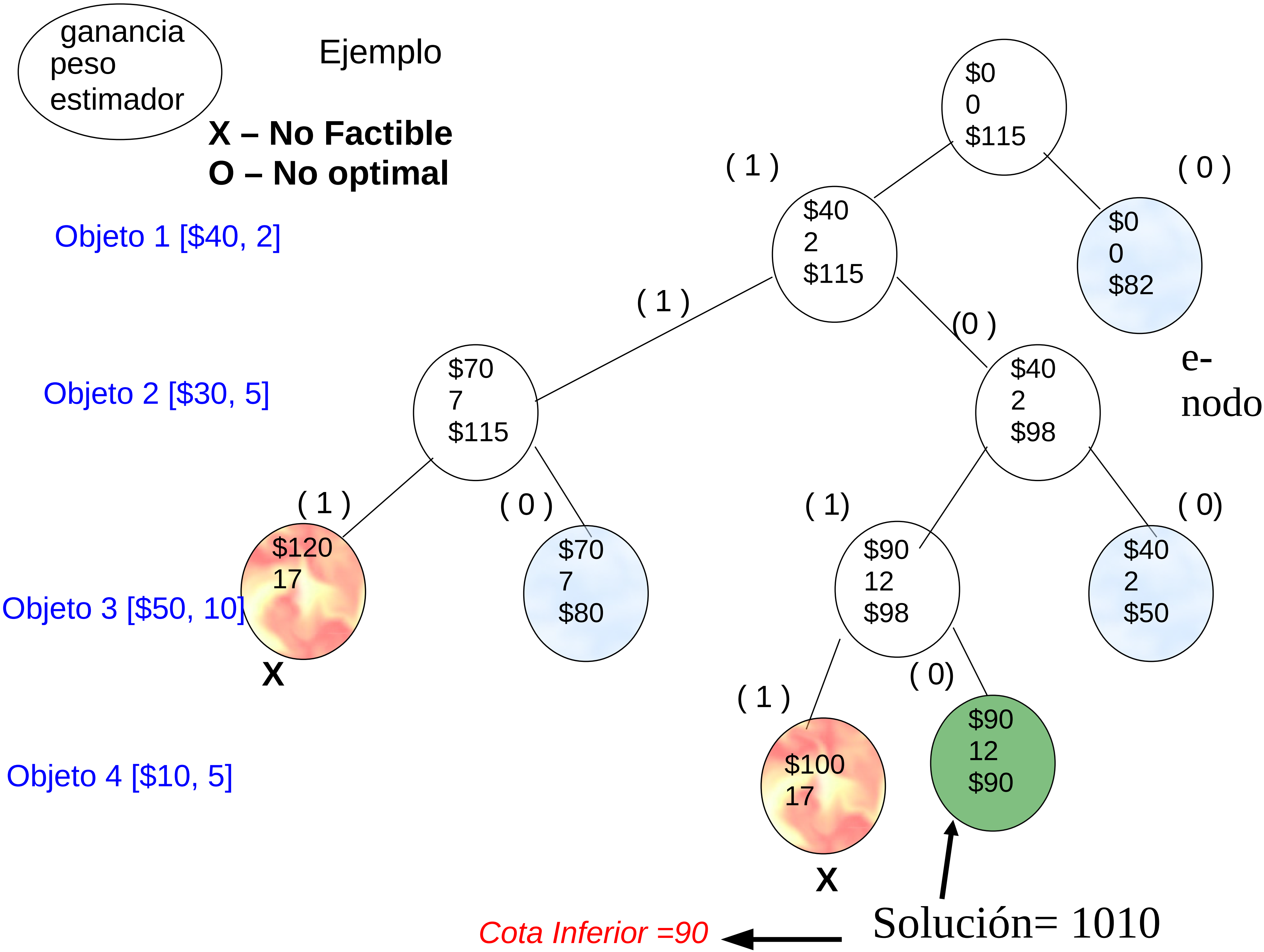
Objeto 2 [$30, 5]

Objeto 3 [$50, 10]

Objeto 4 [$10, 5]



Ejemplo


ganancia
peso
estimador
X – No Factible
O – No optimal
Objeto 1 [$40, 2]
Objeto 2 [$30, 5]
Objeto 3 [$50, 10]
Objeto 4 [$10, 5]
$0
0
$115
(1)
(0)
$40
2
$115
$0
0
$82
(1)
(0)
$70
7
$115
$40
2
$98
e-nodo
(1)
(0)
(1)
(0)
$120
17
X
$70
7
$80
$90
12
$98
$40
2
$50
(1)
(0)
$100
17
X
$90
12
$90
Solución= 1010
Cota Inferior =90

```
class Solucion {
public:
  Solucion(const int tam_max);
  bool Factible( ) const;
  void PrimerValorComp(int k);
  void SigValComp(int k);
  TipoBase Comp(int k) const;
  float CotaLocal() const
  bool HayMasValores(int k) const
  float Evalua();
  int NumComponentes() const;
  bool EsSolucion() const
  int CompActual() const
  bool operator<( const Solucion & s) const
   {return Estimador < s.Estimador;}

private:
  vector<int> X;   // Almacenamos la solucion
  int pos_e;     // Posicion  de la ultima decision en X
  float GA; // Ganancia Acumulada
  float VO; // Volumen ocupado
  float Estimador;  // Valor del estimador para el nodo X

};
```

```
Solucion Branch_and_Bound(int n_objetos )
{
  priority_queue<Solucion> Q;
  Solucion n_e(n_objetos), mejor_solucion(n_objetos) ;  //nodoe en expansion
  int k;
  float CG=0;   // Cota Global
  float ganancia_actual;

 Q.push(n_e);
 while ( !Q.empty() && (Q.top().CotaLocal() > CG) ){
    n_e = Q.top();
   Q.pop();
    k = n_e.CompActual();
    for ( n_e.PrimerValorComp(k+1); n_e.HayMasValores(k+1);n_e.SigValComp(k+1)) {
            if ( n_e.EsSolucion() ){
              ganancia_actual = n_e.Evalua();
             if (ganancia_actual > CG) { CG = ganancia_actual; mejor_solucion = n_e; }
           }  else if ( n_e.Factible( ) && n_e.CotaLocal()>CG )
                 Q.push( n_e );
     }
  }
 return mejor_solucion;
}
```

