

PAPER

A Heuristic Algorithm for Solving the Aircraft Landing Scheduling Problem with a Landing Sequence Division

Wen SHI^{†a)}, *Member*, Shan JIANG^{††}, Xuan LIANG[†], and Na ZHOU[†], *Nonmembers*

SUMMARY Aircraft landing scheduling (ALS) is one of the most important challenges in air traffic management. The target of ALS is to decide a landing scheduling sequence and calculate a landing time for each aircraft in terminal areas. These landing times are within time windows, and safety separation distances between aircraft must be kept. ALS is a complex problem, especially with a large number of aircraft. In this study, we propose a novel heuristic called CGIC to solve ALS problems. The CGIC consists of four components: a chunking rule based on costs, a landing subsequence generation rule, a chunk improvement heuristic, and a connection rule. In this algorithm, we reduce the complexity of the ALS problem by breaking it down into two or more subproblems with less aircraft. First, a feasible landing sequence is generated and divided into several subsequences as chunks by a chunking rule based on aircraft cost. Second, each chunk is regenerated by a constructive heuristic, and a perturbative heuristic is applied to improve the chunks. Finally, all chunks constitute a feasible landing sequence through a connection rule, and the landing time of each aircraft is calculated on the basis of this sequence. Simulations demonstrate that (a) the chunking rule based on cost outperforms other chunking rules based on time or weight for ALS in static instances, which have a large number of aircraft; (b) the proposed CGIC can solve the ALS problem up to 500 aircraft optimally; (c) in dynamic instances, CGIC can obtain high-quality solutions, and the computation time of CGIC is low enough to enable real-time execution.

key words: air traffic management, aircraft landing scheduling, heuristic

1. Introduction

Aircraft fly into a terminal area from different routes to land on the runways of the airport. A scheduled landing time is allocated to each aircraft by a controller because aircraft may be unable to land at the planned arrival time. The scheduled landing time is in the time window, which is between the earliest and the latest landing time of the aircraft. Scheduled landing time separations between aircraft must be no less than safety standards because of wake turbulence and other factors. Therefore, the scheduled landing time of an aircraft may not be exactly equal to the target landing time. In aircraft landing scheduling (ALS) problem, the objective is to minimize the total deviation of the scheduled landing time from the target landing time for each aircraft, whereas the scheduled landing time is in time window, and the separation constraints are satisfied. Obviously, the more the aircraft interfere with one another, the more complex the ALS problem will be.

Numerous attempts have been made to solve an ALS problem. These approaches can be classified into two categories: exact algorithm and heuristic algorithm. Beasley et al. [1] presented a mixed-integer zero-one formulation of ALS problems and performed a linear programming-based tree search to solve them. Faye et al. [2] proposed a method based on the approximation of a separation time matrix and on time discretization to solve ALS problems.

Many heuristics are proposed to solve ALS problems. Sabar et al. [3] proposed an iterated local search algorithm, which is a single-solution-based search methodology, to minimize the total penalty of landing deviations from the target landing time. Beasley et al. [4] developed a population heuristic to schedule aircraft landings at London Heathrow. In 2006, two heuristic techniques, namely, scatter search and bionomic algorithm, were presented for ALS problems [5]. The two population heuristic algorithms were applied to test instances involving up to 500 aircraft.

Some studies have introduced a heuristic by using a local search to obtain a scheduled landing sequence, and an exact algorithm to calculate the landing time of each aircraft based on this landing sequence. Ernst et al. [6] developed a heuristic approach and a simplex algorithm to calculate the upper and lower bounds of a branch-and-bound scheme to evaluate solutions. Hu et al. [7] designed a genetic algorithm based on a binary representation of arriving sequences. Yu et al. [8] proposed a cellular automata optimization(CAO), which consists of three components: a cellular automata model to generate a landing sequence, a local search to improve the landing sequence, and a deterministic operator to calculate the optimal landing times based on the landing sequence. Salehipour et al. [9] designed a hybrid meta-heuristic applying a simulated annealing framework. The framework includes a problem-dependent construction heuristic and a variable neighborhood descent meta-heuristic with three neighborhood structures as an improvement algorithm. In 2014, we proposed a dynamic hyper-heuristic algorithm for the ALS problem [10]. A scatter search is adopted as a high level heuristic, which builds a chain of priority rules as low level heuristics to generate a landing sequence.

The longer the landing sequence generated by a heuristic algorithm is, the larger the computation time of the exact algorithm for calculating the landing time of each aircraft will be. Therefore, the set of aircraft can be divided into several subsets, called chunks. The landing time of each aircraft in every chunk is calculated, and the computation

Manuscript received November 20, 2018.

Manuscript revised March 13, 2019.

[†]The authors are with the Tianjin University of Commerce, China.

^{††}The author is with the Tianjin Medical University, China.

a) E-mail: shiwen@tjcu.edu.cn

DOI: 10.1587/transfun.E102.A.966

time is short enough even in dynamic ALS instances. Furini et al. [11], [12] proposed three chunking rules to generate chunks with a length constraint and found that different rules iteratively applied to various sequences can obtain better results than those determined without chunking or those with a single chunking rule.

