

Análisis y diseño de algoritmos II

Ramificación y poda *(Branch and bound)*

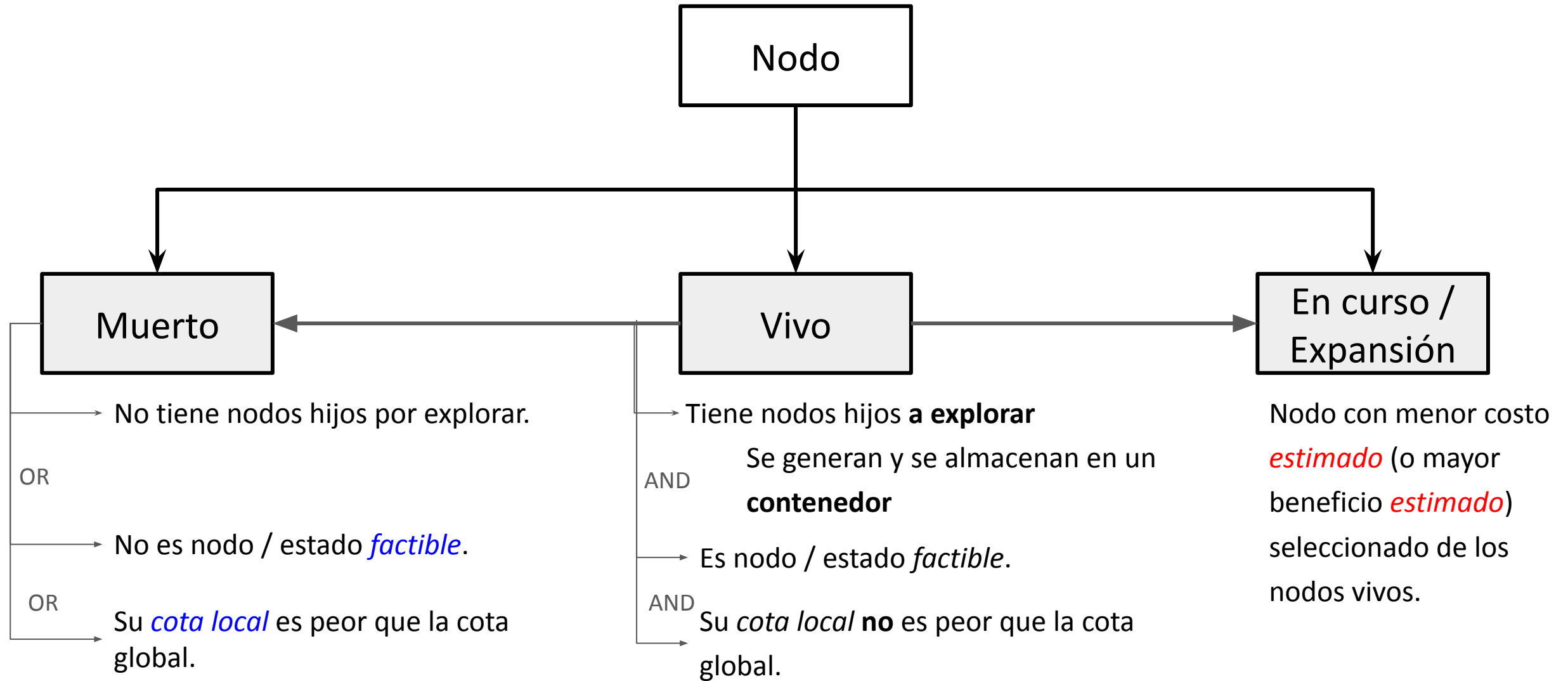
Objetivos

- Aplicar la técnica de ramificación y poda para la resolución de un problema de asignación uno a uno
 - Comprender y adquirir un proceso para resolución de problemas con la técnica
 - Poner en práctica los conceptos vistos en teoría
- Adquirir herramientas para analizar, diseñar e implementar soluciones algorítmicas basadas en ramificación y poda
- Diferenciar de *backtracking*

Agenda

- Breve repaso de los conocimientos necesarios para entender el tema
- Presentación del problema de asignación 1:1
- Resolución aplicando los conceptos generales de ramificación y poda vistos en teoría
- Presentación y resolución del problema de la mochila
- Consideraciones finales

Repaso ramificación y poda



En cualquier instante pueden existir varios nodos vivos y muertos y sólo uno en expansión.

Problema I

Para mejorar su puntuación en redes sociales, un bar está evaluando la posibilidad de armar promociones para ofrecer una **pinta** con un **tapeo**. Para ello, hicieron una entrevista a un grupo de sus clientes más fieles para que puntuen que tan buena les parecía una serie de combinaciones de tapeo + cerveza. En base a esa calificación y a la ganancia según el costo, elaboraron una matriz de beneficios.

El dueño del bar decidió que para armar las promociones, cada pinta debe ir con un solo plato y cada tapeo con una única cerveza.

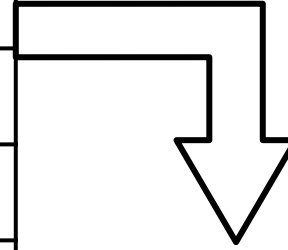
Nos piden encontrar la asignación entre tapeo y cerveza que maximice el beneficio.

Resolver mediante el diseño de un algoritmo de ramificación y poda.

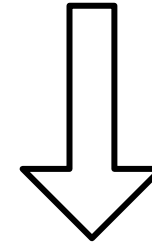
	Rabas	Hamburguesa	Milanesa	Cazuelas <i>veggie</i>
<i>Porter</i>	5	40	60	15
<i>IPA</i>	80	35	10	50
<i>Blonde</i>	15	20	10	40
<i>Amber</i>	10	35	40	30

Sabiendo que nos dan una matriz de beneficios:

	Rabas	Hamburguesa	Papas	Cazuelas <i>veggie</i>
<i>Porter</i>	5	40	60	15
<i>IPA</i>	80	35	10	50
<i>Blonde</i>	15	20	10	40
<i>Amber</i>	10	35	40	30



Y nos piden *asignar* una pinta a cada tapa

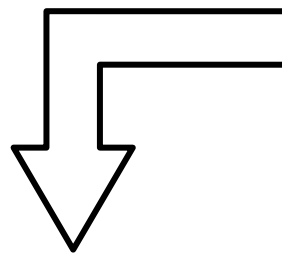


	Rabas	Hamburguesa	Papas	Cazuelas <i>veggie</i>
<i>Porter</i>	5	40	60	15
<i>IPA</i>	80	35	10	50
<i>Blonde</i>	15	20	10	40
<i>Amber</i>	10	35	40	30

¿Cómo modelamos un **estado** para cualquier asignación?

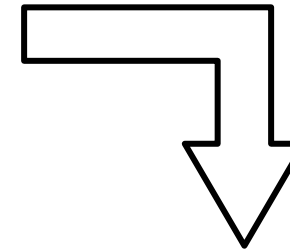
por ejemplo

¿Cómo modelar el *estado*?



	Rabas	Hamburguesa	Papas	Cazuelas veggie
<i>Porter</i>	5	40	60	15
<i>IPA</i>	80	35	10	50
<i>Blonde</i>	15	20	10	40
<i>Amber</i>	10	35	40	30

<i>Blonde</i>	<i>Porter</i>	<i>IPA</i>	<i>Amber</i>
Rabas	Hamburguesa	Papas	Cazuelas veggie



Podemos hacerlo mediante una **tupla**:

{Blonde, Porter, IPA, Amber}

- Cada elemento de la tupla es una pinta
- El índice hace referencia al plato.

¿Cómo modelar el **estado** intermedio?

	Rabas	Hamburguesa	Papas	Cazuelas veggie
Porter	5	40	60	15
IPA	80	35	10	50
Blonde	15	20	10	40
Amber	10	35	40	30

Opción con valores en blanco
{Blonde; Porter; “; “}

Blonde	Porter		
Rabas	Hamburguesa	Papas	Cazuelas veggie

Opción con valores discernibles
{Blonde; Porter; NADA; NADA}

Blonde	Porter	NADA	NADA
Rabas	Hamburguesa	Papas	Cazuelas veggie

Opción estructura incompleta
{Blonde; Porter}

Blonde	Porter
Rabas	Hamburguesa

Modelo de Estado para nuestro problema

Generalizamos los posibles valores de las pintas como “*pinta*” y las posibles tapas como “*tapa*”

- *pinta* en PINTA
 - $PINTA = \{Blonde, IPA, Porter, Amber\}$
- *tapa* en TAPA
 - $TAPA = \{ Rabas, Hamburguesa, Papas, Cazuelas veggie \}$
- N cantidad de pintas (o tapas)
 $N = |PINTA| = |TAPA|$

Ejemplo de estado E:

Blonde	Porter	IPA	Amber
Rabas	Hamburguesa	Papas	Cazuelas veggie

Estado : tupla de N elementos $\{p_0, p_1, \dots, p_{N-1}\}$; donde cada $p_i \in PINTA$

¿Con qué plato i se toma la pinta p_i ? El i referencia el elemento i de TAPA

Estamos en condiciones de identificar las restricciones

¿Cómo sabemos que un **estado es válido**?

Estado : tupla de N elementos $\{p_0, p_1; \dots p_{N-1}\}$;
donde cada $p_i \in \text{PINTA}$

$\{Blonde, Porter, IPA, Amber\}$

Blonde	Porter	IPA	Amber
Rabas	Hamburguesa	Papas	Cazuelas veggie

- Las pintas no se repiten (es factible)
- Todos los platos tienen una pinta asignada (es solución)

Restricciones explícitas (dominio):

- Los valores de p_i están definidos sobre el conjunto $\text{PINTA} = \{Blonde, IPA, Porter, Amber\}$

Restricciones implícitas (solución):

- Cada p_i es único en el estado
- Están todos los p_i posibles

Tenemos la función de factibilidad

¿Cómo sabemos que un **estado** es solución?

Estado : tupla de N elementos $\{p_0, p_1; \dots p_{N-1}\}$;
donde cada $p_i \in \text{PINTA}$

Estado **factible**

- Las pintas no se repiten

$\{\text{Blonde}, \text{Porter}, \text{IPA}, \text{Amber}\}$

Blonde	Porter	IPA	Amber
Rabas	Hamburguesa	Papas	Cazuelas veggie

Estado interno o en proceso:

- Estado **factible** que todavía no terminó de asignar los N platos.

Estado solución parcial:

- Estado **factible** que tiene las N pintas y el *mayor beneficio* encontrado hasta el momento.
 - La *cota global* coincide con el **beneficio** de la solución parcial.

Estado solución:

- Estado con el **mayor beneficio** de todos los estados solución parcial.

Al tratarse de un problema de optimización, tenemos un único estado solución

Estado solución

	Rabas	Hamburguesa	Papas	Cazuelas <i>veggie</i>
<i>Porter</i>	5	40	60	15
<i>IPA</i>	80	35	10	50
<i>Blonde</i>	15	20	10	40
<i>Amber</i>	10	35	40	30

<i>Blonde</i>	<i>IPA</i>	<i>Porter</i>	<i>Amber</i>
Rabas	Hamburguesa	Papas	Cazuelas veggie

Función de optimización

Dado un estado solución parcial **S**, la función a **maximizar** es

$\forall \text{pinta}_n \in \mathbf{S}, \text{tapa}_n \in \mathbf{TAPA}$

$\sum_{n=0}^{n = N-1} B[\text{pinta}_n][\text{tapa}_n]$

$\{Blonde, IPA, Porter, Amber\}$

$\sum_{n=0}^{n = 3} B[\text{pinta}_n][\text{tapa}_n] = B[Blonde][Rabas] + B[IPA][Hamburguesa] + B[Porter][Papas] + B[Amber][Cazuelas] =$
15 + 35 + 60 + 30

Para facilitar la comprensión, reducimos la matriz.

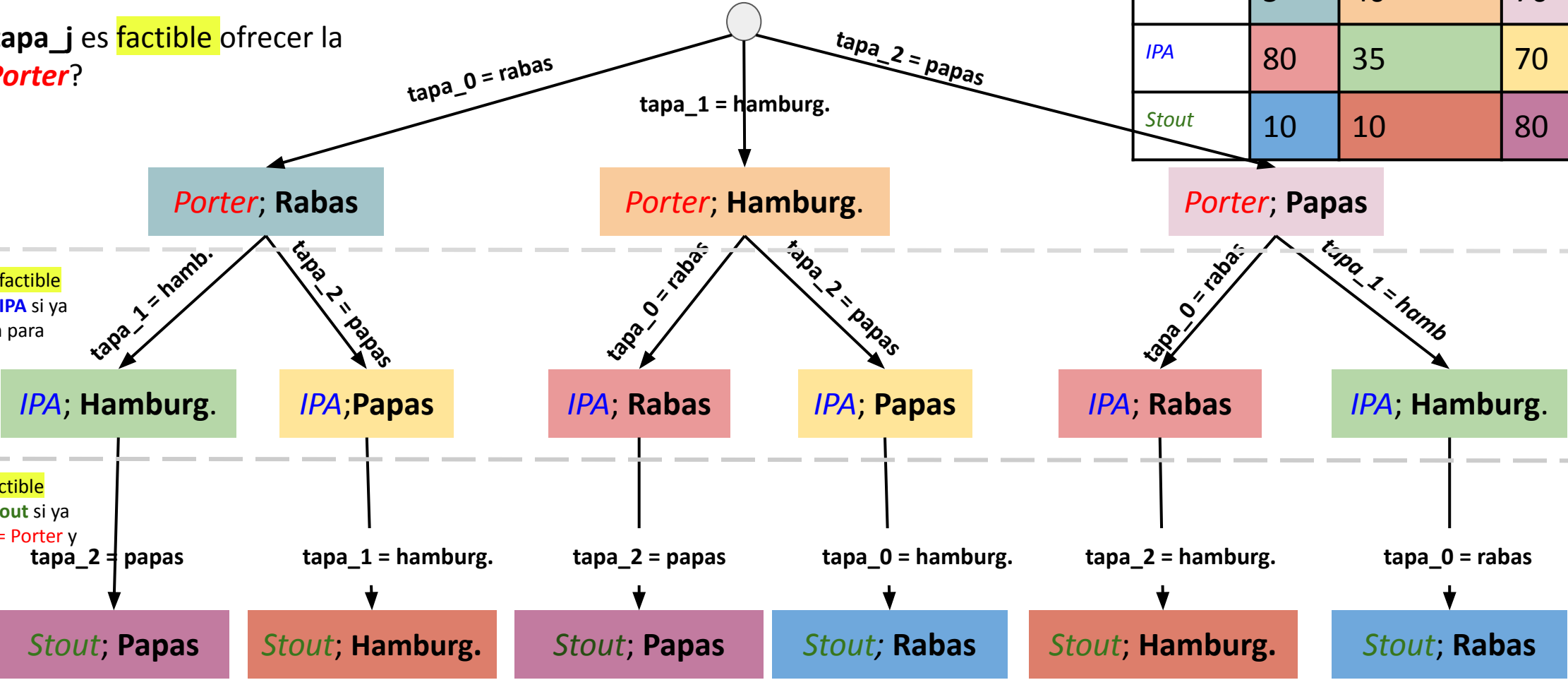
Modelo de árbol: hijos y hermanos

¿Con cuál **tapa_j** es **factible** ofrecer la **pinta_0 = Porter**?

¿Con cuál **tapa_j** es **factible** ofrecer la **pinta_1 = IPA** si ya tenemos una opción para **pinta_0 = Porter**?

¿Con cuál **tapa_j** es **factible** ofrecer la **pinta_2 = Stout** si ya asignamos la **pinta_0 = Porter** y **pinta_1 = IPA**?

	Rabas	Hamburguesa	Papas
Porter	5	40	70
IPA	80	35	70
Stout	10	10	80



Modelo de árbol: hijos y hermanos

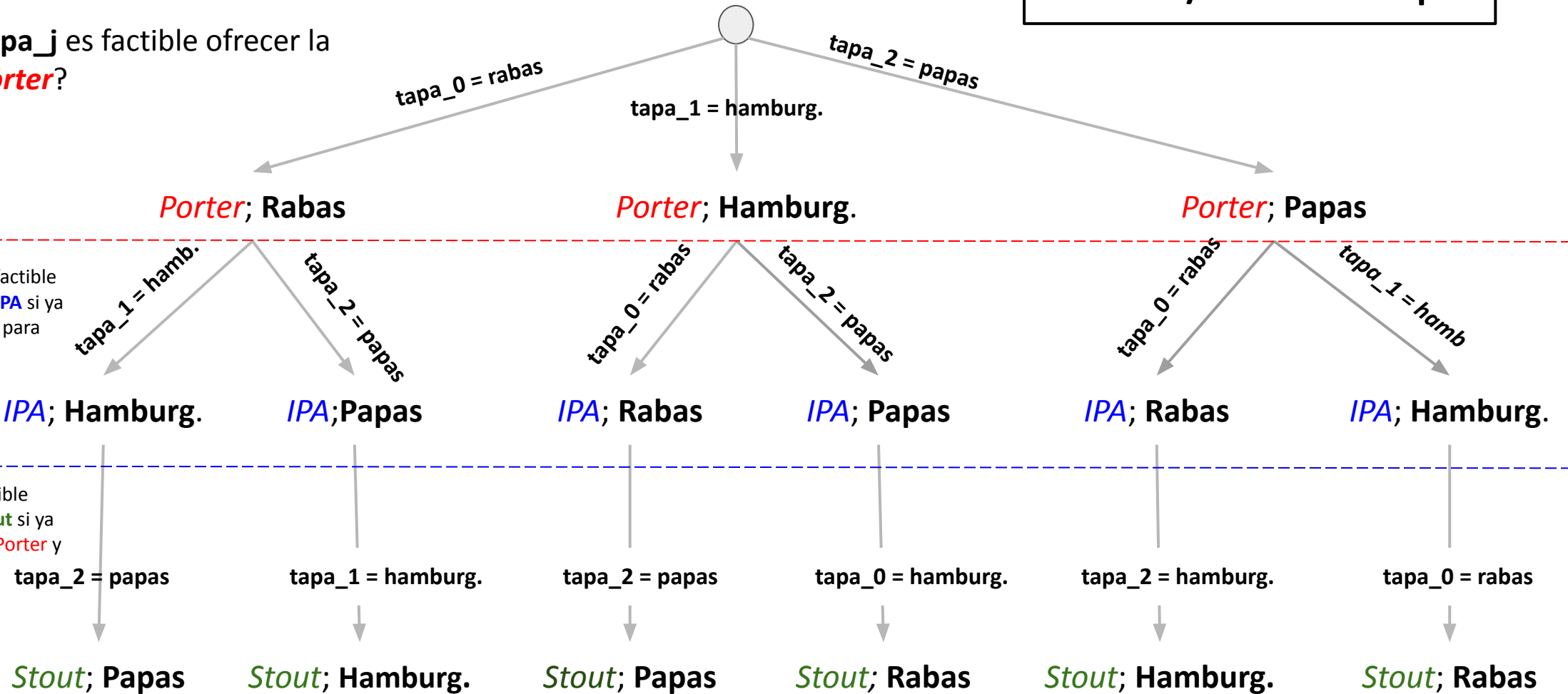
Nivel / hijo: *pinta*

Decisión / hermano: *tapa*

¿Con cuál **tapa_j** es factible ofrecer la *pinta₀* = **Porter**?