ganancia
peso
estimador

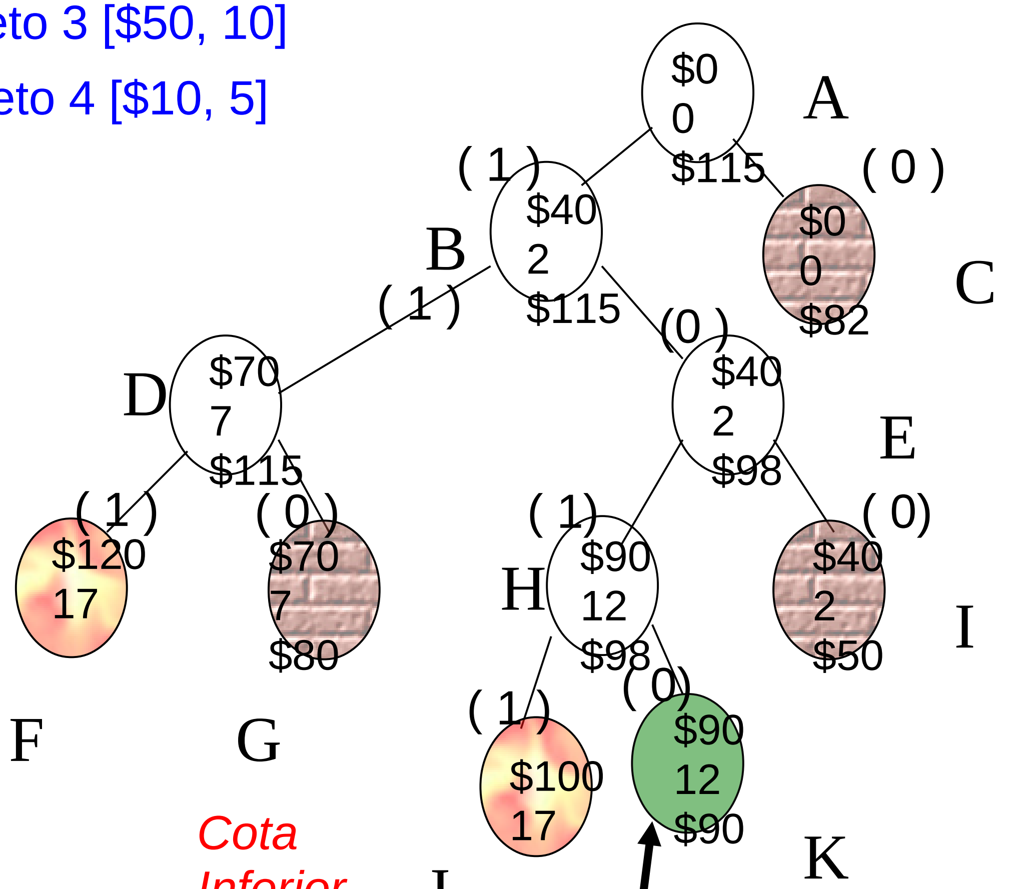
Ejemplo

Objeto 1 [$40, 2]

Objeto 2 [$30, 5]

Objeto 3 [$50, 10]

Objeto 4 [$10, 5]



Cota Inferior =90

Solución= 1010

Cola con Prioridad C

Estado		Cota Local
A	X X X X	115
B	1 X X X	115
C	0 X X X	82
D	1 1 X X	115
E	1 0 X X	98
C	0 X X X	82
E	1 0 X X	98
C	0 X X X	82
G	1 1 0 X	80
H	1 0 1 X	98
C	0 X X X	82
G	1 1 0 X	80
I	1 0 0 X	50
K	1 0 1 0	90 Soluc
C	0 X X X	82
G	1 1 0 X	80
I	1 0 0 X	50

Problema de asignación

Dadas n personas y n trabajos, con C[i][j] el costo que tiene que la persona i-ésima realice el trabajo j-ésimo.

Ejemplo

	Tr 1	*Tr* 2	*Tr* 3	*Tr* 4
Persona *a*	9	2	7	8
Persona *b*	6	4	3	7
Persona *c*	5	8	1	8
Persona *d*	7	6	9	4

Se pide:

Realizar una asignación de tareas de forma que:

- Todos los trabajos sean ejecutados
- Todas las personas realicen un trabajo (no hay persona ociosa)
- El costo de la asignación es mínimo

Problema de asignación

Espacio de estados:

hay que tomar n decisiones, cada decisión d_i se corresponde con el trabajo que se le asigna a la persona i-ésima.

Restricciones explícitas:

x[i] toma valores en {1..n}

Restricciones implícitas:

Un trabajo se le asigna a una única persona:

x[i] != x[j], para todo i!=j

Espacio de estados:
Árbol de permutaciones

Problema de asignación: Cotas

Cota Global:

Es un problema de minimización, por tanto actúa como cota superior

Greedy: (Cual?)

Ejemplo

	Tr 1	*Tr* 2	*Tr* 3	*Tr* 4
Persona *a*	9	2	7	8
Persona *b*	6	4	3	7
Persona *c*	5	8	1	8
Persona *d*	7	6	9	4

Problema de asignación: Cotas

Cota Global:

Es un problema de minimización, por tanto actúa como cota superior

Greedy: (Cual?)

Ejemplo

	Tr 1	*Tr* 2	*Tr* 3	*Tr* 4
Persona *a*	9	2	7	8
Persona *b*	6	4	3	7
Persona *c*	5	8	**1**	8
Persona *d*	7	6	9	4

Cota = 1+2+4+6 = 13

Problema de asignación: Cotas

Cotas Locales

Para cada nodo, una cota inferior del coste de la mejor solución que se puede alcanzar desde el nodo

Para el nodo raíz: Cualquier solución debe ser mayor que:

Mínimos filas 2 + 3 + 1 + 4

(o mínimos columnas 5 + 2 + 1 + 4)

Ejemplo

	Tr 1	*Tr* 2	*Tr* 3	*Tr* 4
Persona *a*	9	2	7	8
Persona *b*	6	4	3	7
Persona *c*	5	8	1	8
Persona *d*	7	6	9	4

Problema de asignación: Cotas

Cotas Locales

<u>Para un nodo con algunas asignaciones</u>: p.e. a<--Tr 3:

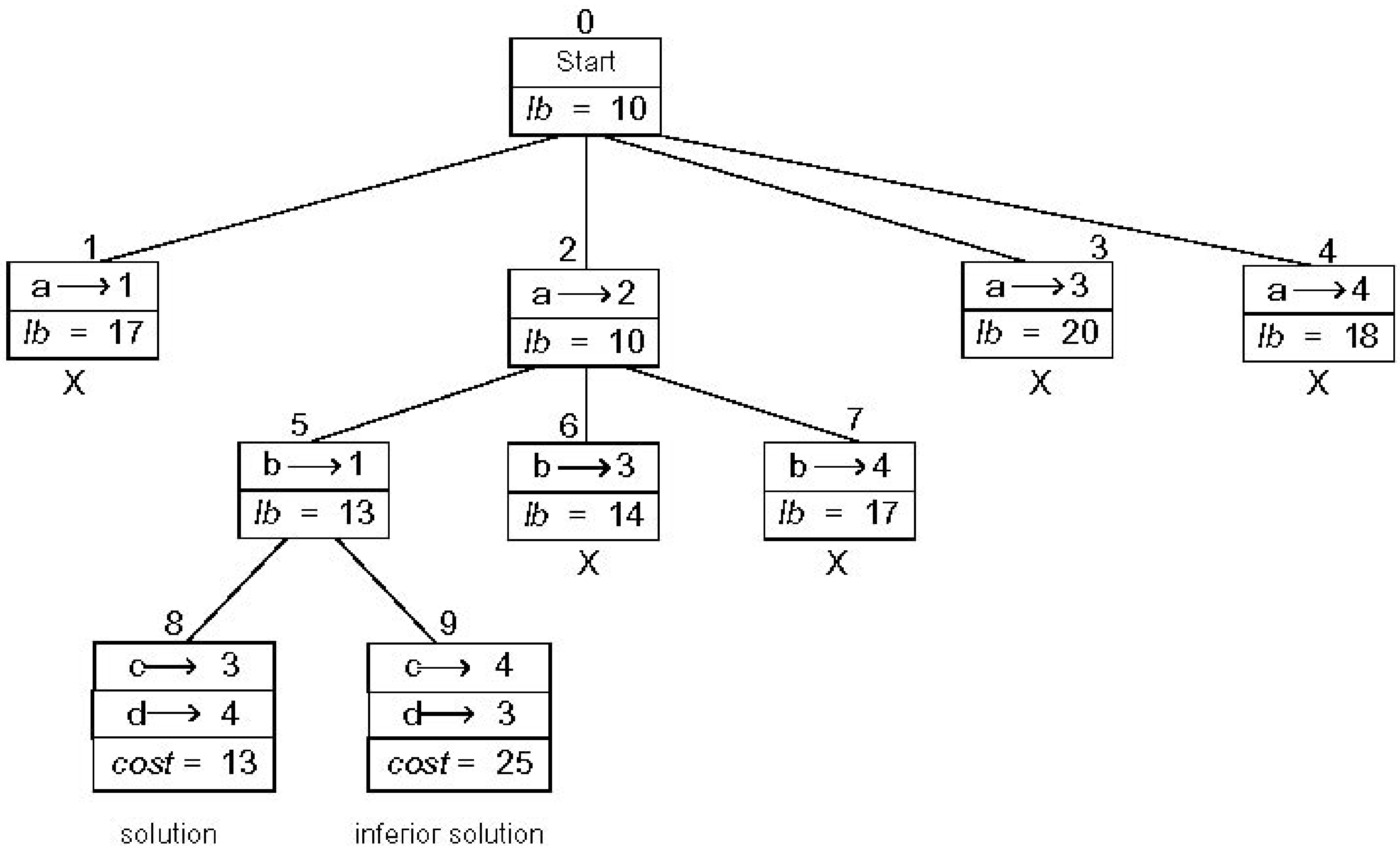
7 + Mínimos filas no asignadas 4 + 5 + 4

