SCHEDULING AIRCRAFT LANDINGS DYNAMICALLY USING STOCHASTIC AND DETERMINISTIC ELEMENTS

I. MOSER

1Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia
E-mail: imoser@ict.swin.edu.au

Abstract

The characteristics of a dynamic implementation of the single-runway aircraft landing problem were analysed using some well-known datasets. Given the nature and degree of complexity of this problem, a specific hybrid algorithm was developed, consisting of a stochastic and a deterministic element. The deterministic scheduler divides the aircraft into logical landing sequences as part of the recursive time line optimisation process. A relatively new stochastic algorithm called Extremal Optimisation uses the deterministic scheduling element for finding the optimal permutation. The new hybrid algorithm was observed to produce better results in a dynamic environment than other approaches, while the only available measure for solution quality is not unequivocal in a dynamic context. Insights on the reliability of the solver were gained testing variations of both problem instances and solver.

Keywords: artificial intelligence, stochastic algorithms, extremal optimisation, aircraft landing, scheduling, dynamic problems

1 Introduction

Extremal Optimisation (EO) is a recent addition to the available range of stochastic solvers. In its current form, it was first described in [14]. While similar to a Genetic Algorithm (GA), it also bears some resemblance to Simulated Annealing (SA). It is often possible and even meaningful to choose the same representation for the single EO individual as for an individual of a
GA or an SA application. There are, however, significant differences between these approaches, as discussed in detail in [9]. GAs are based on the notion of ‘breeding’ good solutions, preserving desirable features in the population and recombining them to find the optimal solution. SA draws its inspiration from the concept of thermal equilibrium, allowing for more exploration at the start of the search and narrowing the focus to better areas toward the end. While GAs and SA share the principle of ‘finding the good’, EO was designed to ‘exclude the bad’. More importantly to a dynamic environment, EO was not intended to show any convergence behaviour. Good solutions will be stored for reporting only.

Based on the Bak-Sneppen model [1] of self-organised criticality (SOC), EO was conceived to eliminate the worst component at each iteration to exchange it for another element. Crucially, the quality of the supplanting element is not assessed beforehand, enabling moves which do not improve the solution quality but prevent entrapment in local optima. According to the SOC paradigm, the solution will slowly evolve to a comparatively high quality. Given an interdependence between the components, moves have an influence on the ‘adaptedness’ of other components. When the components are at a generally high level of fitness, a further move is likely to lead to a deteriorating ‘avalanche’, causing substantial change and allowing for the exploration of other areas in the search space.

It is therefore intuitive to apply EO to graph-based problems, where natural neighbourhoods of interdependent components emerge through the connectivity of the graph. Not surprisingly, the first and very successful implementations of EO solved graph bipartitioning [8], [13], [15], spin glass problems [10], [11], [16] and different variations of graph colouring [11]. The seminal paper [14] also described an application to the travelling salesperson problem (TSP). Unlike the graph-based problems, which provide an intuitive way to discover the current fitness of each component, the TSP requires a specific neighbourhood definition to describe the possible state the component could be in. Most contemporary combinatorial optimisation problems will pose a similar neighbourhood modelling challenge, as does the aircraft landing problem (ALP).

The ALP is easily construed as a dynamic problem. Any practical implementation would have to accommodate arriving aircraft on the fly. A preliminary analysis, however, led to the conclusion that part of the ALP search space—the timeline—would lend itself to deterministic optimisation, as it turns into a unimodal problem once the permutation problem has been solved. This was not encouraging, as EO was observed to be less successful at solving another permutation problem, the TSP. Implementing a hybrid of EO and
a deterministic timeline solver both proved very successful and also revealed some interesting features in both the algorithm and the problem.

2 The Aircraft Landing Problem

For better comparison, the existing datasets from the OR library in [3] were chosen as a benchmark for this approach. Naturally, the algorithm implementation was optimised according to the specific features of the datasets. Detailed descriptions of the problem are found in [4] and [5]. As illustrated in Fig.1, all aircraft report to the Air Traffic Controller (ATC) at their appearance time $A$. They can arrive as early as $E$ (speeding up the approach), ideally land at $T$ and must land no later than $L$.