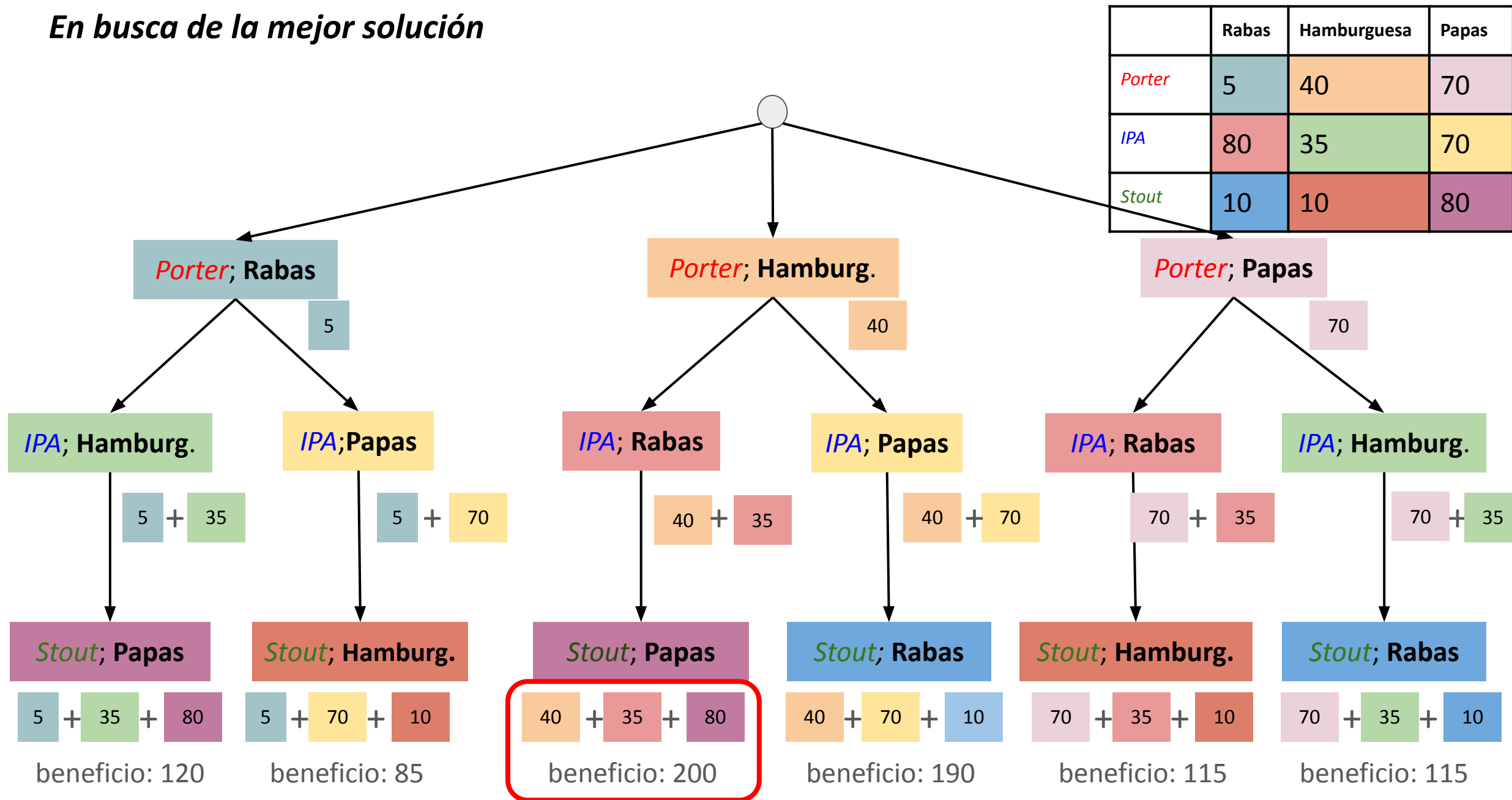
¿Con cuál **tapa_j** es factible ofrecer la *pinta₁* = **IPA** si ya tenemos una opción para *pinta₀* = **Porter**?

¿Con cuál **tapa_j** es factible ofrecer la *pinta₂* = **Stout** si ya asignamos la *pinta₀* = **Porter** y *pinta₁* = **IPA**?



En busca de la mejor solución

Para facilitar la comprensión, reducimos la matriz.



Ramificación y poda: cotas

Recordar de teoría

Una **COTA GLOBAL** expresa que la solución óptima nunca será peor que la cota.

Es el valor de la mejor solución estudiada hasta el momento y debe ser “peor” o igual al “costo” de la solución óptima.

Una **COTA-LOCAL_i** (para cada nodo en curso i) expresa que no alcanzaremos nada mejor al expandir el nodo.

Al iniciar el algoritmo, podemos obtener una **cota global inicial** buscando una solución por Greedy o alguna otra técnica de baja complejidad computacional.

Ejemplo:

	Rabas	Hamburguesa	Papas	Cazuelas <i>veggie</i>
<i>Porter</i>	25	40	30	10
<i>IPA</i>	40	35	5	50
<i>Blonde</i>	15	5	50	40
<i>Amber</i>	10	35	40	30

$$\text{cota_global} = \sum_{n=0}^{n=N-1} B[n][n] = 140$$

Ya tenemos una primer solución posible:

La solución debe tener un beneficio mayor a

Si desde un nodo no podemos obtener un beneficio mayor a , no tiene sentido seguir buscando en la rama.

Pero, ¿cómo sabemos que beneficio máximo podemos llegar a obtener desde un nodo?

Con la cota local, que se calcula mediante una función de evaluación

¿cómo sabemos que beneficio máximo podemos llegar a obtener desde un nodo?

Con la cota local, que se calcula mediante una función de evaluación

	Rabas	Hamburguesa	Papas	Cazuelas <i>veggie</i>
Porter	25	40	30	10
IPA	40	35	5	50
Blonde	15	5	50	40
Amber	10	35	40	30

El mayor valor posible que podemos obtener, mirando toda la matriz es $40+40+50+50=180$
Estamos mirando el **máximo** valor posible para cada plato.

Ahora supongamos que hemos asignado Porter con cazuela *veggie*. Entonces nos resta buscar una pinta para el resto de los platos.

¿Cuál es el valor máximo que le puede ser asignado a cada uno de los restantes platos?

Rabas->40; Hamburguesa->35; Papas->50

Entonces, el beneficio máximo que podemos alcanzar luego de asignar porter a cazuela es $10+40+35+50 = 135$

	Rabas	Hamburguesa	Papas	Cazuelas <i>veggie</i>
Porter	25	40	30	10
IPA	40	35	5	50
Blonde	15	5	50	40
Amber	10	35	40	30

Los ~~tachados~~ no son tenidos en cuenta porque ya asignamos la pinta de la fila 0 y el plato de la col 3.

Cota local
Ejemplos

Para facilitar la comprensión,
reducimos la matriz.

Porter; Rabas

	Rabas	Hamburguesa	Papas
Porter	5	40	70
IPA	80	35	70
Stout	10	10	80

beneficio_acumulado: 5
cota_local: 5 + 35 + 80 = 120

IPA; Hamburg.

	Rabas	Hamb	Papas
Porter	5	40	70
IPA	80	35	70
Stout	10	10	80

beneficio_acumulado: 5 + 35 = 40
cota_local: 40 + 80 = 120

IPA;Papas

	Rabas	Hamb	Papas
Porter	5	40	70
IPA	80	35	70
Stout	10	10	80

beneficio_acumulado: 5 + 70 = 75
cota_local: 75 + 10 = 85

Porter; Papas

	Rabas	Hamburguesa	Papas
Porter	5	40	70
IPA	80	35	70
Stout	10	10	80

beneficio_acumulado: 70
cota_local: 70 + 35 + 80 = 185

IPA; Rabas

	Rabas	Hamb	Papas
Porter	5	40	70
IPA	80	35	70
Stout	10	10	80

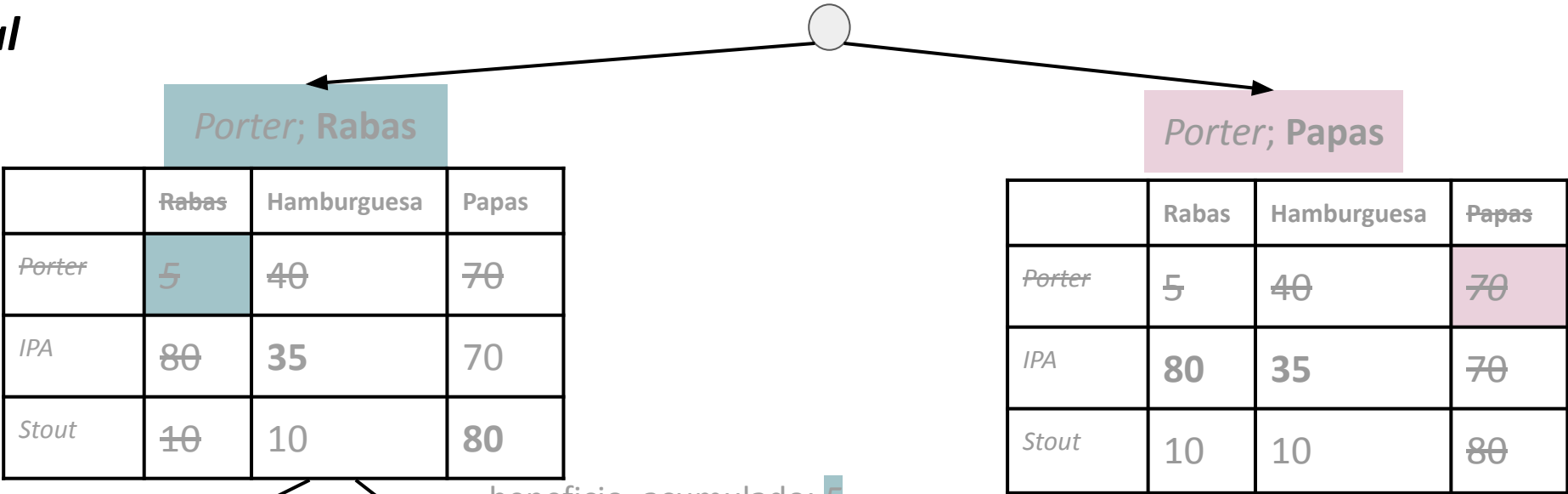
beneficio_acumulado: 70 + 80 = 75
cota_local: 75 + 10 = 85

IPA; Hamburg.

	Rabas	Hamb	Papas
Porter	5	40	70
IPA	80	35	70
Stout	10	10	80

beneficio_acumulado: 70 + 35 = 75
cota_local: 75 + 10 = 85

Cota local
Ejemplos



beneficio_acumulado: 5
cota_local: 5+35+80 = 120

$$cota_local = beneficio_acumulado + B[pinta][tapa] + \sum_{i=pinta+1} \max(B[i][tapa_x]) \text{ tq } tapa_x \text{ no est\'a en el estado actual}$$

En este problema, la *cota local* es el beneficio m\'aximo que **podr\'ia** llegar a obtener desde el estado actual.

IPA; Hamburg.

	Rabas	Hamb	Papas
<i>Porter</i>	5	40	70
<i>IPA</i>	80	35	70
<i>Stout</i>	10	10	80

beneficio_acumulado: 5 + 35 = 40
cota_local: 40 + 80 = 120

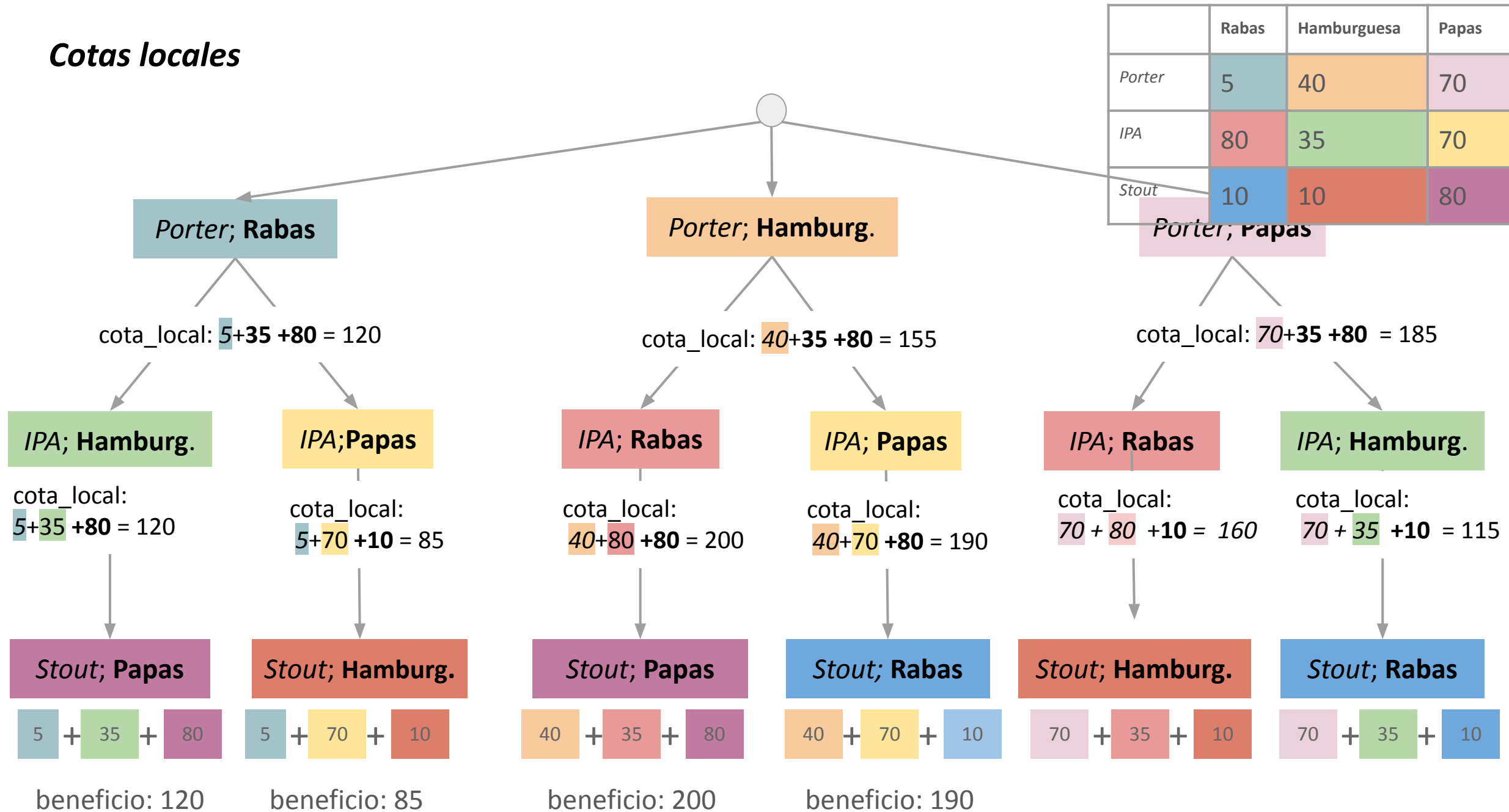
IPA; Papas

	Rabas	Hamb	Papas
<i>Porter</i>	5	40	70
<i>IPA</i>	80	35	70
<i>Stout</i>	10	10	80

beneficio_acumulado: 5 + 70 = 75
cota_local: 75 + 10 = 85

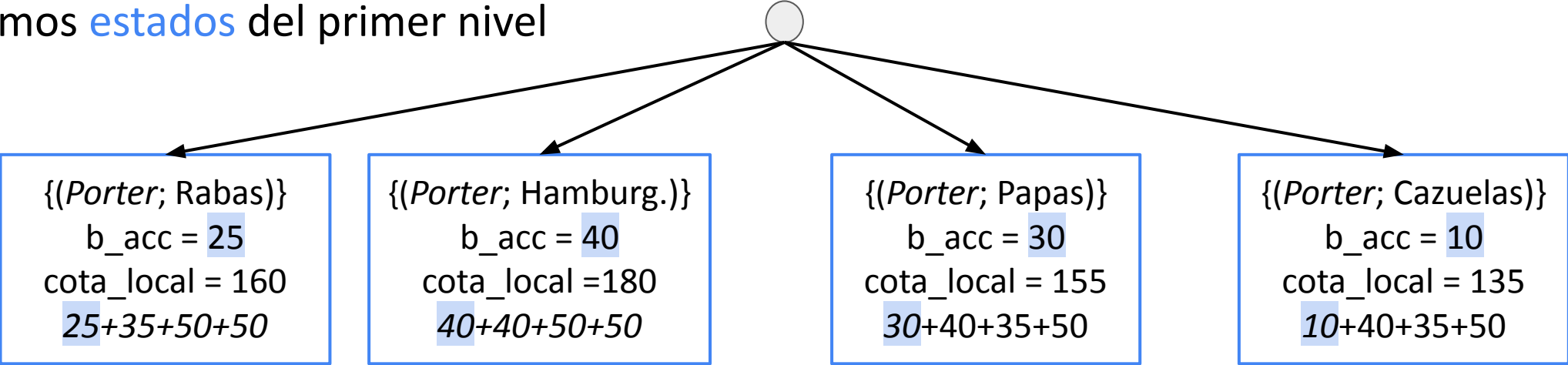
Cotas locales

Para facilitar la comprensión, reducimos la matriz.