Ejemplo

	Tr 1	*Tr* 2	*Tr* 3	*Tr* 4
Persona *a*	9	2	7	8
Persona *b*	6	4	3	7
Persona *c*	5	8	1	8
Persona *d*	7	6	9	4

	Tr 1	*Tr* 2	*Tr* 3	*Tr* 4
Persona *a*	~~9~~	~~2~~	~~7~~	~~8~~
Persona *b*	6	4	~~3~~	7
Persona *c*	5	8	~~1~~	8
Persona *d*	7	6	~~9~~	4

Árbol de estados



En cada nodo se indica la decisión y el valor de la cota inferior (lb)

Problema del viajante de comercio

- Podemos usar cualquiera de las cotas inferiores comentadas anteriormente (aunque en B&B interesa que las cotas sean bastante precisas).
- Emplearemos en el ejemplo una variación de la cota 2 (el costo del camino acumulado más la suma de los menores arcos salientes de cada vértice del que aun no se ha salido)

Ejemplo

- Dada la siguiente matriz de adyacencias, ¿cuál es el costo mínimo posible de visitar todos los nodos una sola vez?

$$
\begin{pmatrix}
0 & 14 & 4 & 10 & 20 \\
14 & 0 & 7 & 8 & 7 \\
4 & 5 & 0 & 7 & 16 \\
11 & 7 & 9 & 0 & 2 \\
18 & 7 & 17 & 4 & 0
\end{pmatrix}
\begin{array}{l}
\longrightarrow \text{Mínimo} = \boldsymbol{4} \\
\longrightarrow \text{Mínimo} = \boldsymbol{7} \\
\longrightarrow \text{Mínimo} = \boldsymbol{4} \\
\longrightarrow \text{Mínimo} = \boldsymbol{2} \\
\longrightarrow \text{Mínimo} = \boldsymbol{4}
\end{array}
$$

TOTAL = 21

Ejemplo

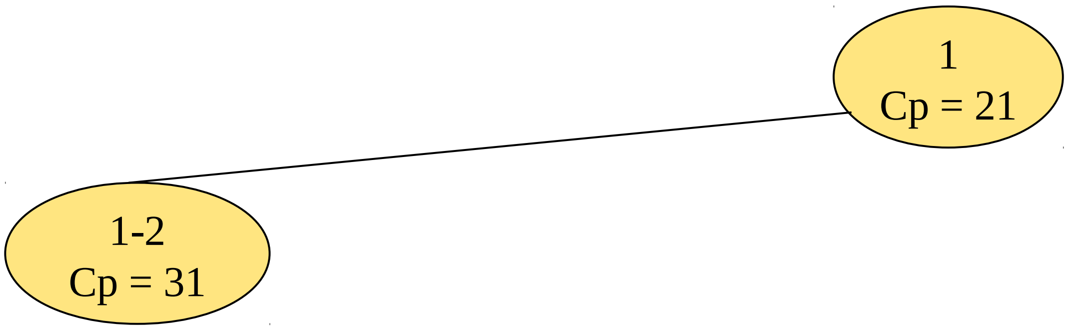
$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$

1
Cp = 21

Costo mínimo = ∞

Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$



Costo mínimo = ∞

Cálculo del Costo posible:

Acumulado de 1-2 : **14**

Más mínimo de 2-3, 2-4 y 2-5: **7**

Más mínimo de 3-1, 3-4 y 3-5: **4**

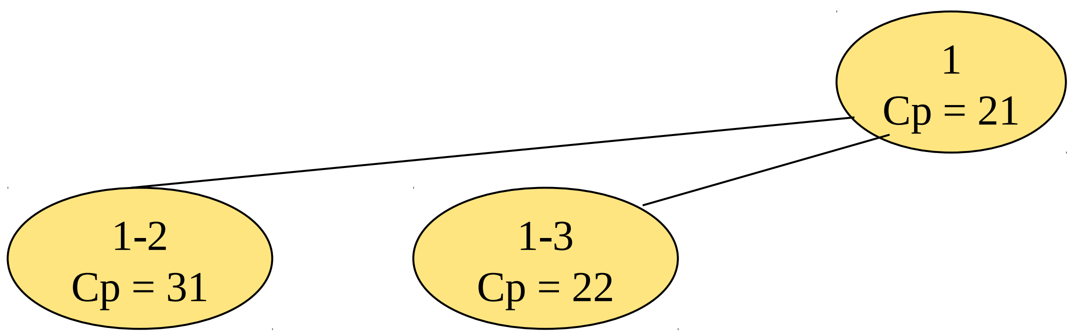
Más mínimo de 4-1, 4-3 y 4-5: **2**

Más mínimo de 5-1, 5-3 y 5-4: **4**

TOTAL = 31

Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$



Costo mínimo = ∞

Cálculo del Costo posible:

