for each: foreach
do
end foreach

Algorithm 1: Overlapping-Event-Scheduler

Input : EventList : List< *Event* >, NewEvent : Event
Output: ValidEventsFlag : Boolean, OverlappingEvent:Event

```
1  foreach e1 ∈ EventList do
2  |  dummyst1 = e1.startTime
3  |  dummyst2 = NewEvent.startTime
4  |  if e1.chosenMobility! = NULL then
5  |  |  dummyst1 = dummyst1 - e1.chosenMobility.TravelDuration
6  |  end
7  |  if NewEvent.chosenMobility! = NULL then
8  |  |  dummyst2 = dummyst - NewEvent.chosenMobility.TravelDuration
9  |  end
10 |  // Checks overlapping
11 |  if e1.startTime < NewEvent.endTime and e1.endTime >
   |   NewEvent.startTime then
12 |  |  //Overlap occurs
13 |  |  ValidEventsFlag = False
14 |  |  return ValidEventsFlag,e1
15 |  else
16 |  |  ValidEventsFlag = True
17 |  |  return ValidEventsFlag,NULL
18 |  end
19 end
20
```

Algorithm 2: Mobility-Option-Recommender-For-Events

Input : PreferenceList : List< $Mobility$ >,NewEvent:$Event$,EventList: List< $Event$ >

Output: RecommendedMobilityList : List< $Mobility$ >

```
1  EventList.push(NewEvent) RecommendedMobilityList = ∅
2  foreach m ∈ PreferenceList do
3  |   dummye = new Event
4  |   dummye.startTime = NewEvent.startTime - m.TravelDuration
5  |   dummye.endTime = NewEvent.startTime
6  |   dummye.chosenMobility = NULL
7  |   ValidEventsFlag = Overlapping-Event-Scheduler(EventList,dummye)
8  |   if ValidEventsFlag then
9  |   |   RecommendedMobilityList.push(m)
10 |   end
11 end
12 return RecommendedMobilityList
13
```

Algorithm 3: Empty-Slot-Generator

Input : EventList : List $< Event >$
Output: EmptySlotList : List $< Break >$

```
1  // Initialization
2  EmptySlotList =
     Break(startTime = 0, endTime = 24.00, chosenMobility = None)
3  dummye = new < Break >
4  dummyst = new < DateTime >
5  foreach e1 ∈ EventList do
6      foreach e2 ∈ EmptySlotList do
7          if e2.startTime < e1.startTime and e1.endTime < e2.endTime
             then
8              if e1! = Break then
9                  dummyst = e1.startTime -
                     e1.chosenMobility.travelDuration
10             end
11             // Partition of the empty slots as two new events
12             EmptyList.delete(e2)
13             dummye.startTime = e2.startTime
14             dummye.endTime = dummyst
15             dummye.Duration = dummye.endTime - dummye.startTime
16             EmptyList.push(dummye)
17             dummye.startTime = e1.endTime
18             dummye.endTime = e2.endTime
19             dummye.Duration = dummye.endTime - dummye.startTime
20             EmptyList.push(dummye)
21         end
22     end
23 end
```

Algorithm 4: Locator-For-Break

Input : BreakList:List< *Break* >,EventList: List< *Event* >,EmptyList : List< *Break* >

Output: ValidScheduleWithBreaks:Boolean , newBreakList :List< *Break* >,EmptyList : List< *Break* >

```
1  newBreakList = List< Break > foreach Break ∈ BreakList do
2      foreach empty in EmptyList do
3          if empty.Duration >
             NewBreak.Duration+NewBreak.chosenMobility.TravelDuration
             then
4              empty.Duration = empty.Duration - (New-
                Break.Duration+NewBreak.chosenMobility.TravelDuration)
               newBreakList.push(Break)
5          end
6      end
7  end
8  // If all the breaks are schedulable, schedule is valid with breaks
9  ValidScheduleWithBreaks = IsSame(newBreakList, BreakList)
10 return ValidScheduleWithBreaks,newBreakList,EmptyList
11
```

Algorithm 5: Mobility-Option-Recommender-For-Breaks

Input : PreferenceList,EventList : List < *Event* >, NewBreak< *Break* >,EmptyList : List< *Break* >

Output: RecommendedMobilityList: List< *Mobility* >

```
1  RecommendedMobilityList = ∅
2  foreach empty in EmptyList do
3      foreach m in PreferenceList do
4          if emptyDuration > NewBreak.Duration+m.TravelDuration then
5              RecommendedMobilityList.push(m)
6          end
7      end
8  end
9  return RecommendedMobilityList
```