Ramificación

Generamos **estados** del primer nivel



cota_global = 140

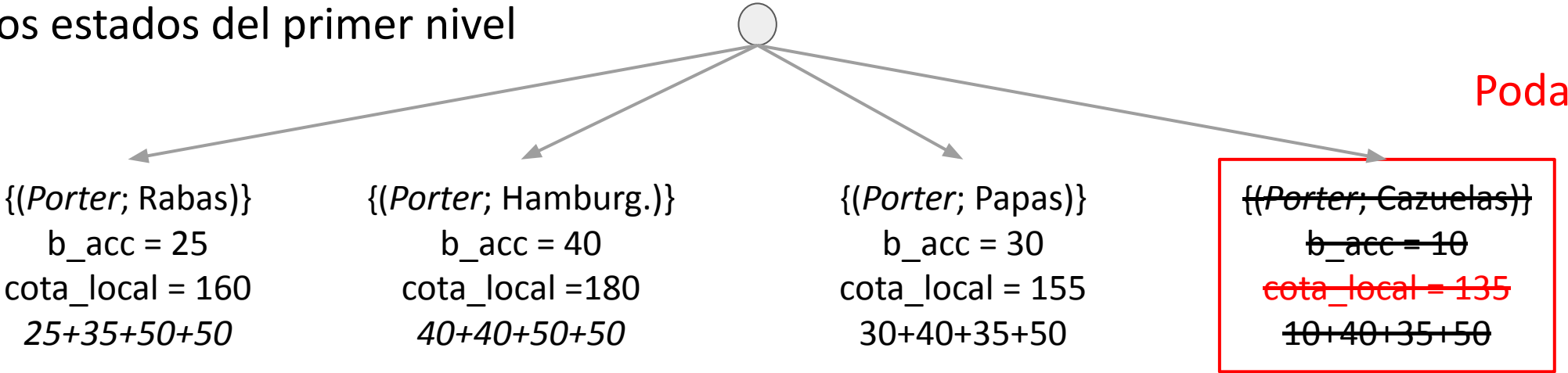
	Rabas	Hamburguesa	Papas	Cazuelas veggie
Porter	25	40	30	10
IPA	40	35	5	50
Blonde	15	5	50	40
Amber	10	35	40	30

Recordar de teoría:
En ramificación y poda, se generan todos los hijos del nodo en curso antes de que cualquier otro nodo vivo pase a ser el nuevo nodo en curso

Ramificación

Nivel 0: Porter

Generamos estados del primer nivel

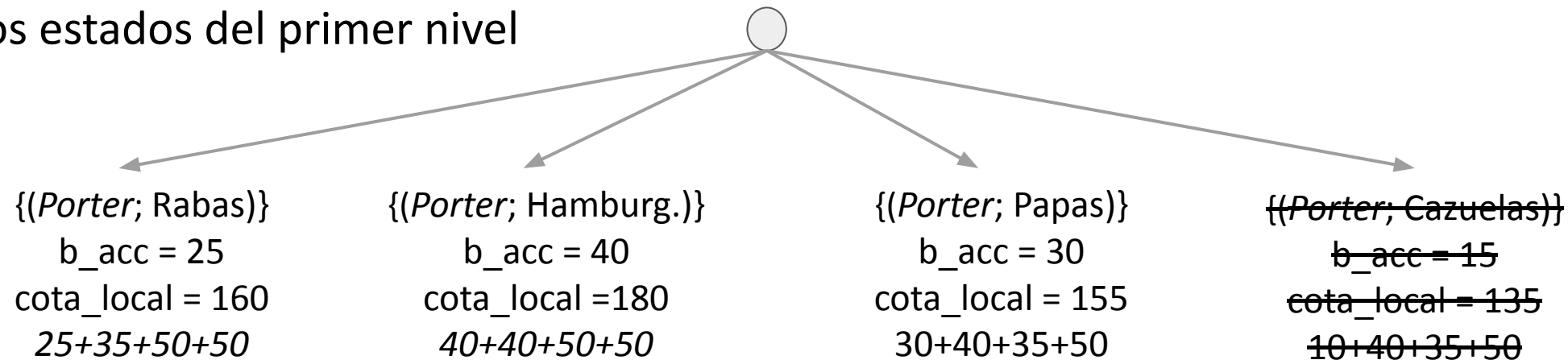


cota_global = 140

	Rabas	Hamburguesa	Papas	Cazuelas <i>veggie</i>
<i>Porter</i>	25	40	30	10
<i>IPA</i>	40	35	5	50
<i>Blonde</i>	15	5	50	40
<i>Amber</i>	10	35	40	30

Ramificación

Generamos estados del primer nivel



Cola de nodos vivos (cota local)
$\{(Porter; Hamburguesa)\}; 180$
$\{(Porter; Rabas)\}; 160$
$\{(Porter; Papas)\}; 155$

Recordar de teoría:

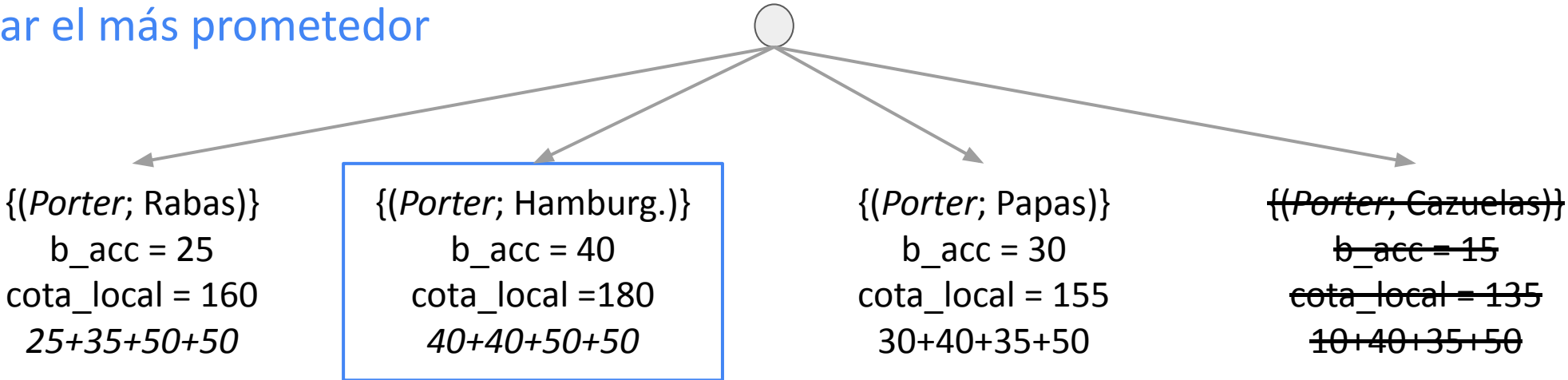
En Ramificación y Poda, puede haber nodos vivos en varios caminos que se almacenan en una cola de prioridad de nodos vivos

cota_global = 140

	Rabas	Hamburguesa	Papas	Cazuelas veggie
Porter	25	40	30	10
IPA	40	35	5	50
Blonde	15	5	50	40
Amber	10	35	40	30

Ramificación

Seleccionar el más prometedor



Cola de nodos vivos (cota local)
$\{(Porter; Hamburg.)\}; 180$
$\{(Porter; Rabas)\}; 160$
$\{(Porter; Papas)\}; 155$

cota_global = 140

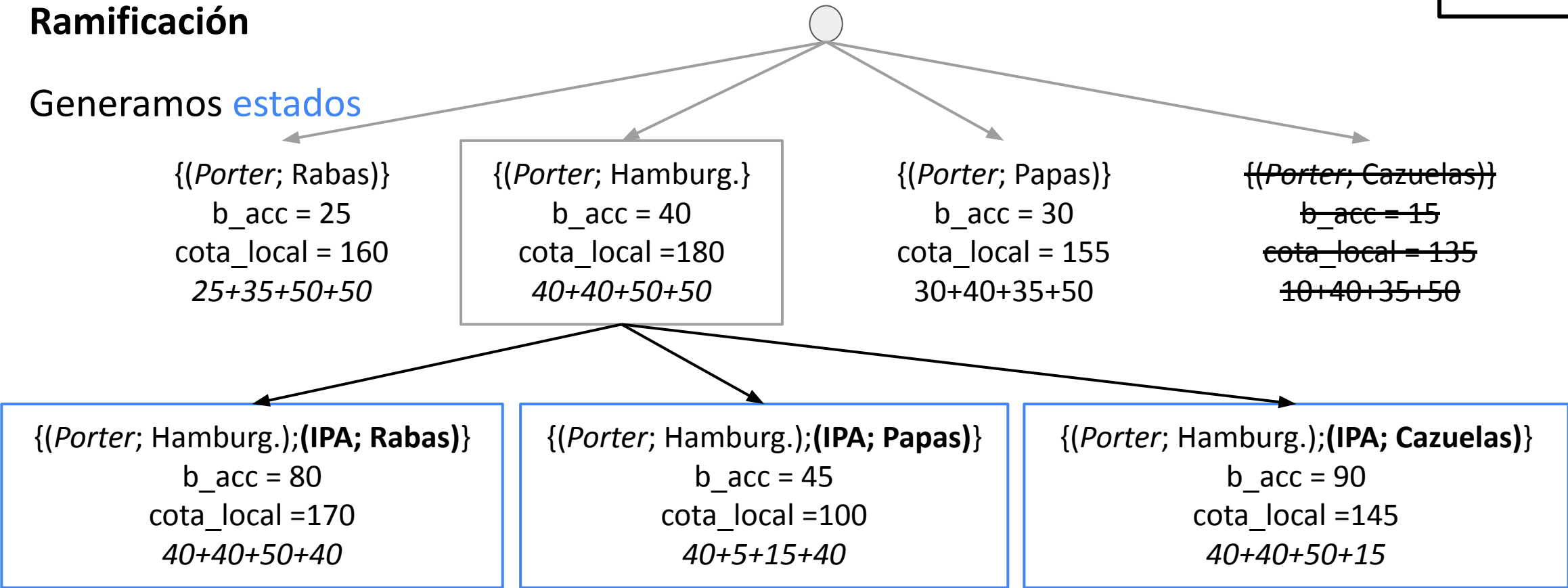
	Rabas	Hamburguesa	Papas	Cazuelas veggie
Porter	25	40	30	10
IPA	40	35	5	50
Blonde	15	5	50	40
Amber	10	35	40	30

El tachado en la matriz es a fines didácticos.

Ramificación

Nivel 1: IPA

Generamos estados



cota_global = 140

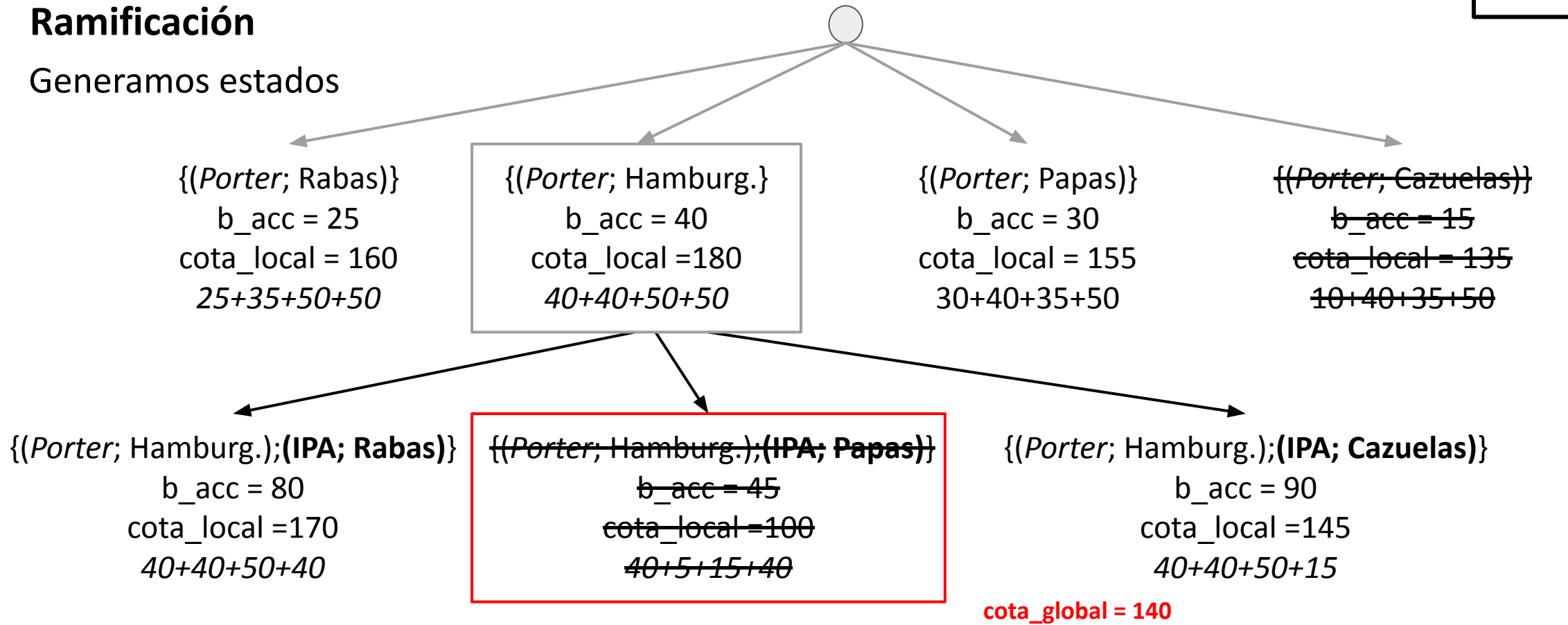
	Rabas	Papas	Cazuelas veggie
IPA	40	5	50
Blonde	15	50	40
Amber	10	40	30

Cola de nodos vivos (cota local)
$\{(Porter; Rabas)\}; 160$
$\{(Porter; Papas)\}; 155$

Ramificación

Generamos estados

Nivel 1: IPA

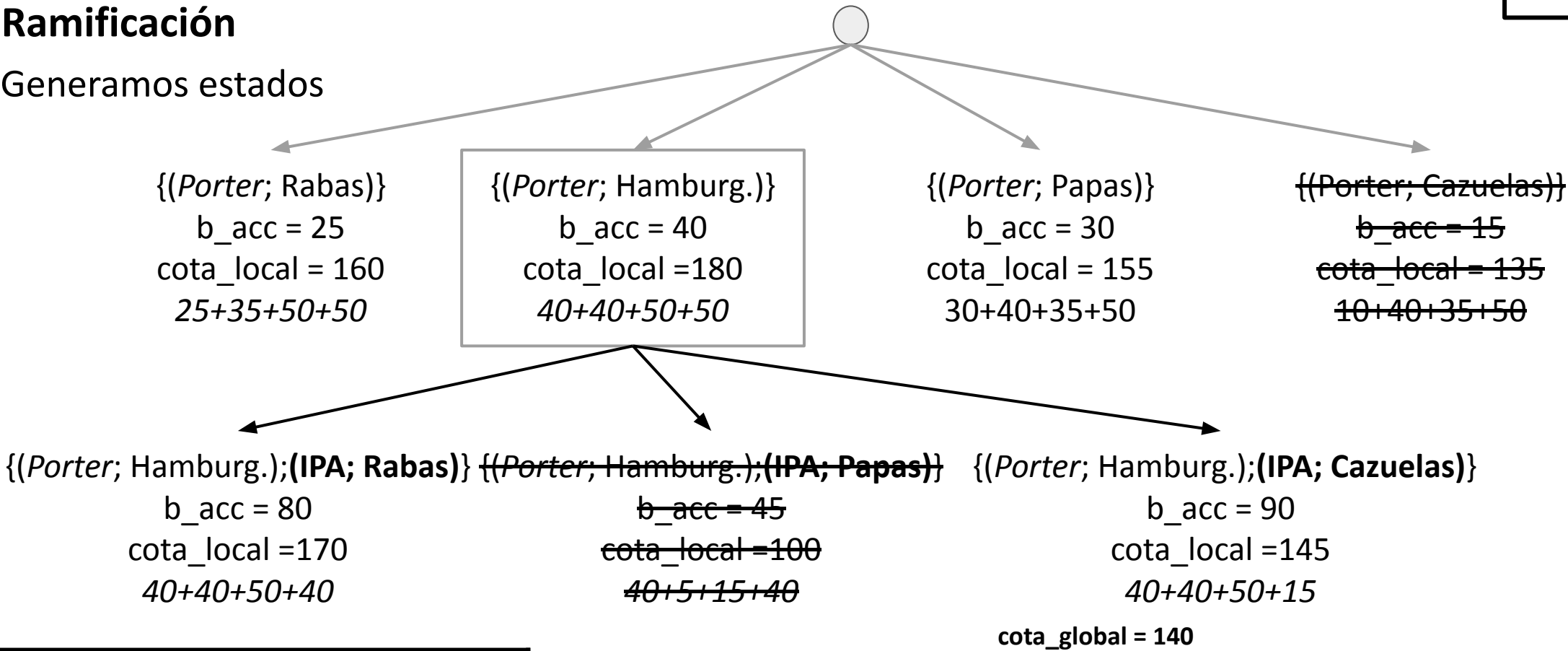


Cola de nodos vivos (cota local)
$\{(Porter; Rabas)\}; 160$
$\{(Porter; Hamburguesa)\}; 155$

	Rabas	Papas	Cazuelas veggie
IPA	40	5	50
Blonde	15	50	40
Amber	10	40	30

Ramificación

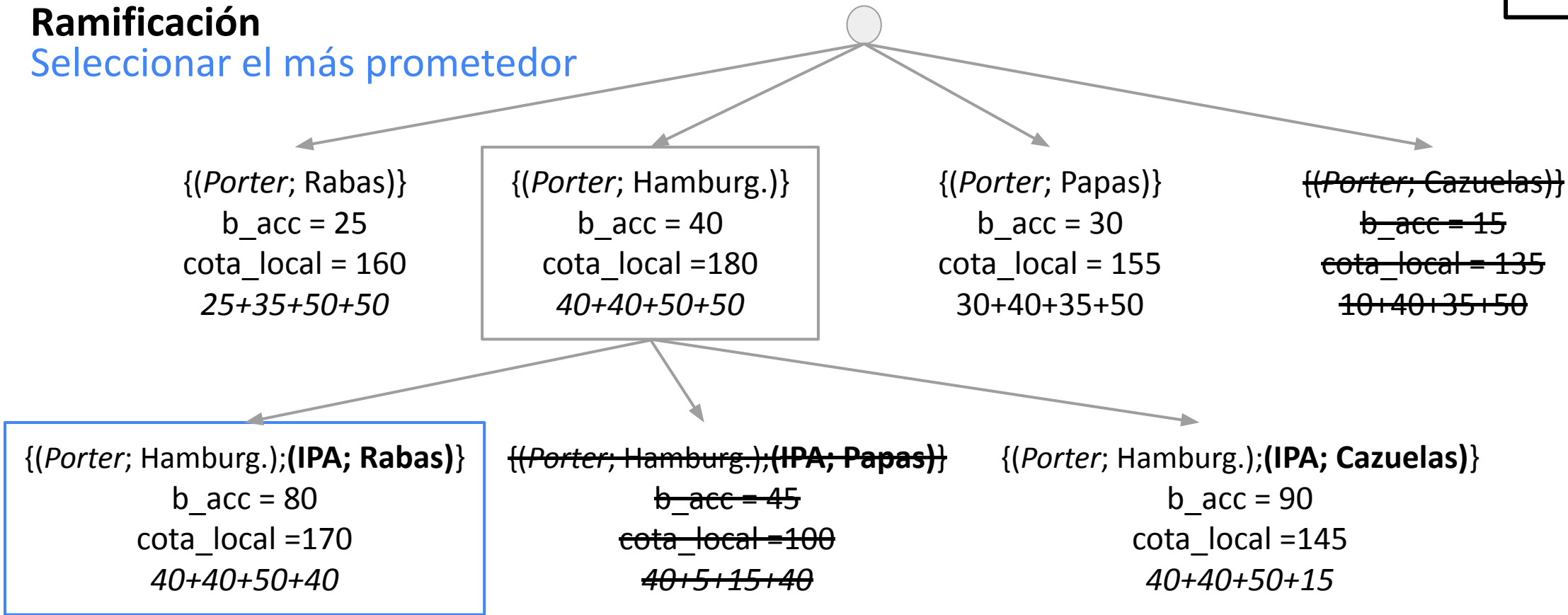
Generamos estados



Cola de nodos vivos (cota local)
$\{(Porter; Hamburguesa);(IPA; Rabas)\} ; 170$
$\{(Porter; Rabas)\}; 160$
$\{(Porter; Papas)\}; 155$
$\{(Porter; Hamburguesa);(IPA; Cazuelas)\} ; 145$