In this study, we proposed a novel algorithm *CGIC* to solve ALS problems. Our approach consists of four components: a chunking rule based on costs (CR_{cost}), a landing subsequence generation rule (*GR*), a chunk improvement heuristic (*IH*), and a connection rule (*CNR*). First, a landing sequence generated by a non-optimization method First-Come-First-Served (*FCFS*) is divided into several chunks through the chunking rule. Chunking rules proposed in [12] are based on time or weight and with constraints of length of chunks. CR_{cost} is based on cost of each aircraft and without any constraints of length. Therefore, CR_{cost} aims to place most aircraft that interfere with one another in the same chunk. Second, all aircraft in one chunk are rescheduled by *GR*. *GR* is based on cost priority. The aircraft with a larger cost is inserted into a new landing subsequence in the chunk with a higher priority. Third, the generated landing subsequence is improved using *IH*. In *IH*, the aircraft with the highest cost in the subsequence is selected, and several aircraft that land continuously before or after this aircraft with the highest cost are resequenced randomly to decrease the total cost of the subsequence. Finally, all of the subsequences in every chunk are combined into a complete final landing sequence through *CNR* and the landing time of each aircraft is calculated by using an exact algorithm based on the final landing sequence.

The remaining parts of this paper are organized as follows: Section 2 describes the problem statement. Section 3 provides the details of the proposed algorithm *CGIC*. Section 4 reports the simulation results to evaluate the efficacy of the *CGIC*. Section 5 concludes this paper and presents recommendations for some future work.

2. Aircraft Scheduling Landing: Problem Definition

The ALS problem is divided into two types: static and dynamic. We formulate the static and dynamic ASL problems as mixed integer linear problems. Thus, parameters and variables are defined as follows:

Parameters:

- F_{all} : the set of all aircraft.
- F_f : the set of aircraft that are scheduled to land and could not be rescheduled.
- F_s : the set of aircraft that are scheduled to land and could be rescheduled during an update.
- F_a : the set of aircraft scheduled to land and $F_a = F_f \cup F_s$.
- T_i^t : the target landing time of aircraft i .
- T_i^a : the appearance time of aircraft i .
- T_i^e : the earliest landing time of aircraft i .
- T_i^l : the latest landing time of aircraft i .

- S_{ij} : the separation time between aircraft i and j when aircraft i lands before j .
- c_i^- : penalty cost per unit of time for landing before target time of aircraft i .
- c_i^+ : penalty cost per unit of time for landing after target time of aircraft i .
- T_f : the freeze time in which a scheduled landing time allocated for any aircraft within T_f of the current time could not be rescheduled.

Variables:

- x_i : the scheduled landing time of aircraft i .
- δ_{ij} : a Boolean variable; if aircraft i lands before j , it takes 1.
- t_{cur} : the current time.

2.1 Static Model

In a static model, all aircraft appear and can be scheduled. Therefore, $F_s = F_a = F_{all}$. No aircraft is frozen to allocate a scheduled landing time, that is, $F_f = \emptyset$. For each aircraft in F_s , the scheduled landing time x_i must be within the time window:

$$T_i^e \leq x_i \leq T_i^l \quad (i \in F_s). \quad (1)$$

The landing time of any two aircraft in F_s must meet the safety standards:

$$(x_j - x_i) \geq S_{ij} - M * \delta_{ji} \quad (i \in F_s, j \in F_s, i \neq j). \quad (2)$$

where M is a sufficiently large number used to ensure that Eq. (2) is redundant in the case when aircraft j lands before i ($\delta_{ji} = 1$). The objective function minimizes the total cost of landing deviation from the target landing time of each aircraft in F_s .

$$\text{Min } z = \sum_{i \in F_s} z_i \quad z_i = \begin{cases} c_i^-(T_i^t - x_i) & \text{if } x_i \leq T_i^t \\ c_i^+(x_i - T_i^t) & \text{if } x_i > T_i^t. \end{cases} \quad (3)$$

2.2 Dynamic Model

In a dynamic ALS problem, all aircraft F_{all} consists of aircraft in F_a and aircraft that have no appearance. Aircraft that have no appearance do not affect scheduling aircraft in F_a . F_a can be divided into two parts: F_f and F_s . In F_f , aircraft are too close to land at the current time and landing time cannot be further changed. Thus, the dynamic ALS model contains not only inequality (1)(2) but also other constraints, such as

$$F_f = \{i | x_i - t_{cur} < T_f\}. \quad (4)$$

In every iteration, the landing time of aircraft in F_s is calculated. When an aircraft is frozen, it no longer appears in F_s . Thus,

$$F_s = F_a - F_f. \tag{5}$$

All aircraft in F_s must land after the aircraft in F_f . The scheduled landing time x_i of aircraft i in F_s is larger than any landing time x_j of aircraft j , which is in F_f . Therefore, the lower bound of the time window is as follows:

$$x_i \geq \text{Max}(T_i^e, x_j) \quad (i \in F_s, j \in F_f). \tag{6}$$

The separation time between an aircraft in F_s and an aircraft in F_f must be satisfied:

$$(x_i - x_k) \geq S_{ki} \quad (i \in F_s, k \in F_f). \tag{7}$$

The objective function in the dynamic model is the same as in the static model.

3. CGIC Algorithm

The CGIC algorithm proposed in this study contains four main components: CR_{cost} , GR , IH , and CNR . First, $FCFS$ generates an initial landing sequence from F_s . Second, the initial landing sequence is divided into several chunks by CR_{cost} . Third, aircraft are rescheduled in each chunk through GR . Furthermore, the rescheduled chunks are improved using the perturbative heuristic IH . Subsequently, CNR connects all chunks and constructs the final landing sequence. Finally, an exact method calculates the landing time of all aircraft based on the final landing sequence. The flowchart of the CGIC algorithm is shown in Fig. 1. Some variables and parameters in the algorithm are defined as follows:

- seq_{ini} : the initial landing sequence.
- seq_{fin} : the final landing sequence.
- seq_t : the landing subsequence of chunk t .
- $seq(p)$: the p th aircraft in seq .
- $x_{seq(p)}$: the landing time of the p th aircraft in seq .
- $cost(seq(p))$: the cost of the p th aircraft in seq .
- $cost(seq)$: the sum of the cost of each aircraft in seq .
- $|seq|$: the length of the landing sequence seq .
- $F(p)$: the p th aircraft in F .
- $|F|$: the number of aircraft in aircraft set F .
- C_{ini} : the initial chunk set.
- C_{imp} : the improved chunk set.
- $|C|$: the number of chunks in chunk set C .