Acumulado de 1-3 : **4**

Más mínimo de 3-2, 3-4 y 3-5: **5**

Más mínimo de 2-1, 2-4 y 2-5: **7**

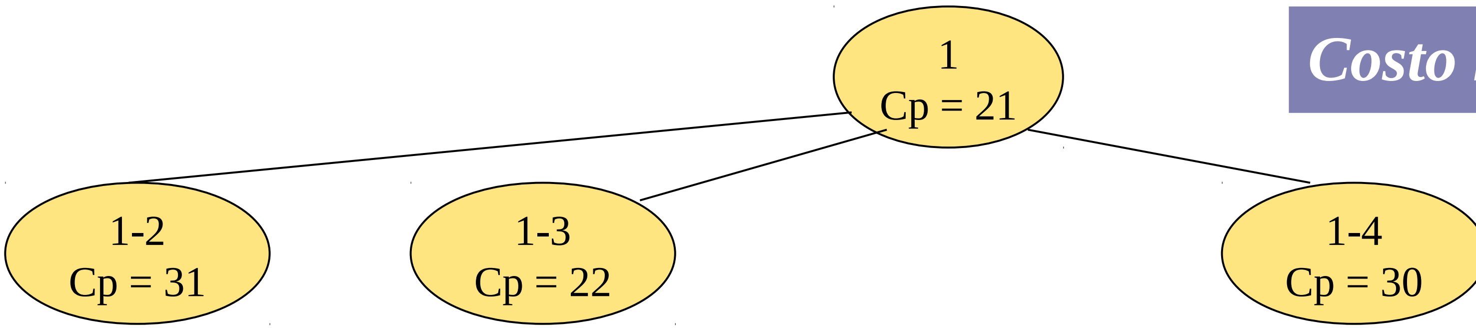
Más mínimo de 4-1, 4-2 y 4-5: **2**

Más mínimo de 5-1, 5-2 y 5-4: **4**

TOTAL = 22

Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$



Costo mínimo = ∞

Cálculo del Costo posible:

Acumulado de 1-4 : **10**

Más mínimo de 4-2, 4-3 y 4-5: **2**

Más mínimo de 3-1, 3-2 y 3-5: **4**

Más mínimo de 2-1, 2-3 y 2-5: **7**

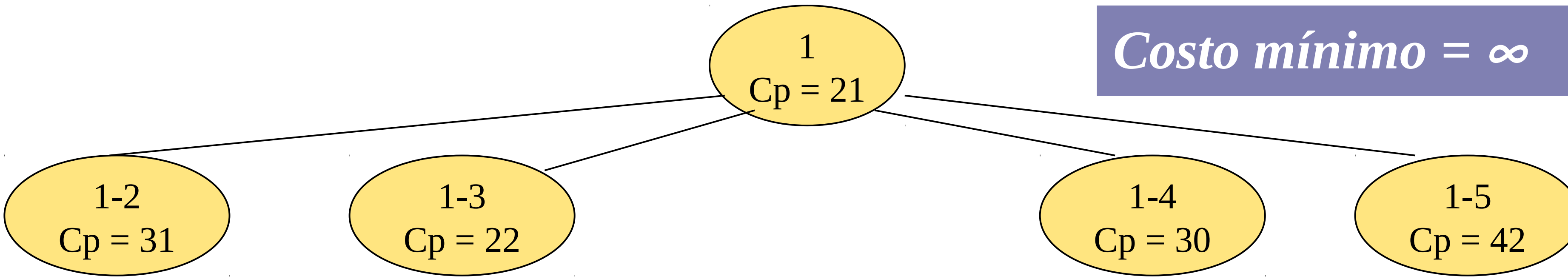
Más mínimo de 5-1, 5-2 y 5-3: **7**

TOTAL = 30

Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$

Costo mínimo = ∞



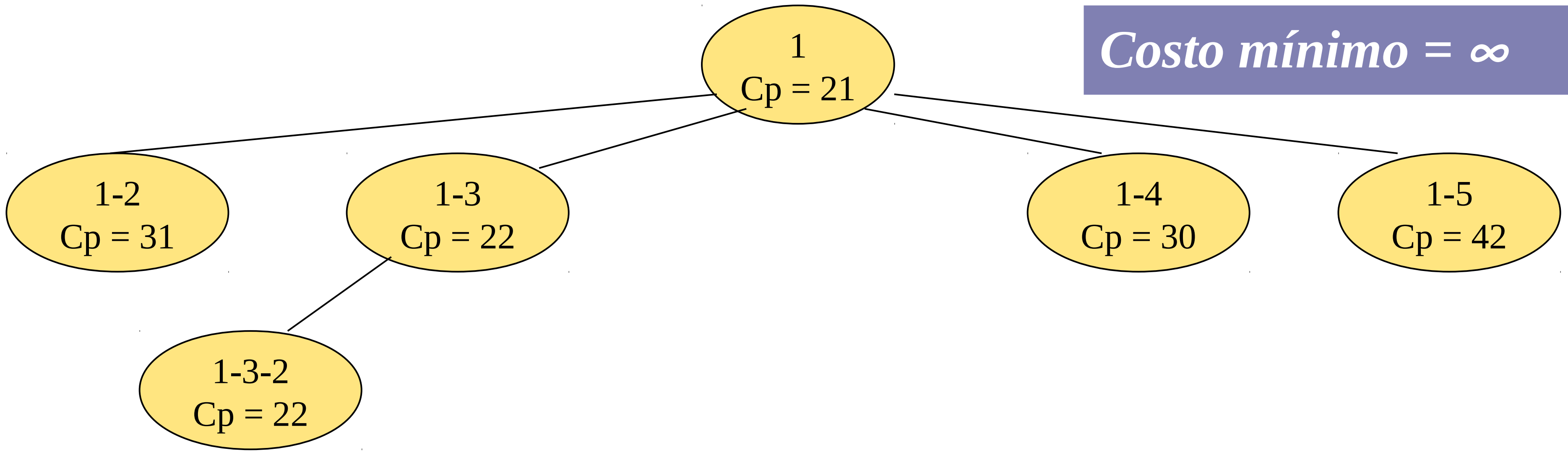
¿Cuál es el mejor nodo para expandir?

Cálculo del Costo posible:
Acumulado de 1-5 : **20**
Más mínimo de 5-2, 5-3 y 5-4: **4**
Más mínimo de 4-1, 4-2 y 4-3: **7**
Más mínimo de 3-1, 3-2 y 3-4: **4**
Más mínimo de 2-1, 2-3 y 2-4: **7**
TOTAL = 42

Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$

Costo mínimo = ∞



<u>Cálculo del Costo posible:</u>

Acumulado de 1-3-2 : **9**

Más mínimo de 2-4 y 2-5: **7**

Más mínimo de 4-1 y 4-5: **2**

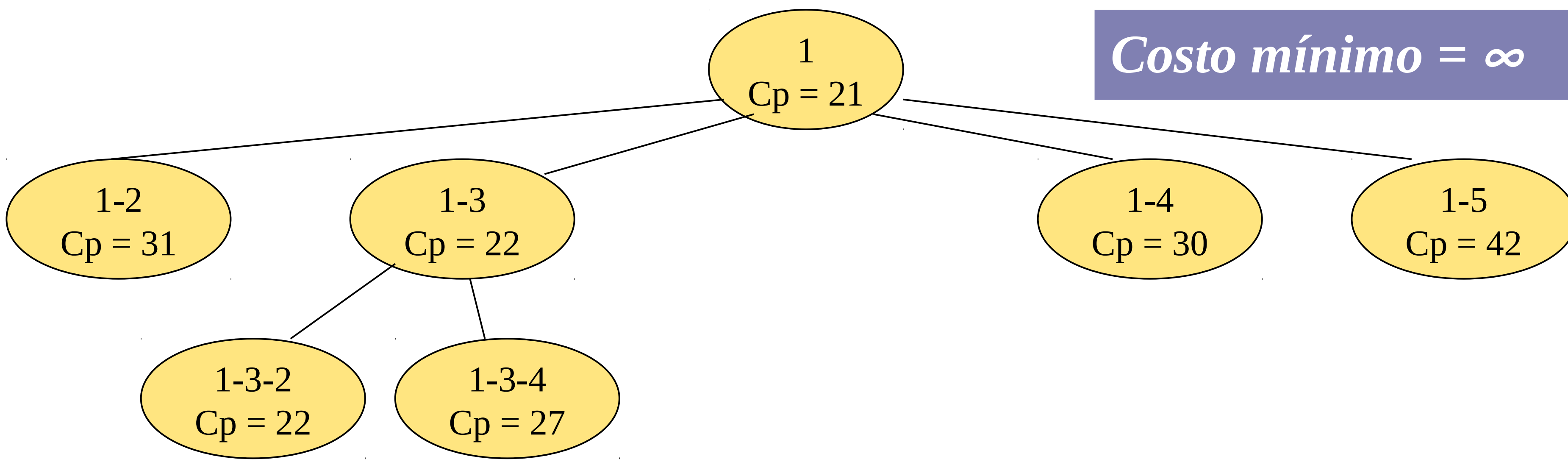
Más mínimo de 5-1 y 5-4: **4**

TOTAL = 22

Ejemplo

0	14	4	10	20
14	0	7	8	7
4	5	0	7	16
11	7	9	0	2
18	7	17	4	0

Costo mínimo = ∞



<u>Cálculo del Costo posible:</u>

Acumulado de 1-3-4 : **11**

Más mínimo de 4-2 y 4-5: **2**

Más mínimo de 2-1 y 2-5: **7**

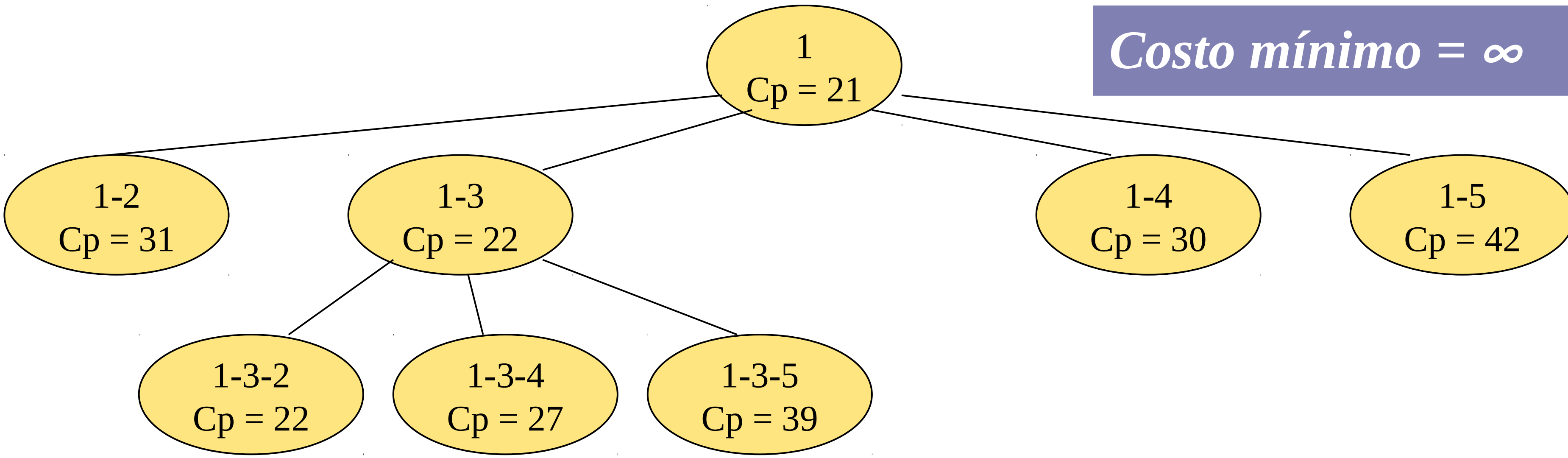
Más mínimo de 5-1 y 5-2: **7**

TOTAL = 27

Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$

Costo mínimo = ∞



¿Cuál es el mejor nodo para expandir?

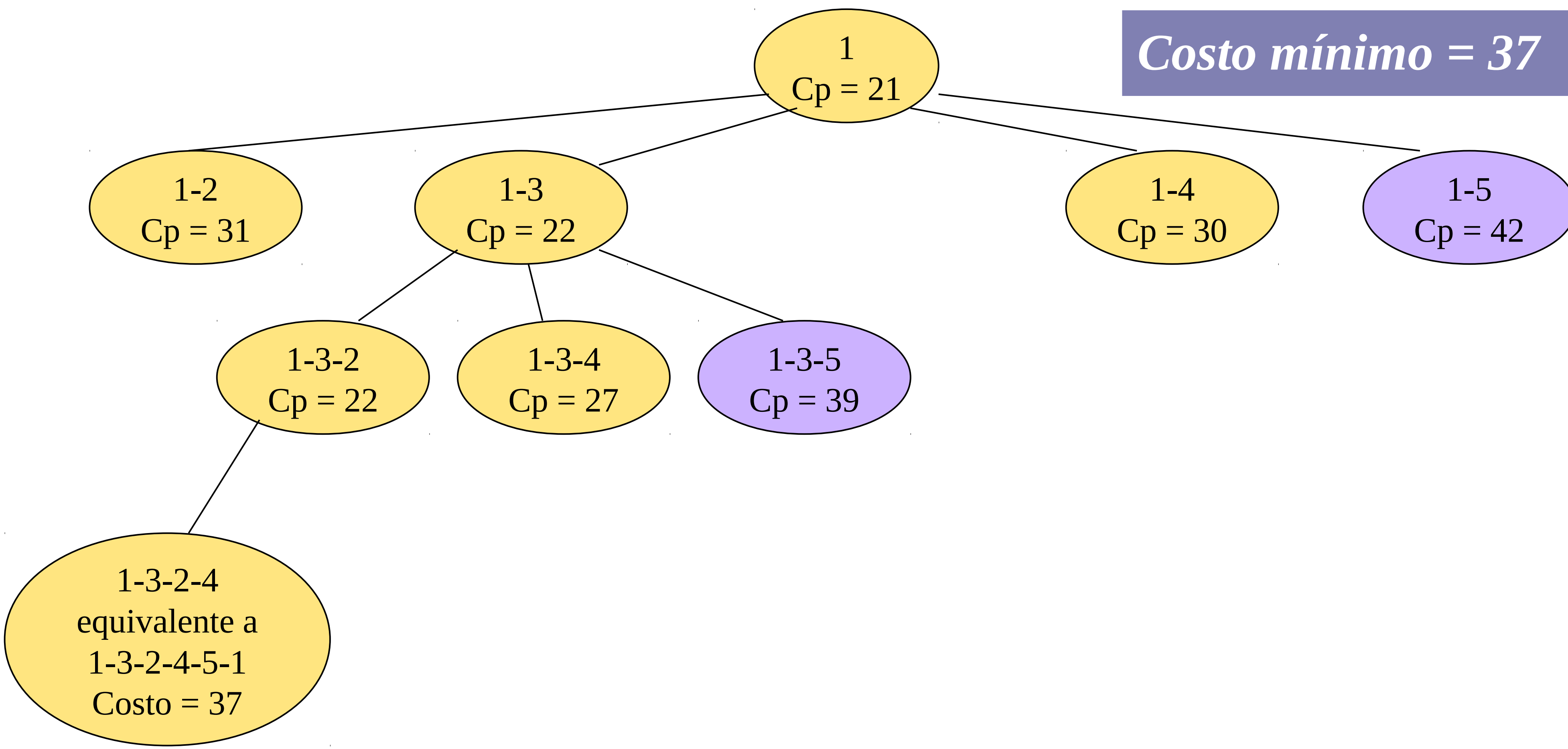
Cálculo del Costo posible:

Acumulado de 1-3-5 : **20**
Más mínimo de 5-2 y 5-4: **4**
Más mínimo de 2-1 y 2-4: **8**
Más mínimo de 4-1 y 4-2: **7**
TOTAL = 39

Ejemplo

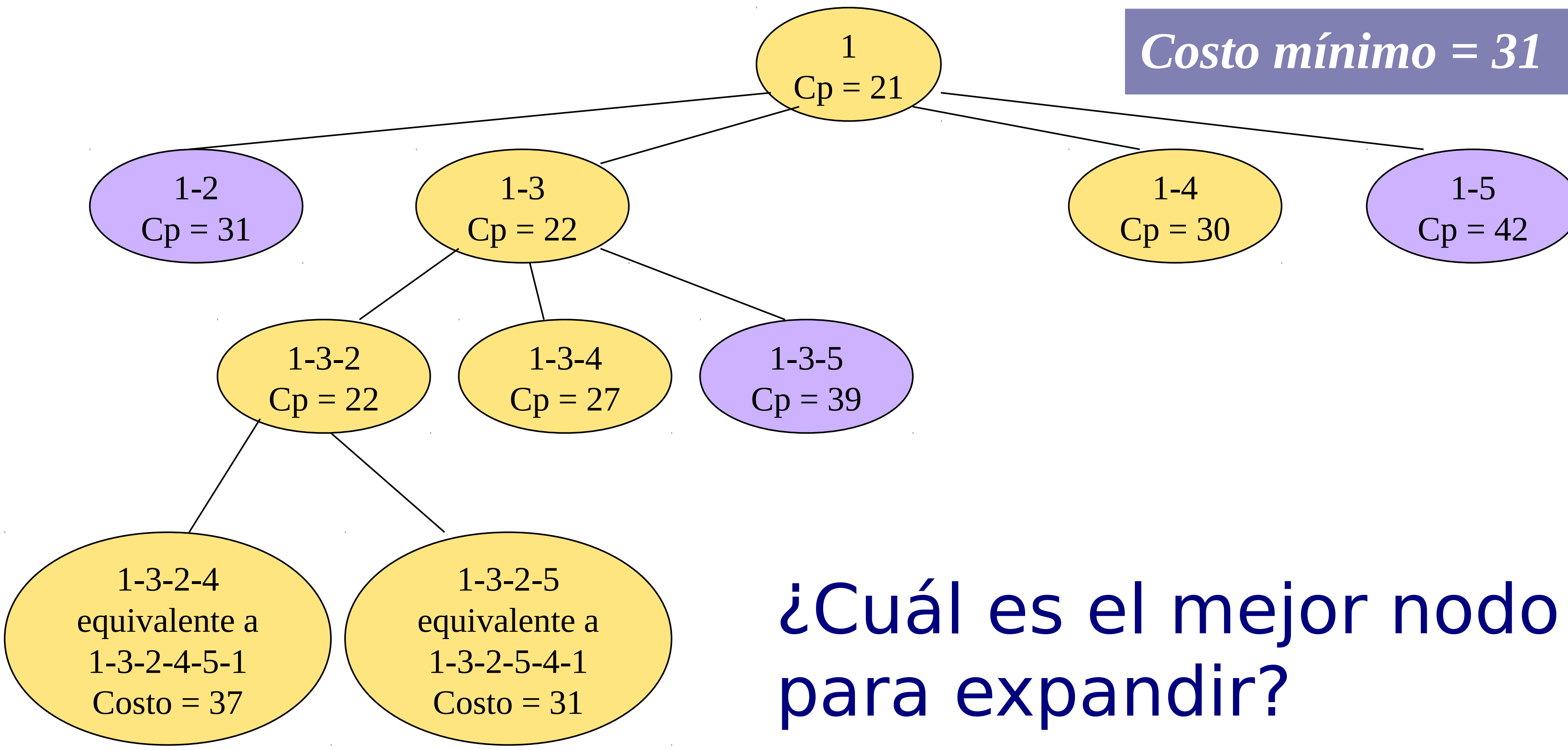
$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$

