# A Ripple-Spreading Genetic Algorithm for the Aircraft Sequencing Problem

**Xiao-Bing Hu** xiaobing.hu@warwick.ac.uk
School of Engineering, University of Warwick, Coventry, CV4 7AL, United Kingdom

**Ezequiel A. Di Paolo** ezequiel@sussex.ac.uk
Ikerbasque, Department of Logic and Philosophy of Science, University of the Basque Country, San Sebastian, 20080, Spain

**Abstract**

When genetic algorithms (GAs) are applied to combinatorial problems, permutation representations are usually adopted. As a result, such GAs are often confronted with feasibility and memory-efficiency problems. With the aircraft sequencing problem (ASP) as a study case, this paper reports on a novel binary-representation-based GA scheme for combinatorial problems. Unlike existing GAs for the ASP, which typically use permutation representations based on aircraft landing order, the new GA introduces a novel ripple-spreading model which transforms the original landing-order-based ASP solutions into value-based ones. In the new scheme, arriving aircraft are projected as points into an artificial space. A deterministic method inspired by the natural phenomenon of ripple-spreading on liquid surfaces is developed, which uses a few parameters as input to connect points on this space to form a landing sequence. A traditional GA, free of feasibility and memory-efficiency problems, can then be used to evolve the ripple-spreading related parameters in order to find an optimal sequence. Since the ripple-spreading model is the centerpiece of the new algorithm, it is called the ripple-spreading GA (RSGA). The advantages of the proposed RSGA are illustrated by extensive comparative studies for the case of the ASP.

**Keywords**

Arrival sequencing problem, ripple-spreading model, feasibility, optimization, binary representations.

## 1 Introduction

Genetic algorithms (GAs) were originally developed based on binary representation to solve problems whose solutions are based on value (Holland, 1975; Eiben and Schoenauer, 2002). They were soon extended to many combinatorial problems where solutions are based on permutation (Bäck et al., 1997; Eiben and Smith, 2003). In the design of GAs for combinatorial problems, the basic binary representation is difficult to apply, and instead, various problem-specific permutation representations have been proposed. Unfortunately, such permutation representations must confront the common problems of infeasible chromosomes and memory inefficiency. As a result, in many applications of GAs to combinatorial problems, evolutionary operators have to be modified mainly to serve feasibility purposes. Some applications even discard certain evolutionary operators altogether, such as crossover, because it proves to be more destructive than effective to evolve chromosomes. Many classic evolutionary operators, such as uniform crossover, which were developed on the basis of binary representation, are hard to apply to permutation-based GAs in combinatorial problems.

Recently, a novel GA scheme, based on binary representations, free of both feasibility and memory-efficiency problems, has been applied to quite a few combinatorial problems (the traveling salesman problem, Hu and Di Paolo, 2007, the topology optimization of complex networks, Hu, Di Paolo, and Barnett, 2008, the airport gate assignment problem, Hu and Di Paolo, 2009b, and the network coding problem, Hu, Leeson, et al., 2010). The results have illustrated its advantages and potential in terms of feasibility, memory efficiency, and compatibility. The basic idea of this scheme is the following. First, a combinatorial problem is transformed by creating a problem-specific artificial space and then projecting the elements that compose the solutions to the combinatorial problem onto this space. Secondly, this is followed by designing a parameterized ripple-spreading process, so that for each given set of values for the parameters, the ripple-spreading process will connect all elements on the space to form a unique solution to the original combinatorial problem. And lastly, a basic binary-representation based GA is used to evolve these ripple-spreading parameters in order to find an optimal or near-optimal solution. Since the ripple-spreading process plays a crucial role in this new GA scheme, it is called the ripple-spreading GA (RSGA). The work of Hu and colleagues (Hu and Di Paolo, 2007; Hu, Di Paolo, and Barnett, 2008; Hu and Di Paolo, 2009b; Hu, Leeson, et al., 2010) provided some preliminary results on the RSGA, which were reviewed in Hu, Wang, et al. (2010). This paper aims to make a deeper and more systematic investigation of the RSGA scheme using another challenging task, the aircraft sequencing problem (ASP), as a study case.

The ASP is a major issue in daily airport operations (Pelegrin, 1994; Carr et al., 1999, 2000). In the past decades many methods have been used to address this problem (Psaraftis, 1978, 1980; Venkatakrishnan et al., 1993; Bianco et al., 1997, 1988; Robinson et al., 1997; Beasley et al., 2001; Hansen, 2004; Cheng et al., 1999; Hu and Chen, 2005; Hu and Di Paolo, 2008, 2009a; Ciesielski and Scerri, 1998). The ASP is an NP-hard problem with no known algorithm for finding a global optimal solution within a polynomial-bounded amount of time. As large-scale parallel stochastic search algorithms, GAs are effective for tackling NP-hard problems such as the ASP (e.g., see Beasley et al., 2001; Hu and Di Paolo, 2009a; Ciesielski and Scerri, 1998). GAs have even been tested at airports and produced satisfactory results (Beasley et al., 2001; Ciesielski and Scerri, 1998). Permutation representations have often been used in these GAs. For instance, the evolutionary algorithms reported in previous works (Hansen, 2004; Cheng et al., 1999; Hu and Chen, 2005) used the order of each aircraft in a candidate landing sequence to construct chromosomes. The GA in Hu and Di Paolo (2008) used 0-1-valued matrix to record the relative positions of aircraft in the landing sequence, and this GA was extended to multi-runway systems by using integer-valued matrix in Hu and Di Paolo (2009a). Apart from permutation representations, some value-based representations have also been successfully used in some GAs for the ASP. For example, the GAs in Beasley et al. (2001) and Ciesielski and Scerri (1998) used a chromosome to directly record scheduled times of arrival (STAs) for aircraft. These representations of landing sequences explicitly reflect the underlying physical meanings of chromosomes in a straightforward way, and therefore can easily be understood by human air traffic controllers and others. However, feasibility problems often result from these representations. The stochastic evolutionary process can often make chromosomes invalid/infeasible in terms of the representations underlying physical meaning. For this reason, feasibility checking and other correcting measures, such as deterministic methods and heuristic rules, have had to be introduced to help evolutionary operators to produce valid/feasible chromosomes. Some GAs have even discarded the crossover operator altogether, simply because it turned out to be more destructive than effective in terms of feasibility (Cheng et al., 1999; Hu and Chen, 2005). The matrix-representation-based GAs in Hu and Di Paolo (2008) and Hu and Di Paolo (2009a) were claimed to have achieved better performance than others, but they suffered from $O(n^2)$ memory problem, which means that while suitable for the ASP, they might not be able to generalize their advantages to other large-scale problems like complex network optimization with the same underlying representation issues.

This paper applies the RSGA scheme to the ASP. Thanks to the novel ripple-spreading model, the proposed RSGA can employ a value-based representation that has nothing (directly) to do with aircraft landing sequence. As a result, the new algorithm is compatible with all classic evolutionary operators, and free of both feasibility and memory-efficiency problems. It is also for this reason easy to generalize to other combinatorial problems. From this case study, a better understanding of the RSGA scheme at work is achieved, which is important to push the scheme toward real-world applications in the future. The remainder of this paper is organized as follows. The basic idea of the RSGA scheme is explained in Section 2. The mathematical description of the ASP is given in Section 3. The associated transformed problem, that is, the ripple-spreading model of the ASP, is described in Section 4. Some important GA-related issues are discussed in Section 5. Further analyses are given in Section 6. An extensive simulation study is reported in Section 7, which is followed by some conclusions in Section 8.

## 2 The RSGA Scheme

As discussed in Section 1, permutation representations often make it difficult to design effective and efficient evolutionary operators because of feasibility and memory-efficiency issues. In order to address these issues, an often used methodology is to transform the original problem into a new problem for which solutions are based on value rather than order or permutation, and then apply a basic binary GA to resolve the transformed problem. Properly transforming the original problem is the centerpiece of this methodology. Inspired by the natural ripple spreading phenomenon that occurs when liquid surfaces are perturbed, this paper reports a novel problem transforming process, which has good potential to become a generalized technique for transforming a wide range of combinatorial problems.

A solution to a combinatorial problem determines how to organize a set of elements, particularly, in what kind of order. Suppose such elements are associated with some points in a space where some spreading ripples will be generated randomly (like when a stone is thrown in a pond). Then as the ripples are spreading out in this space, the points representing the components of a solution can be organized according to, say, the order by which the ripples reach each of them (or by some other choice). Obviously, such an order is determined by the values of the ripple-spreading related parameters, for example, the epicenter, speed, and amplitude of ripples. In this way, a combinatorial solution (a sequence of elements) is transformed into a value-based solution (the parameters of a spatial ripple spreading process). This is the original inspiration of remodeling/transforming a combinatorial problem into a ripple spreading process that leads to the proposed RSGA scheme.

The RSGA scheme thus permits the use of very basic binary GAs to solve those combinatorial problems that would usually require sophisticated permutation representations. In order to stress advantages of the model, the basic binary GAs used is

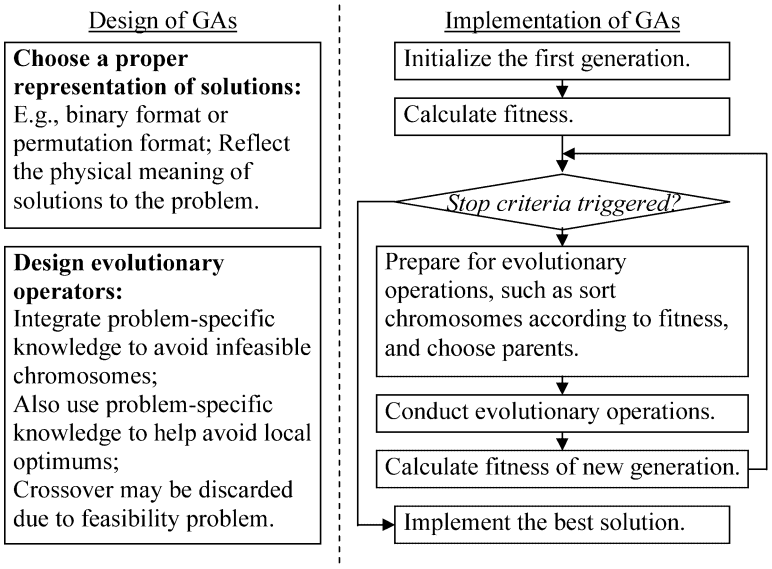
Design of GAs
Choose a proper representation of solutions: E.g., binary format or permutation format; Reflect the physical meaning of solutions to the problem.
Design evolutionary operators: Integrate problem-specific knowledge to avoid infeasible chromosomes; Also use problem-specific knowledge to help avoid local optimums; Crossover may be discarded due to feasibility problem.
Implementation of GAs
Initialize the first generation.
Calculate fitness.
Stop criteria triggered?
Prepare for evolutionary operations, such as sort chromosomes according to fitness, and choose parents.
Conduct evolutionary operations.
Calculate fitness of new generation.
Implement the best solution.

Figure 1: Conventional way of applying GAs.

free of feasibility and memory-efficiency problems and compatible to all classic evolutionary operators. Figure 1 summarizes the conventional way of applying GAs and the central idea of the RSGA scheme is illustrated in Figure 2.

In the implementation of the RSGA, the only thing different from a classic GA is that, before the fitness of a chromosome is calculated, the represented solution to the transformed problem, that is, a set of values for the ripple-spreading related parameters, needs to be mapped back into the associated solution to the original combinatorial problem. Obviously the most important and also the most difficult step in the RSGA scheme is to design a proper ripple-spreading model. This largely depends on each individual original problem. Despite this, there is still great freedom and flexibility for designing a ripple-spreading model. For instance, regarding the design of artificial space, previous studies (Hu and Di Paolo, 2007; Hu, Di Paolo, and Barnett, 2008; Hu and Di Paolo, 2009b; Hu, Leeson, et al., 2010) have shown the following.

