BRANCH & BOUND

RAMIFICACIÓN Y PODA (BRANCH & BOUND) ES UN ESQUEMA DE EXPLORACIÓN EXHAUSTIVA, CON UNA CARACTERIZACIÓN SIMILAR A VUELTA-ATRÁS, PERO EL ÁRBOL DE ESTADO SE RECORRE POR ORDEN DE COSTES (O BENEFICIOS) ESTIMADOS.

* BRANCH & BOUND ES UN ESQUEMA DE BÚSQUEDA INFORMADO (PRIMERO SE EXPLORAN LAS ALTERNATIVAS MÁS PROMETEDORAS), MÁS EFICIENTE QUE BACKTRACKING (ES UN ESQUEMA "CIEGO").
* B & B PODA EN GENERAL MUCHISIMO MÁS QUE BACKTRACKING CON PBMSC
* B & B ES MÁS COSTOSO EN MEMORIA, PUES HAY QUE MANTENER EXPLÍCITAMENTE UNA LISTA DE NODOS VIVOS

BRANCH & BOUND

```
struct nodo { double cost; ... }

branch_and_bound(datos D, ... ) {
  P_queue<double, nodo> VIVOS;
    ...

  y = nodo raíz;        VIVOS.insert(coste_estimado(y), y);

  coste_mejor = ∞ u obtenido mediante una heurística
                  externa

  while ( ! VIVOS.empty() ) {
      y = VIVOS.inf (VIVOS.find_min());
    VIVOS.del_min();
      L = lista de los hijos de y;
      forall (x, L) {
        if ( x es solucion ) {
            if ( coste(x) < coste_mejor ) {
                coste_mejor = coste(x);
                mejor_sol = x;
                depurar la lista de VIVOS
            }
        } else if( x es factible &&
                coste_estimado(x) < coste_mejor)
                VIVOS.insert(coste_estimado(x), x);
            //else Poda PBMSC
      }
  }
```

el nodo a expandir es el más prometedor (el de menor coste estimado); se expande totalmente y "muere"

BRANCH & BOUND

1. Los <u>nodos</u> contienen información explícita sobre la solución en curso, el nivel, el coste de la solución en curso a la que representan, cualesquiera otras marcas, etc.

2. Al generar la lista L se usará la información contenida en y para calcular toda la información correspondiente a cada nodo x

3. La <u>depuración</u> de la lista <u>VIVOS</u> consiste en eliminar todos los nodos de VIVOS cuyo coste estimado sea $\geq$ que el coste mejor recién obtenido