Costo mínimo = 37



Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$



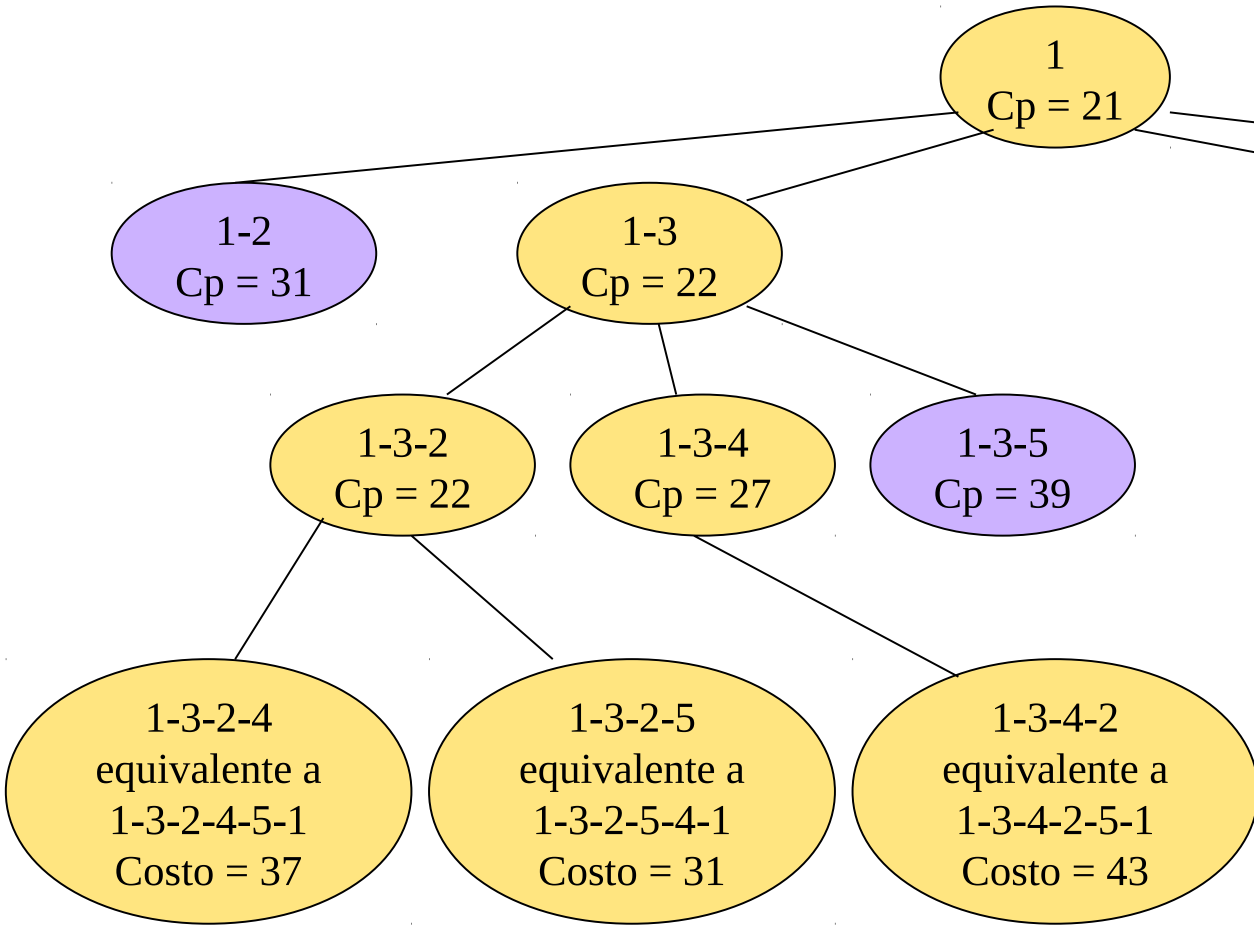
Costo mínimo = 31

¿Cuál es el mejor nodo para expandir?

Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$

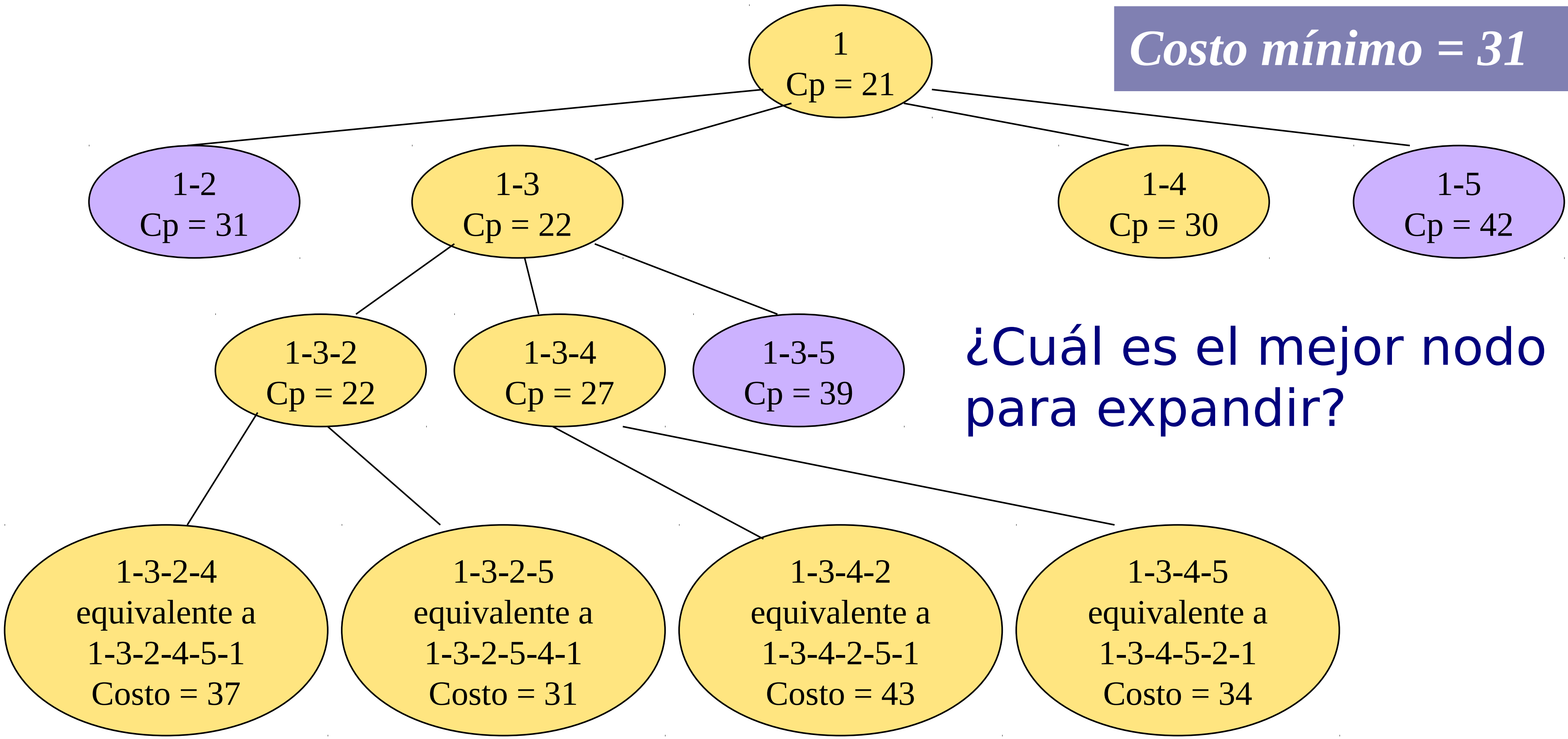
Costo mínimo = 31



Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$

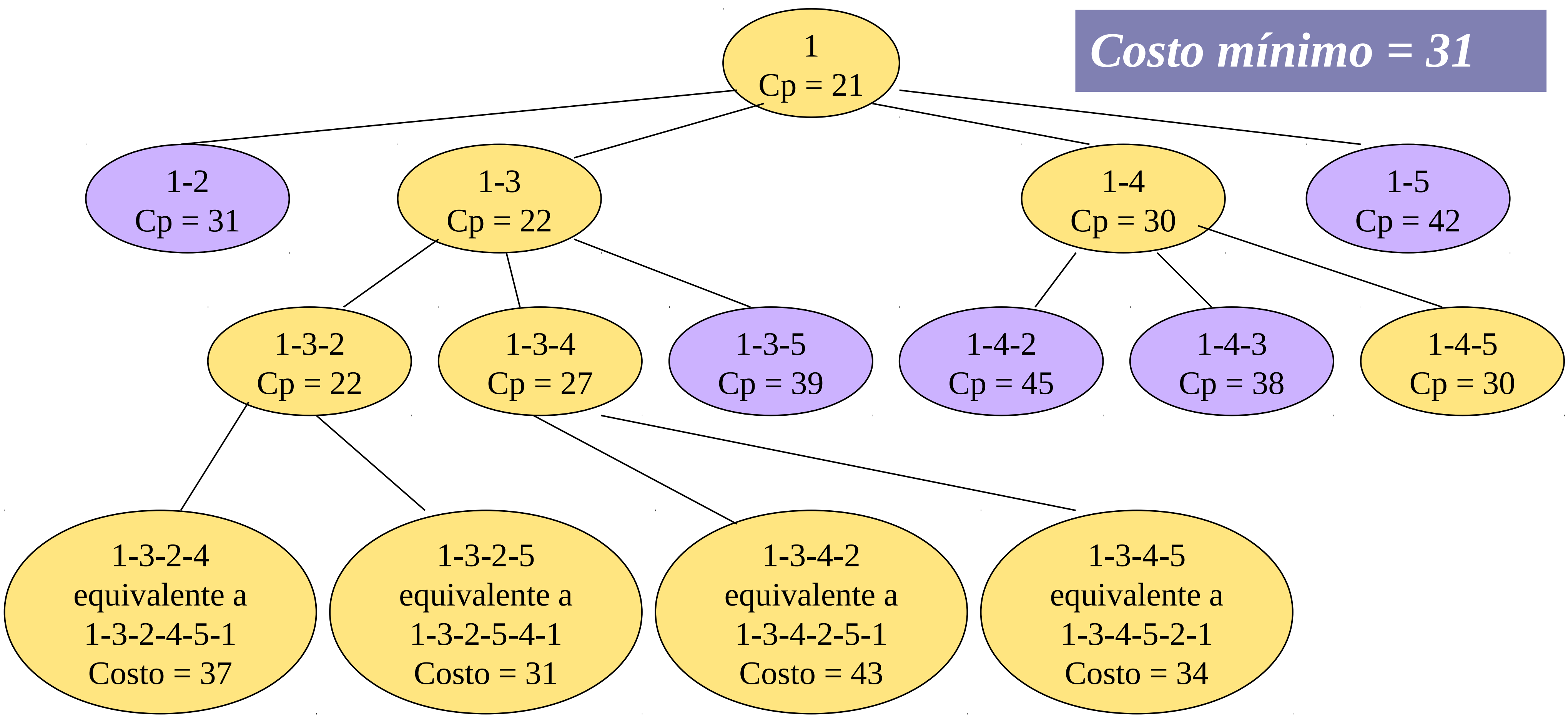
Costo mínimo = 31



¿Cuál es el mejor nodo para expandir?

Ejemplo

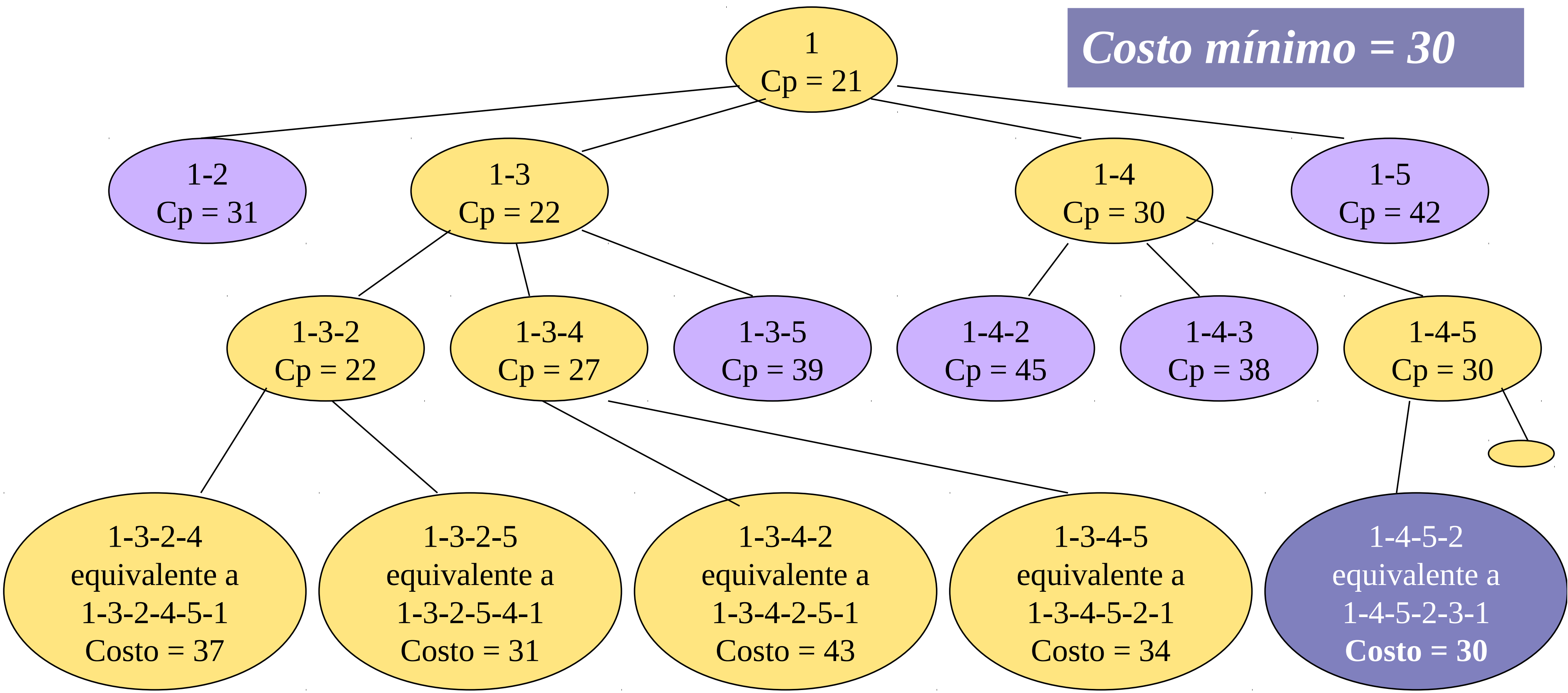
$$
\begin{pmatrix}
0 & 14 & 4 & 10 & 20 \\
14 & 0 & 7 & 8 & 7 \\
4 & 5 & 0 & 7 & 16 \\
11 & 7 & 9 & 0 & 2 \\
18 & 7 & 17 & 4 & 0
\end{pmatrix}
$$



¿Cuál es el mejor nodo para expandir?

Ejemplo

$$\begin{pmatrix} 0 & 14 & 4 & 10 & 20 \\ 14 & 0 & 7 & 8 & 7 \\ 4 & 5 & 0 & 7 & 16 \\ 11 & 7 & 9 & 0 & 2 \\ 18 & 7 & 17 & 4 & 0 \end{pmatrix}$$



Eficiencia en B&B

- Al igual que BK depende de:
  - El tiempo necesario para generar la siguiente componente de la solución, sol(k)
  - El número de sol(k) que satisfacen las restricciones explícitas
  - El tiempo de determinar la función de factibilidad (acota las soluciones)
  - El número de sol(k) que satisfacen la función de factibilidad
- Pero además de
  - El tiempo necesario para gestionar el contenedor de nodos vivos

Eficiencia en Branch and bound

- Básicamente, el tiempo de ejecución depende de:
  - **Número de nodos recorridos**: depende de la efectividad de la poda.
  - **Tiempo gastado en cada nodo**: tiempo de hacer las estimaciones de coste y tiempo de manejo de la lista de nodos vivos.
- En el peor caso, el tiempo es igual que el de un algoritmo con backtracking (o peor si tenemos en cuenta el tiempo que requiere la LNV).
- En el caso promedio se suelen obtener mejoras respecto al backtracking.
- ¿Cómo hacer que un algoritmo B&B sea más eficiente?
  - **Hacer estimaciones de costo muy precisas**: Se realiza una poda exhaustiva del árbol. Se recorren menos nodos pero se gasta mucho tiempo en realizar las estimaciones.
  - **Hacer estimaciones de costo poco precisas**: Se gasta poco tiempo en cada nodo, pero el número de nodos puede ser muy elevado. No se hace mucha poda.
- Se debe buscar un equilibrio entre la exactitud de las cotas y el tiempo de calcularlas.