- The artificial space may originate from a real space, like the artificial space used in the travelling salesman problem, which is actually the geographic map of cities (Hu and Di Paolo, 2007). Therefore, there is no need to project any elements onto a new space. However, in most problems, it is difficult to refer to any real space to design the artificial space. In this case, we may construct an imaginary space based on the categories/features of the elements composing combinatorial solutions, that is, each axis of the artificial space corresponds to a certain category/feature of elements, if possible following some relation of order. In the study on the airport gate assignment problem, the artificial space has an axis of time, an axis of passenger number, and an axis of aircraft grounding time (Hu and Di Paolo, 2009b). The elements composing combinatorial solutions can then be projected into the artificial space according to their categories/features. Sometimes there is no useful information about the categories/features of elements, such as the nodes in

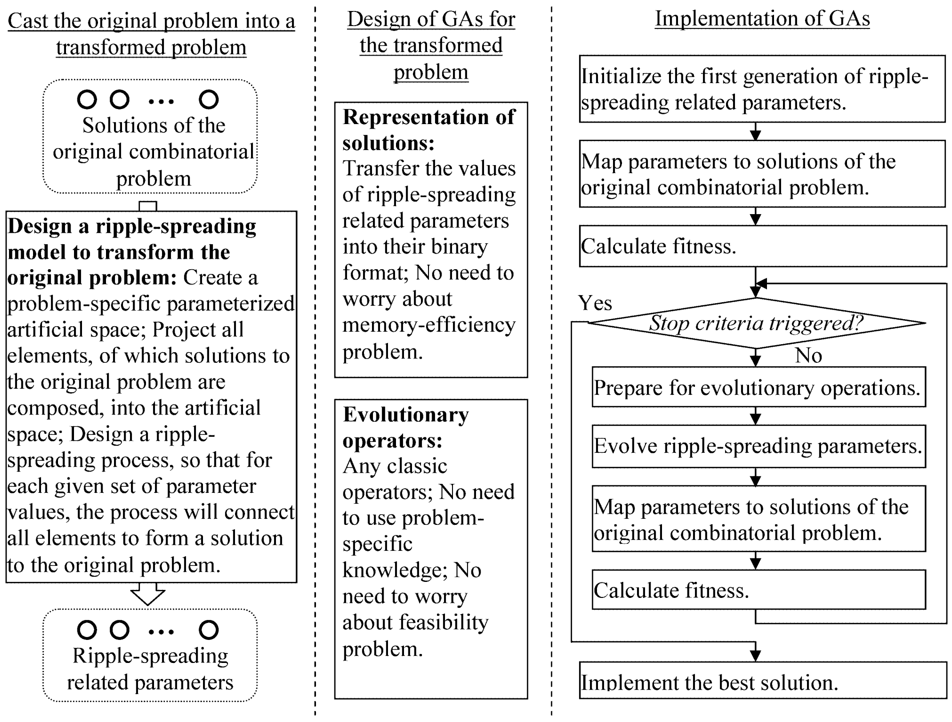
Cast the original problem into a transformed problem
Solutions of the original combinatorial problem
Design a ripple-spreading model to transform the original problem: Create a problem-specific parameterized artificial space; Project all elements, of which solutions to the original problem are composed, into the artificial space; Design a ripple-spreading process, so that for each given set of parameter values, the process will connect all elements to form a solution to the original problem.
Ripple-spreading related parameters
Design of GAs for the transformed problem
Representation of solutions: Transfer the values of ripple-spreading related parameters into their binary format; No need to worry about memory-efficiency problem.
Evolutionary operators: Any classic operators; No need to use problem-specific knowledge; No need to worry about feasibility problem.
Implementation of GAs
Initialize the first generation of ripple-spreading related parameters.
Map parameters to solutions of the original combinatorial problem.
Calculate fitness.
Stop criteria triggered?
Yes
No
Prepare for evolutionary operations.
Evolve ripple-spreading parameters.
Map parameters to solutions of the original combinatorial problem.
Calculate fitness.
Implement the best solution.

Figure 2: Basic idea of the RSGA.

non-geographic networks or graph and information theories (Hu, Di Paolo, and Barnett, 2008). The artificial spaces used in these studies are purely imaginary, and the elements are projected in a rather arbitrary manner.

- The artificial space can in principle be of any dimension, for example, in the existing studies, there are one-dimensional (Hu, Leeson, et al., 2010), two-dimensional (Hu and Di Paolo, 2007; Hu, Di Paolo, and Barnett, 2008), and three-dimensional artificial spaces (Hu and Di Paolo, 2009b). In an artificial space with more than one dimension, different axes may have different distance units, like in Hu and Di Paolo (2009b), the distance units for the time axis and the passenger number axis are completely different and even incommensurable. In this case, we need to introduce an artificial coefficient to homogenize two different distance scales. Such coefficients, along with other ripple-spreading parameters, can be later evolved by the GA. This can help to optimize the ripple-spreading model, in order to improve the performance of RSGA.

Regarding the design of the ripple-spreading process, we can also get some useful observations from existing studies.

- For sequencing problems, such as the traveling salesman problem (Hu and Di Paolo, 2007) and the airport gate assignment problem (Hu and Di Paolo, 2009b), the time by which a ripple reaches an element point is the key factor to determine how to combine all elements. For a combinatorial problem where each element has many

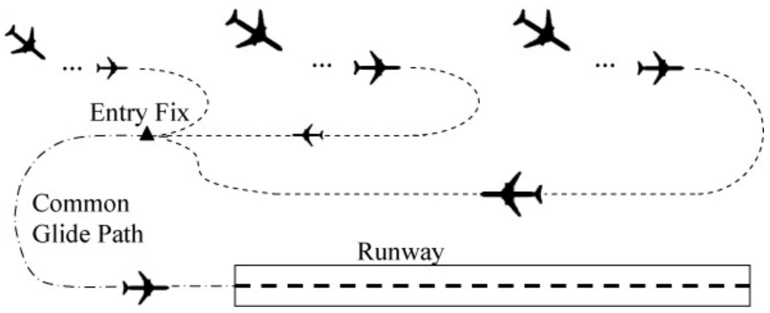

Figure 3: Aircraft waiting to land.

states/features/values to choose from, such as the network coding problem (Hu, Leeson, et al., 2010), the amplitude of a ripple when it reaches an element point plays the most important role in determining how to generate combinatorial solutions. Some other combinatorial problems could be very complicated, where there are complex interactions or relationships between elements in a combinatorial solution, for example, how to distribute connections between nodes to generate a random complex network (Hu, Di Paolo, and Barnett, 2008). In this case, an element point can be activated by a ripple to generate new ripples, which may activate other elements in return.

- The choice for ripple-spreading related parameters is also broad. In the existing studies, the location of ripple epicenter, initial states of a ripple (including initial amplitude, initial point energy, frequency, initial phase, etc.), coefficients in the dynamic functions which define the behavior of ripples, and even some artificial space related parameters were used as ripple-spreading related parameters.

Hu, Wang, et al. (2010) provide a comprehensive review and some useful guidelines regarding how to design an RSGA for a combinatorial problem. The general idea is easier to understand when seen at work on a specific case. More details about how to design a good RSGA will be explained in the following sections through a case study on the ASP.

## 3 The Aircraft Sequencing Problem (ASP)

A simple way to perform an ASP is to schedule arriving aircraft in a first-come-first-served (FCFS) order based on their estimated times of arrival (ETAs) at the runway. Although FCFS scheduling establishes a fair order in terms of ETA, it ignores other useful information which can help make the most of the capacity of the airport, reduce airborne delays and/or improve the airport service to airlines (Pelegrin, 1994; Carr et al., 1999, 2000). As is well known, shifting positions of aircraft according to landing time interval (LTI), which is the minimum permissible time interval between two successive landings, can significantly reduce airborne delays during the arriving and landing phase (Andreatta and Romanin-Jacur, 1987; Bianco and Odoni, 1993; Dear, 1976).

Suppose a set of aircraft need to land at the same runway of an airport during a period of time of interest, as illustrated by Figure 3. Assume that the number of the aircraft under consideration is $N_{AC}$, and the period of time is $T$ seconds long. Then $N_{AC}$

Table 1: Minimum landing time intervals (LTI; Bianco et al., 1997).

| | | Category of following aircraft: $j$ | | | |
|---|---|---|---|---|---|
| $S_{ij}$ (s) | | 1 | 2 | 3 | 4 |
| Category of leading aircraft: $i$ | 1 | 96 | 200 | 181 | 228 |
| | 2 | 72 | 80 | 70 | 110 |
| | 3 | 72 | 100 | 70 | 130 |
| | 4 | 72 | 80 | 70 | 90 |

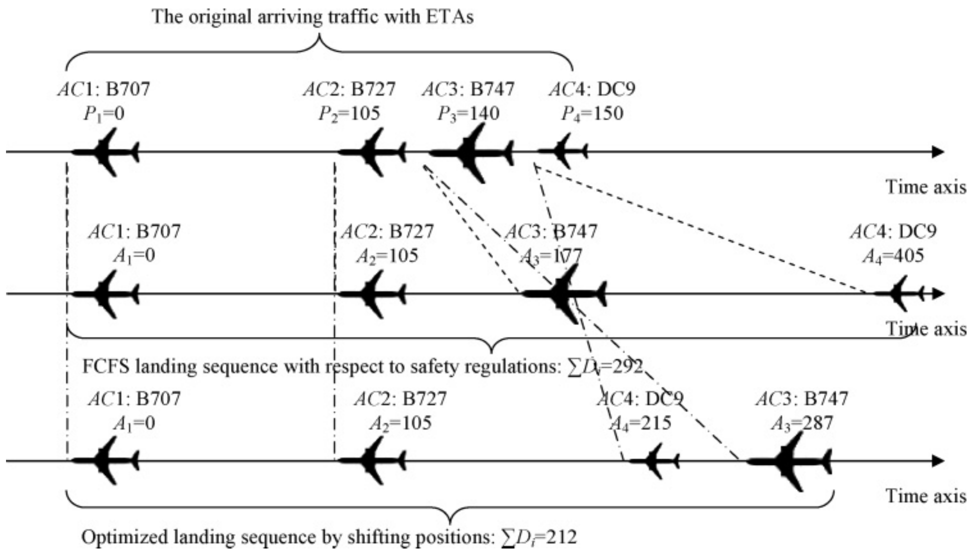

Figure 4: FCFS and position shifting.

and $T$ can be used to estimate the degree of congestion at the runway of the airport. For the $i$th aircraft, $\mathrm{AC}_i$, in the original arriving traffic, $i = 1, \ldots, N_{\mathrm{AC}}$, there is an ETA at the runway, denoted as $P_i$. Based on this set of ETAs, $P_i, i = 1, \ldots, N_{\mathrm{AC}}$, an FCFS landing sequence can be directly worked out with respect to safety regulations.

The safety separation, that is, the minimum LTI between a pair of successive aircraft, is a function of the types and of the relative positions of the two aircraft. There are hundreds of aircraft types in the airspace system, and many different aircraft types may be grouped into a single category regarding minimum LTI. For the sake of simplicity, (as in Bianco et al., 1997), aircraft waiting to land are classified into a relatively small number of distinct categories, according to speed, capacity, weight, and other technical characteristics. Table 1 shows the minimum LTI relative to main categories of commercial aircraft (typical aircraft types in categories 1 to 4 are the B747, B727, B707, and DC9, respectively). It is evident that the LTIs in Table 1 are asymmetric with respect to order in a sequence. For example, a minimum LTI of 200 s is required for a B727 to follow a B747, while a minimum LTI of only 72 s needs to be satisfied for the same pair of aircraft in reverse order. By taking advantage of the asymmetries of the LTIs, in other words, by shifting positions of aircraft in an FCFS landing sequence, it is possible to reduce delays and to improve the capacity of the airport. The potential benefits resulting from position shifting, considering airborne delay, are illustrated by Figure 4 in an intuitive way, where $A_i$, $i = 1, \ldots, N_{\text{AC}}$, is the scheduled time of arrival (STA) for the $i$th aircraft in the original arriving traffic.

The ASP based on position shifting can be mathematically described as a minimization problem. Let $Q(n)$ record the $n$th aircraft in the optimized landing sequence, where $Q(n) = i$ means the $i$th aircraft in the original arriving traffic turns out to be the $n$th aircraft in the optimized landing sequence. With $Q$ one can calculate $A_i$ as

$$A_{Q(n)} = \begin{cases} P_{Q(n)} & n = 1; \\ \max(P_{Q(n)}, A_{Q(n\text{-}1)} + S(C_{Q(n\text{-}1)}, C_{Q(n)})) & n > 1, \end{cases} \tag{1}$$

where $C_i$ is the category of the $i$th aircraft in the original arriving traffic, and $S(i, j)$ is the LTI for an aircraft of category $j$ to follow an aircraft of category $i$ to land. Then the airborne delay of the $i$th aircraft in the original arriving traffic is

$$D_i = A_i - P_i, i = 1, \ldots, N_{\text{AC}}. \tag{2}$$

The objective of the ASP in the period of $T$ is to find an optimal landing sequence, $Q(1), \ldots, Q(N_{\text{AC}})$, such that the total airborne delay can be minimized, as formulated as following

$$\min_{Q(1),\ldots,Q(N_{\text{AC}})} J, J = \sum_{i=1}^{N_{\text{AC}}} D_i. \tag{3}$$

One should be aware of that the above mathematical formulation of the ASP is simplified. The real-world ASP operation needs to consider many more factors and constraints which are missing or ignored here for the sake of simplicity. For instance, arriving aircraft should be delayed as equally as possible in a fair landing sequence. It is not the same to delay five aircraft by 10 min as to delay only one by 50 min. For each aircraft, a maximal allowable delay must be observed due to the limited fuel on board. Also for each aircraft, the STA should be within a landing time window. In the real-world ASP practice, sometimes a negative delay may apply to aircraft, which means that an aircraft may be required to speed up to land before its ETA in order to allow other aircraft to land on time. Different aircraft may have different landing priority in terms of fuel and passenger convenience. Furthermore, how to execute an optimized landing sequence in a dynamic environment (the situation may change during the time window $T$) is also an important issue which needs to be addressed in real world.

Addressing these real-world considerations is not outside the possibilities of the RSGA. For example, as will be explained later, the proposed RSGA naturally stands a higher chance of swapping positions between two closer aircraft, which means large delay to individual aircraft will rarely occur and therefore the delay distribution among aircraft will be reasonably even. As for the landing priority of aircraft, one may apply different weights to the delay of different categories of aircraft. This should not affect the implementation of the RSGA, which would be in these respects a variation of the simplified implementation studied in this paper. And the dynamic issue in the ASP can be resolved by integrating the strategy of receding horizon control into the RSGA, just as the GAs in Hu and Chen (2005) and Hu and Di Paolo (2008) have done. Since the RSGA scheme is the focus of the paper, here we only use the above simplified ASP.