	Rabas	Papas	Cazuelas veggie
IPA	40	5	50
Blonde	15	50	40
Amber	10	40	30

Ramificación
Seleccionar el más prometedor

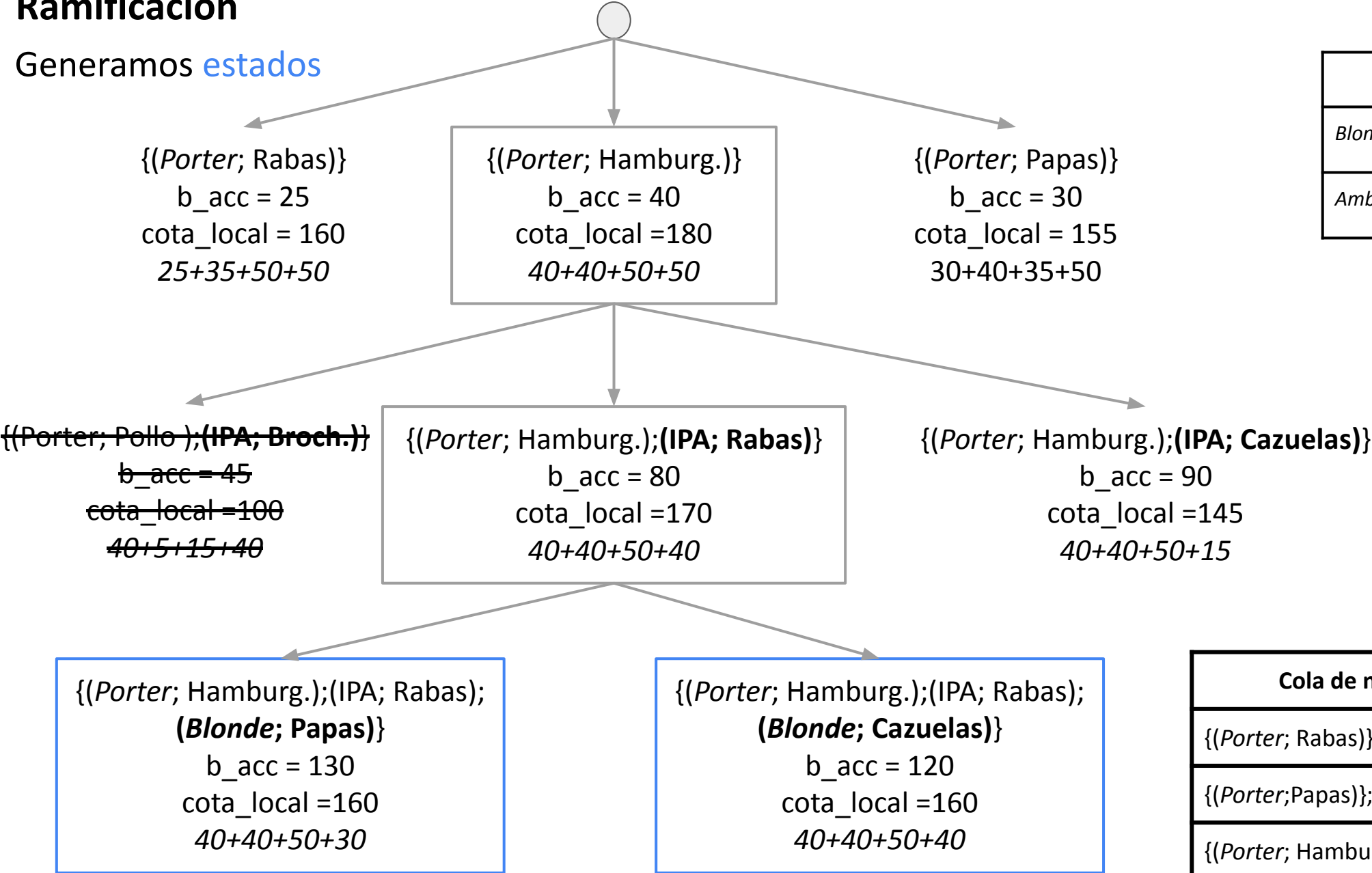


Cola de nodos vivos (cota local)	
$\{(Porter; Hamburg.);(IPA; Rabas)\}; 170$	
$\{(Porter; Rabas)\}; 160$	
$\{(Porter; Papas)\}; 155$	
$\{(Porter; Hamburguesa);(IPA; Cazuelas)\}; 145$	

cota_global = 140			
	Rabas	Papas	Cazuelas veggie
IPA	40	5	50
Blonde	15	50	40
Amber	10	40	30

Ramificación

Generamos estados



Nivel 2: Blonde

cota_global = 140

	Papas	Cazuelas veggie
Blonde	50	40
Amber	40	30

Cola de nodos vivos (cota local)
$\{(Porter; Rabas)\}; 160$
$\{(Porter; Papas)\}; 155$
$\{(Porter; Hamburguesa); (IPA; Cazuelas)\}; 145$

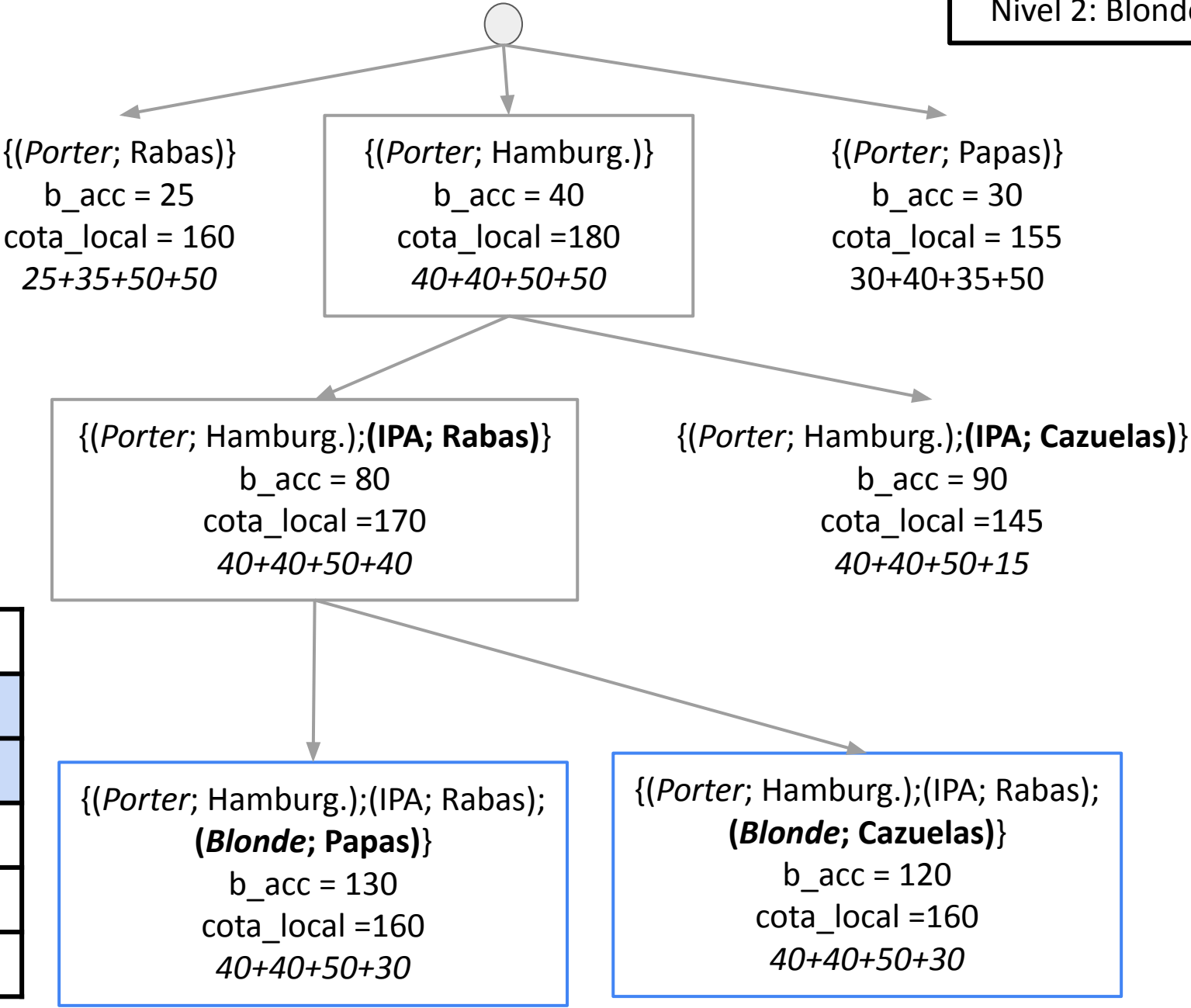
Ramificación

Generamos estados

cota_global = 140

	Papas	Cazuelas veggie
Blonde	50	40
Amber	40	30

Nivel 2: Blonde



Cola de nodos vivos (cota local)	
$\{(Porter; Hamburg.);(IPA; Rabas)); (Blonde; Papas)\}$; 160	
$\{(Porter; Hamburg.);(IPA; Rabas); (Blonde; Cazuelas)\}$; 160	
$\{(Porter; Rabas)\}$; 160	
$\{(Porter; Papas)\}$; 155	
$\{(Porter; Hamburguesa);(IPA; Cazuelas)\}$; 145	

Ramificación

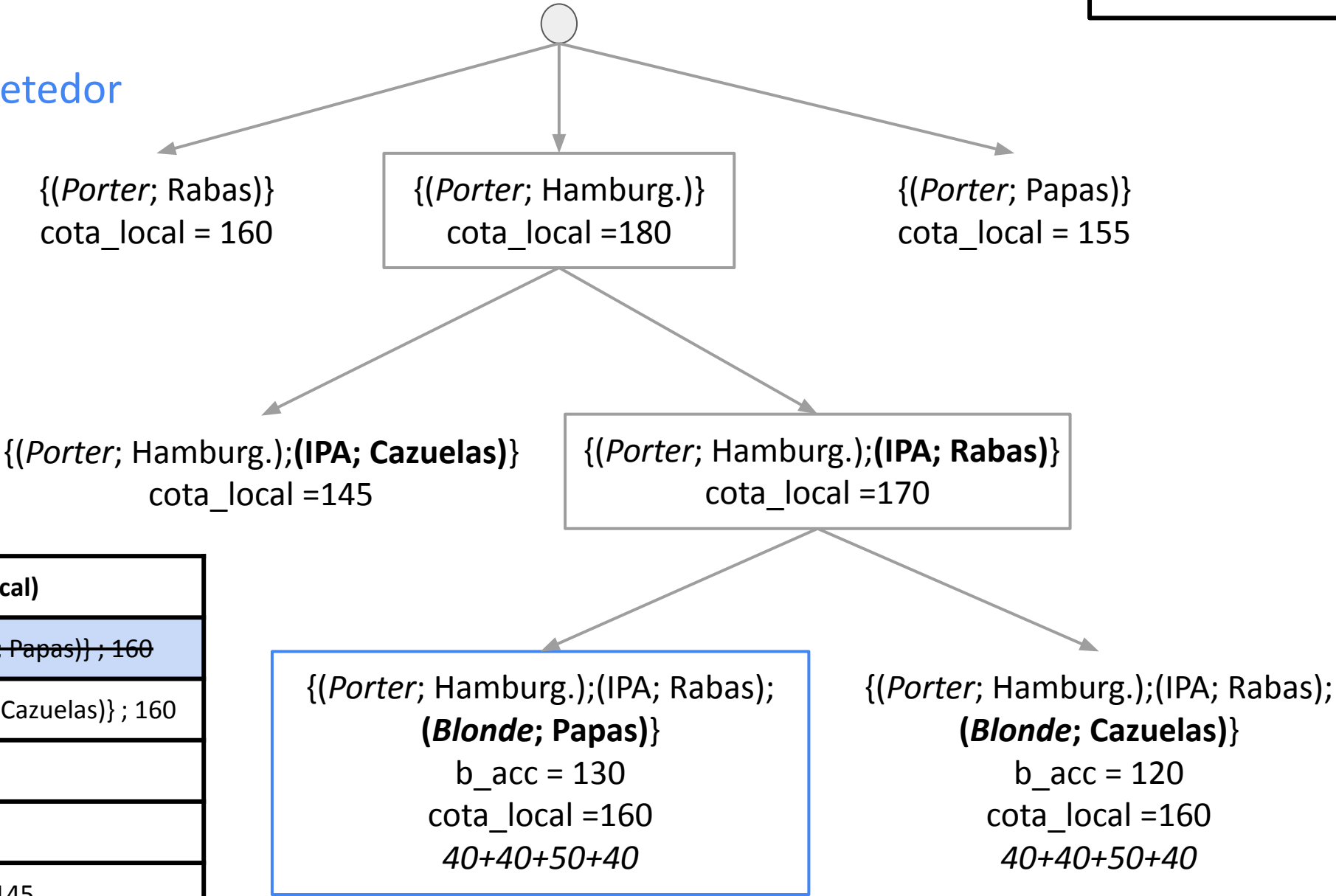
Seleccionar el más prometedor

Nivel 2: Blonde

cota_global = 140

	Papas	Cazuelas veggie
Blonde	50	40
Amber	40	30

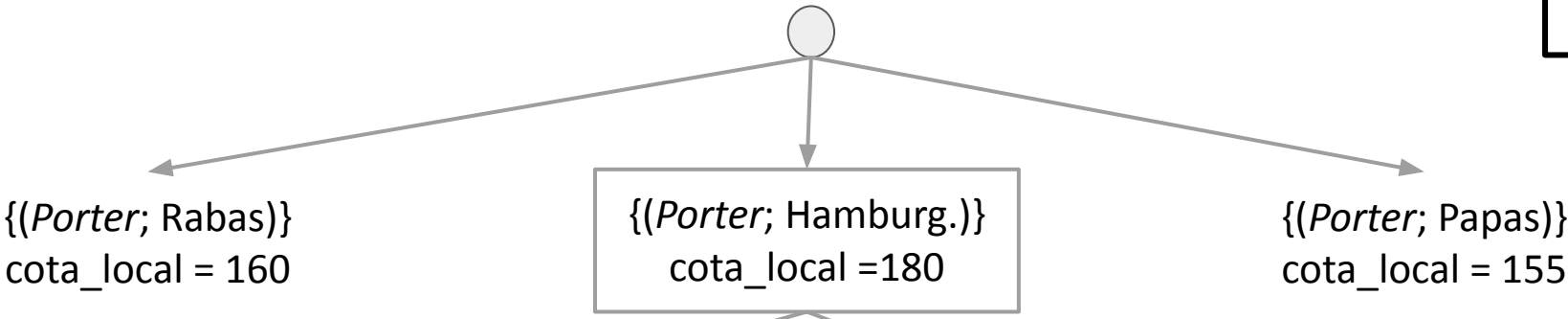
Cola de nodos vivos (cota local)
{{(Porter; Hamburg.);(IPA; Rabas)}}; (Blonde; Papas)} ; 160
{{(Porter; Hamburg.);(IPA; Rabas)}}; (Blonde; Cazuelas)}
{{(Porter; Rabas)}}; 160
{{(Porter; Papas)}}; 155
{{(Porter; Hamburguesa)}}; (IPA; Cazuelas)}



Ramificación

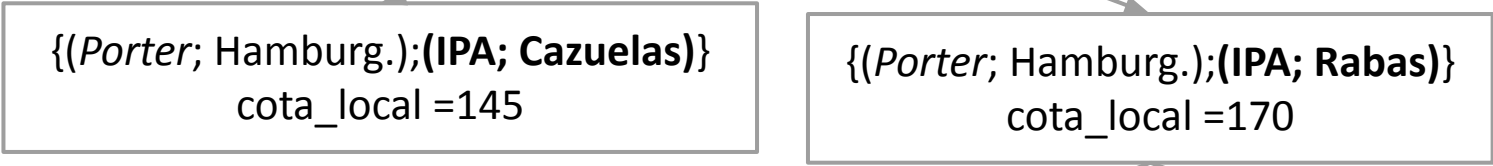
Generamos *estados*

Nivel 3: Amber

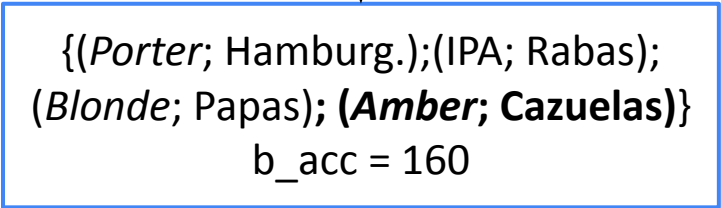
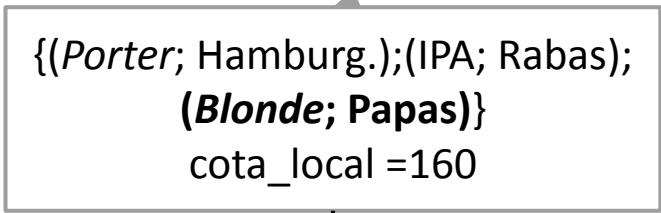


cota_global = 140

	Cazuelas veggie
Amber	30



Cola de nodos vivos (cota local)	
{(Porter; Hamburg.);(IPA; Rabas); (Blonde; Cazuelas)} ; 160	
{(Porter; Rabas)}; 160	
{(Porter; Papas)}; 155	
{(Porter; Hamburguesa);(IPA; Cazuelas)} ; 145	



