Algorithm 1 Insertion tache

```
1: function ADD_TASK(du_t, cpu_t)                                  ▷ duration, number of cpu
2:     start_key ← [0, du_t, cpu_t]
3:     end_key ← [+∞, +∞, +∞]
4:
5:     profile_tree → node_loop(start_key, end_key, {
6:         starting_time_min ← (freespace → {starting_time})
7:         processor_range_t ← (freespace → {cpu})
8:         return 1
9:     })
10:                                          ▷ Remove the number of cpu for the range
11:    processor_range_t ← (processor_range_t → reduce_to_basic(cpu_t))
12:    start_key ← [0, 0, 0]
13:
14:    end_key ← [starting_time_min + du_t, +∞, +∞]
15:    profile_tree → node_loop(start_key, end_key, {              ▷ Research with contraints
16:        push freespace_impacted, freespace
17:        return 0
18:    })
19:
20:    for freespace ← freespace_impacted do
21:        cut_freespace(freespace, starting_time_min, du_t, processor_range_t)
22:    end for
23: end function
```

Algorithm 2 Decoupage Freespace

```
1: function CUT_FREESPACE(freespace, start_time, duration, processor_range)
2:     if Intersection(freespace → {cpu}, processor_range) > 0 then
3:         profile_tree → remove(freespace)
4:
5:         if freespace → {starting_time} < start_time then
6:             new freespace right_freespace ← (freespace → {starting_time}, start_time − freespace →
   {starting_time}, freespace → {cpu})
7:             profile_tree → add(right_freespace)
8:         end if
9:
10:        if freespace → {starting_time} < (start_time + duration) then
11:            new freespace left_freespace ← ((start_time + duration), freespace → {duration} −
   (start_time + duration) − freespace → {starting_time}, freespace → {cpu})
12:            profile_tree → add(left_freespace)
13:        end if
14:
15:        range_test ← (freespace → {cpu} → copy())
16:        range_test → remove(processor_range)
17:
18:        if range_test → size() > 0 then
19:            new freespacenew_freespace ← (freespace → {starting_time}, freespace →
   {duration}, range_test)
20:            profile_tree → add(new_freespace)
21:        end if
22:
23:    end if
24:
25: end function
```

Algorithm 3 Suppression tache

```
1: function REMOVE_TASK(st_t, d_t, cpu_t)                    ▷ Starting time, duration, cpu range
2:     et_t ← (st_t + d_t)                                    ▷ End time of job
3:
4:     for t ← st_t to et_t do                                ▷ Update cpu range for the job placement
5:         add tab{t} ← cpu_t)
6:     end for
7:
8:     start_key ← [0, 0, 0]
9:     end_key ← [st_t + d_t, +∞, +∞]
10:
11:    profile_tree → node_loop(start_key, end_key, {
12:
13:    if (cpu_t inter freespace{cpu}) > 0 then
14:        push freespace_impacted, freespace
15:        st_t2 ← (freespace → {starting_time})
16:        et_t2 ← (freespace → {starting_time} + freespace → {duration})
17:
18:        if (st_t ≤ st_t2 and st_t2 ≤ et_t) then
19:            direction{freespace} ← −1
20:        else if (st_t2 ≤ st_t and st_t ≤ et_t2) then
21:            direction{freespace} ← 1
22:        end if
23:
24:        if exist direction{freespace} then
25:            for t ← st_t2 to et_t2 do                      ▷ Update cpu range for freespace placement
26:
27:                if exist tab{t} then
28:                    add tab{t} ← (freespace → {cpu} inter tab{t})
29:                else
30:                    add tab{t} ← (freespace → {cpu})
31:                end if
32:
33:            end for
34:        end if
35:    end if
36:    }
37:
38:    for freespace ← freespace_impacted do
39:        extend_freespace(freespace)
40:    end for
41: end function
```

Algorithm 4 Augmentation Freespace

```
1: function EXTEND_FREESPACE(freespace)
2:     profile_tree → remove(freespace)
3:
4:     if direction{freespace} = 1 then
5:         t_next ← (freespace → {starting_time} + freespace → {duration})
6:     else
7:         t_next ← freespace → {starting_time}
8:     end if
9:
10:    new_t ← t_next
11:    while (exist tab{new_t} and (tab{new_t} inter freespace → {cpu}) ≥ freespace → {cpu} → size()) do
12:        t_max ← new_t
13:        new_t ← (new_t + direction{freespace})
14:    end while
15:
16:    if direction{freespace} = 1 then
17:        freespace → {duration} ← (t_next − freespace → {starting_time})
18:    else
19:        freespace → {duration} ← (freespace → {starting_time} − new_t + freespace → {duration})
20:        freespace → {starting_time} ← new_t
21:    end if
22:
23:    TODO :VERIFIER SI L'ESPACE N'EST PAS COMPRIS DANS UN AUTRE
24: end function
```

Algorithm 5 Suppression des Freespaces inutiles

```
1: function REMOVE_UNNECESSARY_FREESPACE(freespace, freespace_list)
2:     for space ← freespace_list do
3:         if (freespace → {cpu} inter space → {cpu}) = (freespace → {cpu} → size) then
4:             if ((freespace → {starting_time} ≤ space → {starting_time}) and (freespace → {starting_time} + freespace → {duration} ≤ space → {starting_time} + space → {duration})) then
5:                 profile_tree → remove(freespace)
6:                 Return
7:             end if
8:         end if
9:     end for
10: end function
```