## 4 Transformed Problem: A Ripple-Spreading Model of ASP

The ripple-spreading model is the core technique in the RSGA scheme. In this section, we will describe a ripple-spreading model for the ASP. To this end, arriving aircraft are projected as points onto an especially parameterized artificial space, and then a ripple-spreading process is designed in order to form a landing sequence by connecting all aircraft points according to certain parameters.

First of all, we need to set up an artificial space for projecting arriving aircraft. Here a two-dimensional space is designed, where the $x$ axis is time and the $y$ axis is an aircraft category axis but measured in the same time units as the $x$ axis. Then we project all aircraft as points onto the artificial space according to their ETA and category. Here we define three parameters which can partially determine how aircraft points are distributed in the artificial space: $\delta_{12}$, $\delta_{23}$, and $\delta_{34}$. They are, along the $y$ axis, the distances between category 1 and category 2, between category 2 and category 3, and between category 3 and category 4, respectively. Note that $\delta_{12}$, $\delta_{23}$, and $\delta_{34}$ are real numbers, and may have negative values. Besides the aircraft points, we also have a reference point in the artificial space denoted as RP and defined by its coordinates $(x, y)$. This RP will determine the epicenter of the ripple spreading process. Figure 5(a) and Figure 5(b) illustrate how the original arriving traffic is projected onto the artificial space.

With the above artificial space ready, we now need to design a ripple-spreading process that connects all aircraft points as a function of $\delta_{12}$, $\delta_{23}$, $\delta_{34}$, and RP in order to generate a unique landing sequence. It is desirable that such a process should generate a landing sequence that is absolutely feasible and reasonably good in terms of airborne delay. By mimicking the natural ripple-spreading phenomenon, we propose the following process.

```
Step 1: Calculate the distances between aircraft points to the given RP. Initialize $U$ as
        the set of all original arriving aircraft, set the current landing sequence $Q$ as
        an empty vector, and let $j = 0$.
Step 2: Do while $U \neq \emptyset$
    Step 2.1: Let $j = j + 1$. Choose all aircraft points that have the $j$th shortest
              distance to the RP. Suppose in these chosen aircraft points, $N_E$ aircraft
              have an ETA smaller than $x$, and $N_L$ aircraft have an ETA larger than
              $x$. Remove these aircraft from $U$.
    Step 2.2: Insert the $N_E$ aircraft that have an ETA smaller than $x$ to the front
              of the current landing sequence $Q$ in the following way. An aircraft
              with a larger ETA should be inserted earlier to the front of the current
              $Q$. If some aircraft have the same ETA, then they should be inserted
              in a specified order which is predetermined by aircraft categories in
              order to give the least total delay for those aircraft with the same
              ETA. If some aircraft have the same ETA as well as the same category,
              then insert them randomly to the front of the current $Q$.
    Step 2.3: Append the $N_L$ aircraft that have an ETA larger than $x$ to the end
              of the current $Q$ in the following way. An aircraft with a smaller
              ETA should earlier be appended to the end of the current Q. If some
              aircraft have the same ETA, then they should be appended according
              to the reverse version of the specified order used in Step 2.2. If some
```

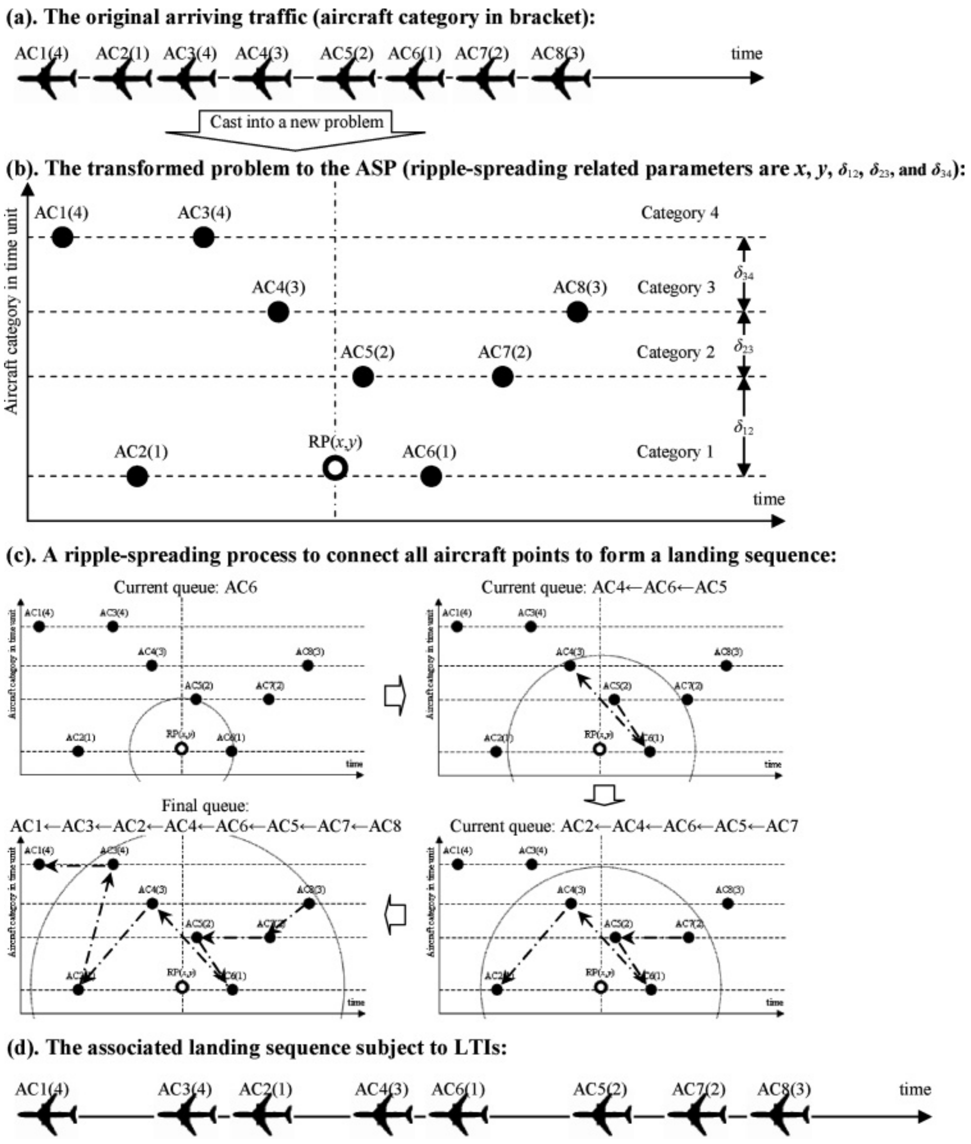
(a). The original arriving traffic (aircraft category in bracket):
AC1(4) AC2(1) AC3(4) AC4(3) AC5(2) AC6(1) AC7(2) AC8(3)
time
Cast into a new problem
(b). The transformed problem to the ASP (ripple-spreading related parameters are x, y, δ12, δ23, and δ34):
Aircraft category in time unit
Category 4
Category 3
Category 2
Category 1
RP(x,y)
(c). A ripple-spreading process to connect all aircraft points to form a landing sequence:
Current queue: AC6
Current queue: AC4←AC6←AC5
Final queue: AC1←AC3←AC2←AC4←AC6←AC5←AC7←AC8
Current queue: AC2←AC4←AC6←AC5←AC7
(d). The associated landing sequence subject to LTIs:
AC1(4) AC3(4) AC2(1) AC4(3) AC6(1) AC5(2) AC7(2) AC8(3)

Figure 5: Illustration of the transformed problem and ripple-spreading process.

aircraft have the same ETA as well as the same category, then append them randomly to the end of the current $Q$.

The above process can be roughly likened to throwing a stone into a pool where there stand $N_{\mathrm{AC}}$ stakes, that is, aircraft points in the artificial space. When the ripple spreads out from the point where the stone hits the pool, that is, the RP, it reaches every stake sooner or later according to the distance from each stake to the hit point. Based on the order in which the ripple reaches each stake, plus referring to the ETA and the aircraft category, we can work out a landing sequence to connect all $N_{\mathrm{AC}}$ aircraft.

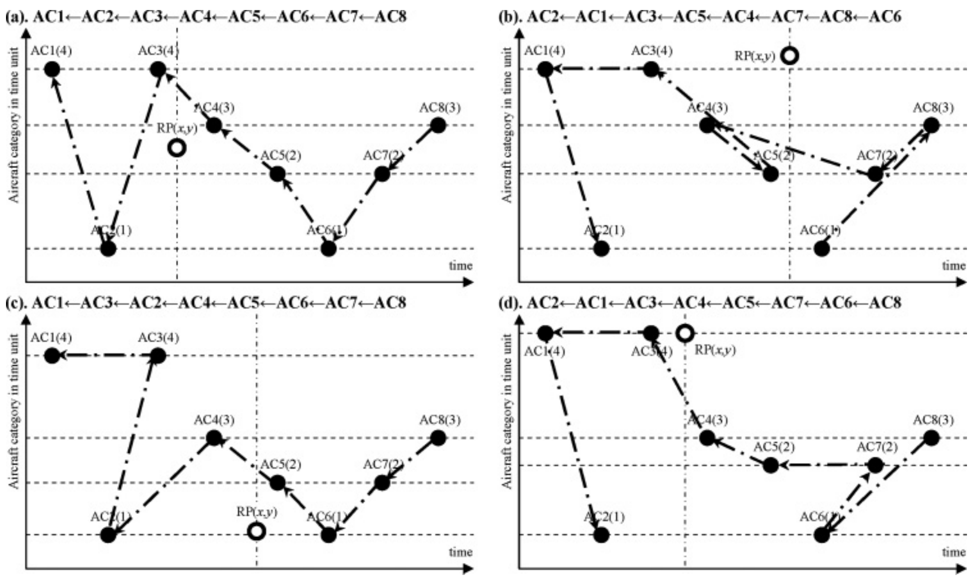
(a). AC1←AC2←AC3←AC4←AC5←AC6←AC7←AC8
(b). AC2←AC1←AC3←AC5←AC4←AC7←AC8←AC6
(c). AC1←AC3←AC2←AC4←AC5←AC6←AC7←AC8
(d). AC2←AC1←AC3←AC4←AC5←AC7←AC6←AC8
AC1(4)
AC2(1)
AC3(4)
AC4(3)
AC5(2)
AC6(1)
AC7(2)
AC8(3)
RP(x,y)
Aircraft category in time unit
time

Figure 6: Effect of ripple-spreading parameters on landing sequence.

Figure 5(c) and Figure 5(d) illustrate how the above ripple-spreading process calculates a landing sequence step by step.

It is clear that different $\delta_{12}$, $\delta_{23}$, $\delta_{34}$, and/or RP may result in different landing sequences. For example, Figure 6(a) and Figure 6(b) use the same set of $\delta_{12}$, $\delta_{23}$, $\delta_{34}$ as Figure 5(b), but they have different RPs, and therefore different landing sequences. Figure 6(c) has the same RP as Figure 5(b), but uses a different set of $\delta_{12}$, $\delta_{23}$, and $\delta_{34}$, which also results in a different landing sequence. The $\delta_{12}$, $\delta_{23}$, $\delta_{34}$, and RP in Figure 6(d) are different from those in Figure 5(b), and so is the landing sequence. Therefore, $\delta_{12}$, $\delta_{23}$, $\delta_{34}$, and RP are all ripple-spreading parameters in this model. An interesting observation is that if $\delta_{12} = \delta_{23} = \delta_{34} = 0$, then the resulting landing sequence is actually a FCFS sequence, regardless of the location of RP.

One may argue that the output of the above ripple-spreading model might not cover all possible combinations of arriving aircraft. This is true. However, it is worth noting that many possible combinations are actually bad landing sequences in terms of airborne delay. For instance, a landing sequence which swaps the first aircraft and the last aircraft in the original arriving traffic will impose a heavy delay on the first aircraft of the original arriving traffic. Therefore, in reality, air traffic controllers only shift the positions of those aircraft which have similar ETAs. Another example is that the total airborne delay will definitely increase if two aircraft of the same category swap positions. As will be proved in Section 6, the proposed ripple-spreading process mainly causes position-shifting between those aircraft of different categories and close to each other in the original arriving traffic. In other words, the ripple-spreading model helps to filter out many bad landing sequences, which is what is expected for a good transformed problem. Actually, many GAs use problem-specific knowledge and heuristic rules to keep chromosomes away from bad solutions, which, in the RSGA, is largely achieved by the ripple-spreading model. Furthermore, we can integrate into the model as many problem-specific knowledge and heuristic rules as we like, without causing any problem to the design of evolutionary operators in the RSGA. This is, however, often difficult for permutation-representation-based GAs for the ASP, which directly integrate problem-specific knowledge and/or heuristic rules into their evolutionary operators.

It should be noted that the design of the ripple-spreading model is highly problem-dependent. Different problems may need quite different ripple-spreading models (e.g., see Hu and Di Paolo, 2007, 2009b; Hu, Di Paolo, and Barnett, 2008). Even for the same problem, there may be many different suitable ripple-spreading models, probably with equivalent or similar effects. For instance, the ripple-spreading model described in this section can be modified by putting the RP on the $x$ axis and defining $\delta_1$, $\delta_2$, $\delta_3$, and $\delta_4$ as the locations of category lines with reference to the $x$ axis. One can easily derive that the modified model is actually equivalent to the first model.