![Figure 1. Constraints of the problem.](image)

The objective function (Equation 1) minimises the weighted deviations from target landing times $T$. $\alpha$ and $\beta$ are the units of deviation from target times, $g$ and $h$ the penalty weights for landing early/late.

$$\min \sum_{i=1}^{p} \{g_i \alpha_i + h_i \beta_i\}$$

(1)

In the dynamic case, the solving process follows a time line measured as the CPU time used by the program. The aircraft are made available to the hybrid EO solver at their given appearance times and remain in the active window of the solver until the landing time found by the solver is too close to the current time to make any further change (i.e. within freeze time). In the dynamic case, the solver solves a sequence of smaller but interrelated problem instances instead of the overall problem of all aircraft in the static instance.

2.1 Datasets

The publicly available datasets [3] consist of 13 problem instances featuring between 10 and 500 aircraft. The datasets and their optimal solutions were
analysed in the interest of designing a good EO neighbourhood for solving the permutation problem. Judging from the smaller instances, the best strategy to optimise Equation 1 is to start by ordering the aircraft by target times. Only the minimum separation times and the penalty weights justify deviating from the target-ordered sequence and create a need for optimisation.

The results analysed in Table 1 were obtained using the EO hybrid described below. In the case of the smaller problems, the results’ optimality was confirmed comparing the costs to the results in [5]. For the datasets airland8 and above, the results analysed were the best outcomes obtained from the experiments with the dynamic EO hybrid.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Aircraft</th>
<th>Active</th>
<th>Max Sequence</th>
<th>Moves</th>
<th>Longest Move</th>
</tr>
</thead>
<tbody>
<tr>
<td>airland1</td>
<td>10</td>
<td>9</td>
<td>7 (5)</td>
<td>0 (0)</td>
<td>0</td>
</tr>
<tr>
<td>airland2</td>
<td>15</td>
<td>10</td>
<td>7 (5)</td>
<td>2 (1.3)</td>
<td>2</td>
</tr>
<tr>
<td>airland3</td>
<td>20</td>
<td>11</td>
<td>7 (3)</td>
<td>1 (0.5)</td>
<td>1</td>
</tr>
<tr>
<td>airland4</td>
<td>20</td>
<td>11</td>
<td>15 (15)</td>
<td>0 (0)</td>
<td>0</td>
</tr>
<tr>
<td>airland5</td>
<td>20</td>
<td>10</td>
<td>15 (15)</td>
<td>3 (1.5)</td>
<td>4</td>
</tr>
<tr>
<td>airland6</td>
<td>30</td>
<td>6</td>
<td>27 (15)</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>airland7</td>
<td>44</td>
<td>4</td>
<td>36 (11)</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
<tr>
<td>airland8</td>
<td>50</td>
<td>18</td>
<td>6 (2.3)</td>
<td>3 (0.6)</td>
<td>3</td>
</tr>
<tr>
<td>airland9</td>
<td>100</td>
<td>11</td>
<td>17 (2.8)</td>
<td>13 (1.3)</td>
<td>5</td>
</tr>
<tr>
<td>airland10</td>
<td>150</td>
<td>15</td>
<td>25 (3.5)</td>
<td>27 (1.8)</td>
<td>8</td>
</tr>
<tr>
<td>airland11</td>
<td>200</td>
<td>12</td>
<td>14 (3.8)</td>
<td>32 (1.6)</td>
<td>10</td>
</tr>
<tr>
<td>airland12</td>
<td>250</td>
<td>13</td>
<td>23 (3.4)</td>
<td>36 (1.4)</td>
<td>10</td>
</tr>
<tr>
<td>airland13</td>
<td>500</td>
<td>17</td>
<td>41 (4.5)</td>
<td>78 (1.5)</td>
<td>7</td>
</tr>
</tbody>
</table>

In all datasets, there are groups of aircraft which plan to land within short distances of each other. These are the planes that require optimisation, as their minimum separation times force them to land outside their target times. The ‘Max Sequence’ column lists the maximum number of aircraft in the dataset which form such a group, with the mean value in parentheses. Especially in the case of the larger problems, these numbers lend themselves as an approximate measure of complexity, as no gain can be expected from swapping a plane with one outside its group of influence.

