2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Aircraft Landing Problem using Hybrid Differential Evolution and Simple Descent Algorithm

Nasser R. Sabar and Graham Kendall, *Senior Member IEEE*

***Abstract*—The aircraft landing problem (ALP) is a practical and challenging optimization problem for the air traffic industry. ALP involves allocating a set of aircrafts to airport runways and allocating landing times for which the goal is to minimize the total cost of landing deviation from the preferred target times. Differential evolution (DE) is a population based algorithm that has been shown to be an effective algorithm for solving continuous optimization problems. However, DE can suffer from slow convergence when utilized for combinatorial optimization problems, thus hindering its ability to return good quality solutions in these domains. To address this we propose a hybrid algorithm that combines differential evolution with a simple descent algorithm. DE is responsible for exploring new regions in the search space, whilst the descent algorithm focuses the search around the area currently being explored. Experimenting with widely used ALP benchmark instances, we demonstrate that the proposed hybrid algorithm performs better than DE without the simple descent algorithm. Furthermore, performance comparisons with other algorithms from the scientific literature demonstrate that our hybrid algorithm performs better, or at least comparably, in terms of both solution quality and computational time**.

I. Introduction

Over the past 20 years, the demand for air transportation has increased significantly [1]. This leads to congestion in airspace, resulting in airports not being able to cope with all the demands placed upon it. Consequently, airport managers face challenges in providing efficient services, and having to shift or change the landing and/or the departure time of some aircrafts. These changes lead to inefficient use of airport resources and potentially poor customer service. The aircraft landing problem (ALP) plays a pivotal role in determining the landing time of the arrival aircrafts [1], [2]. The ALP involves the construction of a landing schedule for a set of aircrafts, such that each aircraft is assigned to land on a specific runway, at a specific time, while ensuring that all the problem and safety constraints are respected. The aim is to minimize the overall penalty that will be incurred when an aircraft lands before or after its preferred time [1], [2].

ALP is an NP-hard problem. For this reason heuristic and meta-heuristic algorithms have been widely used to seek good quality solutions, within an acceptable time, instead of using exact methods [1], [2]. Although exact methods can provide optimal solutions, their computational time tends to grow exponentially as the problem size increases, which makes them only suitable for small/medium-sized problems [1]. Examples of heuristic and meta-heuristic algorithms that have been proposed for ALP are: first-come-first-serve [2], scatter search [3], bionomic [3], genetic algorithm [4], cellular automata [5], simulated annealing [6], and hybrid algorithms [6], [4]. Despite the numerous algorithms for ALP, no one algorithm has been shown to be an efficient solution method over all problem instances and their performance decreases as the instances grow larger.

In this work, we propose a hybrid algorithm to tackle ALP. The proposed algorithm combines differential evolution (DE) with a simple descent local search algorithm. DE is a well-known population based algorithm that has been shown to be very efficient in handling continuous optimization problems [7], [8]. Unfortunately, the application of DE to combinatorial optimization problems is not without any modification on DE operators and is not as good as continuous optimization [8]. This is because the DE evolutionary operators (mutation and crossover), which are used to generate new solutions, were introduced to deal with real values and thus some modifications are needed to deal with combinatorial optimization problems that, typically, use integers to represent their solutions. In other words, a basic DE algorithm cannot be applied to combinatorial optimization problems without modification to the main DE operators. In addition, despite the success in solving continuous optimization problems, DE has been criticized for its slow convergence and this is also apparent when dealing with combinatorial optimization problems, due to the exploration bias of DE [8].

To deal with ALP, a combinatorial optimization problem, we need an appropriate solution representation that allows DE to use its original evolutionary operators to generate new solutions, and we also need to accelerate convergence. To deal with the solution representation, we utilize a real value representation, mapping each solution onto an ALP schedule. To overcome slow convergence, the proposed DE is coupled with a simple descent algorithm in such a way that DE is responsible for exploring new regions of the search space, whilst the simple descent algorithm focuses the search around the area currently being explored. The computational experiments on the widely used ALP benchmark instances [2] demonstrate that the proposed hybrid algorithm outperforms DE when used without the simple descent algorithm. The proposed hybridized approach is also highly competitive, even better on some instances, when compared to the algorithms reported in the scientific literature.

