Algorithm 1 Initial Solution

```
Require: $data$, $regulation$, $threshold$, $delta$, $taxiingTime$
Ensure: $gateList$
$intervalData \leftarrow$ GetInterval($data, 0$)
    $\triangleright$ Data within 12am to 1am is based on the actual time, while the rest uses target time
$intervalData \leftarrow$ ExtendDepartureSide($intervalData$, $delta$)
$objectiveMatrix \leftarrow$ MinimizeTaxiingTime($data$, $threshold$, $taxiingTime$)
$gateMatrix \leftarrow$ ChooseAvailableGate($data$)    $\triangleright$ airlines and wingspan
$obstructionDict \leftarrow$ FindConflictedFlight($intervalData$)
$gateList \leftarrow$ Optimization($intervalData$, $objectiveMatrix$, $gateMatrix$, $obstructionDict$)
return $gateList$
```

Algorithm 2 GetInterval

```
Require: $data$
Ensure: $intervalData$
$m \leftarrow$ length of $intervalData$
for $i \leftarrow 0$ to $m - 1$ do
    if $data[i] < 60 \times 60$ then
        $tot \leftarrow$ ATOT
        $ldt \leftarrow$ ALDT
    else
        $tot \leftarrow$ TTOT
        $ldt \leftarrow$ TLDT
    end if
    if $ldt - tot <= 60 * 60$ and $ldt - tot >= 40 * 60$ then
        $interval \leftarrow$ ShortTime$(ldt, tot)$
    else
        $interval \leftarrow$ LongTime$(ldt, tot)$
    end if
end for
$intervalData \leftarrow interval$
return $intervalData$
```

Algorithm 3 MinimizeTaxiingTime

```
Require: $data$, $threshold$, $taxiingTime$
Ensure: $objectiveMatrix$
while $hour < 24$ do
    $amount \leftarrow$ the amount of the departing or landing aircraft
    if $amount > threshold$ then
        $objectiveMatrix \leftarrow$ 16R for departing aircraft and 16L for landing aircraft
    else
        $objectiveMatrix \leftarrow$ 16R for departing aircraft and landing aircraft
    end if
end while
return $objectiveMatrix$
```

Algorithm 4 Sliding Windows

```
Require: data, regulation, threshold, delta, taxiingTime
Ensure: gateList
quarter ← 0                                         ▷ Recalculate every 15 minutes.
while quarter < $24 \times 4$ do
    intervalData ← GetInterval(data, quarter)
    ▷ Data with actual time comes from within the next hour from the current quarter, while the rest uses target time
    intervalData ← ExtendDepartureSide(intervalData, delta)
    objectiveMatrix ← SelectNearestGate(gateList, intervalSet)
    gateMatrix ← ChooseAvailableGate(data)          ▷ airlines and wingspan
    fixList ← FixedFlight(intervalData, quarter)
    obstructionDict ← FindConflictedFlight(intervalData)
    gateList ← Optimization(intervalData, objectiveMatrix, gateMatrix, obstructionDict)
end while
return gateList
```

Algorithm 5 SelectNearestGate

```
Require: gateList, intervalSet
Ensure: objectiveMatrix                             ▷ Recalculate Target
gate ← the gate allocated to the intervalSet in the previous iteration
no1 ← gates belonging to Terminal 1
no2 ← gates belonging to Terminal 2
no3 ← gates belonging to remote parking stands set 1
no4 ← gates belonging to remote parking stands set 2
m ← length of intervalSet
for i ← 0 to $m - 1$ do
    g ← the allocated gate for i
    objectiveMatrix ← cost(g, i, no_1, target_matrix)
    objectiveMatrix ← cost(g, i, no_2, target_matrix)
    objectiveMatrix ← cost(g, i, no_3, target_matrix)
    objectiveMatrix ← cost(g, i, no_4, target_matrix)
end for
return objectiveMatrix
```

Algorithm 6 FixedFlight

```
Require: $intervalData$, $quarter$, $gateList$
Ensure: $fixList$
    ▷ $intervalData$ within the next 30 minutes from the current $quarter$ use the gate allocated in the previous iteration
$m \leftarrow$ length of $intervalData$
for $i \leftarrow 0$ to $m - 1$ do
    if $intervalData < quarter * 15 * 60 + 60 * 30$ then
        $fixList \leftarrow gateList$
    end if
end for
return $fixList$
```

Algorithm 7 Local Search

```
Require: $oldGateList, intervalSet$
Ensure: $newGateList$
$counter \leftarrow 0$
$\left\{\begin{array}{r} newGateList \\ UnallocatedList \end{array}\right. \leftarrow$ GenerateInitialSolution($intervalSet$)
while counter ¡ 1000 do
    $fitness \leftarrow$ SelectBestGate($oldGateList$, $newGateList$)
    $gateList \leftarrow$ Swap($newGateList, UnallocatedList$)
    $gateList \leftarrow$ Change($newGateList, UnallocatedList$)
    $newFitness \leftarrow$ SelectNearestGate($oldGateList$, $gateList$)
    if $newFitness < fitness$ then
        $newGateList \leftarrow gateList$
    end if
end while
if $UnallocatedList \neq \emptyset$ then
    $newGateList \leftarrow$ AllocateToRemoteGate($newGateList, UnallocatedList$)
end if
return $newGateList$
```

Algorithm 8 Swap

```
Require: $gateList, UnallocatedList$
Ensure: $gateList, UnallocatedList$
$\begin{cases} s1 \\ s2 \end{cases} \leftarrow$ ChooseRandowFlight($gateList$)
$gateList \leftarrow$ ExchangeGate($s1, s2$)
for $i$in$UnallocatedList$ do
    if  i is not conflicted with gateList then
        $gateList \leftarrow$ ChooseAvailableGate($i$)
    end if
end for
return $fixList$
```

Algorithm 9 Change

```
Require: $gateList, UnallocatedList$
Ensure: $gateList, UnallocatedList$
$s1 \leftarrow$ ChooseRandowFlight($gateList$)
$gateList \leftarrow gateList \setminus \{s1\}$
$UnallocatedList \leftarrow Unallocation + [s1]$
for $i$in$UnallocatedList$ do
    if  i is not conflicted with gateList then
        $gateList \leftarrow$ ChooseAvailableGate($i$)
    end if
end for
return $fixList$
```