In the dynamic case, the length of the active time window may further reduce the complexity. If the time window is shorter than the interdependent sequence, the problem becomes less complex as the number of possible combinations is reduced. At the same time, when active aircraft have an influence on the schedule of planes which have not appeared yet, it becomes virtually...
impossible to create the optimal result in a global sense. In the four largest datasets in Table 1, the active window (column ‘Active’) is shorter than the largest interdependent number of aircraft. Therefore, the results obtained from the dynamic environment cannot be expected to be optimal in the sense of the overall result of the objective function, as some time windows were solved on partial information.

The ‘Longest Move’ gives some indication as to how long a displacement from target-time ordering was needed to achieve the best solution. The solutions found for the smaller problems (up to airland5) have the same quality as the best-known published and can be assumed to be optimal. For the larger problems, the numbers are only a rough estimation, as the results may not be the optimal solutions over the whole dataset (as sliding time windows were used). However, they do show that it is possible to achieve competitive results with relatively small displacements, indicating that a small-step neighbourhood is likely to be the best choice.

![Figure 2. Solving airland7.](image)

Instances airland6 and airland7 are different from the other datasets in that the aircraft arrive one by one, most often populating a time window of less than three aircraft. Obviously, this is not a situation that lends itself to combinatorial optimisation. In Fig.2, the blue dots stand for the target times. Green dots indicate landing on target. Red lines represent aircraft within freeze time, white aircraft are still active (being optimised). Yellow lines depict aircraft that have landed off target (if the landing dot is yellow), or on target (if it is green). Red dots are landings after the latest landing times \( L \) and make the solution invalid (the equivalent of aircraft running out of fuel and crash-landing). It is clear to see that the fate of the first crashing plane was already decided when the first
of the three aircraft landed on target. To generalise the algorithm to cover these cases, strategic decisions would have to be included in the algorithm.

3 Known Approaches

Many solution approaches to the ALP problem have been published. Indeed, many known job shop scheduling applications would lend themselves to the current problem. Only applications to the ALP are discussed here, especially the ones featuring dynamic applications or adaptations to the current datasets [3]. Only two approaches discuss the dynamic case, Beasley, Krishnamoorthy, Sharaisha and Abramson [5], who created the datasets, and Ciesiel斯基 and Scerri [18] who solved an Easter schedule procured from the Sydney, Australia airport. The latter accommodates the dynamics of the problem very flexibly, in that it solves the problem three minutes at a time, making use of the GA’s population, from which a feasible—if suboptimal—solution can be extracted at any time. The representation encodes seven periods of 30 seconds each. These are added to the first possible landing time, which is ‘now’. However, rather than stabilising over time, the quality of the solutions is reported to keep fluctuating.

Beasley’s dynamic implementation is based on some approaches devised for a static context, [19] and [6]. Ernst, Krishnamoorthy and Storer [19] introduced a simplex-based approach for the scheduling part of the problem. Dividing the aircraft into trees where only the root nodes are allowed to land on target, they exploited the same problem characteristics employed in the current work (Equation 2 in Section 4.3) for grouping the aircraft into independent sequences. The permutation part uses a GA implementation with an exponential distribution over the swaps with a bias toward shorter range, thus confirming our observation that a strong deviation from target-time order does not benefit the outcome. As all aircraft are optimised at the same time (the static case), the smaller problem instances up until airland8 were used for the experiments.

In the same work [19], the authors also presented a simplex-based branch-and-bound formulation which outperforms the GA in finding the optimum reliably for the first eight datasets. It was, however, observed to be slow when random instances with very similar target times were used, where many solutions cannot be ruled out early.

Beasley, Sonander and Havelock [6] applied a GA to a real dataset obtained from London Heathrow. Encoding the landing times of aircraft in terms of proportions of their admissible landing ranges, the separation times between
the aircraft are not enforced, but infeasible solutions are discouraged using penalty weights for fitness evaluations.