Nasser R. Sabar is with The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih, Selangor, Malaysia (e-mail: Nasser.Sabar@nottingham.edu.my)
Graham Kendall is with The University of Nottingham, UK and also with The University of Nottingham Malaysia Campus, Jalan Broga, 43500 Semenyih, Selangor, Malaysia (e-mail: Graham.Kendall@nottingham.edu.my)

II. PROBLEM DESCRIPTION

The aircraft landing problem (ALP) can be defined as follows [2]. When a set of arrival aircrafts enter the radar coverage of the destination airport, the air traffic controller assigns, for each aircraft, a landing runway and a specific landing time on the determined runway. The assignment is subject to a set of constraints that should be respected under any circumstances. These are [2]:

- Each aircraft is assigned to exactly one runway.
- No more than one aircraft should be assigned the same landing time on the same runway.
- Each aircraft should be assigned a landing time, which has a predefined landing time window.
- The separation time between the aircrafts that have been assigned to land on the same runway should be respected.

In APL, each aircraft is associated with a predefined, preferred landing time and landing time window. The main goal is that each aircraft should land at its landing time. A penalty cost will be added if the aircraft is assigned to land before or after the preferred landing time. The main aim of the optimization algorithm is to minimize the total penalty cost by generating, for a given set of aircrafts, the best landing sequence on given runways and assigning a landing time for each aircraft. The problem inputs and the formulation are as follows [2], [5]:

- n: the number of the arrival aircrafts.
- m: the number of runways.
- s_{ij}: the separation time ($s_{ij} > 0$) between aircrafts i and j when they are assigned to same runway.
- T_i: the preferred landing time (target time) of aircraft i.
- E_i: the earliest landing time of aircraft i.
- L_i: the latest landing time of aircraft i.
- $C1_i$: the incurred penalty per unit of time for late landing of aircraft i.
- $C2_i$: the incurred penalty per unit of time for early landing of aircraft i.
- x_i: the assigned landing time of aircraft i ($i = 1, 2, \ldots, n$)

The penalty cost of the i^{th} aircraft that has been assigned for landing before or after the target landing time is calculated as follows:

$$p_i = \begin{cases} C1_i(T_i - x_i) & if\ x_i \le T_i \\ C2_i(x_i - T_i) & if\ T_i > x_i \end{cases} \quad (1)$$

And the total penalty cost for all aircrafts is calculated as follows:

$$\text{Minimize } f = \sum_{i=1}^{n} p_i \quad (2)$$

Subject to:

$$E_i \le x_i \le L_i, \quad i = 1, 2, \ldots, n \quad (3)$$

$$x_j - x_i > s_{ij}, (i = 1, \ldots, n, j = 1, \ldots, n) \quad (4)$$

Equation (3) ensures that each aircraft is assigned to land within its time windows, while equation (4) verifies that the safety constraint between the aircrafts assigned to the same runway is respected.

III. THE PROPOSED ALGORITHM

In this section, we first discuss the basic differential evolution algorithm followed by the proposed hybrid algorithm.

A. Differential evolution algorithm

Differential evolution (DE) is a stochastic population based algorithm introduced by [7] as a variant of an evolutionary algorithm. DE was originally proposed to deal with continuous optimization problems or to optimize real parameter and real valued functions. DE has a greedy nature that generates, at each generation, a new population of solutions to replace the old population, provided that the new solutions are better in terms of fitness function [8]. In DE, a new population of solutions is created using two main operators (mutation and crossover). The mutation operator selects three different solutions from the current population and then combines them to generate a new solution. Then, the generated solution is combined with the initial solution via the crossover operator. The new solution is added to the new population if its quality is better than the initial solution. The basic steps of the DE (DE/rand/1/bin) are as follows [7]:

Step 1: Randomly generate a population of solutions, *NP*.

Step 2: Evaluate the fitness, *f*, of the population.

Step 3: For each solution x_i^G in the current population *NP* (i is the solution index and G is the current generation) generate a mutated solution (m_i^G) using (5):

$$m_{i,j}^G = x_{1,j}^G + F * (x_{2,j}^G - x_{3,j}^G), \forall j \in \{1, \ldots, n\} \quad (5)$$

where j represent the decision variable, n represent the maximum number of decision variables in a given problem instance, F is the scaling factor ($F \in [0, 1]$) and x_1^G, x_2^G and x_3^G are three randomly selected solutions from the current population where $x_1^G \neq x_2^G \neq x_3^G$.