Es solución
Actualizar cota global

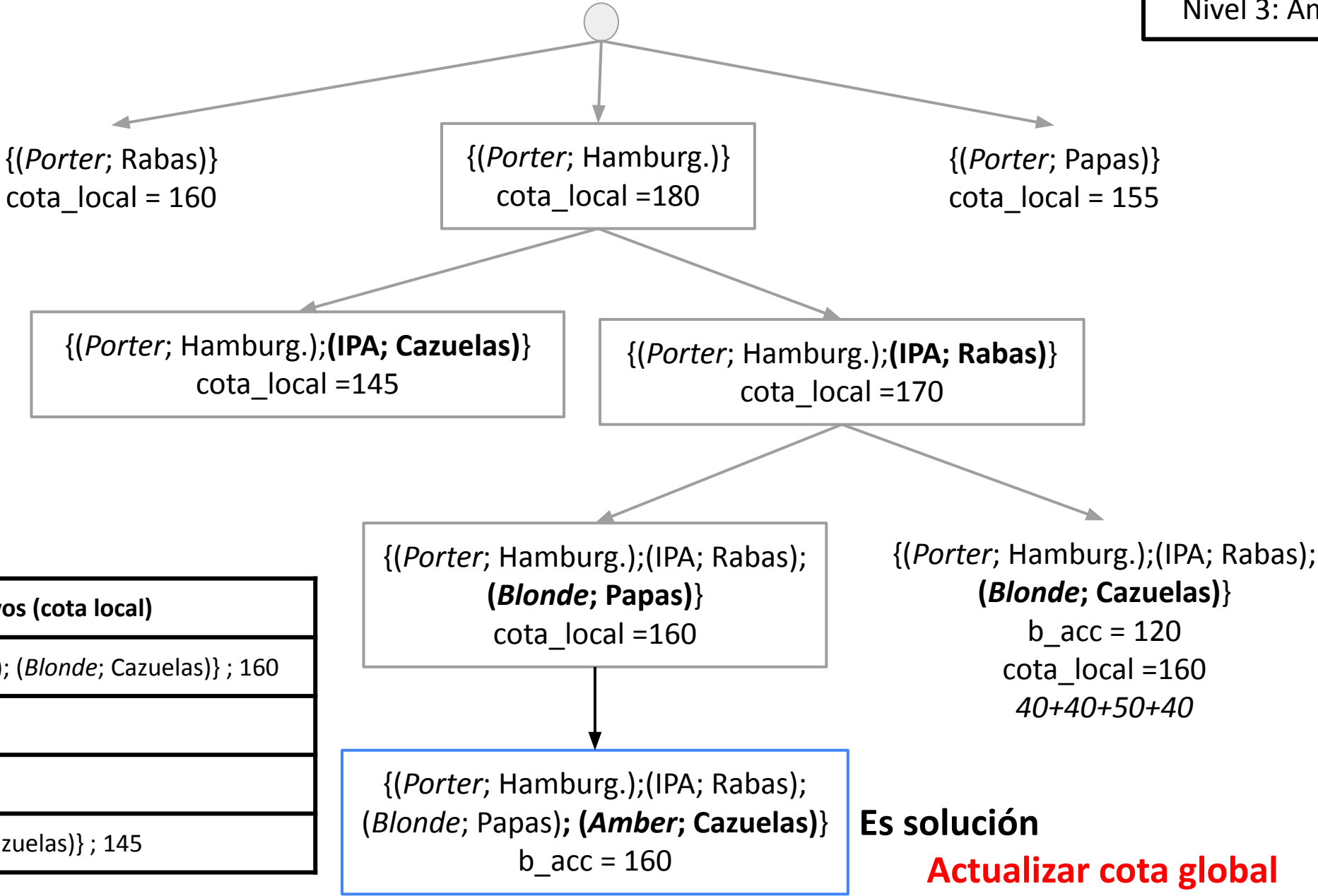
Ramificación

Nivel 3: Amber

cota_global = 140
cota_global = 160

	Cazuelas veggie
Amber	30

Cola de nodos vivos (cota local)	
{(Porter; Hamburg.);(IPA; Rabas); (Blonde; Cazuelas)} ; 160	
{(Porter; Rabas)}; 160	
{(Porter;Papas)}; 155	
{(Porter; Hamburguesa);(IPA; Cazuelas)} ; 145	



Ramificación

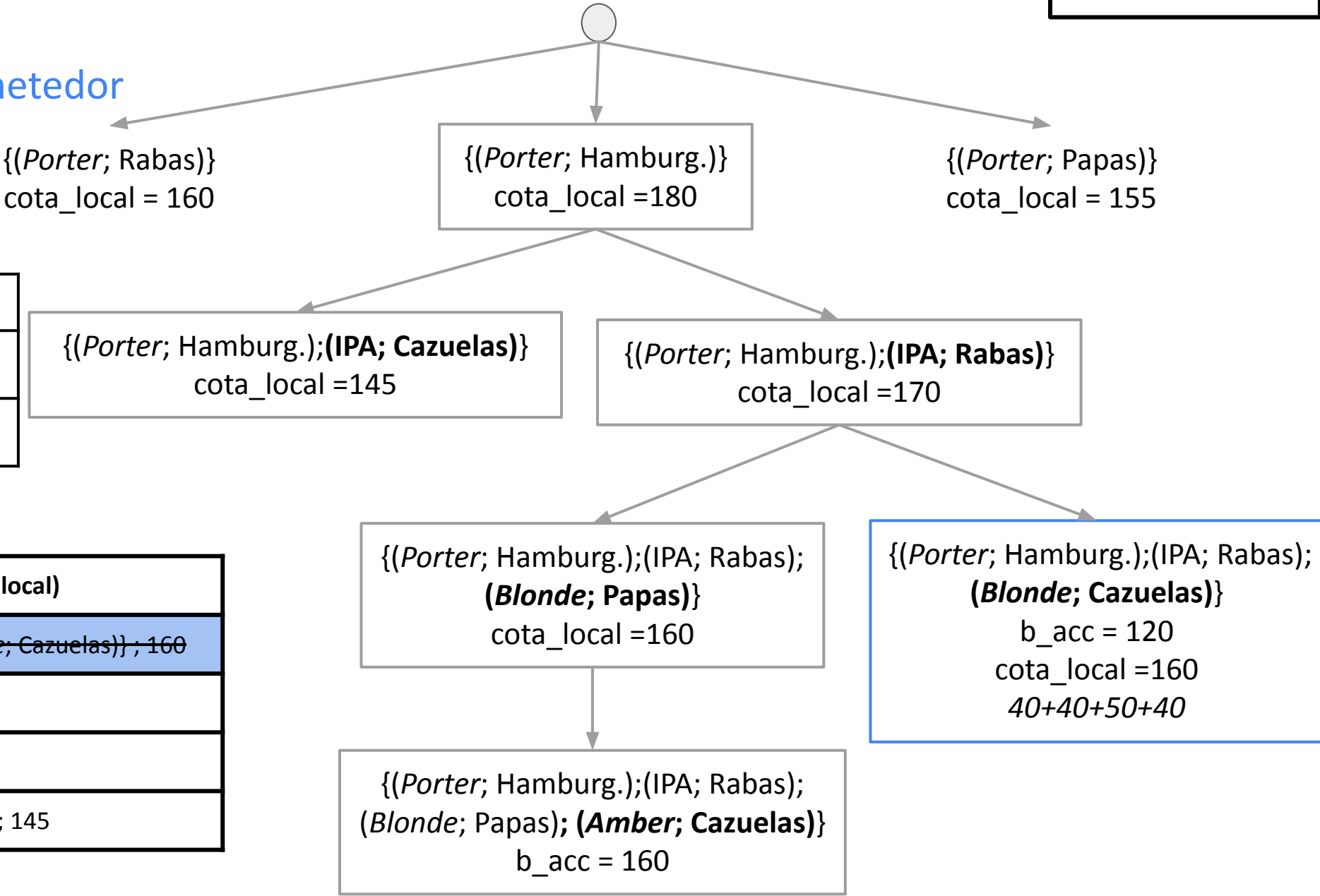
Seleccionar el más prometedor

cota_global = 160

	Papas	Cazuelas veggie
Blonde	50	40
Amber	40	30

Cola de nodos vivos (cota local)
{{(Porter; Hamburg.);(IPA; Rabas); (Blonde; Cazuelas)}}; 160
{{(Porter; Rabas)}}; 160
{{(Porter;Papas)}}; 155
{{(Porter; Hamburguesa);(IPA; Cazuelas)}}; 145

Nivel 2: Blonde



Ramificación

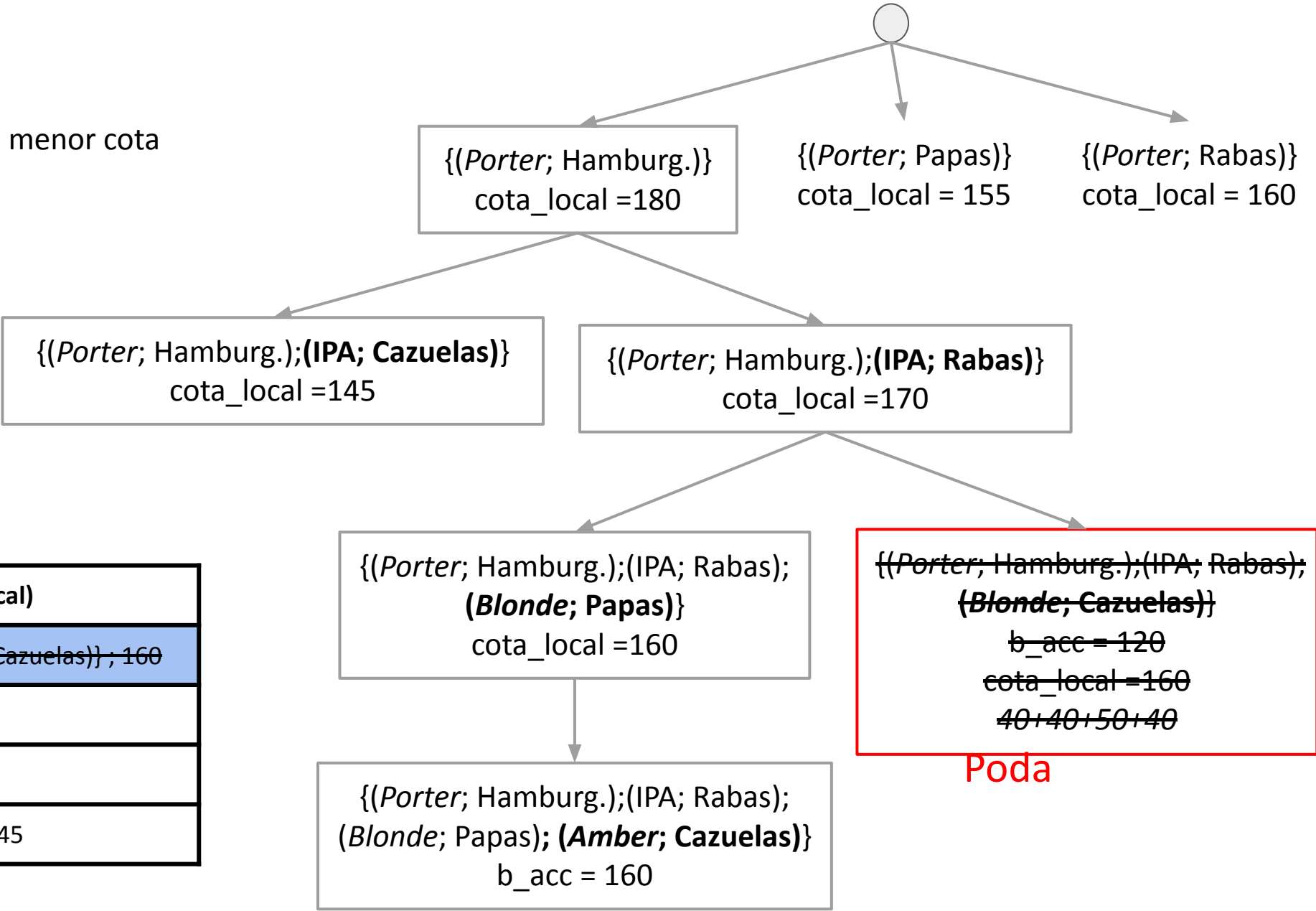
El más prometedor es podado

Los que siguen en cola tienen menor cota
Termina el algoritmo

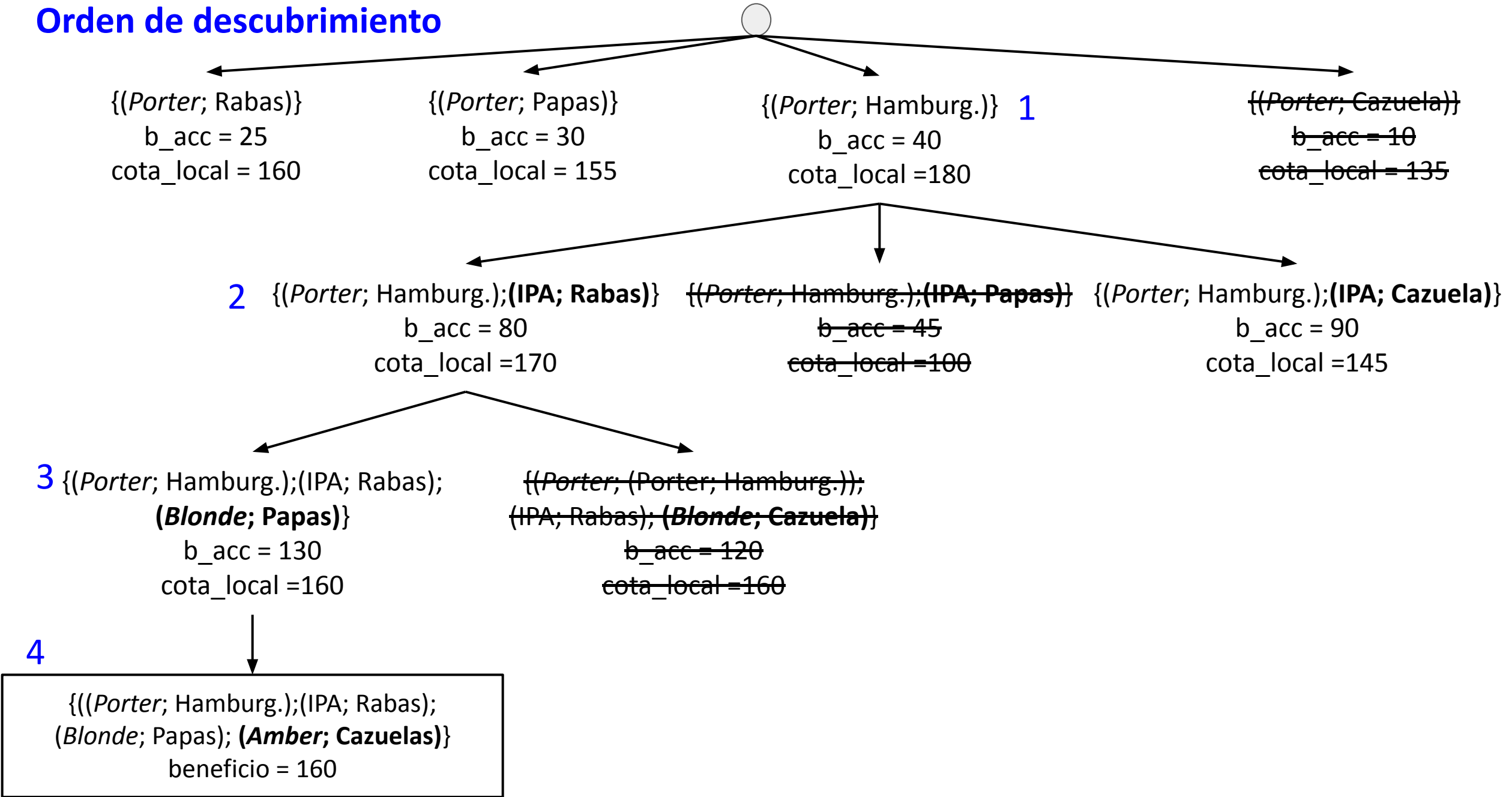
cota_global = 160

	Papas	Cazuelas-veggie
Blonde	50	40
Amber	40	30

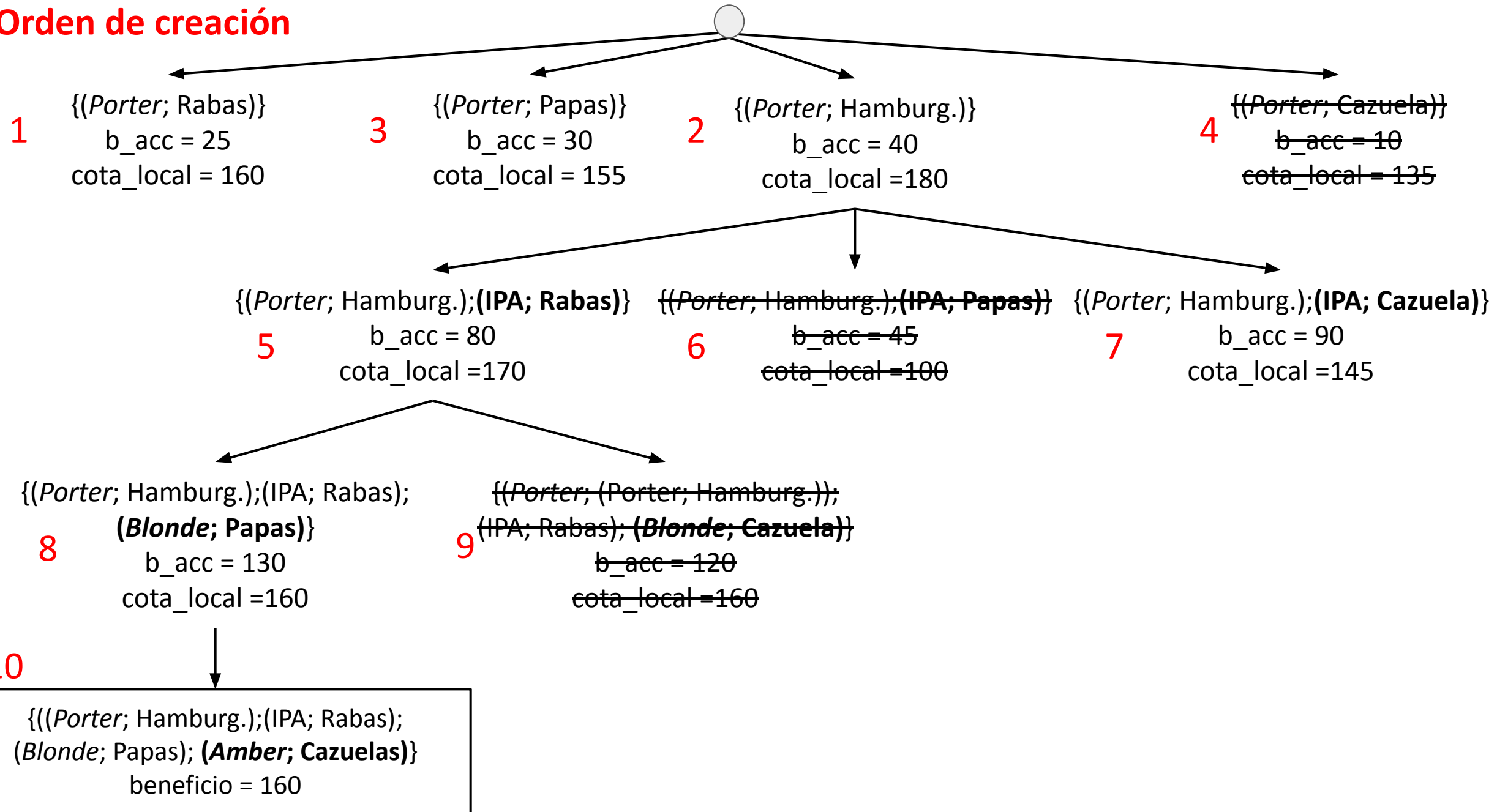
Cola de nodos vivos (cota local)
{{(Porter; Hamburg.);(IPA; Rabas);(Blonde; Cazuelas)}}; 160
{{(Porter; Rabas)}}; 160
{{(Porter;Papas)}}; 155
{{(Porter; Hamburguesa);(IPA; Cazuelas)}}; 145



Orden de descubrimiento



Orden de creación



Para pasar en limpio

Entradas del algoritmo

- $PINTA = \{Blonde, IPA, Porter, Amber\}$
- $TAPA = \{Rabas, Hamburguesa, Papas, Cazuelas\}$ *veggie*
- $B[][]$: cada valor de la matriz $B[pinta_i][tapa_j]$ es un número natural y representa el beneficio

Estado : tupla de N elementos $\{p_0, p_1, \dots, p_{N-1}\}$; donde cada $p_i \in PINTA$ y cada i representa la TAPA i

Árbol: los niveles representan las pintas y las decisiones las tapas.