Beasley, Krishnamoorthy, Sharaiha and Abramson [4] solved the static case as a tree search using a linear program. In a preprocessing step, a preliminary upper bound was found by optimising the time line of a sequence ordered by target times. This approach was later used as a reference algorithm (denoted ‘DALP-OPT’) when solving the dynamic case [5]. As an exact approach, its running time does not scale well with the number of aircraft. Its use becomes practical in the dynamic case when the complexity is reduced to the active aircraft. The algorithms from [19] (‘DALP-H1’) and [6] (‘DALP-H2’) were applied to the dynamic implementation using constrained position shifting, limiting the length of possible swaps from target-time order. Both are hybrid approaches based on a stochastic algorithm and a deterministic scheduler implemented as a linear program. The stochastic parts of both DALP-H1 and DALP-H2 were implemented as a GA with a constructive preprocessing stage. Both rely on target-time ordering for their primary solution structure. The larger datasets airland9–13 were created for these experiments. Interestingly, the stochastic approaches outperform the exact algorithm on one of these large problems (see Table 2 in Section 5).

One of the oldest approaches, Psaraftis [22] created a deterministic model based on classifying the planes into groups of identical types of aircraft. As the minimum separation times depend on the types, preoptimising the order of the types is possible. Applying the sequence pattern of the types, the aircraft are first ordered by appearance (FCFS). Then, each aircraft is repositioned within the range of constrained position shifting such that it succeeds its optimal predecessor. A practical approach, this alternative could easily be applied to a dynamic environment.

Balakrishnan and Chandran [2] examined the static case, arguing that ATCs schedule incoming aircraft in batches, a task which has to be completed in approximately 45 minutes. The objective function optimises the latest arrival. The number of possible permutations, which are to be optimised according to the predefined pattern, is further restricted by the rule that aircraft using the same jet route during the approach cannot overtake one another. Since each runway is being serviced by two routes only, the possibility of shifts from FCFS (here equivalent to target-time ordering) is significantly reduced.
4 Algorithm

Most often, stochastic approaches to the ALP [4], [17], [18], [23]—the bulk of them GA implementations—designed the algorithm as a single-phase solver. The most successful approaches [5], [19] solved the sequencing and the scheduling tasks separately. It did not seem practical to leave a hill-climbing task to a stochastic algorithm; an EO implementation tackling both tasks in one cannot possibly be expected to outperform a hybrid. These considerations led to a three part-solution: In addition to the EO implementation, a deterministic Scheduler was developed. A Controller was also added to manage the time window and the reporting.

![Diagram of algorithm components]

Figure 3. Parts of the algorithm.

The respective tasks of each algorithm component are explained in the following sections.

4.1 Controller

Managing the time line and the active window, the Controller performs the following tasks:

1. At the start of the program, the Controller stores the CPU time as the start time of the algorithm.

2. After each iteration, the Controller checks the elapsed time, removing aircraft that are within freeze time of their scheduled landing times. New aircraft are added according to their appearance times.

3. If new aircraft have appeared, they are added in order of their target landing times. A preliminary landing time is assigned to them according to the minimum separation time from previous aircraft.

4. The active aircraft are left to the EO solver for optimisation.
5. The Controller stops execution when all aircraft have landed, or when the latest of the latest landing times has elapsed, whichever comes first.

4.2 EO Solver

From the very first EO implementations, which mutated the worst component at each iteration, it was observed that a power-law based choice of component to mutate could further improve the results. In their seminal work, Boettcher [8] and Boettcher and Percus [14] applied both the SOC-based original version and the power-law-distributed choice over the ranks of components. Ordering the components by increasing fitness ranks \( k \), the component to be changed is determined according to \( k^{-\tau} \) where the only free parameter \( \tau \) is a value between 1.1 and 3.0. While retaining an emphasis on the worst components, this scheme allows for a limited number of uphill moves. Using a power-law distribution is almost indispensable when the problem structure prohibits a neighbourhood model where moves are only based on the current component fitness, as is the case with the majority of problems.