3.1 The Chunking Rule

The proposed CR_{cost} procedure aims to collect some adjacent aircraft in seq_{ini} into a chunk and does not affect the landing time of the aircraft in other chunks as much as possible. For example, some adjacent aircraft are selected. The costs of the first and last aircraft are 0 and the total cost of these aircraft without constraints from the other aircraft in seq_{ini} is equal to the sum of the costs of the same aircraft in the initial landing sequence. Then these aircraft have less impact on the other aircraft because the

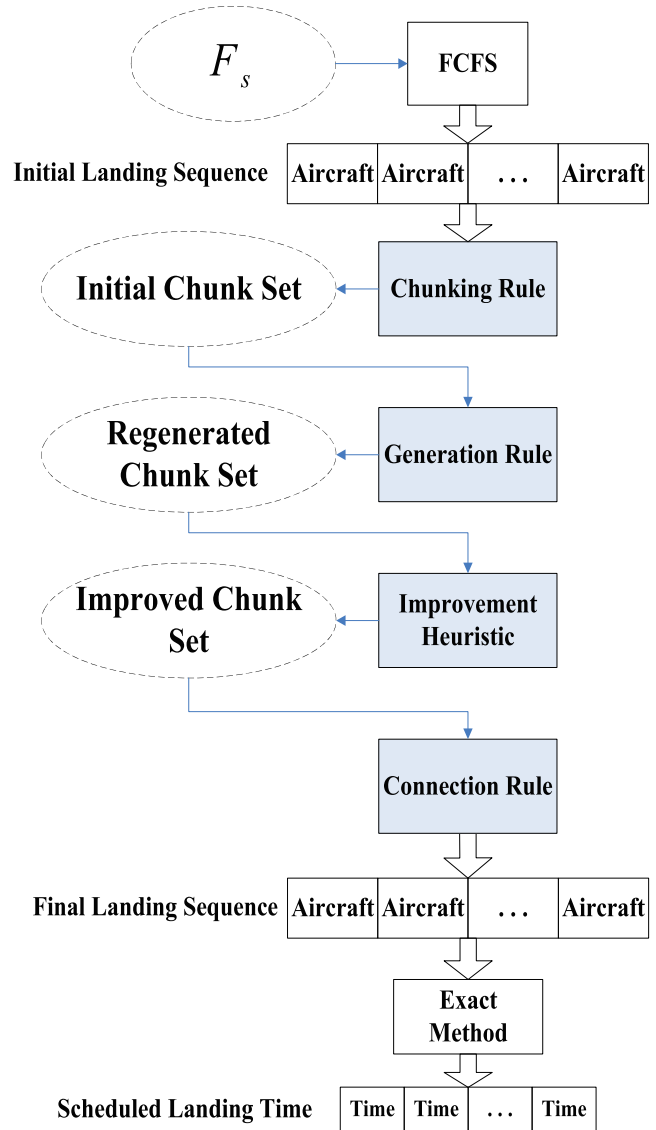


Fig. 1 Flow of CGIC algorithm.

total cost is unchanged after these aircraft are selected out of seq_{ini} . Therefore, in contrast to other chunking rules, such as CR_{fixed} , CR_{time} , and CR_{weight} proposed in [11], [12], CR_{cost} generates chunks based on the cost of aircraft without any length constraints. CR_{cost} measures the cost of each aircraft in seq_{ini} and selects a subsequence seq^{pq} . In this subsequence, the costs of the first aircraft $seq(p)$ and the last aircraft $seq(q)$ are 0. If $cost(seq^{pq})$ is equal to the sum of the costs of the same aircraft in seq_{ini} , the aircraft in this sequence have less impact to the other aircraft in seq_{ini} . Furthermore, the subsequence is taken as a new chunk into the C_{ini} . If the total cost of the subsequence is not equal to the sum of the original cost of the same aircraft, more aircraft are added to this subsequence until two conditions are satisfied: the costs of the first and the last aircraft in the subsequence are 0, and the total cost of this subsequence is equal to the sum of the costs of the same aircraft in seq_{ini} . Two adjacent chunks share the same aircraft. This aircraft

Algorithm 1: The Pseudo Code of CR_{cost}

```

1 Set  $p = 0; q = p + 1; p, q$  are the index of aircraft in  $seq_{ini}$  ;
2 while  $p < q \leq |F_s|$  do
3   if  $(cost(seq_{ini}(p)) = 0 || p = 0) \& cost(seq_{ini}(q)) = 0$ 
4     then
5       Generate a subsequence  $seq^{pq}$ , which contains
        aircraft from the  $p$ th to the  $q$ th in  $seq_{ini}$ ;
6       Calculate a landing time of each aircraft in  $seq^{pq}$ 
        without constraints from the other aircraft in  $seq_{ini}$  ;
7       if  $cost(seq^{pq}) = \sum_{k=p}^q cost(seq_{ini}(k))$  then
8         Construct the subsequence  $seq^{pq}$  as a new chunk
        and place it in  $C_{ini}$  ;
9          $p = q$  ;
10         $q = q + 1$  ;
11        Continue ;
12      end
13    else
14      if  $\exists r, p \leq r \leq q \ \& \ x_{seq^{pq}}(r) < x_{seq_{ini}}(r)$ 
15        then
16          Get the subsequence  $seq^{sp}$  which is before
         $seq^{pq}$  ;
17          Connect the two subsequences as a new
        subsequence and set  $p = s$  ;
18          Continue ;
19        end
20      end
21    end
22  end
23   $q = q + 1$  ;
24 end

```

can be seen as a key node aircraft, and its cost is 0.