## 5 GA Related Techniques

Based on the above ripple-spreading model for the ASP, a traditional GA with the basic binary representation and all classic evolutionary operators can be designed to evolve the ripple-spreading parameters in order to find an optimal landing sequence.

### 5.1 Chromosome Structures

In the RSGA, a chromosome is simply a binary string for the values of $\delta_{12}$, $\delta_{23}$, and $\delta_{34}$, and the coordinates of RP, that is, $x$ and $y$. For comparative purposes, here we also discuss two permutation representations for the ASP: one is an $N_{\mathrm{AC}}$ long vector recording the absolute positions of all aircraft in the represented landing sequence, that is, $g(i) = j$ means the $j$th aircraft in the original arrival traffic is the $i$th to land, as illustrated in Figure 7(c) (Hansen, 2004; Cheng et al., 1999; Hu and Chen, 2005). The other, as shown in Figure 7(d), is a 0-1-valued matrix in a size of $N_{\mathrm{AC}} \times N_{\mathrm{AC}}$, with each entry $g(i, j)$ indicating the relative positions of aircraft in the associated landing sequence (Hu and Di Paolo, 2008), that is, $g(i, j) = 1$ means the $j$th aircraft in the original arrival traffic will follow the $i$th aircraft in the original arrival traffic to land. The contents of a chromosome in the RSGA have nothing directly to do with the associated landing sequence, but are simply the ripple-spreading parameters, that is, a chromosome in the RSGA is a binary string of the values for the ripple-spreading parameters, as depicted in Figure 7(f), where

$$L_X = ceil(\log_2((\overline{X} - \underline{X})/X_S)), \tag{4}$$

$$L_Y = ceil(\log_2((\overline{Y} - \underline{Y})/Y_S)), \tag{5}$$

$$L_{\delta_{12}} = ceil(\log_2((\overline{\delta} - \underline{\delta})/\delta_S)), \tag{6}$$

$$L_{\delta_{23}} = ceil(\log_2((\overline{\delta} - \underline{\delta})/\delta_S)), \tag{7}$$

$$L_{\delta_{34}} = ceil(\log_2((\overline{\delta} - \underline{\delta})/\delta_S)), \tag{8}$$

and $ceil$ is a function which rounds a number to the nearest integer towards infinity, and $\overline{X}/\overline{Y}/\overline{\delta}$, $\underline{X}/\underline{Y}/\underline{\delta}$, and $X_S/Y_S/\delta_S$ are the upper bound, the lower bound, and the searching step of $x/y/\delta$, respectively. In our study, $\overline{X}/\overline{Y}$ and $\underline{X}/\underline{Y}$ are set such that all aircraft points are covered with a reasonable margin. The range for $\delta_{12}$, $\delta_{23}$, and $\delta_{34}$ is set to be twice the largest LTI in Table 1, that is, $\overline{\delta} = 456$ and $\underline{\delta} = 0$. As in many other basic binary GAs, the

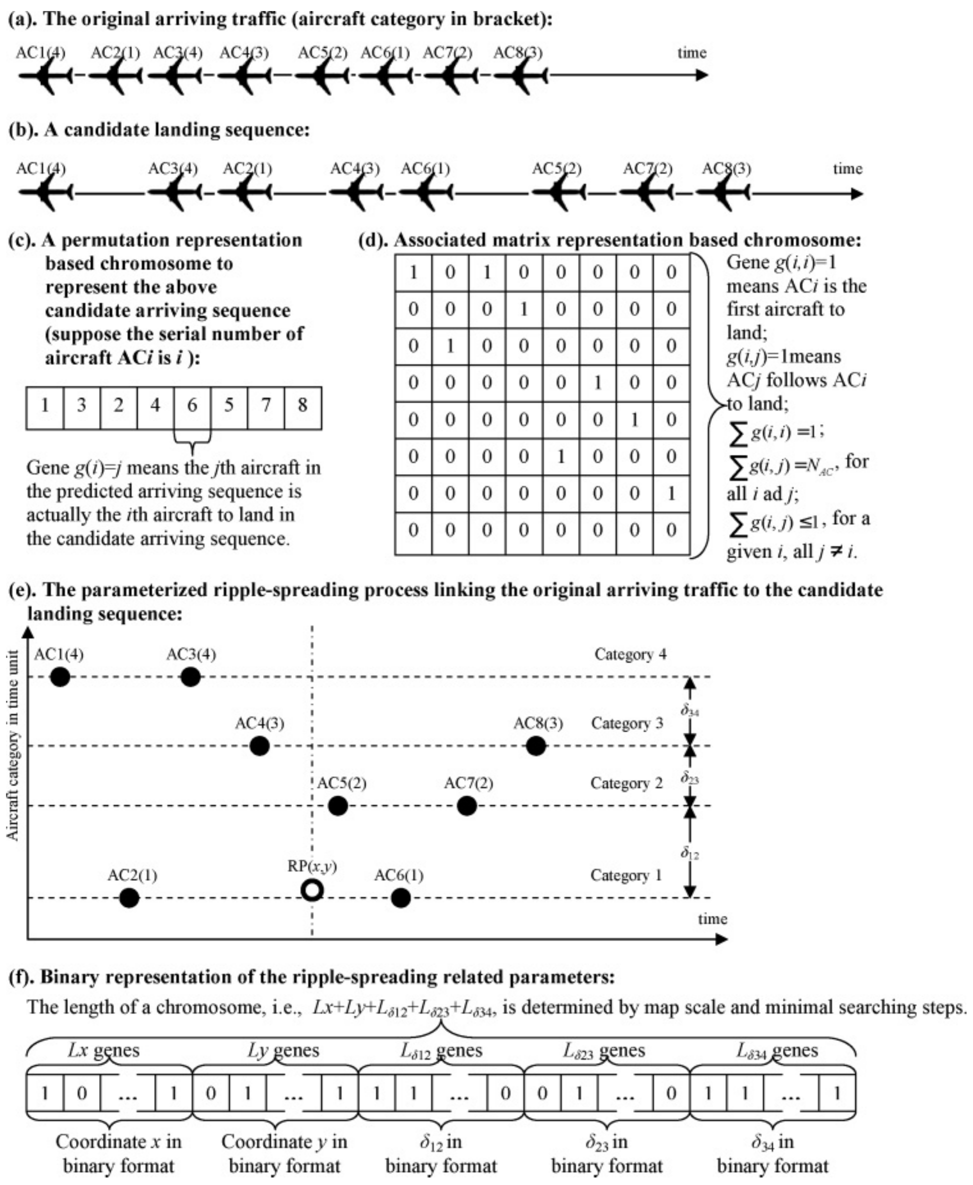
(a). The original arriving traffic (aircraft category in bracket):
AC1(4) AC2(1) AC3(4) AC4(3) AC5(2) AC6(1) AC7(2) AC8(3) time
(b). A candidate landing sequence:
AC1(4) AC3(4) AC2(1) AC4(3) AC6(1) AC5(2) AC7(2) AC8(3) time
(c). A permutation representation based chromosome to represent the above candidate arriving sequence (suppose the serial number of aircraft ACi is i ):
1 3 2 4 6 5 7 8
Gene g(i)=j means the jth aircraft in the predicted arriving sequence is actually the ith aircraft to land in the candidate arriving sequence.
(d). Associated matrix representation based chromosome:
1 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0
Gene g(i,i)=1 means ACi is the first aircraft to land;
g(i,j)=1means ACj follows ACi to land;
∑g(i,i) =1;
∑g(i,j) =N_AC, for all i ad j;
∑g(i,j) ≤1, for a given i, all j ≠ i.
(e). The parameterized ripple-spreading process linking the original arriving traffic to the candidate landing sequence:
Aircraft category in time unit
AC1(4) AC3(4) Category 4
AC4(3) AC8(3) Category 3
AC5(2) AC7(2) Category 2
AC2(1) RP(x,y) AC6(1) Category 1
δ34 δ23 δ12
time
(f). Binary representation of the ripple-spreading related parameters:
The length of a chromosome, i.e., Lx+Ly+Lδ12+Lδ23+Lδ34, is determined by map scale and minimal searching steps.
Lx genes Ly genes Lδ12 genes Lδ23 genes Lδ34 genes
1 0 ... 1 0 1 ... 1 1 1 ... 0 0 1 ... 0 1 1 ... 1
Coordinate x in binary format
Coordinate y in binary format
δ12 in binary format
δ23 in binary format
δ34 in binary format

Figure 7: Different representations.

searching steps can affect the performance of the RSGA. In general, the searching steps should become smaller if the problem is more complicated, for example, in the case of the ASP, if $N_{\mathrm{AC}}$ is larger. In this study, we set $X_S = Y_S = \delta_S = 0.00001$ unless otherwise specified.

The above three chromosome structures are further compared in Table 2, from which one can easily see two advantages of our binary representation. The first advantage is that the binary representation has no feasibility problems because no constraints are imposed on chromosomes. In contrast, both the vector representation and

Table 2: Features of different representations.

| | Meaning of a gene | Meaning of a chromosome | Size of a chromosome | Constraints for evolutionary operators |
|---|---|---|---|---|
| Vector representation | $g(i) = j$ means AC$j$ is the $i$th aircraft to land | Absolute positions of aircraft in a landing sequence | $N_{\mathrm{AC}} \times ceil(\log_2(N_{\mathrm{AC}}))$ bits | $g(i) \in \{1, \ldots, N_{\mathrm{AC}}\}$, $g(i) \neq g(j)$, if $i \neq j$ |
| Matrix representation | $g(i, j) = 1$ means AC$j$ follows AC$i$ to land | Relative positions of aircraft in a landing sequence | $(N_{\mathrm{AC}})^2$ bits | $\sum g(i, i) = 1$, all $i$ $\sum g(:, :) = N_{\mathrm{AC}}$, $\sum g(i, j) \leq 1$, all $j \neq i$ |
| Binary representation | A digital bit in the binary ripple-spreading parameters | The coordinates of RP, $\delta_{12}$, $\delta_{23}$, and $\delta_{34}$ | $(L_X + L_Y + L_{\delta_{12}} + L_{\delta_{23}} + L_{\delta_{34}})$ bits | No constraints |

the matrix representation have to satisfy some special constraints in order to get feasible chromosomes. Actually, the constraints for the vector representation are so restrictive to evolutionary operations that crossover has been discarded in some papers (Cheng et al., 1999; Hu and Chen, 2005). In Hu and Di Paolo (2008), a computationally expensive process had to be added to guarantee the feasibility of crossover. However, the matrix representation used in this case has a potential $O(n^2)$ memory problem. Therefore, the second advantage of the binary representation is its memory efficiency. That is, the RSGA, with similar demands for memory capacity, can easily apply to all problem scales, because the length of its chromosomes has no direct link to $N_{\mathrm{AC}}$. Actually, the RSGA uses the ripple-spreading process in Section 4 to reconstruct online the full information of a landing sequence from the simple data stored in a chromosome. In other words, the ripple-spreading process helps the RSGA to save the memory allocated to chromosomes. The cost is the computational burden caused by the ripple-spreading process. Fortunately, this added computational burden can be largely offset by the reduced computational burden in the evolutionary operations due to the use of binary representation, as will be discussed later. One may argue that the memory efficiency of RSGA is not significant in the ASP, as $N_{\mathrm{AC}}$ will rarely exceed 100. The reason for mentioning the memory efficiency of RSGA here is to note the potential of RSGA. As mentioned before, the RSGA scheme can be applied to many different combinatorial problems, and in some of them, the memory efficiency of RSGA is extremely important. For instance, to represent and optimize a complex network with millions of nodes, a traditional chromosome based on an adjacency matrix requires trillions of bits of memory, while the RSGA only needs to record tens of ripple-spreading related parameters (Hu, Di Paolo, and Barnett, 2008).