Step 4: Recombine (crossover step) the solution generated by a mutation operator (m_i^G) with the target solution (x_i^G) based on the crossover rate *CR* ($CR \in [0, 1]$) to generate a new offspring (m_i^{G+1}) as follows (6):

$$m_{i,j}^{G+1} = \begin{cases} m_{i,j}^G & if\ Rand(j) \le CR\ or\ j = Rnd(i) \\ x_{i,j}^G & if\ Rand(j) > CR\ and\ j \neq Rnd(i) \end{cases} \quad (6)$$

$\forall j \in \{1, \ldots, n\}, \forall i \in \{1, \ldots, | NP |\}$

where *Rand(j)* is a random number (*Rand(j)* $\in [0, 1]$) selected for the j^{th} decision variable, *Rnd(i)* is a

random decision variable index (*Rnd(i)* ∈ {1,…, *n*}). *Rnd* ensure that m_i^{G+1} gets at least one decision variable from m_i^G.

Step 5: Calculate the fitness of m_i^{G+1} and compare it with x_i^G. Replace x_i^G with m_i^{G+1} if m_i^{G+1} fitness is better than x_i^G as follows (7):

$$x_i^{G+1} = \begin{cases} m_i^{G+1} & if\ f(m_i^{G+1}) \le f(x_i^G) \\ x_i^G & if\ f(m_i^{G+1}) > f(x_i^G) \end{cases} \quad (7)$$

$$\forall i \in \{1,\ldots,|NP|\}$$

Step 6: If the termination criterion is satisfied (the number of generations), stop and return the best solution. Otherwise, go to ***Step 3***.

B. The proposed hybrid algorithm for the ALP

In this section, we first discuss the application of DE for the ALP and then the proposed hybrid DE algorithm.

1) DE for ALP

To apply a DE to ALP, we need to define a suitable solution representation. This is because DE mutation and crossover operators were originally proposed to deal with continuous optimization problems or real valued functions. In these kinds of problems, the decision variables are assigned real values and these values are directly used in the calculation of the fitness function *f*(*x*). Unfortunately, the mutation and crossover operators of the basic DE cannot be used to solve combinatorial optimization problems such as ALP [8]. This is because ALP deals with integer values and it has some problem related constraints that have to be respected. Therefore, to have a direct relation between ALP and DE operators, in this work, we represent the ALP solutions using real value numbers, in the same way as is used for continuous optimization problems, but each part of the real number represents a different role in ALP. More precisely, in ALP we need to assign for each aircraft a runway and landing time on the selected runway. Here our solution representation has *n* decision variables and each one represents one aircraft. Each decision variable takes a real value between 1 and *m*, where *m* is the number of runways in a given problem instance. Then, for each decision variable, the integer part of generated number represents the allocated runway, whilst the fraction part denotes the order of the aircraft on this runway. For example, assume an instance of ALP that has 6 aircraft (*n*=6) that need to be scheduled to land on 3 runways (*m*=3). If we assign numbers for the aircrafts from 1 to 6, the decision variables will be (1, 2, 3, 4, 5, 6), as shown in the first row of Table 1. Next, generate for each decision variable a random number *r* (*r* ∈ [1, 3])), as shown in second row of Table 1.

TABLE 1 THE DECISION VARIABLES AND THEIR CORRESPONDING REAL VALUES

Decision variables	1	2	3	4	5	6
Values	2.2	1.45	1.4	3.8	3.65	2.2

To decode Table 1 into an ALP solution, the procedure will be as follows:

- Aircraft 1 is assigned to land on the second runway, aircraft 2 is assigned to land on the first runway, aircraft 3 is assigned to land on the first runway, aircraft 4 is assigned to land on the third runway, and so on.
- On each runway we sort the assigned aircraft in an ascending order in such a way that the order represents the landing time on this runway.

In this work, we use this kind of representation in order to avoid modifying the DE mutation and crossover operators and also to perverse DE original features. We also restricted the decision variable boundaries in order to ensure that their values are within the feasible area. That is, the integer part of each decision variable is restricted between 1 and *m*. In this work, the initial population of solutions of DE is generated by assigning for each decision variable a random value drawn from a uniform distribution in the interval of (1, *m*) and the generated solutions are assigned a fitness values using equation (2). Afterwards, the six steps of DE discussed in section III.A are repeatedly applied until the stopping condition is satisfied.