3.2 The Generation Rule

After all chunks are constructed from seq_{ini} , each chunk in C_{ini} is regenerated, that is to say, aircraft in one chunk are rescheduled by GR . In GR , the aircraft with the largest cost in the chunk is selected and tried to be inserted into a new sequence. If this aircraft conflicts with other aircraft in the new sequence based on its target landing time, the aircraft with the largest cost tries to replace these conflicted aircraft in the new sequence. If the cost of the new sequence decreases after the replacement is completed, the conflicted aircraft is removed from the new sequence and placed in the unscheduled aircraft set. When all of the conflicted aircraft are replaced, the aircraft with the largest cost can be inserted into the new sequence with its target landing time. Otherwise, the aircraft with the largest cost is inserted into the new sequence at a scheduled landing time, and the landing time of the other aircraft is unchanged.

3.3 The Improvement Heuristic

In IH , every chunk is improved to decrease its cost by disordering a subsequence before or after the aircraft with the largest cost. If the aircraft lands after its target landing time, the subsequence to be disordered contains some aircraft landing before this aircraft and the aircraft with the largest cost. Conversely, the subsequence contains the aircraft with the

Algorithm 2: The Pseudo Code of GR

```

1 for  $t=0$  to  $|C_{ini}|$  do
2   for  $p=0$  to  $|seq_t|$  do
3     Calculate  $x_{seq_t(p)}$  and  $cost(seq_t(p))$  ;
4   end
5   Place all of the aircraft in  $seq_t$  in a unscheduled set  $F_{un}$  ;
6   Construct an empty landing sequence  $seq_{new}$  as a new
   chunk;
7   while  $F_{un} \neq \emptyset$  do
8     Get aircraft  $j$  in  $F_{un}$  while the cost of  $j$  is maximum in
      $seq_t$  ;
9     Find all of the aircraft in  $seq_{new}$  which have conflict
     with  $j$  and place them in a set  $F_c$  ;
10    for  $q=0$  to  $|F_c|$  do
11      Copy the  $seq_{new}$  as a new sequence  $seq'$  ;
12      Replace the aircraft  $F_c(q)$  with aircraft  $j$  in  $seq'$  ;
13      if  $cost(seq') \leq cost(seq_{new})$  then
14        Put  $F_c(q)$  into  $F_{un}$  ;
15        Remove  $F_c(q)$  from  $F_c$  and  $seq_{new}$ ;
16      end
17    end
18    if  $F_c = \emptyset$  then
19      Insert aircraft  $j$  into  $seq_{new}$  with the landing
      time  $T_j^t$ ;
20    end
21    else
22      Calculate a landing time with minimum cost for
      aircraft  $j$  based on the landing time of the other
      aircraft in  $seq_{new}$  ;
23      Insert aircraft  $j$  into  $seq_{new}$  with the calculated
      landing time;
24    end
25  end
26  if  $cost(seq_{new}) \leq cost(seq_t)$  then
27     $seq_t = seq_{new}$ ;
28  end
29 end

```

largest cost and those landing after it when the aircraft with the largest cost lands earlier than its target time. The length of the subsequence is limited. After the subsequence is constructed, three aircraft are selected and swapped randomly. If the cost of the disordered chunk is not smaller than the original one, a new subsequence with a shorter length is constructed until the length of the subsequence is 3.

3.4 The Connection Rule

CNR constructs seq_{fin} by connecting all of the chunks. In two adjacent chunks, if the last aircraft in the front chunk and the first aircraft in the later chunk are the same key node aircraft, one of the key node aircraft is removed, and the two chunks are connected to a new chunk. If the two aircraft are not the same, CNR connects two chunks and tries to generate two temporary sequences. One is constructed by removing the key node aircraft in the front chunk, and the other is constructed by removing the key node aircraft in the back chunk. The temporary sequence with an enhanced cost is chosen as a new sequence.

Algorithm 3: The Pseudo Code of *IH*

```

1 for  $t=0$  to  $|C_{ini}|$  do
2   Set a Boolean variable  $sequenceImproved = true$ ,  $N_e$  is
   the maximum length of the subsequence to be disordered ;
3   while  $sequenceImproved$  do
4     Get the aircraft  $seq_t(p)$  which is with the largest cost ;
5     for  $n = N_e$  to 3 do
6       if  $x_{seq_t(p)} > T_{seq_t(p)}^t$  then
7         Get a subsequence  $seq^{(p-n+1)p}$  which is
         from the  $(p-n+1)$ th to the  $p$ th aircraft in
          $seq_t$ ;
8         end
9         else
10        Get a subsequence  $seq^{p(p+n-1)}$  which is
         from the  $p$ th to the  $(p+n-1)$ th aircraft in
          $seq_t$ ;
11        end
12        Choose three aircraft from the subsequence
         randomly ;
13        Swap positions of the three aircraft randomly and
         get a new sequence  $seq'_t$  ;
14        if  $cost(seq'_t) < cost(seq_t)$  then
15           $sequenceImproved = true$  ;
16           $seq_t = seq'_t$  ;
17          break ;
18        end
19        else
20           $sequenceImproved = false$  ;
21           $n = n - 1$  ;
22        end
23      end
24    end
25  end

