Algorithm 1 Checker

Global Variable: edgesData: matrix[s.objectiveValue][E]
Input: d : Data, s : Solution
Output: boolean representing the correctness of s for dataset d

```
 0: function CHECK(d,s)
 1: for int t in 0..s.objectiveValue do
 2:    for edge e in 0..E do
 3:       edgesData[t][e] ⇐ d.capacity(e)
 4:    end for
 5: end for
 6: for all node aNode to be evacuated do
 7:    tInit ⇐ s.startEvacuationDate(aNode)
 8:    remainPeople ⇐ d.PeopleToBeEvacuated(aNode)
 9:    while remainPeople > 0 do
10:       t ⇐ tInit
11:       for all edge e ∈ d.evacuationPath(aNode) do
12:          if t > s.objectiveValue then
13:             return false
14:          end if
15:          edgesData[t][e] ⇐ edgesData[t][e] − s.evacuationRate(aNode)
16:          remainPeople ⇐ remainPeople − s.evacuationRate(aNode)
17:          if edgesData[t][e] < 0 then
18:             return false
19:          end if
20:          t ⇐ t + d.length(e)
21:       end for
22:       tInit ⇐ tInit + 1
23:    end while
24: end for
25: for edge e in 0..E do
26:    for t in dueDate(e)..s.objectiveValue do
27:       if edgesData[t][e] ≠ d.capacity(e) then
28:          return false
29:       end if
30:    end for
31: end for
32: return false
```

Algorithm 2 Lower Bound calculation (function computeInfValue)

Input: d : Data
Output: Integer representing an upper bound value for the evacuation time of the instance d

```
 0: function computeEvacTimes(d)
 1: evacTimesList : List < Integer >
 2: for all node aNode to be evacuated do
 3:    Integer minCapa ⇐ +∞
 4:    Integer travelTime ⇐ 0
 5:    for all edge e ∈ d.evacuationPath(aNode) do
 6:       if d.capacity(e) < minCapa then
 7:          minCapa ⇐ d.capacity(e)
 8:       end if
 9:       travelTime ⇐ travelTime + d.length(e)
10:    end for
11:    Integer nbPackets ⇐ ⌈d.PeopleToBeEvacuated(aNode) ÷ minCapa⌉
12:    add(evacTimesList,(nbPackets + travelTime))
13: end for
14: return evacTimesList
14:    function computeInfValue(d)
15: return min(computeEvacTimes(d))
```

Algorithm 3 Upper Bound calculation (function computeSupValue)

Input: d : Data
Output: Integer representing an upper bound value for the evacuation time of the instance d

```
 0: function computeEvacTimes(d)
 1: evacTimesList : List < Integer >
 2: for all node aNode to be evacuated do
 3:    Integer minCapa ⇐ +∞
 4:    Integer travelTime ⇐ 0
 5:    for all edge e ∈ d.evacuationPath(aNode) do
 6:       if d.capacity(e) < minCapa then
 7:          minCapa ⇐ d.capacity(e)
 8:       end if
 9:       travelTime ⇐ travelTime + d.length(e)
10:    end for
11:    Integer nbPackets ⇐ ⌈d.PeopleToBeEvacuated(aNode) ÷ minCapa⌉
12:    add(evacTimesList,(nbPackets + travelTime))
13: end for
14: return evacTimesList
14:    function computeSupValue(d)
15: return sum(computeEvacTimes(d))
```