2) A hybrid DE for ALP

In DE, the mutation operator (equation (5), section III.A) generates a new solution by adding the weighted difference of the two selected solutions into the third one. In ALP, any small modification on a current solution might lead to a big change in the solution fitness value because this modification might shift or swap several aircrafts from their current runway and landing time to new ones. This may cause slow convergence and generate a low quality solution. Therefore, to accelerate the convergence rate of DE, we coupled it with a simple descent algorithm. That is, at each generation of DE, solutions that are generated by the mutation and crossover operators are further improved by the simple descent algorithm. Hence, the simple descent algorithm takes place after the mutation and crossover operators and before DE population update step. Thus DE is responsible for exploring new regions of the search space, whilst the simple descent algorithm focuses the search around the area currently being explored. The simple descent algorithm is an improvement method that starts with an initial solution and iteratively explores its neighborhood solutions using a move operator. A generated neighborhood solution is accepted if its quality is better than the current one (our solution quality is calculated using equation (2)). In this work, a neighborhood solution is generated by applying one of the following move operators which is selected at random:

- MO1: randomly select a runway and examine all possible swaps between each pair of aircrafts.
- MO2: randomly select two different runways and examine all possible swaps of aircraft between the selected runways.

- MO3: randomly select one aircraft and move it to another runway.

The simple descent algorithm stops the search if there is no improvement in the fitness function after a predefined number of consecutive iterations (20 non-improvement iterations, fixed based on a preliminary test). Generally, invoking the simple descent algorithm at every generation would be computationally expensive and also might cause DE to prematurely converge in the early stages of the search. To avoid this tendency, and to control the number of calls to the simple descent algorithm, the application of the simple descent algorithm will be based on the probability of *Ps* which is calculated as follows:

$$Ps = \frac{iter\_c}{iter\_max} \tag{8}$$

where *iter_c* represents the current iteration and *iter_max* represents the maximum number of iterations. In equation (8), the simple descent algorithm is applied more frequently in the latter stages of the search process than in the early stages. Hence, the search will be more explorative in the early stages and will gradually change to be more exploitive in the latter stages. The pseudocode of the proposed hybrid algorithm is presented in Algorithm 1((DE/rand/1/bin) [8]).

Algorithm 1: The basic DE algorithm (DE/rand/1/bin)

```
1  Set NP, F, CR, Maxg, LSiter, D=n, G=0;
2  Randomly generate a population of solutions
3  Calculate the fitness value (f) for each solution
4  while (G< Maxg) do
5      for i=1 to |NP| do
6          Randomly select r1, r2, and r3 from NP
           where r1≠ r2 ≠ r3≠i
7          for j=1 to D do // eq. (5)
8              m_{i,j}^G = x_{r1,j}^G + F*(x_{r2,j}^G - x_{r3,j}^G)
9          endfor
10         Rnd(j)=rand [1, d]
12         for j=1 to D do      //eq. (6)
13             if(rand(0,1)≤CR or j== Rnd (j) then
14                 u_{i,j}^G = m_{i,j}^G
15             else
16                 u_{i,j}^G = x_{i,j}^G
17         endfor
18         Ps= G/Maxg; r=rand [0,1]
19         if (r ≤ Ps) then //eq. (8)
20         Call the simple descent algorithm to
           improve u_i^G
21         If(f(u_i^G) < f(x_i^G)) then x_i^{G+1} = u_i^G
22         else x_i^{G+1} = x_i^G
23     endfor
24     G=G+1;
25 end while
26 Return the best solution
```

The proposed hybrid algorithm works as follows: first set DE parameters (line 1), generate a population of solutions (line 2) and calculate the fitness value for each solution (line 3). Then, the while-loop is executed (lines 4 to 27). In each generation, a new population of solutions is created (lines 5 to 25) using the mutation (lines 7 to 9) and crossover operators (lines 12 to 19). Next, calculate the probability of calling the simple descent algorithm (line 21) and apply the simple descent algorithm to improve the current solution (line 22). Replace the new solution with the incumbent one if new one is better in term of the fitness value (line 23). Otherwise, keep the old solution (line 24). Update the generation counter (line 26) and check the stopping condition. If the stopping condition is satisfied stop and return the best solution (line 28). Otherwise, start a new generation.

IV. Experimental Setup

In this section, we discuss the main characteristic of the ALP benchmark instances and the parameter settings of the proposed hybrid algorithm.

A. Benchmark Instances

The performance of the proposed algorithm is assessed using the 13 ALP benchmark instances that have been used by other researchers in the scientific literature [2]. The instances are introduced in [2] and are publically available at the OR-library[1]. Table 2 shows the main characteristics of the ALP 13 instances. The first column represents instance names, the second column represents the number of aircraft (*n*), the third column represents the number of runways (*m*) and fourth column represents instance number (instance no.).

In these instances, the number of aircraft and runways is different from one to another. That is, the number of aircraft is 10 to 500 and the number of runways varies from 1 to 5. Based on the number of aircraft in each instance, Airland1 to Airland8 instances (instance no. 1 to 25, Table 2) are categorized as a small-sized instances, while Airland9 to Airland13 (instance no. 26 to 49, Table 2) are classified as a large-sized instances.