Solución parcial: estado que tiene N pares. El beneficio es la cota global.

Función de factibilidad: Cada p_i es único.

Función de optimización: La sumatoria del beneficio de cada tapa con cada pinta a incluir tiene que ser la mayor.

Cota local: beneficio acumulado más la sumatoria del máximo valor de cada fila $B[pinta]$ tal que *pinta* y cada tapa no está en el estado.

Poda: cuando la cota local de un estado es mayor que la global.

Estrategia de ramificación: seleccionar nodo con mayor cota local y expandirlo. Almacenar hijos no podados en cola de prioridad.

Ramificación y poda: Esquema general (1)

```
void Ramificación_y_Poda_Esquema ( Estado inicial, Estado solucion)
//inicialización
    ColaPrioridad <Estado> vivos;

    inicial.CalcularCota(); //la cota es parte del estado

    vivos.Poner (inicial);

    Lista <Estado> hijos;

    //generación del espacio de soluciones
    bool encuentre = false;
```


Ramificación y poda: Esquema general (2)

```
While (vivos.noVacía() && ! encuentre) {  
    Estado en_expansión = vivos.Primer(); //nodo más prometedor  
    vivos.Sacar();  
    if (!Poda (en_expansión, solucion)) { // poda por cota global  
        hijos <- Expandir (en_expansión);  
        for ( c/ hijo h en hijos)  
            if (!Poda2 (h, solucion)) // poda por factibilidad y cota global  
                if (es_solucion (h)  
                    solucion = h;  
                else  
                    vivos.Poner(h);  
            else  
                encuentre= true;
```

Ramificación y poda: Esquema general (3)

Lista <Estado> **Expandir** (Estado)

```
{ // A partir del estado actual, genera los estados hijos y  
  para cada hijo calcula su cota local  
}
```

Bool **Poda** (Estado e, Estado Solucion)

```
{  
  // Compara la cota local de e con la mejor solución  
  encontrada hasta el momento  
}
```

Bool **Poda2** (Estado e, Estado Solucion)

```
{  
  // Verifica que el estado generado sea factible  
  // Compara la cota local de e con la mejor solución encontrada hasta el momento  
}
```

Adaptación del esquema general al problema

```
class EstadoAsignacionBnB {  
public:  
    EstadoAsignacionBnB (int N) ;  
    EstadoAsignacionBnB (const EstadoAsignacionBnB & padre) ;  
    void asignar (int decision, int **B);  
    unsigned int getCotaLocal () const;  
    unsigned int getBeneficio () const;  
    int getNivel () const;  
    bool estaAsignada (int decision) const;  
    int getAsignación (int decision) const;  
}
```

Tarea:

Elegir la estructura más adecuada e implementar.

Considerar que debemos poder reusar el código para otros casos de asignación 1:1 (recursos y procesos; personas y tareas; ...)

Adaptación del esquema general al problema

```
While (!vivos.empty() && ! encuentre) {  
  
    EstadoAsignacionBnB en_expansión = vivos.top(); //nodo más prometedor  
    vivos.pop();  
  
    if (en_expansion.getCotaLocal () > cota_global){  
  
        hijos = Expandir (en_expansión);  
        for ( c/ hijo h en hijos)  
            if (!Poda2 (h, solucion)) // poda por factibilidad y cota global  
                if (es_solucion (h)  
                    solucion = h;  
                else  
                    vivos.poner(h);  
            else  
                encuentre= true;
```

Adaptación del esquema general al problema

```
list <EstadoAsignacionBnB> expandir (const EstadoAsignacionBnB & e, int **B)
{
    // A partir del estado actual e, genera los estados hijos
    list <EstadoAsignacionBnB> hijos;
    for (int h=0;h<N;h++)
    {
        if (!e.estaAsignada(h)) // Función de factibilidad*
        {
            EstadoAsignacionBnB nuevo(e); // El estado "nuevo" se genera a partir de "e"
            nuevo.asignar(h,B);
            hijos.push_back(nuevo);
        }
    }
    return hijos;
}
```

** Recordar que la función de factibilidad no debe ser costosa*

Adaptación del esquema general al problema

```
struct comparator
{
    bool operator()(EstadoAsignacionBnB &p, EstadoAsignacionBnB &q) const {
        return p.getCotaLocal() < q.getCotaLocal();
    }
};

void asignacionBnB(const EstadoAsignacionBnB& inicial, int cota_global, int** B)
{ // begin bnB
    EstadoAsignacionBnB solucion(N);

    //inicialización
    priority_queue<EstadoAsignacionBnB, vector<EstadoAsignacionBnB>, comparator> vivos;
    vivos.push(inicial);
    list <EstadoAsignacionBnB> hijos;

    bool encuentre = false;
```

```

while (!vivos.empty() && !encontre) {
    EstadoAsignacionBnB en_expansion = vivos.top(); //nodo más prometedor
    vivos.pop();

    if (en_expansion.getCotaLocal() > cota_global){// poda por cota global
        hijos = expandir (en_expansion,B);
        for (h: hijos){
            int aux_cota = h.getCotaLocal();
            if (aux_cota > cota_global ) {// poda por factibilidad* y cota global
                if (h.getNivel() == N-1) {
                    solucion = h;
                    cota_global = aux_cota;
                }
                else
                    vivos.push(h);
            }
        }
    } else
        encontre = true;
}
} // end bnB

```

* Cuando implementamos “expandir”, generamos el hijo si cumple la func. de factibilidad.

Consideraciones para implementación

- La implementación asegura que nuestro código funciona
 - Eso no implica que la técnica esté bien aplicada
- Durante la implementación es útil volver a los pasos anteriores donde definimos conceptualmente cómo es un estado, restricciones, funciones, mecanismos para generar hijos / niveles.
 - Esto nos ayuda a definir estructuras y programas eficientes
- También es recomendable incorporar algunas métricas, como por ejemplo contar cantidad de soluciones alcanzadas, estados podados, etc.

Detalles de implementación

```
list <EstadoAsignacionBnB> expandir (...){
    list <EstadoAsignacionBnB> hijos;
    for (int h=0;h<N;h++){
        if (!e.estaAsignada(h)){
            EstadoAsignacionBnB nuevo(e);
            nuevo.asignar(h,B);
            hijos.push_back(nuevo);
        }
    }
    return hijos;
}
```

Ejemplo: desde estado inicial “e”

int **B

	Rabas	Hamburguesa	Papas	Cazuelas veggie
Porter	25	40	30	10
IPA	40	35	5	50
Blonde	15	5	50	40
Amber	10	35	40	30

EstadoAsignacionBnB e

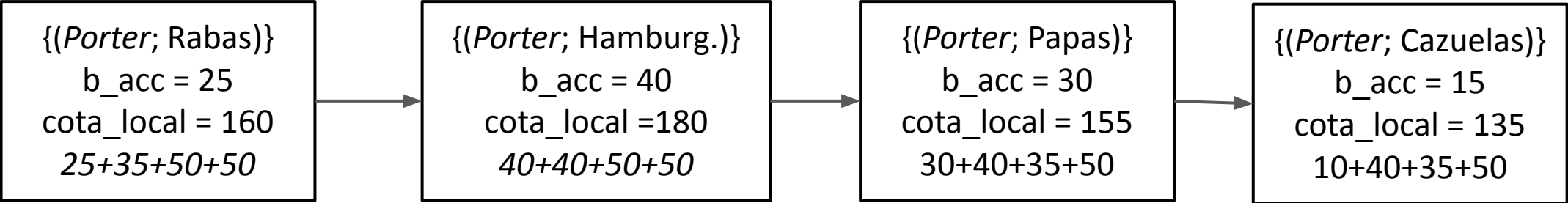
{}

b_acc = 0

cota_local = 180

25+35+50+50

list <EstadoAsignacionBnB> hijos



Detalles de implementación

```
list <EstadoAsignacionBnB> expandir (...){
    list <EstadoAsignacionBnB> hijos;
    for (int h=0;h<N;h++){
        if (!e.estaAsignada(h)){
            EstadoAsignacionBnB nuevo(e);
            nuevo.asignar(h,B);
            hijos.push_back(nuevo);
        }
    }
    return hijos;
}
```

Ejemplo: desde estado “e”

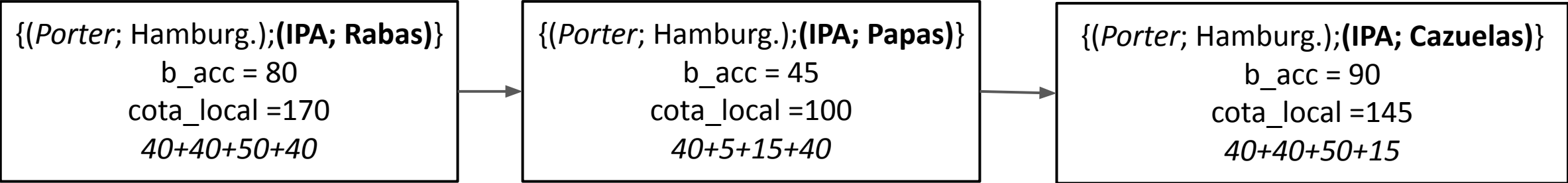
int **B

	Rabas	Hamburguesa	Papas	Cazuelas veggie
Porter	25	40	30	10
IPA	40	35	5	50
Blonde	15	5	50	40
Amber	10	35	40	30

EstadoAsignacionBnB e

{{(Porter; Hamburg.)}
b_acc = 40
cota_local =180
40+40+50+50

list <EstadoAsignacionBnB> hijos

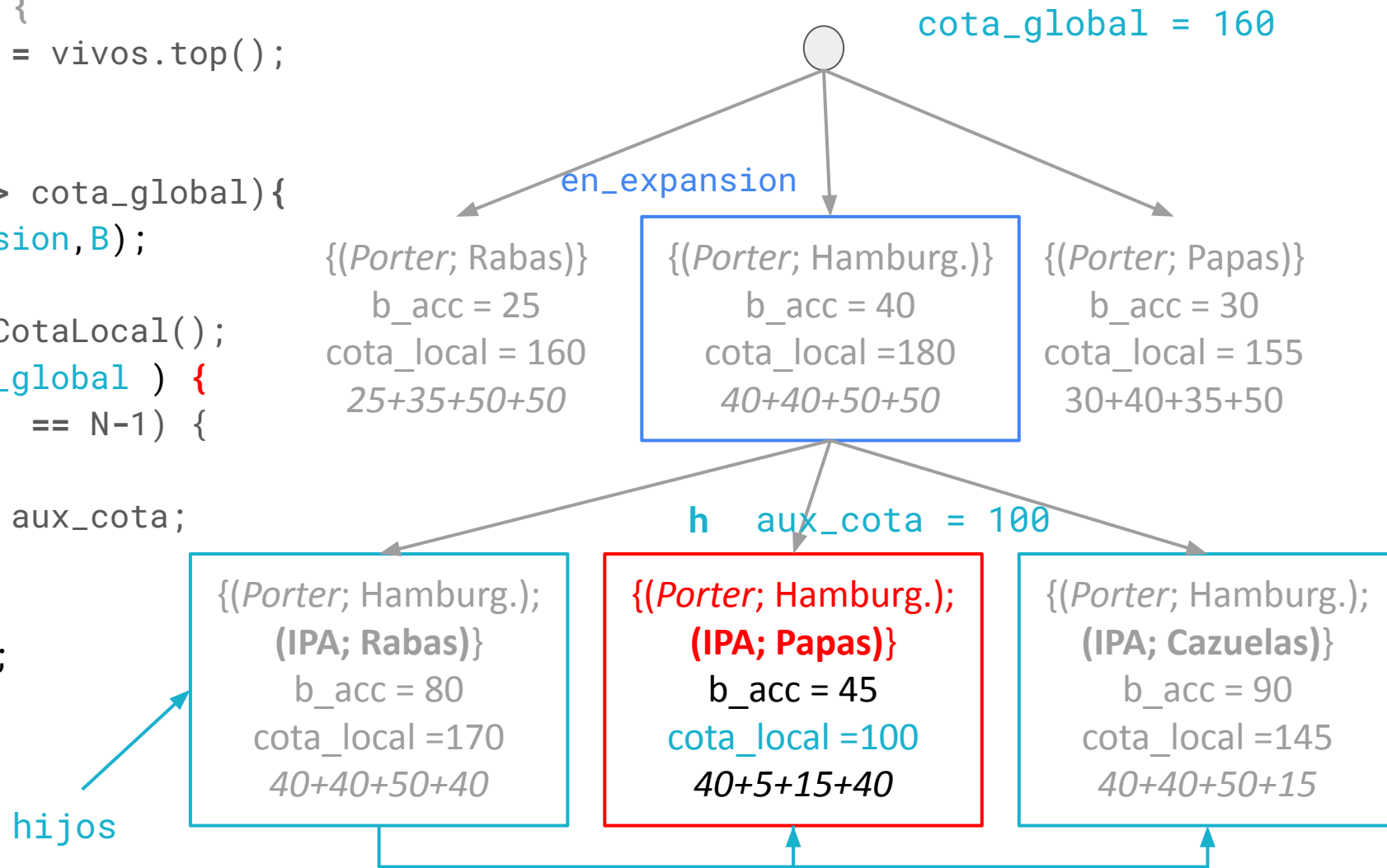


```

while (!vivos.empty() && !encontre) {
    EstadoAsignacionBnB en_expansion = vivos.top();
    vivos.pop();

    if (en_expansion.getCotaLocal() > cota_global){
        hijos = expandir (en_expansion,B);
        for (h: hijos){
            int aux_cota = h.getCotaLocal();
            if (aux_cota > cota_global ) {
                if (h.getNivel() == N-1) {
                    solucion = h;
                    cota_global = aux_cota;
                }
            }
            else
                vivos.push(h);
        }
    } else
        encontre = true;
}
} // end backBnB

```



En este caso, el segundo hijo **h {(Porter; Hamburg.); (IPA; Papas)}** de la lista tiene **aux_cota (100)** menor a la **cota_global (160)**, entonces no se agrega a la cola de vivos, **se poda**

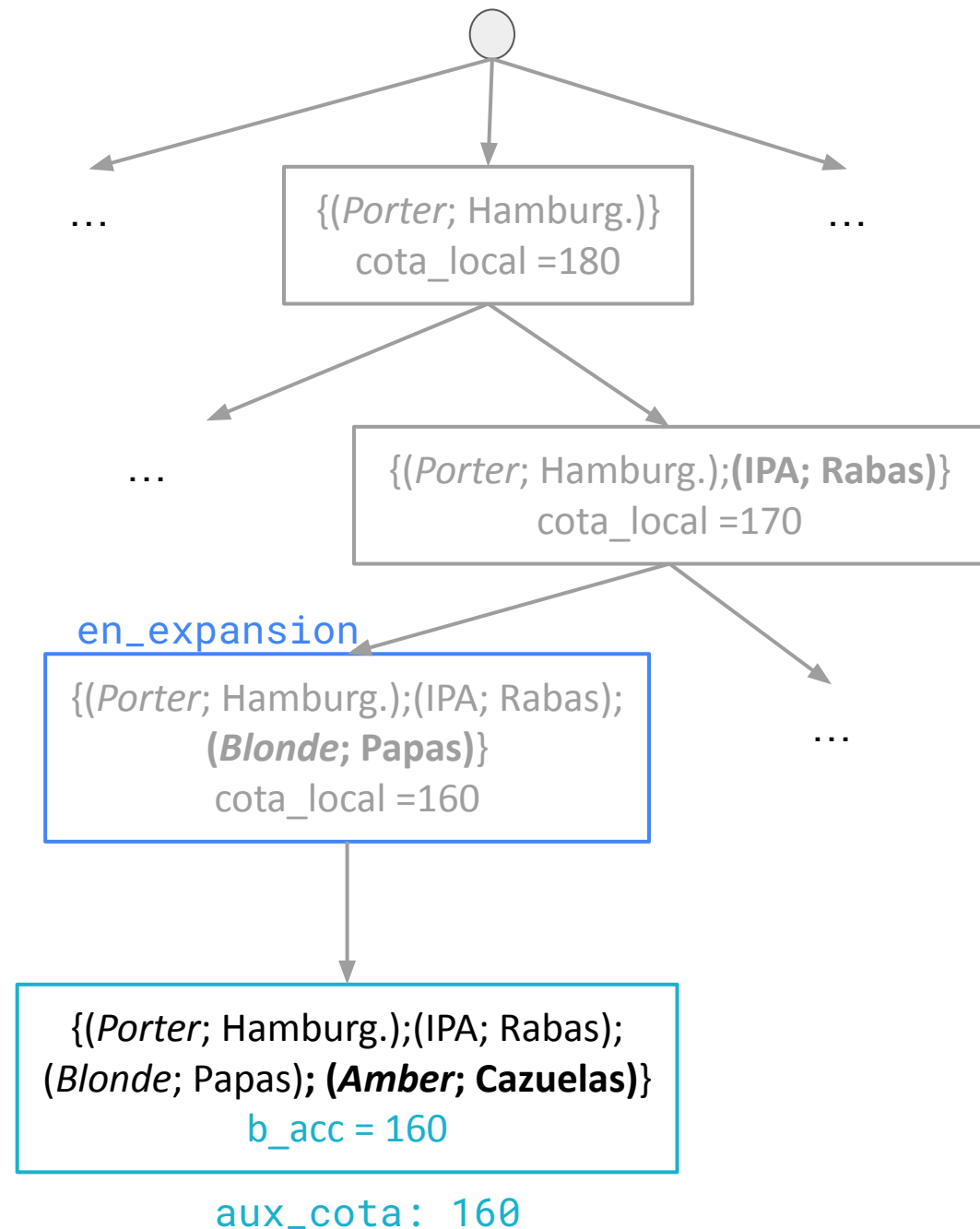
```

while (!vivos.empty() && ! encuentre) {
    EstadoAsignacionBnB en_expansion = vivos.top();
    vivos.pop();

    if (en_expansion.getCotaLocal() > cota_global){
        hijos = expandir (en_expansion,B);
        for (h: hijos){
            int aux_cota = h.getCotaLocal();
            if (aux_cota > cota_global ) {
                if (h.getNivel() == N-1) {
                    solucion = h;
                    cota_global = aux_cota;
                }
                else
                    vivos.push(h);
            }
        }
    } else
        encuentre = true;
}
} // end bnB

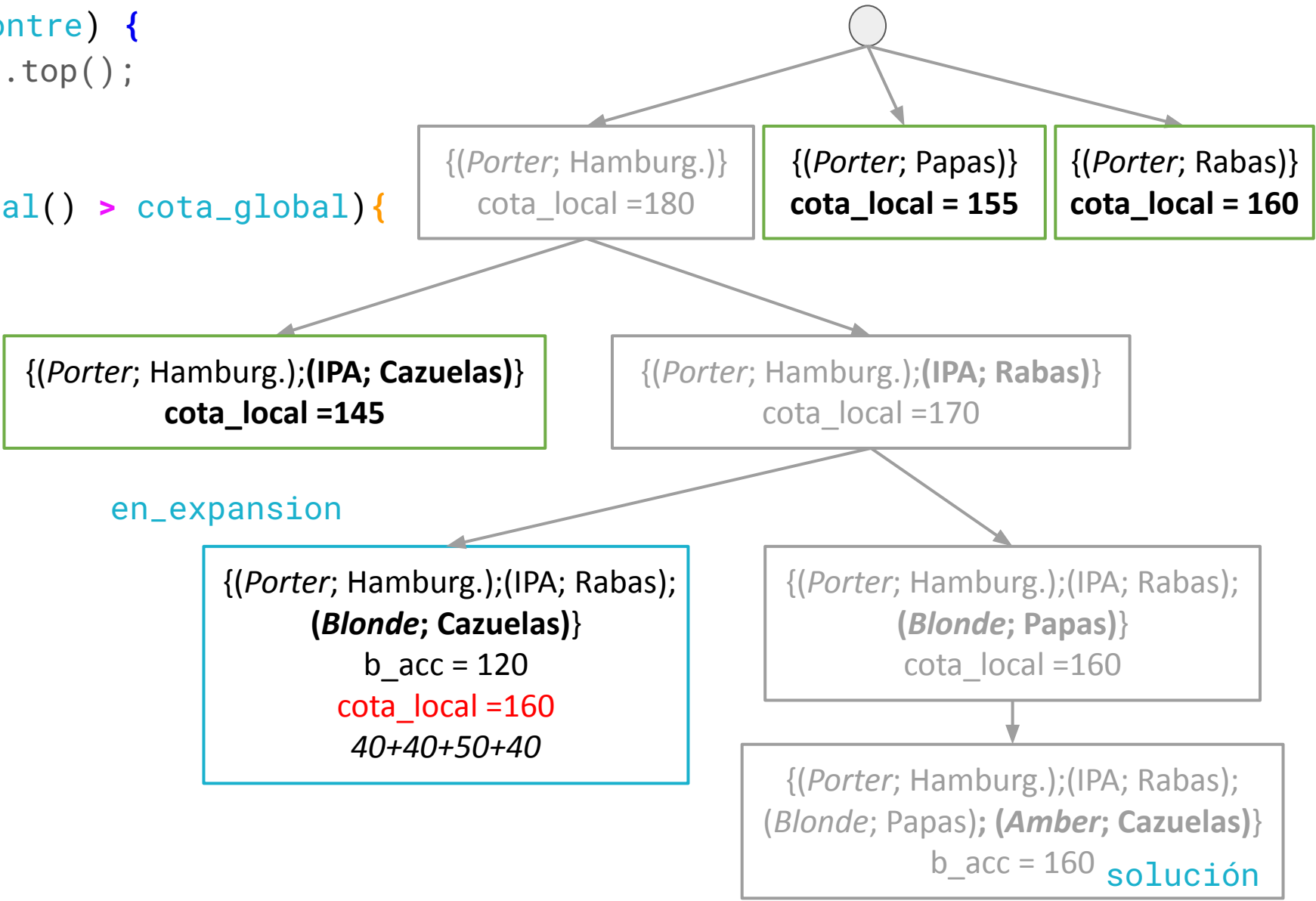
```

