# Branch-and-Bound Algorithm

Björn Barrefors

June 22, 2014

## 1 Branch-and-Bound Algorithm

### 1.1 Equations

Below we introduce the equations used in our algorithm.
Eq. 1 is the payoff of all processors ranked as stated in the equation.
Eq. 2 is the capacity constraint which need to hold at a branch for it to be valid.
Eq. 3 is the current total value.
Eq. 4 is the residual capacity.
Eq. 5 is the lower bound equation.

$$\frac{p_1}{c_1} \leq \frac{p_2}{c_2} \leq \ldots \leq \frac{p_n}{c_n} \tag{1}$$

$$\sum_{j=\hat{j}}^{n} c_j(1 - \hat{x_j}) \geq \hat{c} \tag{2}$$

$$\hat{z} = \sum_{j=1}^{n} p_j \hat{x_j} \tag{3}$$

$$\hat{c} = U_{tot} - \sum_{j=1}^{n} c_j \hat{x_j} \tag{4}$$

$$\hat{b} = \hat{z} + \hat{c}\frac{p_{j+1}}{c_{j+1}} \tag{5}$$

Variable definitions:

- $n$ : Number of processors
- $p_j$ : Power consumption of processor $j$
- $c_j$ : Capacity of processor $j$
- $U_{tot}$ : Total utilization of all tasks
- $\hat{z}$ : Total power consumption of current schedule

- $\hat{c}$ : Total residual task utilization of current solution
- $\hat{b}$ : Lower bound of current solution
- $\hat{j}$ : Current processor of current solution
- $[\hat{X_j}]$ : Current solution, a list of lenght $n$ where $x_j = 1$ if processor $j$ is selected and $x_j = 0$ if processor $j$ is not selected
- $\mathcal{H}$ : Min heap of possible solutions with elements $s$ consisting of a tuple $([\hat{X_j}], \hat{b}, \hat{j})$

We will build a tree of possible solutions. If a solution is not valid or inferior to another solution that branch will not be pursued. The tree is constructed as follows: The processor with best payoff (Eq. 1) is selected as a new node, that node branches into two possible solutions, in one the processor is added to the solution, on the other it is not added to the solution. As long as a solution is valid that leaf of the tree will be kept to be compared to by future solutions. A lower bound (Eq. 5) estimate is used to calculate the possible performance of a branch. The branch with lowest bound is selected. We will use a min heap to keep track of solutions. The root node is popped, based on lowest lower bound, and two new nodes are created based on the above branching. Push valid schedules onto the heap. Repeat. Each node stores its lower bound $\hat{b}$, schedule ($\hat{X_j}$), and current node $\hat{j}$.

**Algorithm 1** Branch-and-Bound Algorithm

**Require:** $n$, $U_{tot}$, $[p_j]$, $[c_j]$; processors sorted based on payoff (Eq. 1)

Initialize:
$[\hat{X_j}] = [0, ..., 0]$
$\hat{j} = 1$
$\hat{z} = 0$
$\hat{c} = U_{tot}$
$\hat{b} =$ Eq. 5
$\mathcal{H}$.push($\hat{b}$, $\hat{j}$, $[\hat{X_j}]$)
**while** $j < n$ **do**
    $s = \mathcal{H}$.pop()
    $[\hat{X_j}] = s.[\hat{X_j}]$
    $\hat{j} = s.\hat{j}$
    Create branch node for processor not added to solution
    $\hat{z} =$ Eq. 3
    $\hat{c} =$ Eq. 4
    $\hat{b} =$ Eq. 3
    $\hat{j} += 1$
    **if** Eq. 2 holds **then**
        $\mathcal{H}$.push($\hat{b}$, $\hat{j}$, $[\hat{X_j}]$)
    **end if**
    $\hat{j} -= 1$
    Create branch node for processor added to solution
    $X_{\hat{j}} = 1$
    $\hat{z} =$ Eq. 3
    $\hat{c} =$ Eq. 4
    $\hat{b} =$ Eq. 3
    $\hat{j} += 1$
    **if** Eq. 2 holds **then**
        $\mathcal{H}$.push($\hat{b}$, $\hat{j}$, $[\hat{X_j}]$)
    **end if**
**end while**
$s = \mathcal{H}$.pop()
$[\hat{X_j}] = s.[\hat{X_j}]$
$\hat{j} = s.\hat{j}$
Create branch node for processor not added to solution
$\hat{z_0} =$ Eq. 3
$\hat{c_0} =$ Eq. 4
Create branch node for processor added to solution
$X_{\hat{j}} = 1$
$\hat{z_1} =$ Eq. 3
$\hat{c_1} =$ Eq. 4
**if** $\hat{c_0} > 0$ **or** $\hat{z_0} > \hat{z_1}$ **then**
    $\hat{z} = \hat{z_1}$
**else**
    $X_{\hat{j}} = 0$
    $\hat{z} = \hat{z_0}$
**end if**
**return** $\hat{z}$, $[\hat{X_j}]$