TABLE 2 THE CHARACTERISTICS OF THE ALP BENCHMARK INSTANCES

Instance name	*n*	*m*	*Instance no.*
Airland1	10	1	1
		2	2
		3	3
Airland2	15	1	4
		2	5
		3	6
Airland3	20	1	7
		2	8
		3	9
Airland4	20	1	10
		2	11
		3	12
		4	13
Airland5	20	1	14
		2	15
		3	16

[1] http://people.brunel.ac.uk/~mastjjb/jeb/orlib/airlandinfo.html (last accessed, 15th Dec 2013)

		4	17
Airland6	30	1	18
		2	19
		3	20
Airland7	44	1	21
		2	22
Airland8	50	1	23
		2	24
		3	25
Airland9	100	1	26
		2	27
		3	28
		4	29
Airland10	150	1	30
		2	31
		3	32
		4	33
		5	34
Airland11	200	1	35
		2	36
		3	37
		4	38
		5	39
Airland12	250	1	40
		2	41
		3	42
		4	43
		5	44
Airland13	500	1	45
		2	46
		3	47
		4	48
		5	49

B. Parameter settings

The parameter settings of proposed hybrid algorithm are presented in Table 3. These settings were determined based on preliminary experiments by taking into consideration the solution quality and the computational time.

TABLE 3 THE PARAMETER SETTINGS

#	*Parameter*	*Value*
1	No. Of generations	200
2	Population size, *NP*	20
3	Scaling factor, *F*	0.1
4	Crossover rate, *CR*	0.4
5	Non-improvement iterations for the simple descent algorithm	20 iterations

V. RESULTS AND COMPARISONS

In this work, we have performed two experimental tests. The goal of the first one is to assess the benefit of coupling DE with the simple descent algorithm by comparing the results of DE with and without using simple descent algorithm (section V.A). In second experiment (section V.B), we compare the results of our algorithm with the best results obtained by the state of the art algorithms.

A. Effectiveness evaluation of the integrated local search

In this experiment, we assess the benefit of hybridizing DE with a simple descent algorithm. Therefore, DE is tested with a simple descent algorithm (denoted as DE-SD) and without a simple descent algorithm (denoted as DE). For both algorithms the initial solutions, random seeds, number of runs, the stopping condition and computer resources are the same for all experiments. In addition, for each ALP instance, both algorithms (DE-SD and DE) are executed for 31 independent runs with different random seeds.

The percentage gap (Δ(%)) of the best results produced by DE-SD and DE over 31 runs from the best known values in the literature (BKV) are presented in Table 4. Δ(%) is calculated as follows: $\Delta(\%)=(B\text{-}BKV/BKV)*100$, where B is the best result retuned by the tested algorithm (DE-SD or DE) over 31 runs and the BKV results are obtained from [3]. The best obtained results are shown in bold.

From the results reported in Table 4, we can see that, on small-sized instances (Airland1 to Airland8), DE-SD reached the best known results on all instances (instances 1 to 25). DE only reached the best known results on 15 instances, being worse than DE-SD on 10 out of 25 tested instances. Considering the large-sized instances (Airland9 to Airland13), the results in Table 4 clearly indicate that DE-SD outperformed DE on all instances (instances 26 to 49). Furthermore, on six instances (instances 26, 28, 35, 36, 37 and 46), DE-SD managed to update the best known results in the literature. From the above results, we can draw the conclusion that DE-SD outperforms DE which clearly justifies the benefit of coupling DE with the simple descent algorithm.

To statistically investigate the effect of the simple descent algorithm on DE performance, we have conducted a pairwise comparison between DE-SD and DE using the Wilcoxon statistical test with a significance level of 0.05. The *p*-value results of DE-SD versus DE are shown in last column of Table 4. In this comparison, we use the "+" symbol if DE-SD is statistically better than DE (*p*-value < 0.05), "-" if DE is statistically better than DE-SD (*p*-value > 0.05), and "=" if both DE-SD and DE have the same performance (*p*-value = 0.05). From Table 4 (last column), we can see that DE-SD is statistically significant than DE on 34 out of 49 tested instances (*p*-value < 0.05). The reported *p*-value (last column, Table 4) shows that, on 15 out of 49 instances, the performance of DE-SD and DE are the same. This finding, again, justifies the benefit of integrating the simple descent algorithm to improve the exploitation process of DE. Indeed, the use of simple descent algorithm does help DE to obtain very good results especially over the large-sized instances. Overall, the results demonstrate that the use of a simple descent algorithm can effectively enhance the performance of DE.