Encuentra la primer solución parcial



```
while (!vivos.empty() && !encontre) {
    Estado en_expansion = vivos.top();
    vivos.pop();

    if (en_expansion.getCotaLocal() > cota_global){
        } else
            encontre = true;
    }
} // end backBnB
```



cota_global = 160

vivos
{(Porter; Rabas)}; 160
{(Porter; Papas)}; 155
{(Porter; Hamburguesa);(IPA; Cazuelas)} ; 145

En este caso, el nodo en_expansion tiene cota_local que no mejora la cota_global, entonces encontre = true y se corta el while. No se continúa expandiendo porque los vivos nodos restantes tendrán una cota peor.

Problema II

Supongamos que estamos de vacaciones por una semana en una ciudad del extranjero. Desde la agencia de viajes nos ofrecen hacer una de actividad cada día. Nos encanta la idea porque además contamos con tiempo suficiente para hacer todas. El problema es que no podemos, porque contamos con solo **200 USD** y los precios son los siguientes:

- safari por la jungla: 140 USD
- excursión con vuelo y parapente: 70 USD
- paseo en catamarán: 50 USD
- tour gastronomico: 40 USD

Entonces, asignamos a cada actividad un número según cuántas ganas tenemos de hacerla.

- safari por la jungla: 180
- excursión con vuelo y parapente: 75
- paseo en catamarán: 70
- *tour* gastronomico: 20

Determinar qué actividades nos conviene hacer para quedarnos contentos.

Resolver mediante el diseño de un algoritmo de ramificación y poda.

Espacio de soluciones factibles

Actividad = (safari por la **jungla**; excursión con vuelo y **parapente**; paseo en **catamarán**; tour **gastronomico**)

Precio = (140; 70; 50; 40)

Entusiasmo = (180; 75; 70; 20)

Limite de dinero C = 200

Teniendo en cuenta los precios, las ganas que tenemos de hacer la actividad y el límite de dinero, podemos elegir entre las combinaciones de la tabla:

Actividad/es elegida/s	Precio	Entusiasmo
jungla	140	180
parapente	70	75
catamarán	50	70
tour gastronomico	40	20
jungla; catamarán	190	250
jungla; tour gastronomico	180	200
parapente; catamarán	120	145
parapente; tour gastronomico	110	95
catamarán; tour gastronomico	90	90
parapente; catamarán; tour gastronomico	160	165

Nos conviene hacer el safari y el paseo en catamarán.

Estado: tupla $\{x_0; x_1; ..x_n\}$ donde $x_i = 0$ o $x_i = 1$ según si la actividad se hace o no.

Espacio de soluciones factibles

Actividad = (jungla; parapente; catamarán; ***tour gastronomico***)

Precio = (140; 70; 50; 40)

Entusiasmo = (180; 75; 70; 20)

Limite de dinero C = 200

Estado: tupla $\{x_0; x_1; ..x_n\}$ donde $x_i = 0$ o $x_i = 1$ según si la actividad se hace o no.

Restricciones explícitas

- cada x_i es 0 ó 1

Restricciones implícitas

- $(C - \sum_{i=0}^{i=N} x_i p_i) \geq 0$

Actividad/es elegida/s	Precio	Entusiasmo
jungla	140	180
parapente	70	75
catamarán	50	70
tour gastronomico	40	20
jungla; catamarán	190	250
jungla; tour gastronomico	180	200
parapente; catamarán	120	145
parapente; tour gastronomico	110	95
catamarán; tour gastronomico	90	90
parapente; catamarán; tour gastronomico	160	165

Diseño de la solución ramificación y poda

Actividad = (jungla; parapente; catamarán; tour gastronomico)

Precio = (140; 70; 50; 40)

Entusiasmo = (180; 75; 70; 20)

Limite de dinero = 200

Analisis: Cada vez que elegimos una actividad, nos queda cierta cantidad de dinero para gastar en otra y además tenemos que evaluar cuánto queremos hacer la actividad actual y cuánto nos puede llegar a gustar las restantes.

- Ordenemos descendentemente por e_i/p_i (ponderación entre precio y entusiasmo)

$1.28 \geq 0.93 \geq 0.7 \geq 0.5$

$180/140 \geq 60/75 \geq 50/70 \geq 20/40$

jungla \geq parapente \geq catamarán \geq gastronómico

Diseño de la solución ramificación y poda

Actividad = (jungla; parapente; catamarán; tour gastronomico)

Precio = (140; 70; 50; 40) Entusiasmo = (180; 75; 70; 20) Limite de dinero C = 200

Ordenemos descendentemente por e_i/p_i

1.28 >= 0.93 >= 0.7 >= 0.5

180/140 >= 60/75 >= 50/70 >= 20/40

Sean S el conjunto de actividades elegidas; k la actividad con mayor relación entusiasmo/peso que no pertenece a S.

$$\sum_{i \in S} x_i e_i + (C - \sum_{i \in S} x_i p_i) (e_k / p_k)$$

Actual **Lo que puedo llegar a alcanzar**

Estado: tupla $\{x_0; x_1; ..x_n\}$ donde $x_i = 0$ o $x_i = 1$ según si la actividad se hace o no.

Estado: tupla {x₀; x₁; ..x_n} donde x_i = 0 ó x_i = 1 según si la actividad se hace o no.

Precio = (140; 70; 50; 40) Entusiasmo = (180; 75; 70; 20)

E/P: 1.28 >= 0.93 >= 0.7 >= 0.5

$$\sum_{i \in S} a_i e_i$$

Actual

+

$$(C - \sum_{i \in S} x_i p_i) (e_k / p_k)$$

Lo que puedo llegar a alcanzar

k la actividad con mayor relación entusiasmo/peso que no pertenece a S

Estado S	Actividad/es elegida/s	Precio	Entusiasmo	k	Restante	Cota local
(1;0;0;0)	jungla	140	180	parapente	(200 - 140) x (0.93) = 55.8	180 + 55.8 = 235.8
(0;1;0;0)	parapente	70	75	jungla	(200 - 70) x (1.28) = 166.4	75 + 166.4 = 241.4
(0;0;1;0)	catamarán	50	70	jungla	(200 - 50) x (1.28) = 192	70 + 192 = 262
(0;0;0;1)	tour gastronomico	40	20	jungla	(200 - 40) x (1.28) = 204.8	20 + 204.8 = 224.8
(1;0;1;0)	jungla; catamarán	190	250	parapente	(200 - 190) x (0.93) = 9.3	250 + 9.3 = 259.3
(1;0;0;1)	jungla; tour gastronomico	180	200	parapente	(200 - 180) x (0.93) = 18.6	200 + 18.6 = 218.6
(0;1;1;0)	parapente; catamarán	120	145	jungla	(200 - 120) x (1.28) = 102.4	145 + 102.4 = 247.4
(0;1;0;1)	parapente; tour gastronomico	110	95	jungla	(200 - 110) x (1.28) = 115.2	95 + 115.2 = 210.2
(0;0;1;1)	catamarán; tour gastronomico	90	90	jungla	(200 - 90) x (1.28) = 140.8	90 + 140.8 = 230.8
(0;1;1;1)	parapente; catamarán; gastronomico	160	165	jungla	(200 - 160) x (1.28) = 51.2	165 + 51.2 = 216.2

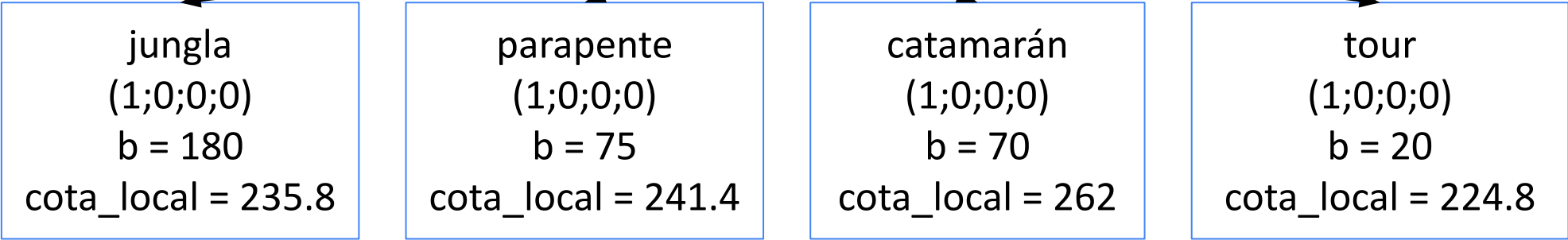
Actividad = (safari por la **jungla**; excursión con vuelo y **parapente**; paseo en **catamarán**; tour **gastronomico**)

Precio = (140; 70; 50; 40) Entusiasmo = (180; 75; 70; 20)

LIMITE = 200 USD

**Generar hijos nivel 0,
tomar 1 actividad**

ninguna
(0;0;0;0)



Vivos
jungla; 235.8
catamarán; 262
parapente; 241.4
tour; 224.8

Estado S	Cota local
(1;0;0;0)	235.8
(0;1;0;0)	241.4
(0;0;1;0)	262
(0;0;0;1)	224.8
(1;0;1;0)	259.3
(1;0;0;1)	218.6
(0;1;1;0)	247.4
(0;1;0;1)	210.2
(0;0;1;1)	230.8
(0;1;1;1)	216.2

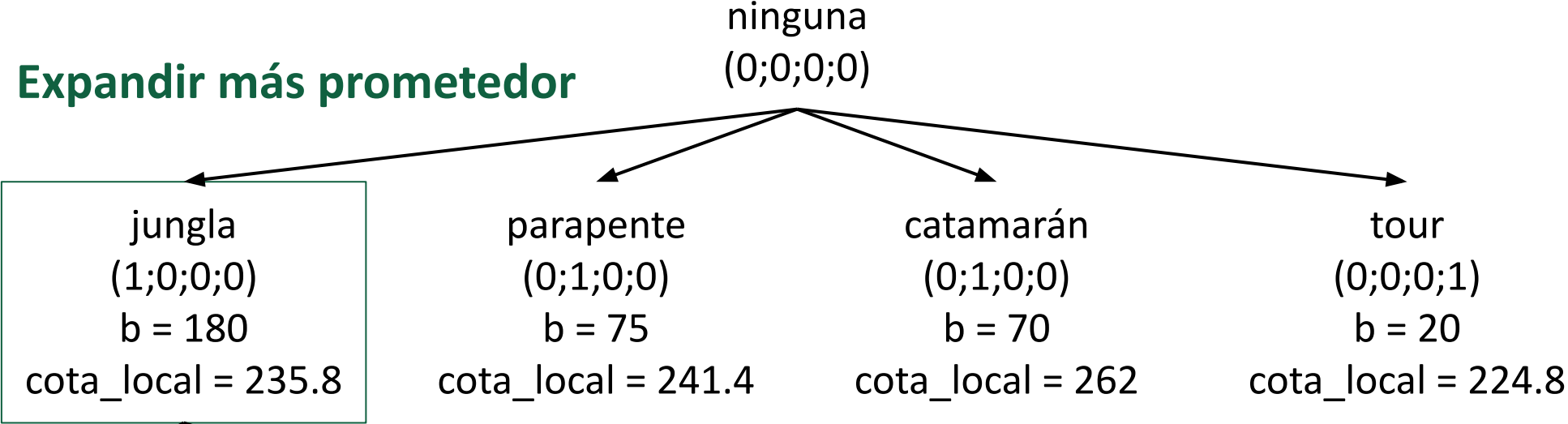
Actividad = (safari por la **jungla**; excursión con vuelo y **parapente**; paseo en **catamarán**; tour **gastronomico**)

Precio = (140; 70; 50; 40) Entusiasmo = (180; 75; 70; 20)

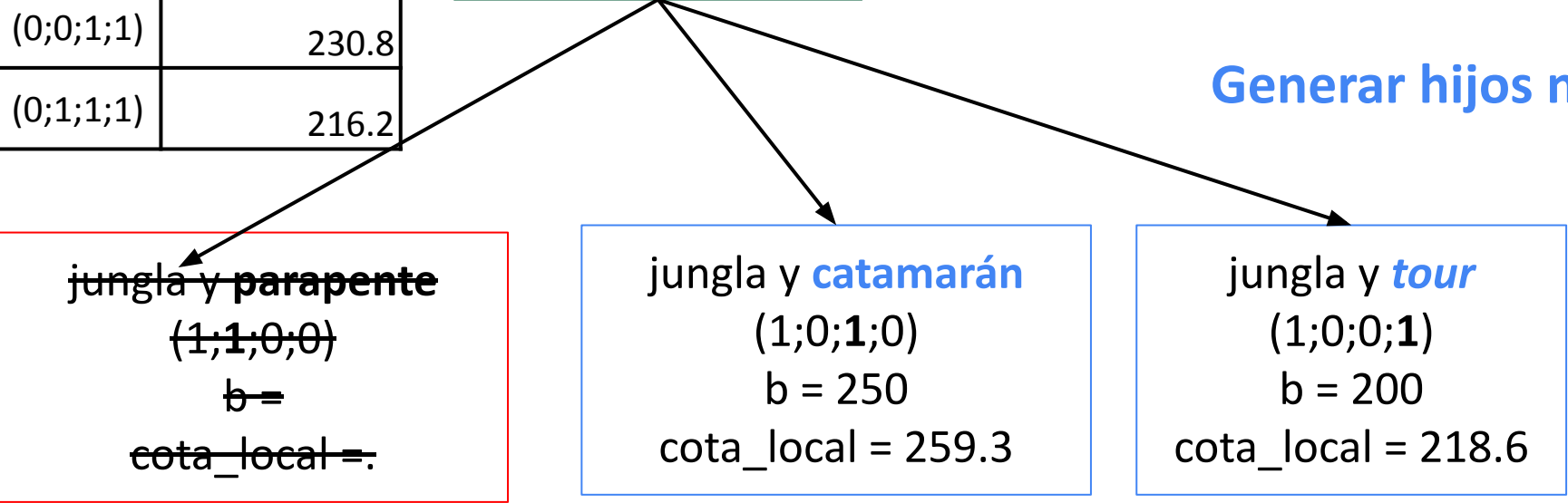
LIMITE = 200 USD

Estado S	Cota local
(1;0;1;0)	259.3
(1;0;0;1)	218.6
(0;1;1;0)	247.4
(0;1;0;1)	210.2
(0;0;1;1)	230.8
(0;1;1;1)	216.2

Expandir más prometedor



Generar hijos nivel 1: tomar otra actividad



Vivos
catamarán; 262
parapente; 241.4
tour; 224.8

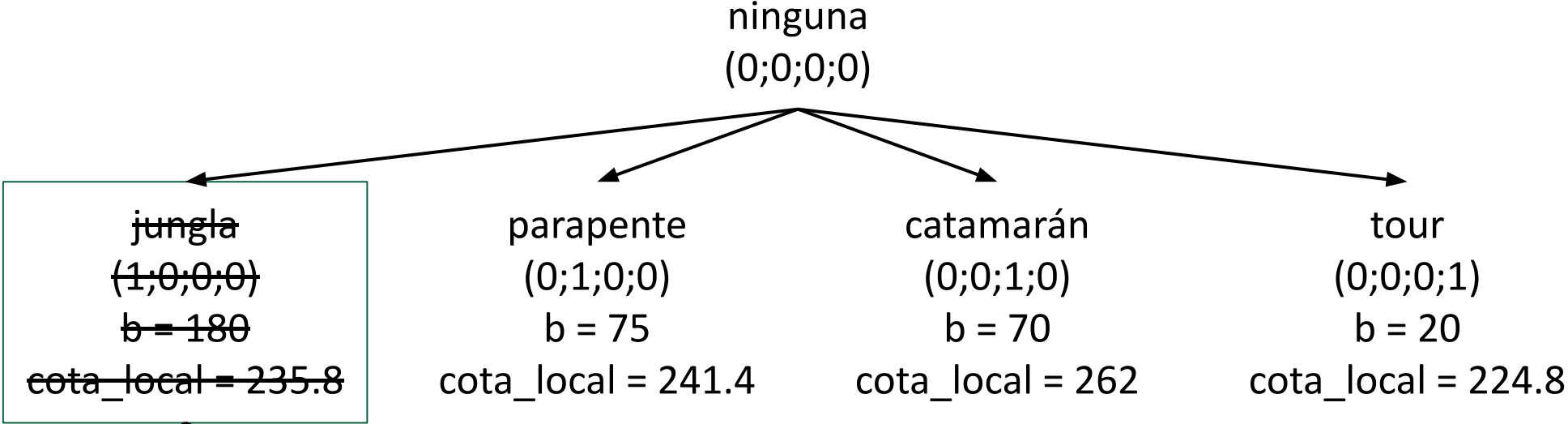
No es factible

Actividad = (safari por la **jungla**; excursión con vuelo y **parapente**; paseo en **catamarán**; tour **gastronomico**)

Precio = (140; 70; 50; 40) Entusiasmo = (**180**; 75; **70**; 20)

LIMITE = 200 USD

Estado S	Cota local
(1;0;1;0)	259.3
(1;0;0;1)	218.6
(0;1;1;0)	247.4
(0;1;0;1)	210.2
(0;0;1;1)	230.8
(0;1;1;1)	216.2



Generar hijos nivel 1: tomar otra actividad

Vivos
catamarán; 262
parapente; 241.4
tour; 224.8

Solución: no podemos contratar más actividades.
Nos quedan 10 USD, no hay actividades por ese precio.