## 5.2 Common Genes

There may be two definitions for common genes in GAs for the ASP: one by absolute positions of aircraft, and the other by relative positions of aircraft, as illustrated in Figure 8(a) and Figure 8(b). As mentioned in Section 3, the cause of the ASP is the asymmetries of the LTIs between aircraft. The relative positions of aircraft are therefore more important to determine a good landing sequence. For the vector representation, one can easily identify the common genes defined by absolute positions of aircraft, but not the common genes by relative positions of aircraft unless some computationally expensive

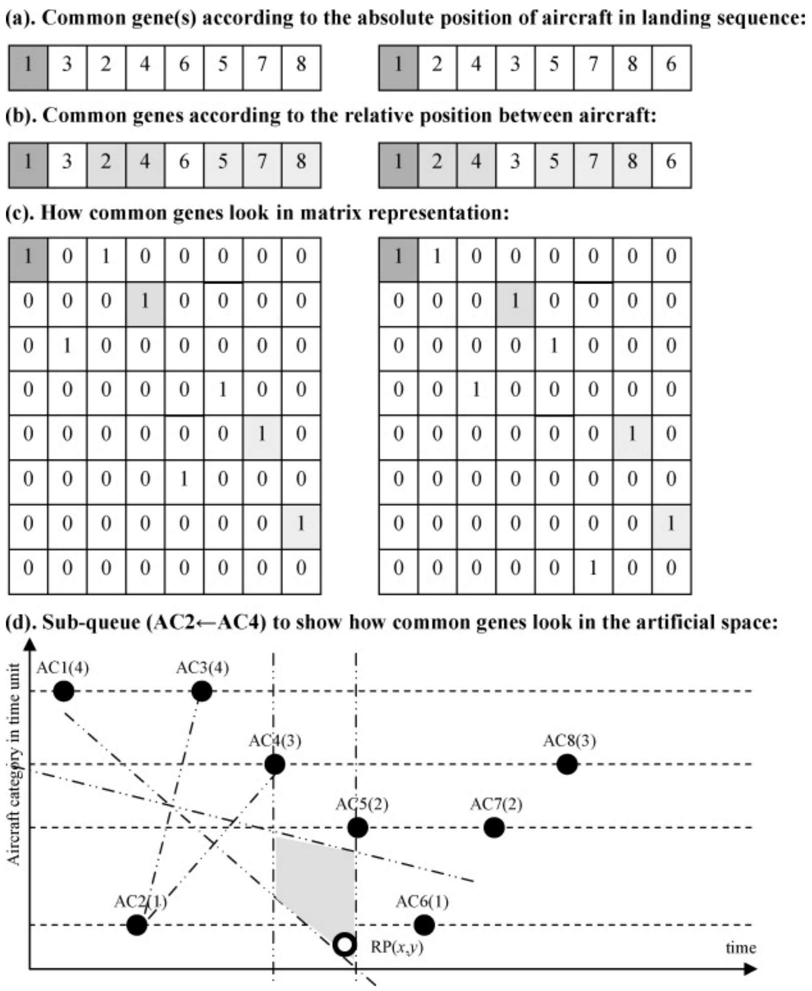
(a). Common gene(s) according to the absolute position of aircraft in landing sequence:
1 3 2 4 6 5 7 8
1 2 4 3 5 7 8 6
(b). Common genes according to the relative position between aircraft:
1 3 2 4 6 5 7 8
1 2 4 3 5 7 8 6
(c). How common genes look in matrix representation:
(d). Sub-queue (AC2←AC4) to show how common genes look in the artificial space:
Aircraft category in time unit
AC1(4)
AC3(4)
AC4(3)
AC8(3)
AC5(2)
AC7(2)
AC2(1)
AC6(1)
RP(x,y)
time

Figure 8: Common genes in different representations.

method is employed. For the matrix representation, identifying common genes based on the relative positions of aircraft is straightforward: conduct an & operation between two matrixes, and then common genes are those resulting entries with value 1 (Hu and Di Paolo, 2008).

For the binary representation in the RSGA, common genes are defined simply based on the values for the ripple-spreading parameters. Surprisingly, similar values for the ripple-spreading parameters often relate to the same relative positions of aircraft, as illustrated in Figure 8(d), where any RP within the dotted area in the parameterized artificial space will cause AC2 to be followed by AC4. Note that Figure 8(d) is only for illustration purposes, and there may be some other areas which can result in the same relative positions of aircraft. Therefore, any evolutionary operation which can identify, inherit, and protect these value-based common genes can also, in some sense, identify, inherit, and protect those common genes based on relations of relative position.

## 5.3 Evolutionary Operators

For the vector representation in Hu and Chen (2005), the design of mutation is very simple: exchange the contents of two genes which are randomly chosen, that is, $g(i) \leftrightarrow g(j)$. This mutation reflects straightforwardly the physical meanings of position-shifting. However, due to potential feasibility problems, the GA in Hu and Chen (2005) failed to give any crossover operator that could effectively identify, inherit, and protect common genes based on relations of relative position. For the matrix representation in Hu and Di Paolo (2008), a relatively complicated procedure was introduced to adopt the mutation in Hu and Chen (2005), and an even more complicated procedure was proposed to conduct uniform crossover aiming to make the most of fit common genes based on relative positions of aircraft.

In the RSGA, since a binary chromosome is based on value rather than on permutation, all classic evolutionary operators can easily apply without any modification. As discussed before, this full freedom of choosing and designing evolutionary operators mainly comes from no constraint to the binary representation. The mutation operator used in the RSGA is to randomly reverse a gene in the chromosome: if gene $C(i)$ is chosen to mutate at a given probability, then we have

$$C(i) \rightarrow |1 - C(i)|. \tag{9}$$

In the RSGA, we choose uniform crossover, which is probably the most powerful crossover operator (Sywerda, 1989). Thanks to the binary representation, the original uniform crossover can directly apply in the RSGA. Uniform crossover uses two parents to produce only one offspring, and the principle is simple: the $i$th gene in the offspring inherits the $i$th gene of either parent with a 50% probability.

## 5.4 Heuristic Rules

In GAs based on a permutation representation for the ASP many problem-specific heuristic rules are integrated into the evolutionary operations. For example, in Hu and Chen (2005) and Hu and Di Paolo (2008), the FCFS sequence is used to initialize a certain proportion of chromosomes, otherwise it is very likely that almost all chromosomes at the beginning of the evolution are very unfit. The probability of applying the mutation operation largely depends on the ETA of the chosen genes, in order to avoid shifting the positions of two aircraft which are far away from each other in the original arriving traffic.

In the evolutionary operations of the RSGA, we mainly focus on some pure GA-related heuristic rules. These are, for example, to evenly distribute a certain proportion of chromosomes within the parameterized artificial space when initializing a generation; to adjust mutation probability and crossover probability online according to the fitness of each individual chromosome as well as the overall fitness level of the current generation of chromosomes (Zhang et al., 2007); to dynamically restrict the mutation operation to a certain part of a chromosome based on its fitness (Hu, Di Paolo, and Wu, 2008). The problem-specific knowledge should have already been taken into account in the design of the ripple-spreading model as described in Section 4, and here we just need to initialize a few chromosomes according to the FCFS principle, that is, we set $\delta_{12} = \delta_{23} = \delta_{34} = 0$.

## 6 Further Analyses on the RSGA

### 6.1 Completeness of Solution Space

An obvious question about the RSGA is: will the ripple-spreading model cover the whole solution space of the original ASP? In other words, for any candidate landing sequence, will there always exist at least one set of values for the ripple-spreading related parameters, such that the output of the ripple-spreading model is the candidate landing sequence?

It is indeed possible to design an RSGA that can in theory cover the whole solution space of the original ASP. Suppose we randomly distribute $N_{\mathrm{AC}}$ RPs (not just one RP) in the artificial space; each RP has a serial number and will generate its own ripple. The first aircraft point which is reached by a ripple will be assigned to the RP where the ripple originates. The aircraft assigned to the $i$th RP will land as the $i$th aircraft in the landing sequence. Then, we encode the coordinates of all RPs in a chromosome, and evolve them by selection, mutation, and crossover. It is easy to see that, by randomly distributing the $N_{\mathrm{AC}}$ RPs, we can get any candidate landing sequence, which means the solution space of the original ASP is completely covered by this RSGA, in other words, it is theoretically possible for this GA to find the global optima. This GA, which can guarantee the completeness of solution space, still follows the basic idea of the RSGA scheme proposed in this paper: it transforms the original order-based ASP solutions into value-based solutions, that is, the coordinates of RPs, which can be easily handled by all classic evolutionary operators free of feasibility problems. Figure 9 illustrates this RSGA with a guarantee of the completeness of solution space.

However, from a practical point of view, it is not necessary to demand GAs to guarantee the completeness of solution space. It is often acceptable that they find good solutions effectively and efficiently. This is because (i) even though a GA can guarantee such completeness, it only theoretically stands a very slim chance to find a global optimum, while practically, it very often ends up with suboptimal or just fairly good solutions, particularly in the case of complex problems, and (ii) it is often the case in various implementations of GAs that finding a global optimum is not essential as long as the algorithms are capable of finding a suboptimal or fairly good solution within a specified time window with limited hardware resources.

Therefore, from a practical point of view, we need to further analyze whether the RSGA described in Section 4 and Section 5 can cover a sufficiently large set of good solutions and find some of them very quickly. As mentioned in Section 4 and Section 5, in real-world ASP use, empirical knowledge informs decisions: (i) two aircraft of same category should not be swapped, (ii) shifting the positions of two aircraft with close ETAs is likely to improve the landing sequence, and (iii) it is rarely of any help to shift the positions of two aircraft which are far away from each other. A candidate landing sequence observing the above knowledge is likely to be a good solution to the ASP. Therefore, conventional GAs based on permutation representations (i.e., Beasley et al., 2001; Hansen, 2004; Cheng et al., 1999; Hu and Chen, 2005; Hu and Di Paolo, 2008, 2009a) need to explicitly integrate the above empirical knowledge into their evolutionary operations in order to achieve a satisfactory performance. For the proposed RSGA, we can theoretically prove as follows that the above knowledge is automatically or naturally reflected in the ripple-spreading model.

For sake of simplicity (but without losing generality), suppose there are two aircraft points in the artificial space as shown in Figure 10, where their coordinates are $(x_{\mathrm{AC1}}, y_{\mathrm{AC1}})$ and $(x_{\mathrm{AC2}}, y_{\mathrm{AC2}})$, respectively, $x_{\mathrm{AC1}} < x_{\mathrm{AC2}}$ and $y_{\mathrm{AC1}} \geq y_{\mathrm{AC2}}$. Let the RP be a

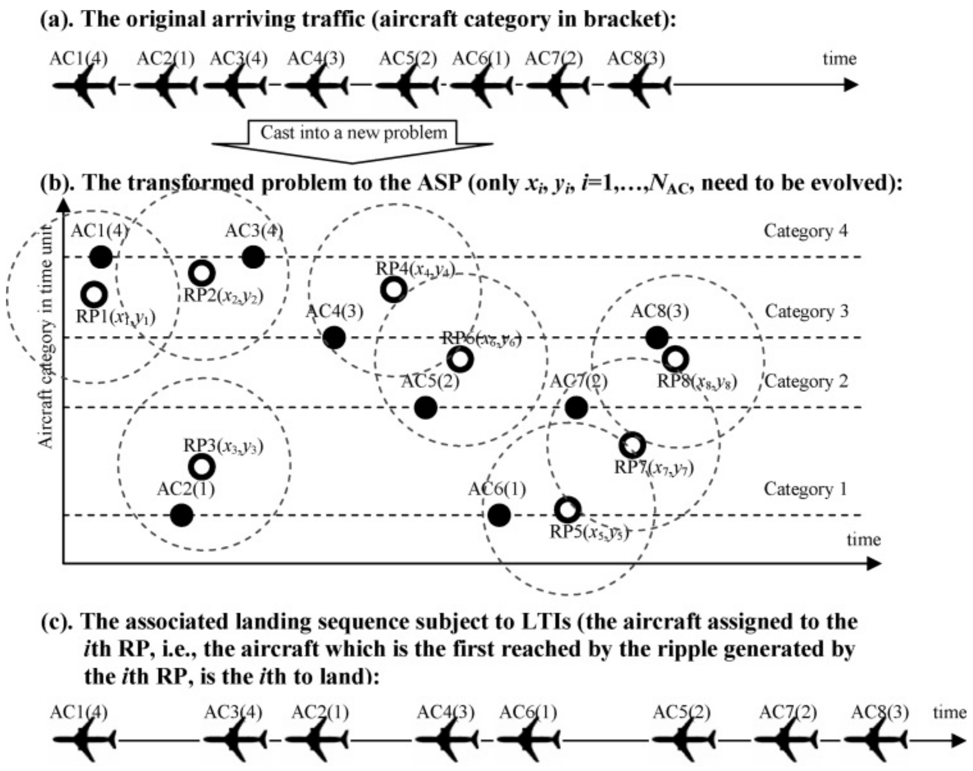
(a). The original arriving traffic (aircraft category in bracket):
AC1(4) AC2(1) AC3(4) AC4(3) AC5(2) AC6(1) AC7(2) AC8(3)
time
Cast into a new problem
(b). The transformed problem to the ASP (only $x_i$, $y_i$, $i$=1,…,$N_{AC}$, need to be evolved):
Aircraft category in time unit
AC1(4)
AC3(4)
RP1($x_1$,$y_1$)
RP2($x_2$,$y_2$)
RP3($x_3$,$y_3$)
AC2(1)
RP4($x_4$,$y_4$)
AC4(3)
RP6($x_6$,$y_6$)
AC5(2)
AC6(1)
RP5($x_5$,$y_5$)
AC7(2)
RP7($x_7$,$y_7$)
AC8(3)
RP8($x_8$,$y_8$)
Category 4
Category 3
Category 2
Category 1
time
(c). The associated landing sequence subject to LTIs (the aircraft assigned to the $i$th RP, i.e., the aircraft which is the first reached by the ripple generated by the $i$th RP, is the $i$th to land):
AC1(4) AC3(4) AC2(1) AC4(3) AC6(1) AC5(2) AC7(2) AC8(3) time

Figure 9: Illustration of an RSGA with guarantee of completeness of the solution space.