For the current ALP application, the neighbourhood model chosen uses simple pairwise swaps of adjacent aircraft (illustrated in Fig.4). Therefore, the future quality for every possible move is known a priori. To ensure that local optima can be overcome, it is necessary to enable deteriorating moves. In this situation, the value to choose for \( \tau \) is obviously crucial to the algorithm performance. Boettcher and Percus discussed the ideal value for \( \tau \) in several works, most exhaustively in [13], and has provided a formula for deriving the ideal setting. However, due to very dissimilar neighbourhood models, different authors have found different ideal values for \( \tau \) in their applications. For the current experiments, a \( \tau \) of 1.6 – 1.8 was found to be most efficient, a value which largely coincides with the recommendations given in [13].

![Figure 4. EO neighbourhood model for the ALP.](image)

Refining step 4 of the controlling algorithm described in Section 4.1, the EO
solver consists of the following steps:

1. Receive target-ordered sequence of active aircraft of length $K$ and use it as a working solution.

2. Build a neighbourhood of $K$ candidate solutions by swapping one pair of aircraft at a time.

3. Submit candidates to Scheduler for optimisation of landing times and establish cost of optimised schedule.


5. Choose one candidate according to $k^{-\tau}$ and adopt it as new working solution unconditionally.

6. If the working solution is better than the existing schedule (either the preliminary one set by the controlling algorithm or subsequent improvements made in this step), adopt the landing times of the working solution.

7. Let the controlling algorithm check for update of active window. If changes have occurred, resume from 1. If not, resume from 2.

### 4.3 Scheduler

The implementation is based on the observation that the permutation at hand can be split up into sequences determined by Equation 2, which can then be scheduled separately.

$$
T_m + \sum_{j=m+1}^{n} \max \{ S_{ij} | m < j < i \} \geq T_n \tag{2}
$$

where $m < n; m \geq 1; n \geq 2; n \leq P$;

$T$ is the target time of an aircraft;

$P$ is the total number of aircraft and

$S_{ij}$ is the minimum separation time between each aircraft $j$ and its preceding aircraft $i$.

Note that Equation 2 enforces that for each plane, the separation time to all its predecessors, not only the immediately preceding plane, must hold. If Equation 2 holds for all aircraft between $m$ and $n$, the sequence between $m$ and $n$, of length $n - m$, is regarded as an interdependent chain of planes that must be optimised in combination.
The landing times of the aircraft in such an interdependent chain can be regarded as linked by the minimum separation times. To preserve the solution validity, they can be changed only in unison. To find out whether moving the sequence to an earlier landing is beneficial, one penalty weight is added for each aircraft to the score (as if landing each plane one time unit earlier). Each aircraft which is currently scheduled with a landing time after target adds a negative \( h \) (penalty weight for late landing) to the score (moving it to an earlier landing would reduce the cost), while each aircraft whose landing time is scheduled on or before target adds a positive \( g \) (as this aircraft would add to the cost if landed earlier). Fig.5 illustrates this.

In Fig.5, the move to an earlier schedule is of benefit (assuming equal penalty weights). The length of the move, in this case, is limited by the first aircraft to reach target time through the combined left shift. In this case, it is the third-last (after the broken line). It would be possible to split the sequence at that line. However, it is clear that a new move in combination is still beneficial, therefore no split is carried out yet. If all deviations from target attracted equal penalties and if no more aircraft were to follow after the ones drawn, the optimum sequence would ultimately split the group in two where indicated with the broken line. Both sequences would be shifted further left, but the lower sequence only about as far as landing the next-to-last aircraft on target, leaving a longer distance than required by the minimum separation time between the fourth and third-last aircraft.

The mirror image of this test is carried out for a move to a later landing time. Obviously, only one of these moves (earlier or later) can have a negative result and thus reduce the cost. If neither move is beneficial, the optimum has already been achieved. If there is a benefit to either move, the possible length

Figure 5. Optimising the schedule of a sequence.
of the shift has to be explored. Fig.5 already demonstrated the limit given by one of the aircraft reaching the target landing time. Shifting the time line to earlier landings, the possible limits are

a) the scheduled landing time of the last plane of an earlier chain and the separation times between the planes;

b) the earliest possible landing time of each craft in the sequence;

c) the target times of the planes currently landing later than their target. This is because they no longer add a negative late penalty to the score once the target time is reached.