TABLE 4 THE RESULTS OF DE-SD COMPARED TO DE

Instance name	*Instance no.*	*m*	*BKV*	*DE-SD*	*DE*	*DE-SD Vs DE*
				Δ(%)	Δ(%)	*p-value*
Airland1	1	1	700	**0**	**0**	=
	2	2	90	**0**	**0**	=
	3	3	0	**0**	**0**	=
Airland2	4	1	1480	**0**	**0**	=
	5	2	210	**0**	**0**	=
	6	3	0	**0**	**0**	=
Airland3	7	1	820	**0**	**0**	=
	8	2	60	**0**	**0**	=
	9	3	0	**0**	**0**	=
Airland4	10	1	2520	**0**	**0**	=

	11	2	640	**0**	**0**	=
	12	3	130	**0**	**0**	=
	13	4	0	**0**	0.24	+
Airland5	14	1	3100	**0**	**0**	=
	15	2	650	**0**	**0**	=
	16	3	170	**0**	**0**	=
	17	4	0	**0**	0.3	+
Airland6	18	1	24442	**0**	3.8	+
	19	2	554	**0**	6.1	+
	20	3	0	**0**	1.5	+
Airland7	21	1	1550	**0**	2.7	+
	22	2	0	**0**	0.91	+
Airland8	23	1	1950	**0**	2.6	+
	24	2	135	**0**	3.4	+
	25	3	0	**0**	0.8	+
Airland9	26	1	5611.70	**-0.22**	21.24	+
	27	2	452.92	**0**	11.52	+
	28	3	75.75	**-0.12**	27.01	+
	29	4	0	**0**	8.81	+
Airland10	30	1	12329.31	**0.22**	60.31	+
	31	2	1288.73	**0**	18.42	+
	32	3	220.79	**0**	25.41	+
	33	4	34.22	**0**	37.26	+
	34	5	0	**0**	3.2	+
Airland11	35	1	12418.32	**-0.14**	8.11	+
	36	2	1540.84	**-1.21**	12.07	+
	37	3	280.82	**-3.12**	7.86	+
	38	4	54.53	**0**	64.54	+
	39	5	0	**0**	14.17	+
Airland12	40	1	16209.78	**0**	57.12	+
	41	2	1961.39	**0**	19.62	+
	42	3	290.04	**0**	43.75	+
	43	4	3.49	**0**	62.78	+
	44	5	0	**0**	48.17	+
Airland13	45	1	44832.38	**0**	35.24	+
	46	2	5501.96	**-0.51**	22.37	+
	47	3	1108.51	**0**	44.41	+
	48	4	188.46	**0**	20.16	+
	49	5	7.35	**0**	11.25	+

B. Comparison of DE-SD with the state of the art algorithms

In this experiment, the performance of DE-SD over 31 independent runs is compared with the state of the art algorithms from both a solution quality and a computational time perspective. The algorithms considered in the comparison are:

- SS: Scatter search algorithm proposed in [3].
- BA: Bionomic algorithm proposed in [3].
- SA1: A hybrid simulated annealing and variable neighborhood descent proposed in [6].
- SA2: A hybrid simulated annealing and variable neighborhood search proposed in [6].

The comparison of DE-SD and other algorithms in terms of the percentage gap (Δ(%)) from the best known value in the literature (BKV) is summarized in Table 5. Best obtained results are indicated in bold font. The results in Table 5 demonstrate that DE-SD able reach the best known values in the literature on all small-sized instances (instances 1 to 25). On large-sized instances (instances 26 to 49), DE-SD reached the best known results on 9 instances and being slightly inferior on 10 out of 24 instances. For 5 instances (instances 26, 28, 35, 37 and 46), DE-SD produced new best results.

In Table 6, we compare the computational times (in seconds) of DE-SD with other algorithms. As Table 6 indicates, across all instances, the computational time of DE-SD is lower than the compared algorithms. Thus, we can conclude that DE-SD is an effective algorithm for ALP.

VI. CONCLUSION

This work has proposed a hybrid algorithm for the aircraft landing problem. The proposed algorithm hybridizes a well-known population based algorithm, Differential Evolution, with a simple descent local search algorithm. The differential evolution algorithm is utilized to effectively explore the solution search space, while the simple descent algorithm focuses the search on the area currently being explored. Experiments have been carried out on the small-sized and large-sized instances that are available in the scientific literature. The computational results demonstrate that hybridizing differential evolution with a simple descent algorithm produces very good results compared to differential evolution without a simple descent algorithm, indicating that is it beneficial to include a local search algorithm within differential evolution for combinatorial optimization problems. The results also show that the proposed hybrid algorithm produced new best results for some instances compared to state of the art algorithms.