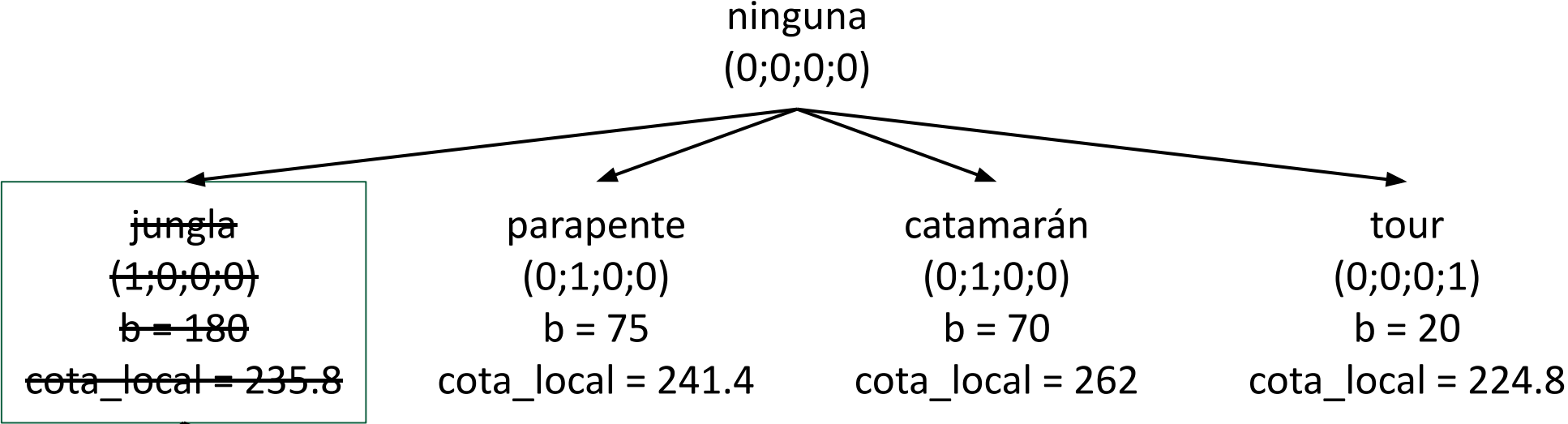
cota_global = 250

Actividad = (safari por la **jungla**; excursión con vuelo y **parapente**; paseo en **catamarán**; tour **gastronomico**)

Precio = (140; 70; 50; 40) Entusiasmo = (**180**; 75; **70**; 20)

LIMITE = 200 USD

Estado S	Cota local
(1;0;1;0)	259.3
(1;0;0;1)	218.6
(0;1;1;0)	247.4
(0;1;0;1)	210.2
(0;0;1;1)	230.8
(0;1;1;1)	216.2



Generar hijos nivel 1: tomar otra actividad

jungla y **catamarán**
(1;0;**1**;0)
b = 250
cota_local = 259.3

Solución actual

jungla y **tour**
(1;0;0;**1**)
b = 200
cota_local = 218.6

No mejora la cota

Vivos
catamarán; 262
parapente; 241.4
tour; 224.8

cota_global = 250

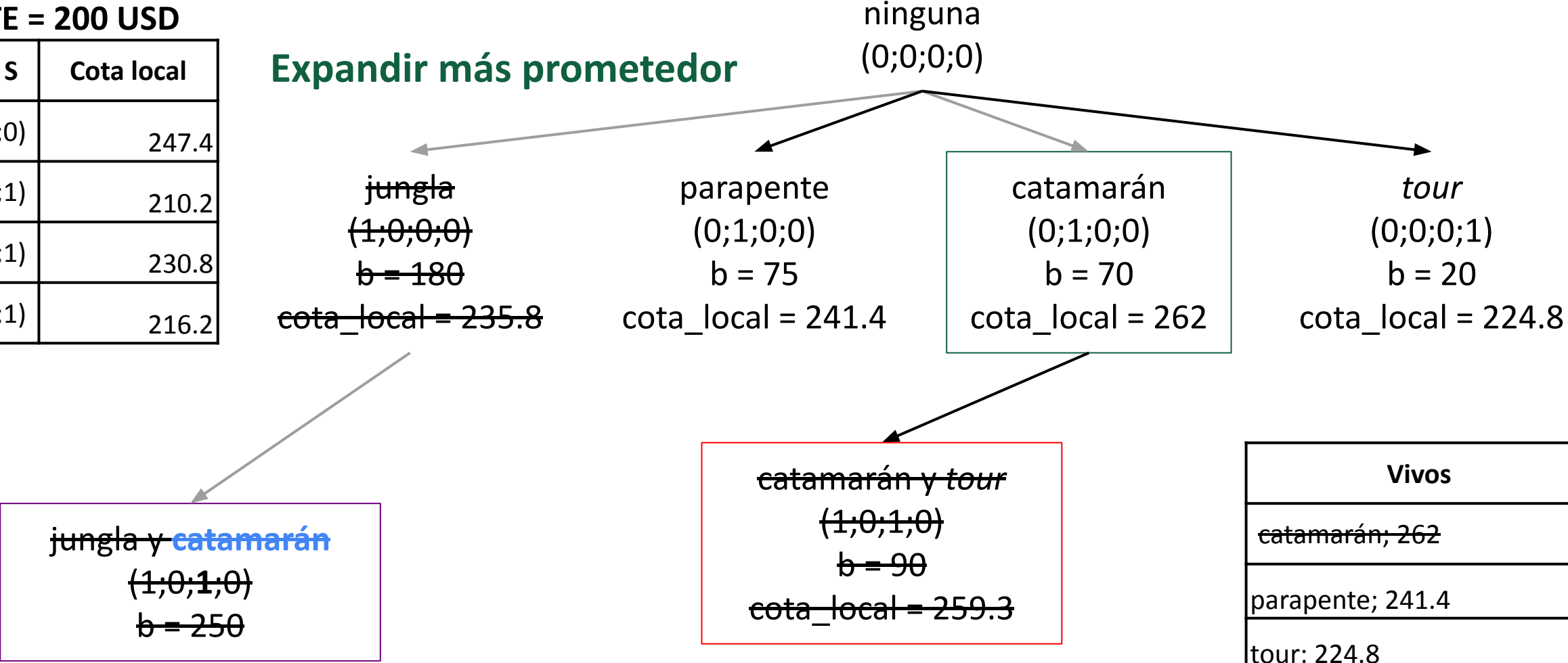
Actividad = (safari por la **jungla**; excursión con vuelo y **parapente**; paseo en **catamarán**; tour **gastronomico**)

Precio = (140; 70; 50; 40) Entusiasmo = (180; 75; 70; 20)

LIMITE = 200 USD

Estado S	Cota local
(0;1;1;0)	247.4
(0;1;0;1)	210.2
(0;0;1;1)	230.8
(0;1;1;1)	216.2

Expandir más prometedor



Poda

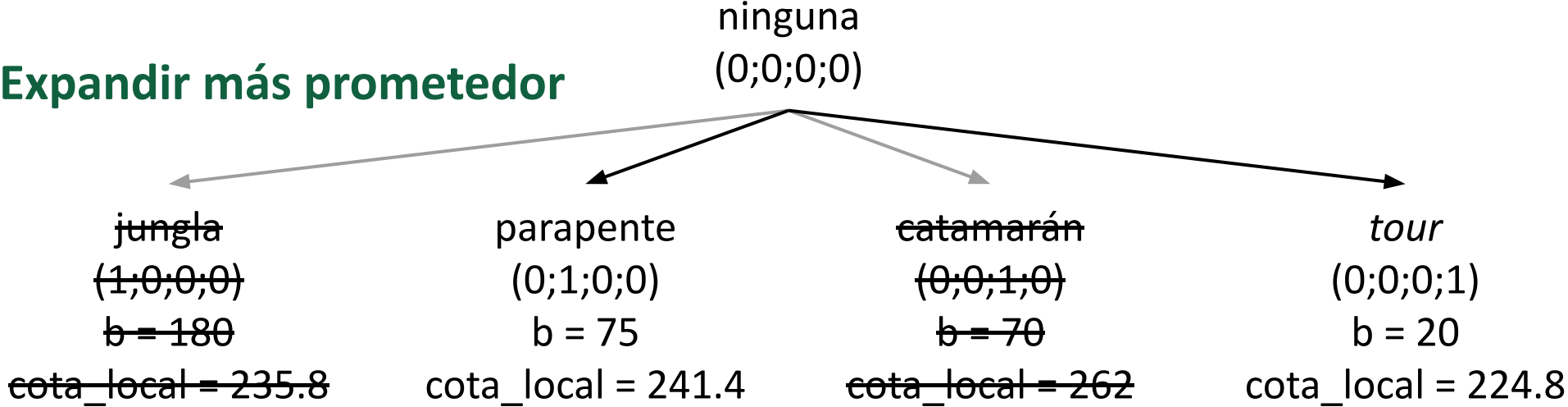
Actividad = (safari por la **jungla**; excursión con vuelo y **parapente**; paseo en **catamarán**; tour **gastronomico**)

Precio = (140; 70; 50; 40) Entusiasmo = (180; 75; 70; 20)

LIMITE = 200 USD

Estado S	Cota local
(0;1;1;0)	247.4
(0;1;0;1)	210.2
(0;0;1;1)	230.8
(0;1;1;1)	216.2

Expandir más prometedor



jungla y **catamarán**
(1;0;1;0)
b = 250

Ningún nodo vivo mejora la cota

Vivos
parapente; 241.4
tour; 224.8

cota_global = 250

Actividad = (safari por la **jungla**; excursión con vuelo y **parapente**; paseo en **catamarán**; tour **gastronomico**)

Precio = (140; 70; 50; 40) Entusiasmo = (180; 75; 70; 20)

LIMITE = 200 USD

jungla y **catamarán**
(1;0;1;0)
b = 250

En el árbol cada nivel es una actividad más de las que siguen en la lista y la decisión es cuál de esas actividades incluir.

Adaptación del esquema general al problema

```
class EstadoMochilaBnB {      En el árbol cada nivel es una actividad más y la decisión es
public:                       qué actividad/objeto incluir

    EstadoMochilaBnB (int N) ;

    EstadoMochilaBnB (const EstadoMochilaBnB & padre) ;

    void incluir (int objeto, int *P, int *B);

    unsigned int getCotaLocal () const;

    unsigned int getBeneficio () const;

    int getNivel () const;

    unsigned int pesoActual () const;

}
```

Adaptación del esquema general al problema

```
While (!vivos.empty() && ! encuentre) {  
  
    EstadoMochilaBnB en_expansión = vivos.top(); //nodo más prometedor  
    vivos.pop();  
  
    if (en_expansion.getCotaLocal () > cota_global){  
  
        hijos = Expandir (en_expansión, P, B);  
        for ( c/ hijo h en hijos)  
            if (!Poda2 (h, solucion)) // poda por factibilidad y cota global  
                if (es_solucion (h)  
                    solucion = h;  
                else  
                    vivos.poner(h);  
            else  
                encuentre= true;
```

Adaptación del esquema general al problema

```
list <EstadoMochilaBnB> expandir (const EstadoMochilaBnB & e, int B[N], int P[N])
{
    // A partir del estado actual e, genera los estados hijos
    list <EstadoMochilaBnB> hijos;
    for (int h=e.getNivel()+1;h<N;h++)
    {
        if (e.pesoActual()+P[h] <= C) // Función de factibilidad
        {
            EstadoAsignacionBnB nuevo(e); // El estado "nuevo" se genera a partir de "e"
            nuevo.asignar(h,B,P);
            hijos.push_back(nuevo);
        }
    }
    return hijos;
}
```

Adaptación del esquema general al problema

```
struct comparator
{
    bool operator()(EstadoMochilaBnB &p, EstadoMochilaBnB&q) const {
        return p.getCotaLocal() < q.getCotaLocal();
    }
};

void mochilaBnB(const EstadoMochilaBnB& inicial,int* B,int* P)
{ // begin bnb
    EstadoAsignacionBnB solucion(N);

    //inicialización
    priority_queue<EstadoAsignacionBnB,vector<EstadoAsignacionBnB>,comparator> vivos;
    vivos.push(inicial);
    list <EstadoAsignacionBnB> hijos;

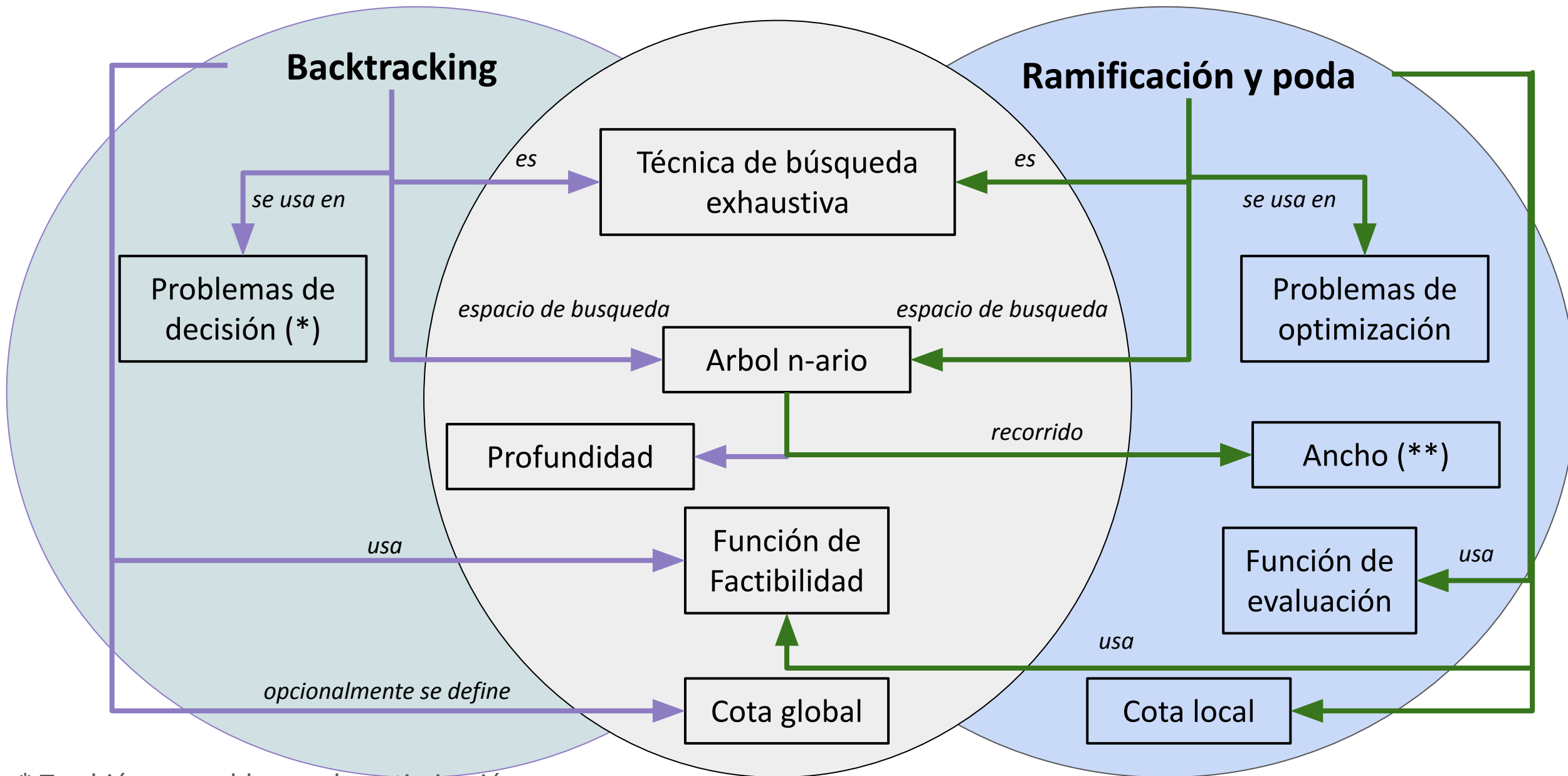
    int cota_global = 0;
    bool encuentre = false;
```

```

while (!vivos.empty() && ! encuentre) {
    EstadoMochilaBnB en_expansion = vivos.top(); //nodo más prometedor
    vivos.pop();

    if (en_expansion.getCotaLocal() > cota_global){// poda por cota global
        hijos = expandir (en_expansion,B,P);
        for (h: hijos){
            int aux_cota = h.getCotaLocal();
            if (aux_cota > cota_global ) {// poda por cota global
                if (C - h.pesoActual() <= min(P)) { // no entran más objetos
                    solucion = h;
                    cota_global = aux_cota;
                }
                else
                    vivos.push(h);
            }
        }
    } else
        encuentre = true;
}
} // end mochilaBnB

```



* También en problemas de optimización

** También profundidad

Eficiencia

Parámetros de la búsqueda que determinan la eficiencia:

- factor de bifurcación (b): número de hijos de un nodo
- profundidad de la solución (p): longitud del camino del nodo inicial a la solución.

Complejidad temporal: $O(b^p)$

A fin de profundizar los temas, se recomienda:

- Realizar la implementación del problema del viajante usando ramificación y poda.
- Implementar y comparar resoluciones ramificación y poda vs. *backtracking*
 - Incluir *métricas* en el código para analizar tiempos de ejecución, cantidad de estados generados, cantidad de estados solución alcanzados y cantidad de estados podados.
 - Comparar la complejidad espacial.

Bibliografía:

- *Fundamentos de Algoritmia.*
Brassard, G.; Bratley, P.
- *Estructuras de datos y algoritmos.*
Aho, A; Hopcroft, J y Ullman, J.