```

Algorithm 4: The Pseudo Code of *CNR*

```

1 Construct an empty landing sequence  $seq_{fin}$  ;
2 Add  $seq_0$  into  $seq_{fin}$  ;
3 for  $t = 1$  to  $|C_{imp}|$  do
4   if  $seq_{t-1}(|seq_{t-1}| - 1) = seq_t(0)$  then
5     Remove  $seq_t(0)$  from  $seq_t$  ;
6     Add  $seq_t$  to the end of  $seq_{fin}$  ;
7   end
8   else
9     Construct a new sequence  $seq_{front} = seq_{fin}$  ;
10    Remove the key node aircraft from the  $seq_{front}$  ;
11    Add  $seq_t$  to the end of  $seq_{front}$  ;
12    Construct a new sequence  $seq_{behind} = seq_{fin}$  ;
13    Remove the key node from the  $seq_t$  ;
14    Add  $seq_t$  to the end of  $seq_{behind}$  ;
15    if  $cost(seq_{front}) \leq cost(seq_{behind})$  then
16       $seq_{fin} = seq_{front}$  ;
17    end
18    else
19       $seq_{fin} = seq_{behind}$  ;
20    end
21  end
22 end

```

Table 1 Best results of different approaches in static instances 1-9.

	N	<i>FCFS</i>	<i>CAO</i>	<i>HHSS</i>	<i>ILS</i>	<i>CGIC</i>
1	10	700	0.00	0.00	0.00	0.00
2	15	1500	1.33	1.33	1.33	1.33
3	20	1730	52.60	52.60	52.60	52.60
4	20	2520	0.00	0.00	0.00	0.00
5	20	5420	42.80	42.80	42.80	42.80
6	30	24442	0.00	0.00	0.00	0.00
7	44	1550	0.00	0.00	0.00	0.00
8	50	2480	21.17	21.37	21.37	21.37
9	100	7310	23.24	23.24	23.24	23.24

4. Computational Experiments**4.1 Simulation Environment**

Our computational tests use two types of instances. The first includes static instances. In these instances, all aircraft information is known whether or not it enters the terminal area. Therefore, all aircraft are scheduled only once, and the result is not changeable by time. The second comprises dynamic instances. The aircraft that appear in the terminal area and not frozen are scheduled. The algorithm updates the landing sequence and the landing time for each aircraft in every time unit. All of the data of 13 instances with different numbers of aircraft of OR-Library [13] can be obtained at <http://people.brunel.ac.uk/~mastjjb/jeb/info.html>. The ILOG's CPLEX is used as the exact method to calculate the landing time for each aircraft based on the landing sequence. The computer runs on Intel Core i7-6500U 2.50 GHz, 2.50 GHz, 8 GB RAM, and Microsoft Windows 7.

4.2 Static Instances

First, we evaluate the performance of our proposed algorithm *CGIC*. The best results over the 30 replications of

CGIC are compared with several previous approaches, including *FCFS*, *CAO* [8], *HHSS* [10], and *ILS* [3], in static instances. N is the number of aircraft in different instances. The results of *CAO*, *HHSS*, *ILS*, and *CGIC*, in terms of the percentage gap from the best results of *FCFS* are calculated as follows: $(BR_{FCFS} - BR) / BR_{FCFS} * 100\%$, where BR_{FCFS} is the best result of *FCFS*, and BR is the best result of the other algorithms.

In Table 1, *CGIC* can obtain the same optimal results as other algorithms in instances 1-9. In Fig. 2, the results of *CGIC* are not the best in all of the algorithms in instances 10 and 11. However, *CGIC* obviously outperforms the other algorithms in instances 12 and 13, which have a large number of aircraft.

Figure 3 shows the average computation time over 30 replications of *CGIC* with other algorithms. When the number of aircraft is small, the computation time of *CGIC* is about 1 second except in instance 7. In instances 9-13, the computation time increases as N enlarges. When the number of aircraft is 500 in instance 13, the average computation time of *CGIC* is shorter than that of other algorithms.

Figure 4 shows the average computation time of the four

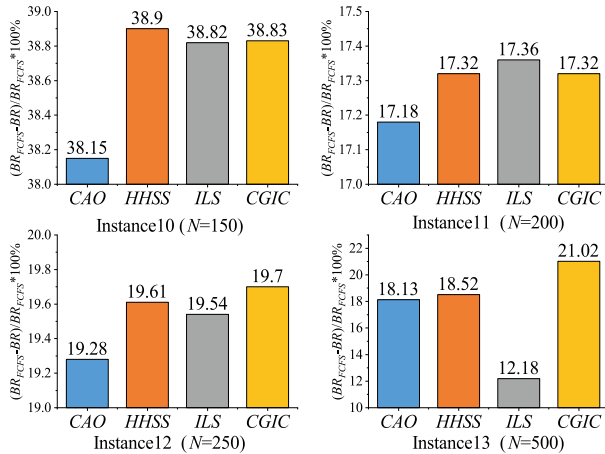


Fig. 2 Best results of different approaches in static instances 10-13.

components (CR_{cost} , GR , IH , and CNR) of $CGIC$. CR_{cost} and CNR are fast algorithms because their computation time is short, especially in instances with a large number of aircraft. As a perturbation heuristic, the computation time of IH occupies most of the total time in most instances except instance 7.