TABLE 5 THE COMPUTATIONAL RESULTS OF DE-SD COMPARED TO THE STATE OF THE ART ALGORITHMS

Instance name	*Instance no.*	*m*	*BKV*	*DE-SD*	*SS*	*BA*	*SA1*	*SA2*
				Δ(%)	Δ(%)	Δ(%)	Δ(%)	Δ(%)
Airland1	1	1	700	**0**	**0**	**0**	**0**	**0**
	2	2	90	**0**	**0**	**0**	**0**	**0**
	3	3	0	**0**	**0**	**0**	**0**	**0**
Airland2	4	1	1480	**0**	**0**	**0**	**0**	**0**
	5	2	210	**0**	**0**	**0**	**0**	**0**
	6	3	0	**0**	**0**	**0**	100	100
Airland3	7	1	820	**0**	**0**	**0**	**0**	**0**
	8	2	60	**0**	**0**	**0**	16.66	16.66
	9	3	0	**0**	**0**	**0**	100	100
Airland4	10	1	2520	**0**	**0**	**0**	**0**	**0**
	11	2	640	**0**	**0**	**0**	3.12	3.12
	12	3	130	**0**	**0**	**0**	23.07	27.07
	13	4	0	**0**	**0**	**0**	100	100
Airland5	14	1	3100	**0**	**0**	**0**	**0**	**0**
	15	2	650	**0**	**0**	3.08	**0**	**0**
	16	3	170	**0**	**0**	**0**	**0**	**0**
	17	4	0	**0**	**0**	**0**	100	100
Airland6	18	1	24442	**0**	**0**	**0**	**0**	**0**
	19	2	554	**0**	**0**	3.61	**0**	**0**
	20	3	0	**0**	**0**	**0**	**0**	**0**
Airland7	21	1	1550	**0**	**0**	**0**	**0**	**0**
	22	2	0	**0**	**0**	**0**	**0**	**0**
Airland8	23	1	1950	**0**	52.05	36.15	**0**	**0**
	24	2	135	**0**	0	**0**	**0**	**0**
	25	3	0	**0**	0	**0**	100	100
Airland9	26	1	5611.70	**-0.22**	30.06	14.51	8.55	8.55
	27	2	452.92	0	5.67	54.73	**-0.58**	0
	28	3	75.75	**-0.12**	0	87.46	0	0
	29	4	0	**0**	**0**	-	**0**	**0**
Airland10	30	1	12329.31	0.22	44.96	33.90	**0**	**0**
	31	2	1288.73	0	7.87	25.95	**-5.39**	0
	32	3	220.79	0	8.88	195.88	**-6.49**	0
	33	4	34.22	**0**	16.74	292.40	3.09	3.09
	34	5	0	**0**	**0**	-	100	100
Airland11	35	1	12418.32	**-0.14**	17.95	16.67	0	0
	36	2	1540.84	-1.21	9.19	38.54	**-8.04**	0
	37	3	280.82	**-3.12**	21.59	290.09	-2.81	0
	38	4	54.53	**0**	2.77	474.47	**0**	**0**
	39	5	0	**0**	**0**	-	**0**	**0**
Airland12	40	1	16209.78	**0**	22.15	23.58	**0**	**0**
	41	2	1961.39	**0**	18.80	50.18	**0**	**0**
	42	3	290.04	0	17.48	198.01	**-3.56**	0
	43	4	3.49	**0**	271.63	13216.91	**0**	**0**
	44	5	0	**0**	**0**	-	**0**	**0**
Airland13	45	1	44832.38	0	3.24	1.03	**-7.54**	0
	46	2	5501.96	**-0.51**	3.72	37.47	**-0.47**	0
	47	3	1108.51	0	1.98	182.69	**-32.79**	0
	48	4	188.46	0	22.98	1186.81	**-46.62**	0
	49	5	7.35	0	0	22308.44	**-48.16**	0

Note: *Best results are shown in bold font. Δ (%) represents the percentage gap from the best result. "-" indicates no feasible solution has been reported.*