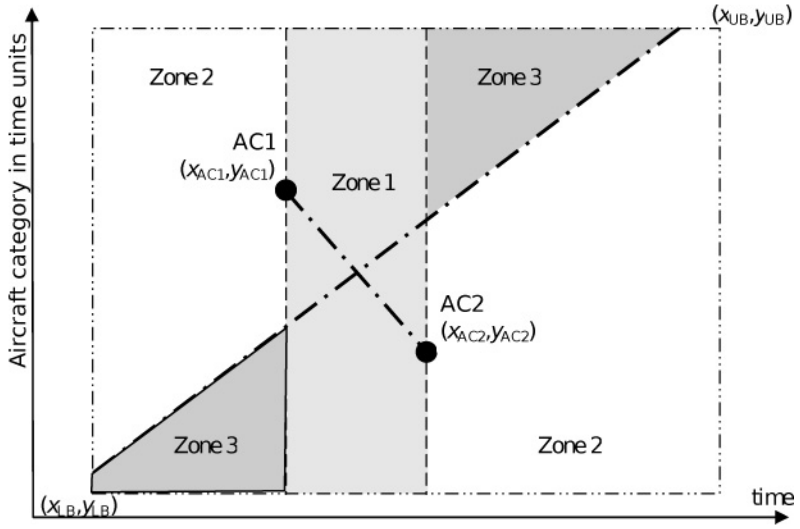

Figure 10: Likelihood of two aircraft being swapped. Only when the RP is located within Zone 3 will two aircraft shift positions.

random point within the rectangular area defined by $(x_{\mathrm{LB}}, y_{\mathrm{LB}})$ and $(x_{\mathrm{UB}}, y_{\mathrm{UB}})$. According to the ripple-spreading model given in Section 4, one has that, only when the RP is located within Zone 3 will AC1 and AC2 shift positions. Therefore, the probability of swapping two aircraft is determined by the probability at which the RP is located within Zone 3. This can be calculated by dividing the area of Zone 3 by the area of the rectangle defined by $(x_{\mathrm{LB}}, y_{\mathrm{LB}})$ and $(x_{\mathrm{UB}}, y_{\mathrm{UB}})$. Due to the limited space, here we skip the detailed mathematical deduction, but from Figure 10, one can easily see: (i) if $y_{\mathrm{AC1}} = y_{\mathrm{AC2}}$, which means two aircraft are of the same category, the volume of Zone 3 is always 0, which implies the probability of shifting positions is 0, while if $y_{\mathrm{AC1}} \neq y_{\mathrm{AC2}}$, then (ii) as the gap between $x_{\mathrm{AC1}}$ and $x_{\mathrm{AC2}}$ decreases, the dot-and-dash lines in Figure 10 will turn clockwise, which leads to an increase in the area of Zone 3 and therefore a larger probability of swapping two aircraft, and (iii) if $(x_{\mathrm{AC2}} - x_{\mathrm{AC1}})$ is larger than a critical value, Zone 3 will disappear, which means no chance to shift positions. Now, we can come to the conclusion that the RSGA proposed in Section 4 and Section 5 naturally observes the empirical knowledge learned in real-world ASP practice. As a result, although the simple RSGA cannot guarantee the completeness of solution space, it is very likely to cover a considerable proportion of good solutions.

### 6.2 Extension to a Multi-Runway System

The RSGA proposed above can be easily extended to handle some more complex situations, for example, the ASP in a multi-runway system (denoted as MRASP). Simply speaking, the MRASP is the problem of assigning arriving aircraft to different runways at the airport and generating efficient landing sequences and landing times so that the safety separation between arriving aircraft is guaranteed, the available capacity at the airport is efficiently used, and airborne delays are minimized. The complexity in the MRASP comes not just from the fact that there are $N_{\mathrm{R}}$ runways to choose from, but also from the fact that the $N_R$ runways may have quite different operational conditions, for example, they start service at different times, and may be exclusive to certain categories of aircraft. The mathematical description of the ASP in Section 3 can easily be extended to a MRASP version. However, due to limited space, here we skip the MRASP formulation (see Hansen, 2004, for details) and mainly focus on how to design a RSGA for the MRASP. It is easy to extend the RSGA for the ASP to a multi-runway version: (i) use $N_{\mathrm{R}}$ RPs rather than just one, and assign a runway to each RP, for example, the $r$th RP is related to runway $r$; (ii) associate two constants with each RP, one is $t_{\mathrm{SS}}(r)$, which is the time for runway $r$ to start service, and the other is a constant vector $V(r)$, which defines the restricted aircraft categories to runway $r$; (iii) in the ripple-spreading process, the $N_{\mathrm{R}}$ RPs compete with each other to connect, subject to $t_{\mathrm{SS}}(r)$ and $V(r)$, those aircraft points which are not connected yet, in order to generate landing sequences to different runways (those aircraft connected by RP $r$ will form the landing sequence to runway $r$). The above MRASP version of RSGA is illustrated in Figure 11, from which one can see that the modified RSGA can effectively arrange arriving aircraft to land at different runways subject to runway operational conditions.

## 7 Simulation Results

### 7.1 Simulation Analysis on the Completeness of the Solution Space

As discussed in Section 6.1, the completeness of solution space is an important issue in the study of the RSGA proposed in this paper. Here we attempt to further address this

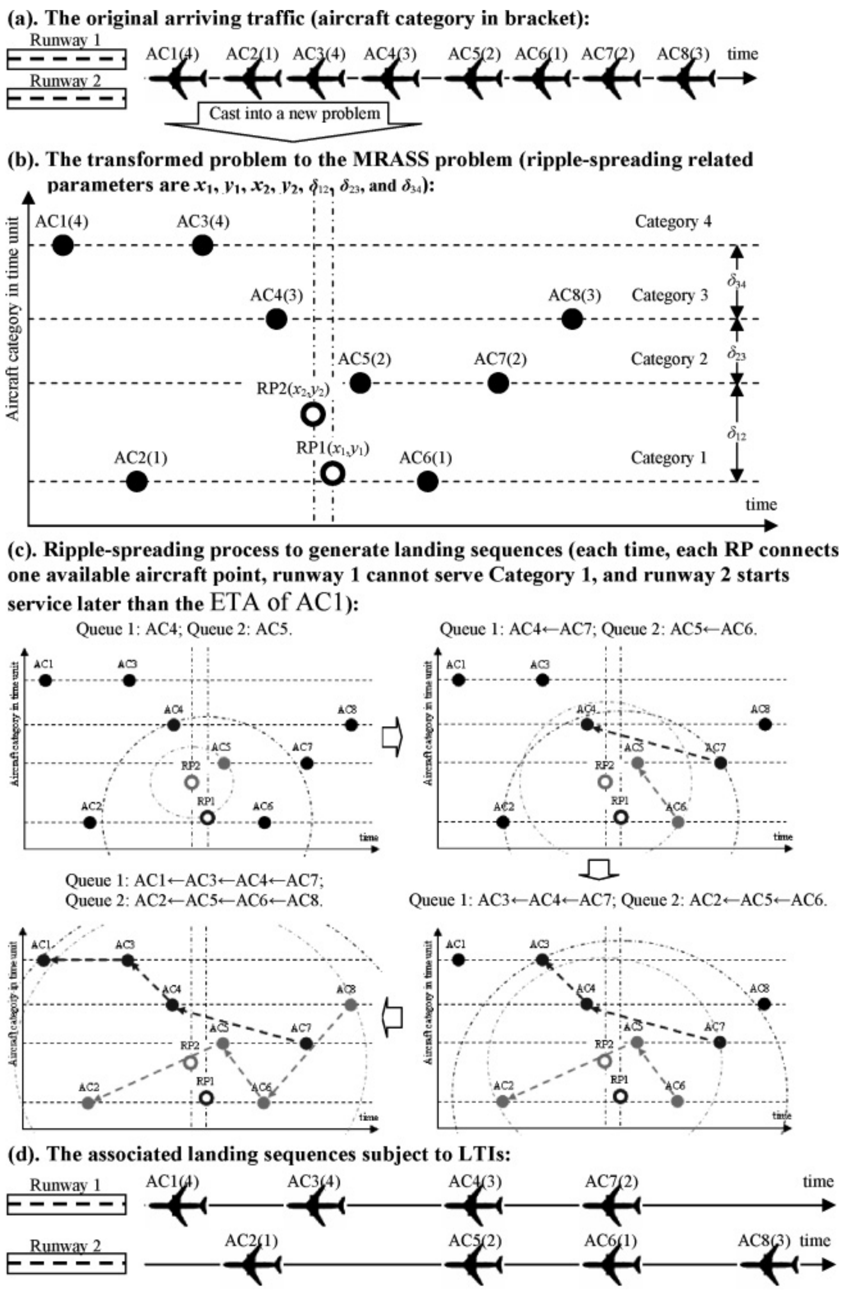
(a). The original arriving traffic (aircraft category in bracket):
Runway 1
Runway 2
AC1(4) AC2(1) AC3(4) AC4(3) AC5(2) AC6(1) AC7(2) AC8(3) time
Cast into a new problem
(b). The transformed problem to the MRASS problem (ripple-spreading related parameters are $x_1$, $y_1$, $x_2$, $y_2$, $\delta_{12}$, $\delta_{23}$, and $\delta_{34}$):
Aircraft category in time unit
Category 4
Category 3
Category 2
Category 1
RP2($x_2$,$y_2$)
RP1($x_1$,$y_1$)
$\delta_{34}$
$\delta_{23}$
$\delta_{12}$
time
(c). Ripple-spreading process to generate landing sequences (each time, each RP connects one available aircraft point, runway 1 cannot serve Category 1, and runway 2 starts service later than the ETA of AC1):
Queue 1: AC4; Queue 2: AC5.
Queue 1: AC4←AC7; Queue 2: AC5←AC6.
Queue 1: AC1←AC3←AC4←AC7; Queue 2: AC2←AC5←AC6←AC8.
Queue 1: AC3←AC4←AC7; Queue 2: AC2←AC5←AC6.
(d). The associated landing sequences subject to LTIs:
Runway 1
Runway 2
AC1(4) AC3(4) AC4(3) AC7(2) time
AC2(1) AC5(2) AC6(1) AC8(3) time

Figure 11: RSGA for the multi-runway problem (MRASP).

issue through extensive simulation tests. The basic idea behind the experiment is: we use a small sampling step to sample the solution space of the transformed problem (i.e., the ripple-spreading model described in Section 4), count the number of different solutions we find, and then compare them with a roughly estimated number of candidate landing sequences by shifting positions between adjacent aircraft.

Table 3: Simulation analysis on completeness of solution space (20 aircraft).

| | Probability that shifting position is potentially useful | Total number of potential good landing sequences | Total number of different solutions covered by the transformed problem |
|---|---|---|---|
| Congested case | ~0.4915 | ≤257,690 | ≥27,769 |
| Normal case | ~0.2901 | ≤152,109 | ≥14,137 |
| Under-congested case | ~0.2046 | ≤107,269 | ≥5,769 |

Suppose 20 arriving aircraft are uniformly randomly distributed within a time window of 1000/2000/3000 s (congested/normal/under-congested case). Basically, it is very likely that shifting positions is not necessary unless the gap between the ETAs of two aircraft is less than the associated LTI. We randomly generate 100 sets of arriving traffic data for each case (each set has 20 aircraft), and then count how many times on average the LTIs in Table 1 are violated. The result is: the probability that shifting positions is potentially useful is about 0.4915/0.2901/0.2046 in the congested/normal/under-congested cases. The actual probability should be much smaller, because, as discussed in Section 4, if two adjacent aircraft not satisfying the required LTI are of the same category, then shifting their positions will only make things worse.

Since the average gap between aircraft is close to the average LTI in Table 1, we assume it is only necessary to shift positions between adjacent aircraft in the original arriving traffic. Then for an arriving traffic of 20 aircraft, the total number of candidate landing sequences is no more than $2^{19} = 524{,}288$. Actually, the number is much less than 524,288, because, for instance, if the first aircraft in the original traffic is put as the second aircraft in a candidate landing sequence, then the second aircraft in the original traffic must take the first place in that landing sequence, otherwise, at least two aircraft that are not adjacent in the original traffic have to shift their positions, which is not allowed under our assumption for this analysis. For conservative purposes again, we use 524,288 as the total number of candidate landing sequences. Therefore, the number of potentially good landing sequences can be estimated by multiplying 524,288 by the probability at which shifting positions is potentially useful.

Then for each of the 100 sets of traffic data in the congested/normal/under-congested case, we use a reasonably small sampling step to sample the solution space of the transformed problem, and count the number of different solutions we find. Obviously, the number of different solutions found by sampling the solution space of the transformed problem is less than the actual number of different solutions in such a solution space. However, still for conservative purposes, we assume the size of the solution space of the transformed problem equals such a number determined by sampling, and we take the average of the 100 sets in each case.

The results of simulation analysis on the completeness of solution space are summarized in Table 3, from which one can see that the number of solutions covered by the transformed problem is about 5–11% of the number of potentially good landing sequences generated by shifting positions between adjacent aircraft. The remaining question is: will the set of solutions covered by the transformed problem overlap, at least reasonably overlap, the set of potentially good landing sequences? This is going to be answered in the following two sections by comparing the proposed RSGA with some existing algorithms which can guarantee the completeness of solution space. If the RSGA proposed in this paper can give similar performance in terms of airborne

Table 4: Definitions of traffic scenarios.

| | | $N_{AC}$ | | | | |
|---|---|---|---|---|---|---|
| | | 20 | 30 | 40 | 50 | 60 |
| $T$ | Under-congested case | 3,000 | 4,500 | 6,000 | 7,500 | 9,000 |
| (s) | Normal case | 2,000 | 3,000 | 4,000 | 5,000 | 6,000 |
| | Congested case | 1,000 | 1,500 | 2,000 | 2,500 | 3,000 |

delay, then it will be safe to say the transformed problem can at least cover many useful solutions exploited by existing algorithms.