Figure 5 shows the comparison of three different chunking rules, namely, CR_{time} , CR_{weight} , and CR_{cost} . First, each chunking rule generates an initial chunk set. Second, seq_{fin} is constructed by CNR . The landing time is calculated by CPLEX based on seq_{fin} . Finally, the percentage gaps from the best results of $FCFS$ are shown in instances 9-13, which have a large number of aircraft. In most instances, CR_{cost} outperforms the two other chunking rules. In instance 11, the result of CR_{cost} is equal to that of CR_{time} .

N_e is the maximum length of the subsequence to be disordered randomly, and it is considered an important parameter in IH . Figure 6 shows the average results generated by $CGIC$ with different N_e . These results are calculated in terms of the percentage gap from the best results of $FCFS$: $(BR_{FCFS} - AR) / BR_{FCFS} * 100\%$, where AR is the average result generated by $CGIC$ with different N_e . The experimental results in Fig. 6 demonstrate that better results can be generated when N_e is 6. If N_e is too small, IH , as a perturbative heuristic, cannot easily escape local optima. A large N_e indicates a long subsequence, which contains more aircraft. Disorder only three aircraft has less impact on the cost of the subsequence.

4.3 Dynamic Instances

We compare $CGIC$ with different Δt in 13 dynamic instances. Δt is the unit of time. The freeze time T_f is set at 300 second. Table 2 shows the average results in instances 1-7 over the 30 replications of $CGIC$. The results show the percentage gap from best results of $FCFS$. Different Δt , including 360, 480, and 600 seconds, are chosen. Table 2 shows that the results with different Δt of instances 1-7 and the results in static instances 1-7 are the same because the

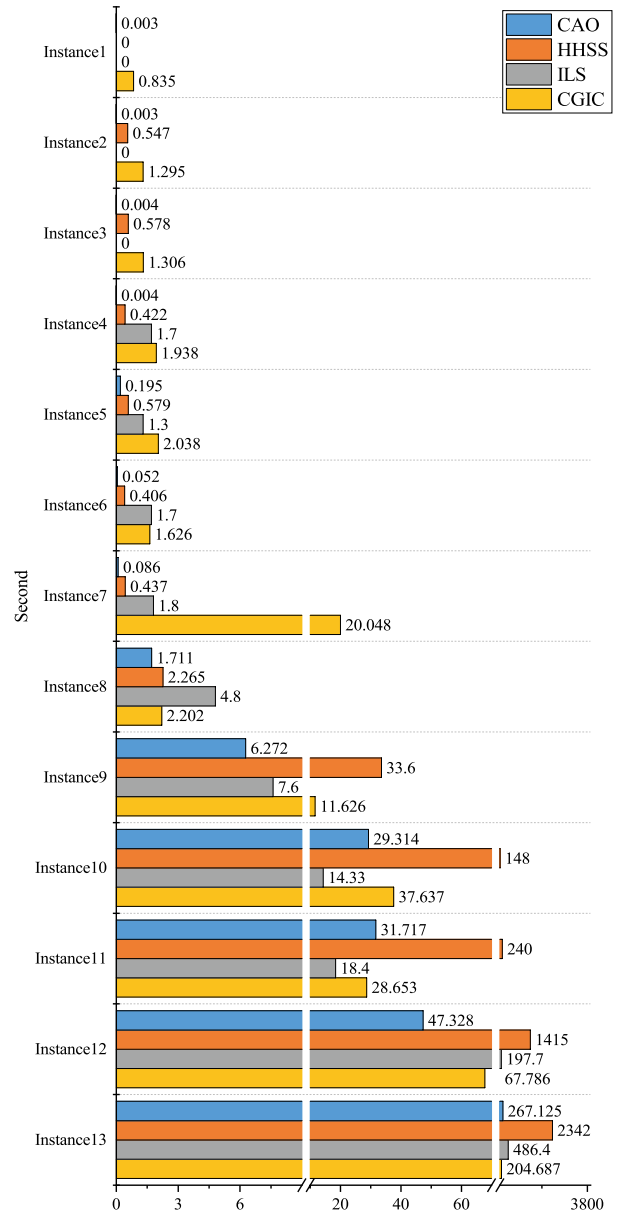


Fig. 3 Computation time of different approaches (second).

number of aircraft in instances 1-7 is small, and $CGIC$ with different Δt can calculate the results as in the static instances. Fig. 7 shows that the result with Δt 600 seconds is the same as that of static instance 8, and the results with Δt 360 and 480 seconds are worse. The number of aircraft in instance 8 is 50, and the total time from the appearance time of the first aircraft to the latest landing time of the last aircraft is 828 seconds, which is close to the Δt with 600 seconds. Therefore, $CGIC$ with a large Δt can generate much better results in instance 8. However, the number of aircraft is larger with longer total time from instances 9 to 13. For example, the total time in instance 9 is 14,122 seconds. Δt with 360, 480, and 600 seconds are much shorter than the total time. In instances 9 to 12, the results of Δt with 480 are optimal. The results of Δt with 360 seconds are not the best, indicating

