

```

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 = dummyst2 - 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* >, EmptyList: List< *Break* >
Output: RecommendedMobilityList : List< *Mobility* >

```
1 EventList.push(NewEvent) RecommendedMobilityList =  $\emptyset$ 
2 foreach  $m \in \text{PreferenceList}$  and  $m.\text{Status} = \text{ACTIVE}$  do
3    $\text{dummye} = \text{new Event}$ 
4    $\text{dummye.startTime} = \text{NewEvent.startTime} - m.\text{TravelDuration}$ 
5    $\text{dummye.endTime} = \text{NewEvent.startTime}$ 
6    $\text{dummye.chosenMobility} = \text{NULL}$ 
7   foreach  $\text{empty}$  in  $\text{EmptyList}$  do
8     // If mobility  $m$  is in a empty interval
9     if  $\text{dummye.startTime} > \text{empty.startTime}$  and  $\text{dummye.endTime}$ 
      <  $\text{empty.endTime}$  then
10      //Add  $m$  to recommendation list
11       $\text{RecommendedMobilityList.push}(m)$ 
12    end
13  end
14 end
15 return  $\text{RecommendedMobilityList}$ 
16
```

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  $e1 \in EventList$  do
6   foreach  $e2 \in EmptySlotList$  do
7     if  $e2.startTime < e1.startTime$  and  $e1.endTime < e2.endTime$ 
8       then
9         // Partition of the empty slots as two new events
10         $EmptyList.delete(e2)$ 
11         $dummye.startTime = e2.startTime$ 
12         $dummye.endTime = dummyst$ 
13         $dummye.Duration = dummye.endTime - dummye.startTime$ 
14         $EmptyList.push(dummye)$ 
15         $dummye.startTime = e1.endTime$ 
16         $dummye.endTime = e2.endTime$ 
17         $dummye.Duration = dummye.endTime - dummye.startTime$ 
18         $EmptyList.push(dummye)$ 
19      end
20    end
  end
```

Algorithm 4: Locator-For-Breaks

Input : BreakList:List< *Break* >, 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 Break.startTime < empty.endTime and Break.endTime >
       empty.startTime then
4       dummyst = max(Break.startTime,empty.startTime)
5       dummyend = min(Break.endTime,empty.endTime)
6       AvailDuration = dummyend - dummyst
7       if AvailDuration > New-
         Break.Duration+NewBreak.chosenMobility.TravelDuration
       then
8         empty.Duration = empty.Duration - (New-
           Break.Duration+NewBreak.chosenMobility.TravelDuration)
           newBreakList.push(Break)
9       end
10    end
11  end
12 end
13 // If all the breaks are schedulable, schedule is valid with breaks
14 ValidScheduleWithBreaks = IsSame(newBreakList, BreakList)
15 return ValidScheduleWithBreaks,newBreakList,EmptyList
16
```

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

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

```
1 RecommendedMobilityList =  $\emptyset$ 
2 foreach empty in EmptyList do
3   // If empty slot and new break overlaps
4   if Break.startTime < empty.endTime and Break.endTime >
     empty.startTime then
5     // Calculate the available duration
6     dummyStart = max(Break.startTime, empty.startTime)
7     dummyEnd = min(Break.endTime, empty.endTime)
8     AvailDuration = dummyEnd - dummyStart
9     foreach m in PreferenceList and m.Status = ACTIVE do
10      if AvailDuration > NewBreak.Duration + m.TravelDuration
11        then
12          RecommendedMobilityList.push(m)
13      end
14    end
15 end
16 return RecommendedMobilityList
```

Algorithm 6: AddEvent

Input : PL:List < *Mobility* >, EventList: List< *Event* >, BreakList:
List< *Break* >, sT:DateTime, eT:DateTime, EL:Location

Output: EventList: < *Event* >

```
1 // This piece of algorithm chart explains
2 // overall flow of the algorithms and user
3 // interaction
4 newEvent = new
   Event(startTime = sT, endTime = eT, eventLocation = EL)
5 ValidEventsFlag, OverlapEvents =
   Overlapping-Event-Scheduler(EventList, newEvent)
6 if ValidEventsFlag == True then
7   EmptyList = Empty-List-Generator(EventList)
8   ValidScheduleWithBreaks , newBreakList , EmptyList =
     Locator-For-Breaks(BreakList, EmptyList)
9   if ValidScheduleWithBreaks = True then
10    RecommendedMobL = Mobility-Option-Recommender-For-
      Events(PL, newEvent, EventList, EmptyList)
11    if isEmpty(RecommendedMobL) then
12      | return UnreachableError
13    else
14      | EventList.push(newEvent)
15      | return EventList
16    end
17  else
18    | return OverlapError
19  end
20 else
21   return OverlapError
22 end
```

Algorithm 7: AddBreak

Input : PL:List < *Mobility* >, EventList: List< *Event* >, BreakList:
List< *Break* >,
sT:DateTime,eT:DateTime,duration:DateTime,EL:Location

Output: EventList: < *Event* >

```
1 // This piece of algorithm chart explains
2 // overall flow of the algorithms and user
3 // interaction
4 newBreak = new Break(startTime = sT, endTime = eT, Duration =
    duration, eventLocation = EL)
5 EmptyList = Empty-List-Generator(EventList)
6 ValidScheduleWithBreaks , newBreakList ,EmptyList =
    Locator-For-Breaks(BreakList,EmptyList)
7 if ValidScheduleWithBreaks = True then
8     RecommendedMobL = Mobility-Option-Recommender-For-
        Breaks(PL,newEvent,EventList,EmptyList)
9     if isEmpty(RecommendedMobL) then
10         return UnreachableError
11     else
12         BreakList.push(Break)
13         return BreakList
14     end
15 else
16     return OverlapError
17 end
```