### 7.2 ASP Performance

In this section, the performance of the RSGA proposed in this paper is studied by extensive simulations. For comparative purposes, the method reported in Bianco et al. (1997) (denoted as DTSPM since it is a deterministic method developed based on traveling salesman problem modeling), the GA reported in Hu and Chen (2005) (whose chromosome structure is based on the absolute position of aircraft), and the GA developed in Hu and Di Paolo (2008) (whose chromosome structure is based on the relative positions of aircraft) are also used to tackle the ASP. Hereafter, for distinguishing purposes, the GA in Hu and Chen (2005) is denoted as GA1, the GA in Hu and Di Paolo (2008) as GA2, and the RSGA proposed in Section 4 and Section 5 as RSGA1. Besides, a FCFS procedure is also used to establish FCFS landing sequences, in order to illustrate the important role of the position-shifting operation in the ASP. For the RSGA, the basic mutation probability and crossover probability are 0.2 and 0.5, respectively, and the population size is determined according to the problem scale, that is, $N_{AC}$, by the same function used in Hu and Di Paolo (2008).

Like Hu and Chen (2005) and Hu and Di Paolo (2008), the initial traffic data used in the simulation are randomly generated by referring to the data used in Bianco et al. (1997). Suppose that the total number of aircraft coming to land varies between 20 and 60, and accordingly, the length of an operating day varies between 3,000 s and 9,000 s in under-congested cases, between 2,000 s and 6,000 s in normal cases, and between 1,000 s and 3,000 s in congested cases, respectively. Table 4 gives the definitions of all traffic scenarios used in the simulation. In each case, we randomly generate 100 sets of initial arriving traffic data. In each test, only one set of initial arriving traffic data is used, one simulation run is conducted under the DTSPM, while 50 simulation runs are conducted under each GA, and then the averages are taken regarding average airborne delay (AAD) and average computational time (ACT) consumed by each algorithm.

All tests are conducted in a MATLAB environment on a PC with a 2.0 GHz CPU. Table 5 gives an example about what the optimized landing sequences look like under different algorithms, where due to limited space, the initial arriving traffic data have just 20 aircraft arriving in 1,000 s in a congested case. From Table 5, one can see that the position-shifting operation can significantly reduce the total airborne delay of a FCFS sequence, and RSGA1 gives a quite satisfactory landing sequence compared with another two GAs. Before giving any general conclusion, more extensive simulation tests need to be conducted and analyzed. Tables 6 through 8 and Figure 12 give the results of these extensive simulation tests.

Table 5: Results of a single test.

| Initial traffic | | | FCFS | | | DTSPM | | | GA1 | | | GA2 | | | GA3 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AC | Cat.* | ETA | Order | STA | Delay | Order | STA | Delay | Order | STA | Delay | Order | STA | Delay | Order | STA | Delay |
| 10 | 1 | 180 | 1 | 180 | 0 | 1 | 180 | 0 | 1 | 180 | 0 | 1 | 180 | 0 | **1** | **180** | **0** |
| 2 | 1 | 190 | 2 | 276 | 86 | 2 | 276 | 86 | 2 | 276 | 86 | 2 | 276 | 86 | **2** | **276** | **86** |
| 20 | 2 | 213 | 3 | 476 | 263 | 3 | 476 | 263 | 3 | 476 | 263 | 3 | 476 | 263 | **3** | **476** | **263** |
| 15 | 2 | 221 | 4 | 556 | 335 | 4 | 556 | 335 | 4 | 556 | 335 | 4 | 556 | 335 | **4** | **556** | **335** |
| 3 | 4 | 319 | 5 | 666 | 347 | 14 | 1,509 | 1,190 | 14 | 1,498 | 1,179 | 5 | 666 | 347 | **6** | **746** | **427** |
| 14 | 2 | 369 | 6 | 746 | 377 | 5 | 636 | 267 | 5 | 636 | 267 | 7 | 826 | 457 | **5** | **636** | **267** |
| 16 | 1 | 456 | 7 | 818 | 362 | 6 | 708 | 252 | 8 | 858 | 402 | 19 | 1,880 | 1,424 | **18** | **1,758** | **1,302** |
| 13 | 1 | 511 | 8 | 914 | 403 | 19 | 1,941 | 1,430 | 20 | 2,026 | 1,515 | 17 | 1,688 | 1,177 | **19** | **1,854** | **1,343** |
| 7 | 2 | 521 | 9 | 1,114 | 593 | 8 | 989 | 468 | 6 | 716 | 195 | 12 | 1,256 | 735 | **12** | **1,256** | **735** |
| 1 | 4 | 603 | 10 | 1,224 | 621 | 15 | 1,599 | 996 | 16 | 1,678 | 1,075 | 16 | 1,616 | 1,013 | **7** | **836** | **233** |
| 17 | 3 | 603 | 11 | 1,294 | 691 | 7 | 889 | 286 | 7 | 786 | 183 | 8 | 896 | 293 | **10** | **1,086** | **483** |
| 18 | 4 | 653 | 12 | 1,424 | 771 | 16 | 1,689 | 1,036 | 15 | 1,588 | 935 | 15 | 1,526 | 873 | **8** | **926** | **273** |
| 9 | 2 | 706 | 13 | 1,504 | 798 | 9 | 1,069 | 363 | 9 | 1,058 | 352 | 6 | 746 | 40 | **13** | **1,336** | **630** |
| 19 | 4 | 720 | 14 | 1,614 | 894 | 17 | 1,779 | 1,059 | 17 | 1,768 | 1,048 | 11 | 1,176 | 456 | **9** | **1,016** | **296** |
| 11 | 2 | 754 | 15 | 1,694 | 940 | 10 | 1,149 | 395 | 12 | 1,298 | 544 | 10 | 1,066 | 312 | **14** | **1,416** | **662** |
| 5 | 2 | 789 | 16 | 1,774 | 985 | 11 | 1,229 | 440 | 10 | 1,138 | 349 | 13 | 1,336 | 547 | **16** | **1,606** | **817** |
| 4 | 3 | 812 | 17 | 1,844 | 1,032 | 13 | 1,379 | 567 | 13 | 1,368 | 556 | 9 | 966 | 154 | **11** | **1,156** | **344** |
| 6 | 2 | 859 | 18 | 1,944 | 1,085 | 12 | 1,309 | 450 | 11 | 1,218 | 359 | 14 | 1,416 | 557 | **17** | **1,686** | **827** |
| 8 | 1 | 952 | 19 | 2,016 | 1,064 | 20 | 2,037 | 1,085 | 19 | 1,930 | 978 | 18 | 1,784 | 832 | **20** | **1,950** | **998** |
| 12 | 4 | 1,001 | 20 | 2,244 | 1,243 | 18 | 1,869 | 868 | 18 | 1,858 | 857 | 20 | 2,108 | 1,107 | **15** | **1,526** | **525** |
| Total delay (s) | | | — | 12,890 | — | — | 11,836 | — | — | 11,478 | — | — | 11,008 | — | — | **10,846** | — |

*Cat. = category.

Table 6: Test results in under-congested cases.

| AAD | $N_{AC}$ | | | | |
|---|---|---|---|---|---|
| (s) | 20 | 30 | 40 | 50 | 60 |
| FCFS | 94.47 | 114.75 | 125.70 | 124.86 | 124.82 |
| DTSPM | 75.73 | 86.51 | 95.60 | 98.40 | 98.91 |
| GA1 | 73.56 | 85.74 | 93.80 | 96.98 | 95.82 |
| GA2 | 73.56 | **84.01** | **91.26** | 93.47 | **94.30** |
| RSGA1 | **72.77** | 85.24 | 92.13 | **93.09** | 95.14 |

Table 7: Test results in normal cases.

| AAD | $N_{AC}$ | | | | |
|---|---|---|---|---|---|
| (s) | 20 | 30 | 40 | 50 | 60 |
| FCFS | 254.07 | 357.50 | 439.14 | 533.00 | 578.14 |
| DTSPM | 181.00 | 251.85 | 312.09 | 359.37 | 389.52 |
| GA1 | 174.55 | 236.25 | 288.39 | 339.32 | 351.93 |
| GA2 | **170.61** | 230.54 | **284.51** | **328.72** | 349.42 |
| RSGA1 | 170.64 | **227.37** | 285.62 | 331.97 | **348.33** |

Table 8: Test results in congested cases.

| AAD | $N_{AC}$ | | | | |
|---|---|---|---|---|---|
| (s) | 20 | 30 | 40 | 50 | 60 |
| FCFS | 598.70 | 912.83 | 1,205.45 | 1,521.50 | 1,809.38 |
| DTSPM | 457.75 | 698.96 | 909.89 | 1,128.07 | 1,326.60 |
| GA1 | 446.83 | 686.58 | 894.32 | 1,106.61 | 1,305.93 |
| GA2 | 443.70 | 675.89 | 887.04 | 1,093.34 | 1,294.08 |
| RSGA1 | **439.60** | **660.88** | **869.11** | **1,075.47** | **1,280.49** |

Tables 6 through 8 compare the performance of FCFS, DTSPM, GA1, GA2, and RSGA1 when they are applied to under-congested cases, normal cases, and congested cases with different $N_{AC}$ values. We make the following observations from these tables:

- With the position-shifting operation, DTSPM, GA1, GA2, and RSGA1 can significantly reduce the airborne delays of arriving traffic compared with FCFS. On average, the position-shifting operation results in 20–30% fewer airborne delays.

- In the same congestion condition, the delay AAD should be at the same level for the same algorithm, regardless of the number of aircraft, $N_{AC}$. However, Tables 6 through 8 show that AAD goes up as $N_{AC}$ increases, which reflects the nature of the ASP. As the ASP is an NP-hard problem, no method used in the simulation can guarantee finding optimal solutions. When the problem scale is small, say $N_{AC} = 20$, it is relatively easy to find optimal or suboptimal solutions, which means a smaller AAD in the same congestion condition; while if the problem scale is large, say, $N_{AC} = 60$, all algorithms often end up only with good feasible solutions, which means a larger AAD.

- From Tables 6 through 8, it can be seen that in under-congested cases, the gaps between the performances of different methods are relatively small, while in

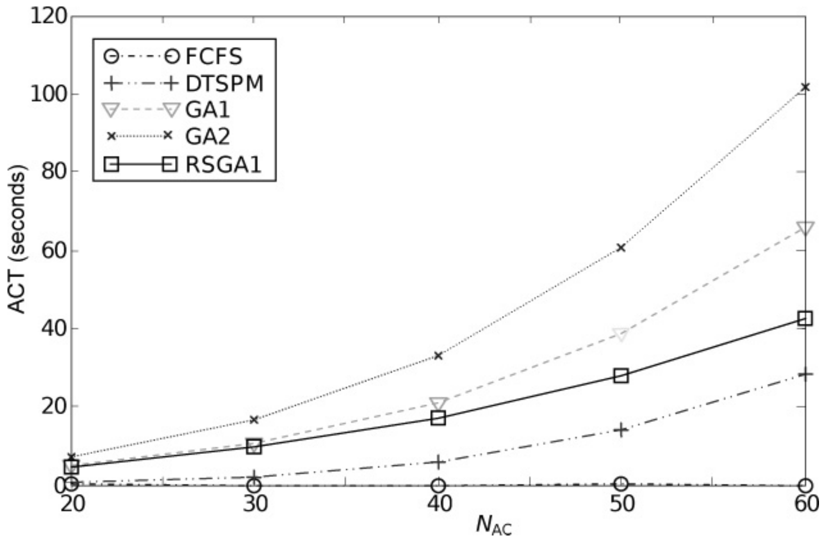

Figure 12: Computational time for the different algorithms.

congested cases, the gaps expand significantly. This is understandable, because, in under-congested cases, the FCFS landing sequences already give reasonably good performance in terms of airborne delay, and a few position-shifting operations can slightly improve the sequences, while in congested cases, the FCFS landing sequences often result in heavy airborne delays, and many position-shifting operations are required to significantly reduce airborne delays. If a method can effectively locate some of these useful positions for shifting in congested cases, it may achieve outstanding performance. Therefore, congested cases can be considered as the most demanding cases in the ASP simulation for evaluating the different methods.

- When their parameters are properly tuned, all GAs achieve fewer airborne delays than DTSPM. GA1 gives about 2% fewer airborne delays than DTSPM, GA2 gives 5% fewer, and RSGA1 gives 6% fewer. This probably implies that stochastic searching algorithms are more suitable than deterministic methods for NP-hard problems like the ASP. However, the cost all GAs have to pay is the much heavier computational burden compared with DTSPM, as shown in Figure 12.

- As reported in Hu and Di Paolo (2008), GA2 outperforms GA1, mainly because the chromosome structure and the uniform crossover used by GA2 are particularly effective for identifying, inheriting, and protecting useful relative positions of aircraft, which, from the position-shifting point of view, are much more important than the absolute positions of aircraft used in GA1. However, to record all relative positions of aircraft and to operate on them, GA2 requires a huge memory capacity as well as some computationally expensive feasibility-guaranteeing measures. As a result, the ACT is much larger than the GA1. Actually, from Figure 12, one can see that GA2 is the most computationally expensive algorithm of all.

- RSGA1 achieves a quite satisfactory performance in terms of airborne delay. In general, RSGA1 has a performance similar to that of GA2 in under-congested cases and normal cases, while in congested cases, which are probably the most important cases in the ASP, RSGA1 beats GA2 by giving 1–2% fewer airborne delays. Therefore, it should now be safe to claim that the RSGA proposed in this paper does work well for the ASP as expected.