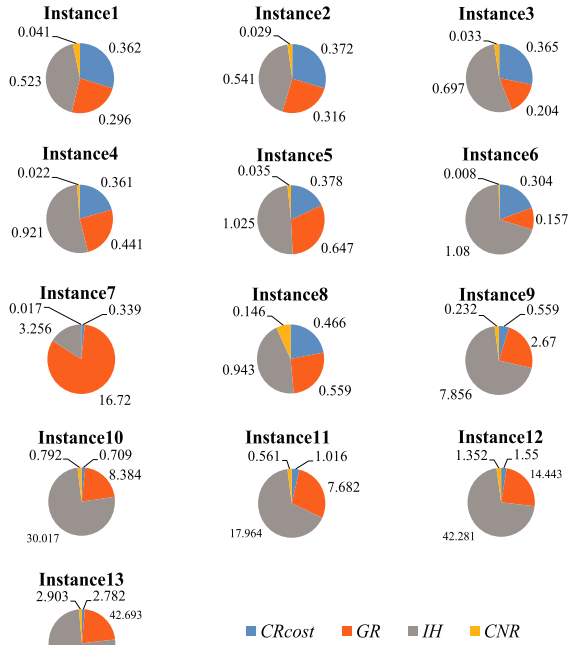


Fig. 4 Computation time of CR_{cost} , GR , IH , and CNR in the total computation time of $CGIC$ (second).

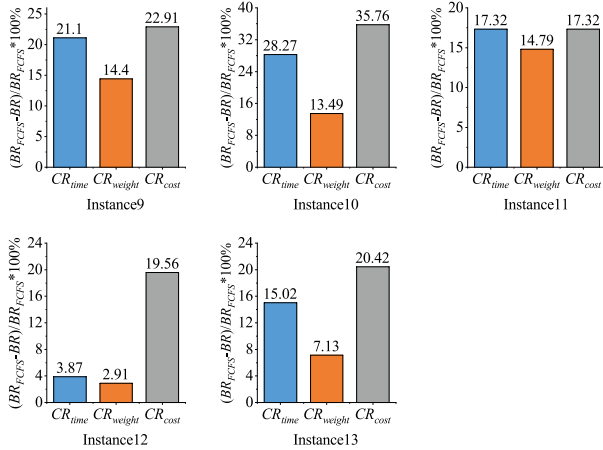


Fig. 5 Best results of different chunking rules.

that the number of aircraft to be scheduled in F_s is small if Δt is small. Moreover, the landing time is calculated on the basis of less constraints from the other aircraft in F_a in early Δt . In later Δt , aircraft have larger deviations from their target landing time. In instance 13, the result of Δt with 600 is optimal when the number of aircraft is 500, and the total time from the appearance time of the first aircraft to the latest landing time of the last aircraft is 56,382 seconds.

Figure 8 shows the average computation time over the 30 replications of $CGIC$ with different Δt . In dynamic instances, with a small number of aircraft, the average calculating time is close to static instances except instance 7. However, the results in dynamic and static instance 7 are the same. When the number of aircraft is large, the average

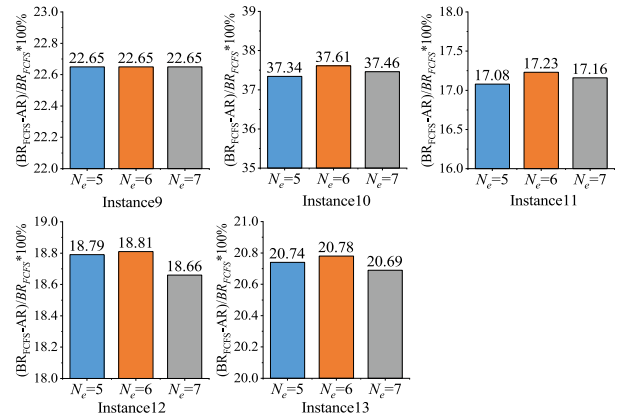


Fig. 6 Effects of N_e .

Table 2 Average results in dynamic instances 1-7 with different Δt .

	N	360	480	600
1	10	0.00	0.00	0.00
2	15	1.33	1.33	1.33
3	20	52.60	52.60	52.60
4	20	0.00	0.00	0.00
5	20	42.80	42.80	42.80
6	30	0.00	0.00	0.00
7	44	0.00	0.00	0.00

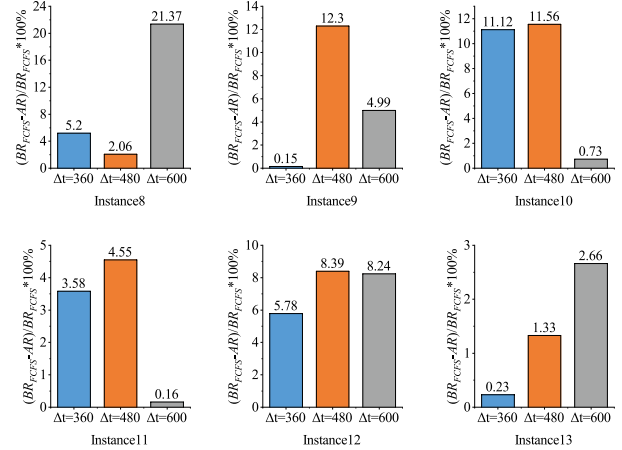


Fig. 7 Average results in dynamic instances 8-13 with different Δt .

computation time with a large Δt is smaller because $CGIC$ needs more iterations when Δt is small.

