Ramificación y Poda

Pedro O. Pérez M., PhD.

Análisis y diseño de algoritmos avanzados
Tecnológico de Monterrey

pperezm@tec.mx

12-2021

Contenido

Introducción

- Ramificación y poda (Branch and Bound, B&B) es una técnica similar a Backtracking. La diferencia es que en el momento en que se encuentre que una solución parcial no puede ser la solución buscada, se detiene el proceso (poda). Esta técnica se utiliza para solucionar problemas de optimización.
- En el argot de los problemas de optimización,
  - Una solución factible es aquella que cumple con todas las restricciones que tenga el problema.
  - La solución óptima es la solución factible que maximiza o minimiza la función, dependiendo de lo que se busque.
  - La función que se quiere maximizar o minimizar se le llama función objetivo.

- Para hacer B&B es necesario que para cada nodo del árbol se pueda establecer un límite para el mejor valor de la función objetivo sobre cualquier solución que pueda ser obtenida.
- No hay una técnica general para establecer esta cota, es decir, es muy particular para cada problema. Sin embargo, una forma que normalmente resulta adecuada es la de relajar el problema original, es decir, quitar o cambiar algunas de las restricciones de tal forma que tengamos un problema más sencillo de resolver y podamos encontrar una solución adecuada para iniciar el proceso, en forma más rápida.

- En cada nodo, se compara el límite de dicho nodo con el mejor valor que se haya obtenido hasta ahora, el cual al inicio es ∞, si el problema es de minimización y $-\infty$, si el problema es de maximización.
- Si el límite no es mejor que la mejor solución obtenida hasta ahora, entonces el nodo no es promisorio y ahí se detiene la búsqueda con esa rama (se poda la rama), lo que significa que ninguna rama que parte de este nodo nos dará una mejor solución.
- También se puede podar una rama si la solución parcial en el nodo actual viola alguna de las restricciones establecida por el problema, es decir, si representa una solución no factible.

- Cuando se llega a una solución completa, se compara con el mejor valor encontrado hasta ahora. Si es mejor, se actualiza el mejor valor. Si no, la solución se desecha.
- A diferencia de Backtracking, donde se generaba un solo hijo del nodo a la vez, en B&B, se deben generar todos los hijos (ramificación) del nodo más prometedor, de entre todos los que se encuentran en la frontera del árbol, que no han sido podados.
- La frontera del árbol está formada por todos los nodos que no han sido expandidos, a los cuales se les llama nodos vivos.
- Mientras que Backtracking emplea "primero profundidad", B&B utiliza "primero el mejor".

Forma general

Procedure 1 BRANCH_AND_BOUND

Input: $frontier : Set < Node >$, $currentSolution : Node$, $currentBest : Number$

$node \leftarrow$ BEST($frontier$)
if NOT(PROMISSORY($node$)) **then**
  return
end if
if SOLUTION($node$) **and** FEASIBLE($node$) **and** LIMIT($node$) IS BETTER THAN $currentBest$ **then**
  $currentSolution \leftarrow node$
  $currentBest \leftarrow$ LIMIT($node$)
  return
end if
NEXT SLIDE

```
for i ← 1 to CHILDS(node) do
  ADD(i, frontier)
end for
if NOT(EMPYT(frontier)) then
  BRANCH_AND_BOUND(frontier)
end if
```

Dado un conjunto W trabajos, $W = w_1, w_2, ..., w_n$ y un conjunto de T de trabajadores, $T = t_1, t_2, ..., t_n$, y una matriz C de tamaño (nxn), donde las filas representan los n trabajadores y las columnas los n trabajos. Cada celda c_{ij} contiene el costo de asignar el trabajador i al trabajo j. Encontrar la mejor asignación de trabajos y trabajadores, es decir, el menor costo total, donde, cada trabajo sólo se puede ser asignado a un solo trabajador y a cada trabajador sólo se le puede asignar un trabajo.

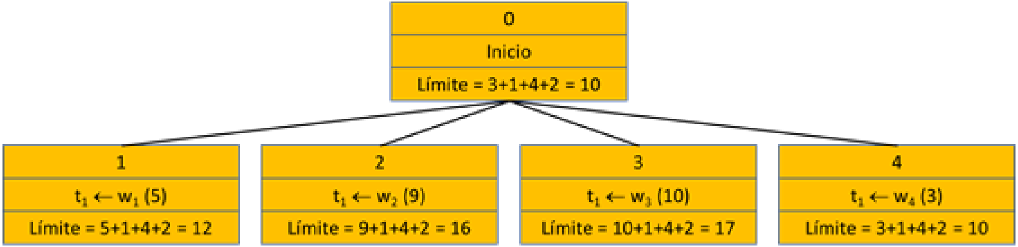
Por ejemplo, 4 trabajadores y 4 trabajos. Los costos de asignación están dados por la siguiente matriz:

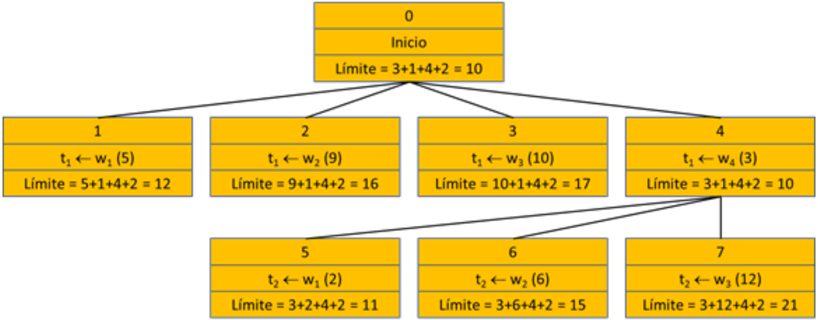
5	9	10	3
2	6	12	1
7	4	4	8
11	16	2	14

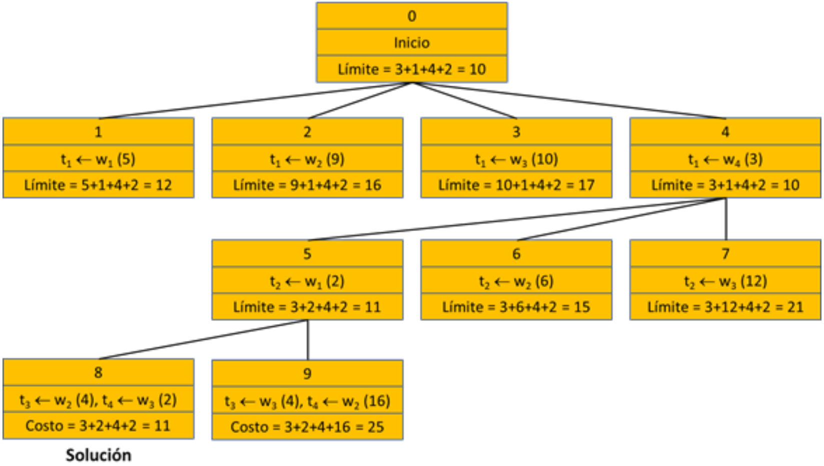
Si tomamos como punto de partida el menor costo de cada columna, obtenemos el siguiente nodo raíz:

0
Inicio
Límite = 3+1+4+2 = 10

IMPORTANTE: Este nodo es factible, se repiten asignaciones; sin embargo, nos sirve como punto de arranque.


0
Inicio
Límite = 3+1+4+2 = 10
1
$t_1 \leftarrow w_1$ (5)
Límite = 5+1+4+2 = 12
2
$t_1 \leftarrow w_2$ (9)
Límite = 9+1+4+2 = 16
3
$t_1 \leftarrow w_3$ (10)
Límite = 10+1+4+2 = 17
4
$t_1 \leftarrow w_4$ (3)
Límite = 3+1+4+2 = 10


0
Inicio
Límite = 3+1+4+2 = 10
1
$t_1 \leftarrow w_1$ (5)
Límite = 5+1+4+2 = 12
2
$t_1 \leftarrow w_2$ (9)
Límite = 9+1+4+2 = 16
3
$t_1 \leftarrow w_3$ (10)
Límite = 10+1+4+2 = 17
4
$t_1 \leftarrow w_4$ (3)
Límite = 3+1+4+2 = 10
5
$t_2 \leftarrow w_1$ (2)
Límite = 3+2+4+2 = 11
6
$t_2 \leftarrow w_2$ (6)
Límite = 3+6+4+2 = 15
7
$t_2 \leftarrow w_3$ (12)
Límite = 3+12+4+2 = 21


0
Inicio
Límite = 3+1+4+2 = 10
1
$t_1 \leftarrow w_1$ (5)
Límite = 5+1+4+2 = 12
2
$t_1 \leftarrow w_2$ (9)
Límite = 9+1+4+2 = 16
3
$t_1 \leftarrow w_3$ (10)
Límite = 10+1+4+2 = 17
4
$t_1 \leftarrow w_4$ (3)
Límite = 3+1+4+2 = 10
5
$t_2 \leftarrow w_1$ (2)
Límite = 3+2+4+2 = 11
6
$t_2 \leftarrow w_2$ (6)
Límite = 3+6+4+2 = 15
7
$t_2 \leftarrow w_3$ (12)
Límite = 3+12+4+2 = 21
8
$t_3 \leftarrow w_2$ (4), $t_4 \leftarrow w_3$ (2)
Costo = 3+2+4+2 = 11
9
$t_3 \leftarrow w_3$ (4), $t_4 \leftarrow w_2$ (16)
Costo = 3+2+4+16 = 25
Solución

Problema de asignación

Procedure 2 ASSIGMENT_PROBLEM

Input: C : Matrix[N][N]
frontier : Priority Queue<Node>,
currentSolution : Node,
currentBest : Number,
assigned : Array[N],
assigment : Array[N],
node : Node,
child : Node

```
currentBest ← ∞
currentSolution ← None
for i ← 1 to N do
  assigned[i] ← false
  minimum ← C[i][0]
  work ← 0
  for j ← 2 to N do
    if C[i][j] < minimum then
      minimum ← C[i][j]
      work ← j
    end if
  end for
  assigment[i] ← work
end for
NEXT SLIDE
```

```
node ← new Node(0, assigment, assigned, -1)
frontier.enqueue(node)
while not frontier.empty() do
   node ← frontier.dequeueMin()
   if node.limit < currentBest then
      NEXT_SLIDE
   end if
end while
```

```
if node.level < N then
  for i ← 1 to N do
    if not node.assigned[i] then
      child ← new Node(node.level + 1, node.assigment, node.assigned, i)
      frontier.push(child)
    end if
  end for
else
  currentBest ← node.limit
  currentSolution ← node
end if
```