- RSGA1 also exhibits very promising performance in terms of computational burden. Actually, Figure 12 shows that RSGA1 is the least computationally expensive algorithm of all the GAs. There are two main reasons: (i) as a deterministic process, the ripple-spreading model is computationally cheap, just like the deterministic method DTSPM, which is computationally much cheaper than all GAs, and (ii) the value-based representation used in RSGA1 makes it possible and easy to apply all classic evolutionary operations in a very straightforward way, in other words, no additional measures are needed in the evolutionary operations to serve any non-evolutionary purposes. The merit of RSGA1 in terms of computational burden, as illustrated in Figure 12, might not be very apparent in a dynamic ASP, where GA1 and GA2 can take advantage of the receding horizon control strategy to improve their real-time properties, as reported in Hu and Chen (2005) and Hu and Di Paolo (2008). However, this merit gives RSGA1 a better potential of applying to other large scale static combinatorial problems, particularly when compared with GA2. Due to its matrix representation, when GA2 is applied to large scale problems, it will face $O(n^2)$ memory problems, of which RSGA1 is totally free.
- Tables 6 through 8 can help answer the question about the completeness of solution space raised in Section 6.1. Since RSGA1 achieves performance similar to the other methods, which are designed directly based on the original ASP, and sometimes, mainly in congested cases, RSGA1 gives better solutions, one may conclude that the transformed problem introduced in Section 4 covers a considerably large proportion of high-quality solutions from the original ASP.

## 7.3 Tests on the MRASP

The MRASP test cases in Hansen (2004) are adopted to test our RSGA for the MRASP, denoted as RSGA2. Here we will compare RSGA2 with Method 4 in Hansen (2004) and GA4 in Hu and Di Paolo (2009a), respectively. Since few details in the design of GAs are given in Hansen (2004), we cannot repeat exactly the methods reported therein. Therefore, we quote directly from the best results achieved by Method 4 in Hansen (2004). Readers should be aware that the LTIs used in this study are different from those given in Table 1, which is adopted in the simulations in Section 7.2. Actually, the LTIs used in Hansen (2004) are based on three categories of aircraft. To illustrate how three different GAs optimize arrival traffic, Table 9 gives the test results of Scenario 1 in Hansen (2004), where 12 aircraft arrive at a three-runway system. In this case study, the optimized values for the ripple-spreading related parameters in RSGA2 are as following: the coordinates of the three RPs are $(x_1, y_1) = (6.9, 11.0)$, $(x_2, y_2) = (8.8, 2.2)$, $(x_3, y_3) = (3.7, 6.4)$, the gaps between the three categories are $\delta_{12} = 14.7$, $\delta_{23} = -8.5$ (Category 1 is always fixed as 0), and they are all in units of time. Table 9 shows clearly that, in Scenario 1, RSGA2 outperforms Method 4 in Hansen (2004) in terms of total airborne delay (TAD). One may also note from Table 9 that RSGA2 assigns aircraft to three runways most evenly in all three GAs, while Method 4 in Hansen (2004) generates the most uneven assignments. This may partially explain why this method has the worst performance in Scenario 1. Due to the limited space, here we only summarize the main

Table 9: Scenario 1 in Hansen (2004).

| Initial traffic data | | | Method 4 in Hansen (2004)* | | | GA4 in Hu and Di Paolo (2009a)** | | | RSGA2† | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| AC No. | Category | ETA | RW No. | STA | Del§ | RW No. | STA | Del§ | RW No. | STA | Del§ |
| DL130 | Heavy | 10 | 3 | 10 | 0 | 3 | 10 | 0 | 2 | 10 | 0 |
| AA335 | Small | 15 | 1 | 15 | 0 | 1 | 15 | 0 | 2 | 15 | 0 |
| UA123 | Heavy | 7 | 1 | 7 | 0 | 1 | 7 | 0 | 1 | 7 | 0 |
| DL1920 | Heavy | 6 | 1 | 6 | 0 | 3 | 6 | 0 | 3 | 6 | 0 |
| UA1133 | Large | 10 | 1 | 10.5 | 0.5 | 2 | 10 | 0 | 1 | 10 | 0 |
| NW2123 | Heavy | 5 | 3 | 5 | 0 | 3 | 5 | 0 | 3 | 5 | 0 |
| AA205 | Large | 15 | 1 | 16 | 1 | 2 | 15 | 0 | 1 | 15 | 0 |
| DL3319 | Heavy | 7 | 1 | 8 | 1 | 3 | 7 | 0 | 3 | 7 | 0 |
| SW200 | Small | 6 | 2 | 7 | 1 | 2 | 7 | 1 | 2 | 7 | 1 |
| DL510 | Heavy | 9 | 1 | 9 | 0 | 1 | 9 | 0 | 2 | 9 | 0 |
| UA410 | Heavy | 4 | 3 | 4 | 0 | 3 | 4 | 0 | 3 | 4 | 0 |
| SW185 | Large | 6 | 3 | 6.5 | 0.5 | 1 | 6 | 0 | 1 | 6 | 0 |
| | TAD | | | 4 | | | **1** | | | **1** | |

§Del: Delays in units of time.

Table 10: Summary of MRASP test results (delays in units of time).

| Test cases in Hansen (2004) | | | | TAD under Method 4 in | TAD under GA4 in | TAD under |
|---|---|---|---|---|---|---|
| Scenario | $N_{AC}$ | $N_R$ | Heavy aircraft | Hansen (2004) | Hu and Di Paolo (2009a) | RSGA2 |
| 1 | 12 | 3 | Yes | 3.5 | **1.0** | **1.0** |
| 2 | 15 | 3 | Yes | 9 | 5.5 | **5.2** |
| 3 | 20 | 5 | Yes | 12 | 7.7 | **7.6** |
| 4 | 12 | 3 | No to runway 3 | 8.5 | **4.3** | 4.9 |

test results. Table 10 lists the average results of 10 runs of RSGA2 in each scenario, as well as the relevant results in Hansen (2004) and Hu and Di Paolo (2009a). From Table 10, one can see the RSGA proposed in this paper works fairly well in the MRASP.

## 8 Conclusions

This paper reports on a novel ripple-spreading genetic algorithm (RSGA) scheme by using the aircraft sequencing problem (ASP) as a case study. In the RSGA, the original ASP is transformed into a ripple-spreading model, where arriving aircraft are projected as points in an artificial space, and a ripple-spreading process is designed to use certain ripple-spreading parameters as input to connect all aircraft points to form a landing sequence. A very basic binary GA is then used to evolve these ripple-spreading parameters in order to find an optimal landing sequence. From this study, we can formulate the following conclusions.

1. The RSGA is free of feasibility problems because the ripple-spreading model requires the GA to evolve only the value of parameters and, therefore, all classic evolutionary operations can be applied without any constraints.

2. The RSGA is also highly memory-efficient and thus scalable because a chromosome stores only the values of a few ripple-spreading parameters regardless of the problem scale.
3. The ripple-spreading model can automatically filter out many bad solutions incorporating empirical knowledge, therefore, from a practical point of view, the RSGA has good potential as an application, although it may not guarantee the completeness of solution space.
4. The proposed RSGA scheme has good flexibility and extendibility, for example, the ripple-spreading model can easily be modified to guarantee the completeness of solution space or to apply to the MRASP.
5. An extensive simulation study shows that the new algorithm can outperform existing GAs based on permutation representations for the ASP, or at least achieve similar performance without their disadvantages.
6. However, the ASP is highly simplified in this study and it is necessary to increase the complexity of the ASP in future studies by taking into account more realistic factors and constraints, such as using more aircraft categories and considering the equity of delays and landing time windows.
7. The simplified ASP still makes a good case study to verify the proposed RSGA scheme, which has a very good potential of being suitable to a wide range of combinatorial optimization problems, such as general job scheduling and network topology optimization, in which permutation representations are typically used to construct chromosomes in the implementation of GAs.

## Acknowledgments

This work was supported by EPSRC Grant EP/C51632X/1. The authors would also like to thank the anonymous reviewers, whose comments were of great help to improve the quality of the paper.

## References

Andreatta, G., and Romanin-Jacur, G. (1987). Aircraft flow management under congestion. *Transportation Science*, 21(4):249–253.

Bäck, T., Hammel, U., and Schwefel, H.-P. (1997). Evolutionary computation: Comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17.

Beasley, J. E., Sonander, J., and Havelock, P. (2001). Scheduling aircraft landings at London Heathrow using a population heuristic. *Journal of the Operational Research Society*, 52(4):483–493.

Bianco, L., Dell'Olmo, P., and Odoni, A. R. (1997). *Modelling and simulation in air traffic management*. Berlin: Springer Verlag.

Bianco, L., and Odoni, A. R. (1993). *Large scale computation and information processing in air traffic control*. Berlin: Springer Verlag.

Bianco, L., Ricciardelli, S., Rinaldi, G., and Sassano, A. (1988). Scheduling task with sequence-dependent processing times. *Naval Research Logistics*, 35(2):177–184.

Carr, G. C., Erzberger, H., and Neuman, F. (1999). Delay exchanges in arrival sequencing and scheduling. *Journal of Aircraft*, 36(5):785–791.

Carr, G. C., Neuman, F., and Erzberger, H. (2000). Fast-time study of aircraft-influenced arrival sequencing and scheduling. *Journal of Guidance, Control and Dynamics*, 23(3):526–531.

Cheng, V. H. L., Crawford, L. S., and Menon, P. K. (1999). Air traffic control using genetic search techniques. In *Proceedings of the IEEE International Conference on Control Applications*, pp. 643–649.

Ciesielski, V., and Scerri, P. (1998). Realtime genetic scheduling of aircraft landing times. In *The 1998 IEEE Congress on Evolutionary Computation*, pp. 360–364.

Dear, R. G. (1976). *The dynamic scheduling of aircraft in the near terminal area*. Cambridge, MA: MIT Press.

Eiben, A. E., and Schoenauer, M. (2002). Evolutionary computing. *Information Processing Letters*, 82(1):1–6.

Eiben, A. E., and Smith, J. E. (2003). *Introduction to evolutionary computing*. Berlin: Springer Verlag.

Hansen, J. V. (2004). Genetic search methods for air traffic control. *Computers and Operations Research*, 31(3):445–459.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: University of Michigan Press.

Hu, X., Di Paolo, E., and Barnett, L. (2008). Ripple-spreading model and genetic algorithm for random complex networks: Preliminary study. In *The 2008 IEEE Congress on Evolutionary Computation (in the 2008 IEEE World Congress on Computational Intelligence)*, pp. 3642–3649. IEEE Press.

Hu, X. B., and Chen, W. H. (2005). Genetic algorithm based on receding horizon control for arrival sequencing and scheduling. *Engineering Applications of Artificial Intelligence*, 18(5):633–642.

Hu, X. B., and Di Paolo, E. (2007). A hybrid genetic algorithm for the travelling salesman problem. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2007)*, pp. 357–367. Berlin: Springer.

Hu, X. B., and Di Paolo, E. (2008). Binary representation based genetic algorithm for aircraft arrival sequencing and scheduling. *IEEE Transactions on Intelligent Transportation Systems*, 9(2):301–310.

Hu, X. B., and Di Paolo, E. (2009a). An efficient genetic algorithm with uniform crossover for air traffic control. *Computers and Operations Research*, 36(1):245–259.

Hu, X. B., and Di Paolo, E. (2009b). A ripple-spreading genetic algorithm for airport gate assignment problem. In *The 2009 IEEE Congress on Evolutionary Computation*, pp. 1857–1864.

Hu, X. B., Di Paolo, E., and Wu, S. F. (2008). A comprehensive fuzzy-rule-based self-adaptive genetic algorithm. *International Journal of Intelligent Computing and Cybernetics*, 1(1):94–109.

Hu, X. B., Leeson, M. S., and Hines, E. L. (2010). A ripple-spreading genetic algorithm for the network coding problem. In *The 2010 IEEE World Congress on Computer Intelligence*.

Hu, X. B., Wang, M., Leeson, M. S., Hines, E. L., and Di Paolo, E. (2010). A review on ripple-spreading genetic algorithms for combinatorial optimization problems. In *The 9th IEEE International Conference on Cognitive Informatics*, pp. 441–448.

Pelegrin, M. (1994). *Towards global optimization for air traffic management*. AGARD-AG-321.

Psaraftis, H. N. (1978). *A dynamic programming approach to the aircraft sequencing problem*. Cambridge, MA: MIT Press.

Psaraftis, H. N. (1980). A dynamic programming approach for sequencing identical groups of jobs. *Operations Research*, 28(6):1347–1359.

Robinson, J. E., III, Davis, T. J., and Isaacson, D. R. (1997). Fuzzy reasoning-based sequencing of arrival aircraft in the terminal area. In *AIAA Guidance, Navigation and Control Conference*.

Sywerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 2–9. Morgan Kaufmann.

Venkatakrishnan, C. S., Barnett, A., and Odoni, A. R. (1993). Landings at Logan Airport: Describing and increasing airport capacity. *Transportation Science*, 27(3):211–227.

Zhang, J., Chung, H. S. H., and Lo, W. L. (2007). Clustering-based adaptive crossover and mutation probabilities for genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 11(3):326–335.