5. Conclusion

In this study, a novel heuristic $CGIC$ is proposed for the ALS problem. The algorithm contains four parts: CR_{cost} as a chunking rule to generate several chunks from the landing sequence based on the cost of aircraft without length constraints, GR as a chunk generation rule to regenerate chunks with the largest cost first, IH as an improvement heuristic to decrease the cost of chunks by disordering a subsequence around the aircraft with the largest cost, and CNR as a connection rule to construct the final landing sequence with

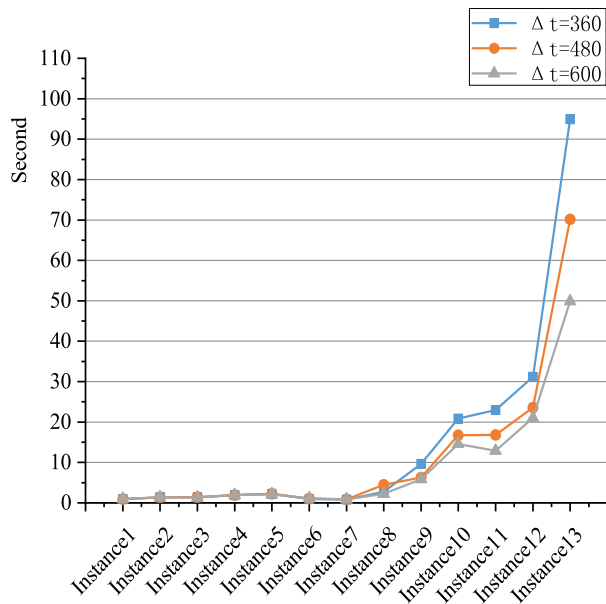


Fig. 8 Average computation time in dynamic instances with different Δt (second).

chunks. The proposed algorithm outperforms other heuristics in most static instances. The computation time is short enough to solve the dynamic ALS problem. Further developments may focus on the automatic generation of chunking rules for the multiple runways ALS problem.

Acknowledgments

This study is supported by the National Natural Science Foundation for Young Scientists of China (No.61802282), Natural Science Foundation for Young Scientists of Tianjin (No.18JCQNJC70000) and National Student Training Program for Innovation and Entrepreneurship of China (No.201710069050).

References

- [1] J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha, and D. Abramson, "Scheduling aircraft landings the static case," *Transport. Sci.*, vol.34, no.2, pp.180–197, May 2000.
- [2] A. Faye, "Solving the aircraft landing problem with time discretization approach," *Eur. J. Oper. Res.*, vol.242, no.3, pp.1028–1038, May 2015.
- [3] N.R. Sabar and G. Kendall, "An iterated local search with multiple perturbation operators and time varying perturbation strength for the aircraft landing problem," *Omega*, vol.56, pp.88–98, Oct. 2015.
- [4] J.E. Beasley, J. Sonander, and P. Havelock, "Scheduling aircraft landings at London Heathrow using a population heuristic," *J. Oper. Res. Soc.*, vol.52, no.5, pp.483–493, May 2001.
- [5] H. Pinol and J.E. Beasley, "Scatter search and biologic algorithms for the aircraft landing problem," *Eur. J. Oper. Res.*, vol.171, no.2, pp.439–462, June 2006.
- [6] A.T. Ernst, M. Krishnamoorthy, and R.H. Storer, "Heuristic and exact algorithms for scheduling aircraft landings," *Networks: An International Journal*, vol.34, no.3, pp.229–241, Oct. 1999.
- [7] X.B. Hu and E. Di Paolo, "Binary-representation-based genetic algorithm for aircraft arrival sequencing and scheduling," *IEEE Trans.*

- [8] *Intell. Transp. Syst.*, vol.9, no.2, pp.301–310, June 2008.
- [8] S.-P. Yu, X.-B. Cao, and J. Zhang, "A real-time schedule method for aircraft landing scheduling problem based on cellular automation," *Appl. Soft Comput.*, vol.11, no.4, pp.3485–3493, June 2011.
- [9] A. Salehipour, M. Modarres, and L.M. Naeni, "An efficient hybrid meta-heuristic for aircraft landing problem," *Comput. Oper. Res.*, vol.40, no.2, pp.207–213, Jan. 2013.
- [10] W. Shi, X.-Y. Song, and J.-Z. Sun, "A dynamic hyper-heuristic based on scatter search for the aircraft landing scheduling problem," *IEICE Trans. Fundamentals*, vol.E97-A, no.10, pp.2090–2094, Oct. 2014.
- [11] F. Furini, C.A. Persiani, and P. Toth, "Aircraft sequencing problems via a rolling horizon algorithm," *Int. Symp. on Comb. Opt.*, pp.273–284, Springer, Berlin, Heidelberg, April 2012.
- [12] F. Furini, M.P. Kidd, C.A. Persiani, and P. Toth, "Improved rolling horizon approaches to the aircraft sequencing problem," *J. Sched.*, vol.18, no.5, pp.435–447, Oct. 2015.
- [13] J.E. Beasley, "OR-library: Distributing test problems by electronic mail," *J. Oper. Res. Soc.*, vol.41, no.11, pp.1069–1072, Nov. 1990.



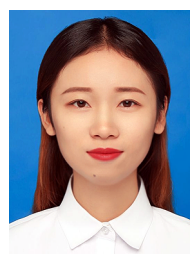
Wen Shi received the B.S. and Ph.D. degrees in Computer Science and Technology from Tianjin University in 2006 and 2015, respectively. He received the M.S. degree in Traffic Information Engineering and Control from Civil Aviation University of China. He is a lecturer with the School of Information Engineering, Tianjin University of Commerce, China. His current research interests include heuristics, machine learning and data mining.



Shan Jiang received the B.S. degree in Tianjin University of Technology and Education. He is an engineer with Tianjin Medical University.



Xuan Liang received the B.S. degree in Tianjin University of Commerce.



Na Zhou is a student in Tianjin University of Commerce.