Algorithm 1 Initial Solution

Require: $data$, $regulation$, $threshold$, $delta$, $taxiingTime$
Ensure: $gateList$

1: $intervalData \leftarrow$ GetInterval($data, 0$)
▷ Data within 12pm to 1am is based on the actual time, while the rest uses target time
2: $intervalData \leftarrow$ ExtendDepartureSide($intervalData$, $delta$)
3: $objectiveMatrix \leftarrow$ MinimizeTaxiingTime($data$, $threshold$, $taxiingTime$)
4: $gateMatrix \leftarrow$ ChooseAvailableGate($data$) ▷ airlines and wingspan
5: $obstructionDict \leftarrow$ FindConflictedFlight($intervalData$)
6: $gateList \leftarrow$ Optimization($intervalData$, $objectiveMatrix$, $gateMatrix$, $obstructionDict$)
7: **return** $gateList$

Algorithm 2 GetInterval

Require: *data*
Ensure: *intervalData*
1: $m \leftarrow$ length of *intervalData*
2: **for** $i \leftarrow 0$ to $m - 1$ **do**
3: **if** $data[i] < 60 \times 60$ **then**
4: $tot \leftarrow$ ATOT
5: $ldt \leftarrow$ ALDT
6: **else**
7: $tot \leftarrow$ TTOT
8: $ldt \leftarrow$ TLDT
9: **end if**
10: **if** $ldt - tot <= 60 * 60$ and $ldt - tot >= 40 * 60$ **then**
11: $interval \leftarrow$ ShortTime(ldt, tot)
12: **else**
13: $interval \leftarrow$ LongTime(ldt, tot)
14: **end if**
15: **end for**
16: $intervalData \leftarrow interval$
17: **return** $intervalData$

Algorithm 3 MinimizeTaxiingTime

Require: *data*, *threshold*, *taxiingTime*
Ensure: *objectiveMatrix*
1: **while** $hour < 24$ **do**
2: *amount* $\leftarrow$ the amount of the departing or landing aircraft
3: **if** $amount > threshold$ **then**
4: $objectiveMatrix \leftarrow$ 16R for departing aircraft and 16L for landing aircraft
5: **else**
6: $objectiveMatrix \leftarrow$ 16R for departing aircraft and landing aircraft
7: **end if**
8: **end while**
9: **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**
    $intervalData \leftarrow$ GetInterval($data, quarter$)
    ▷ Data with actual time comes from within the next hour from the current *quarter*, while the rest uses target time
    $intervalData \leftarrow$ ExtendDepartureSide($intervalData$, *delta*)
    $objectiveMatrix \leftarrow$ SelectNearestGate($gateList$, $intervalSet$)
    $gateMatrix \leftarrow$ ChooseAvailableGate(*data*) ▷ airlines and wingspan
    $fixList \leftarrow$ FixedFlight($intervalData, quarter$)
    $obstructionDict \leftarrow$ FindConflictedFlight($intervalData$)
    $gateList \leftarrow$ 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 $i \leftarrow 0$ to $m - 1$ **do**
    $g \leftarrow$ the allocated gate for i
    $objectiveMatrix \leftarrow$ **cost**(g, i, no_1, $target\_matrix$)
    $objectiveMatrix \leftarrow$ **cost**(g, i, no_2, $target\_matrix$)
    $objectiveMatrix \leftarrow$ **cost**(g, i, no_3, $target\_matrix$)
    $objectiveMatrix \leftarrow$ **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: $newGateList$

$counter \leftarrow 0$
$\begin{cases} newGateList \\ UnallocatedList \end{cases} \leftarrow$ 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: $gateList, UnallocatedList$
1: $\begin{cases} s1 \\ s2 \end{cases} \leftarrow$ ChooseRandowFlight($gateList$)
2: $gateList \leftarrow$ ExchangeGate($s1, s2$)
3: **for** $i$in$UnallocatedList$ **do**
4: **if** i is not conflicted with gateList **then**
5: $gateList \leftarrow$ ChooseAvailableGate(i)
6: **end if**
7: **end for**
8: **return** $fixList$

Algorithm 9 Change

Require: $gateList, UnallocatedList$
Ensure: $gateList, UnallocatedList$
1: $s1 \leftarrow$ ChooseRandowFlight($gateList$)
2: $gateList \leftarrow gateList \backslash \{s1\}$
3: $UnallocatedList \leftarrow Unallocation + [s1]$
4: **for** $i$in$UnallocatedList$ **do**
5: **if** i is not conflicted with gateList **then**
6: $gateList \leftarrow$ ChooseAvailableGate(i)
7: **end if**
8: **end for**
9: **return** $fixList$