TABLE 6 THE COMPUTATION TIME OF DE-SD COMPARED TO OTHER ALGORITHMS

Instance name	*Instance no.*	*m*	*DE-SD*	*SS*	*BA*	*SA1*	*SA2*
Airland1	1	1	**0**	4	60	0	0
	2	2	**0**	24	45	0	0
	3	3	**0**	39	34	0	0
Airland2	4	1	**1.2**	6	90	1.59	1.38
	5	2	**1.1**	45	49	1.66	1.65
	6	3	**1.6**	46	43	1.98	1.91
Airland3	7	1	**0**	8	99	1.78	1.73
	8	2	**2.4**	48	58	3.12	4.22
	9	3	**2.1**	62	63	3.29	5.11
Airland4	10	1	3.2	8	95	**1.98**	2.85

	11	2	**2.2**	52	55	3.56	3.94
	12	3	**3.61**	46	57	3.74	5.05
	13	4	**5.4**	56	52	4.06	7.15
Airland5	14	1	2.3	9	100	**1.85**	1.89
	15	2	5.2	50	61	**3.04**	4.84
	16	3	**3.4**	54	43	4.11	4.92
	17	4	4.1	56	68	4.35	**3.04**
Airland6	18	1	**0.9**	158	274	2.12	2.14
	19	2	**3.5**	70	101	3.98	4.01
	20	3	**4.32**	54	87	4.41	5.91
Airland7	21	1	**1.6**	195	79	2.68	2.65
	22	2	4.12	118	124	2.83	**2.37**
Airland8	23	1	**6.4**	42	287	7.1	7.31
	24	2	**5.6**	121	196	10.73	9.85
	25	3	**9.71**	139	181	14.11	17.39
Airland9	26	1	**7.8**	119	554	11.59	10.12
	27	2	**10.22**	342	487	13.78	13.64
	28	3	**16.4**	390	466	17.95	18.46
	29	4	**16.22**	336	439	19.69	21.18
Airland10	30	1	**15.38**	227	925	20.12	20.75
	31	2	**18.7**	608	845	21.33	22.04
	32	3	**24.5**	668	803	27.62	25.19
	33	4	**34.8**	647	788	30.12	41.28
	34	5	**29.4**	607	762	39.85	40.15
Airland11	35	1	**24.13**	256	1417	24.17	33.84
	36	2	**28**	959	1287	29.09	33.99
	37	3	**36.4**	1021	1203	41.22	37.19
	38	4	**33.7**	993	1168	42.4	45.96
	39	5	**51.96**	956	1158	66.23	61.05
Airland12	40	1	**174**	381	2011	219.03	198.85
	41	2	**310.6**	1266	1835	362.6	313.46
	42	3	**360.14**	1454	1710	412.73	379.91
	43	4	**374.2**	1445	1688	410.33	401.04
	44	5	**382.4**	1386	1662	394.6	386.16
Airland13	45	1	**522.4**	1237	5852	566.82	528.84
	46	2	**1024**	3836	5379	1047.93	1294.23
	47	3	**1120.2**	4560	5158	1241	1334.33
	48	4	**1186**	4413	4977	1201.8	1197.48
	49	5	**1154.3**	4421	4887	1203.93	1185.46

***Note**: The presented times are in seconds. Bold indicate the best computation time.*

REFERENCES

[1] J. Bennell, M. Mesgarpour, and C. Potts, "Airport runway scheduling," *Annals of Operations Research,* vol. 204, pp. 249-270, 2013/04/01 2013.

[2] J. E. Beasley, M. Krishnamoorthy, Y. M. Sharaiha, and D. Abramson, "Scheduling aircraft landings—the static case," *Transportation science,* vol. 34, pp. 180-197, 2000.

[3] H. Pinol and J. E. Beasley, "Scatter search and bionomic algorithms for the aircraft landing problem," *European Journal of Operational Research,* vol. 171, pp. 439-462, 2006.

[4] G. Hancerliogullari, G. Rabadi, A. H. Al-Salem, and M. Kharbeche, "Greedy algorithms and metaheuristics for a multiple runway combined arrival-departure aircraft sequencing problem," *Journal of Air Transport Management,* vol. 32, pp. 39-48, 2013.

[5] S.-P. Yu, X.-B. Cao, and J. Zhang, "A real-time schedule method for Aircraft Landing Scheduling problem based on Cellular Automation," *Applied Soft Computing,* vol. 11, pp. 3485-3493, 2011.

[6] A. Salehipour, M. Modarres, and L. Moslemi Naeni, "An efficient hybrid meta-heuristic for aircraft landing problem," *Computers & Operations Research,* vol. 40, pp. 207-213, 2013.

[7] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization,* vol. 11, pp. 341-359, 1997.

[8] S. Das and P. N. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art," *Evolutionary Computation, IEEE Transactions on,* vol. 15, pp. 4-31, 2011.