If the possible shift range is zero and the boundary is the earlier chain, the benefits of uniting the chains and shifting them together has to be explored.

Similarly, if the benefit of both shifts is zero, we have to explore if one or more of the first aircraft in a chain can be shifted to an earlier landing with benefit while leaving the latter planes in place, or if one or more of the later planes can be shifted to a later landing with benefit if the earlier planes are left at their current landing times. This would indicate that the chain has to be split.

Following the outline sketched above, the exact algorithm consists of the following steps:

1) Pre-schedule all active aircraft by assigning each aircraft whose landing time \( t_j \) is not influenced by the separation time from a preceding aircraft to its target time \( T_j \). If the separation times from previous aircraft prevent assigning the target time, assign the first possible landing time (minimum separation time).

2) Divide all aircraft in the active window into \( x \) separate chains \( s_1 \ldots s_x \) of aircraft whose landing times influence each other according to Equation 1.

3) While there are chains \( s_i \) whose landing times can be further optimised, for each \( s_i \) check the benefit of

   (a) shifting all aircraft in \( s_i \) to an earlier landing time by means of adding an early penalty \( g \) for all aircraft with \( t_j \leq T_j \) and a negative late penalty \( -h \) for each aircraft landing later than the target time, i.e. \( t_j > T_j \)

   (b) deferring the landing of all aircraft in \( s_i \) by adding \( h \) for each aircraft with \( t_j \geq T_j \) and \(-g\) for all aircraft with \( t_j < T_j \).
4) If none of above checks returns a negative cost (an improvement), go to 7. If there is a gain in either (both is not possible), find the maximal displacement

(a) for a shift to earlier landing; the smallest of the following distances:
   i. the earliest landing according to the minimum separation times from planes in the preceding sequences \(\{s_k|k < i\}\), if they exist;
   ii. the earliest landing according to the minimum separation times from already landed aircraft, if they exist;
   iii. \(\min \{t_j - E_j|\forall t_j \in s_i; t_j \leq T_j\}\) where \(E_j\) is the earliest landing time of aircraft \(j\);
   iv. \(\min \{t_j - T_j|\forall t_j \in s_i; t_j > T_j\}\)

(b) for a shift to a later landing; the smallest of the following distances:
   i. the minimum distance to the landing times of the planes of the following sequences \(\{s_l|l > i\}\), if they exist;
   ii. \(\min \{L_j - t_j|\forall t_j \in s_i; t_j \geq T_j\}\) where \(L_j\) is the latest landing time of aircraft \(j\);
   iii. \(\min \{T_j - t_j|\forall t_j \in s_i; t_j < T_j\}\)

5) If either of the shifts (4a or 4b) is beneficial and the maximum possible displacement is nonzero, shift the landing times by the maximum displacement and go to 3. If one of the shifts is beneficial, but the maximum displacement is 0, go to 6.

6) Check if the distance to the aircraft of the previous (following) sequence is 0. If yes, join the chains and go to 3.

7) Check if it is beneficial to split a sequence by either

   (a) shifting the later planes in a sequence to a later landing time while leaving one or more of the first in place, or
   (b) shifting the earlier planes of a sequence to an earlier landing time while keeping the current landing times of one or more later planes in the sequence.

8) If none of these checks is successful, set the ‘optimised’ flag of the sequence to true and continue with the next sequence until all sequences are optimised.
5 Results Discussion

The six largest datasets from the OR-lib [3] were chosen for the dynamic experiments. All datasets have durations listed in time units to be interpreted as seconds. The time units allocated in the last five problems are rather lax, allowing 1122 and 1412 seconds on average to land 10 aircraft, whereas airland8 allocates only an average of 226 seconds for every 10 aircraft. Speeding up the experiments on airland9—13 to one fifth of the time did not have any influence on the results.

<table>
<thead>
<tr>
<th>Instance</th>
<th>DALP-OPT</th>
<th>DALP-H1</th>
<th>DALP-H2</th>
<th>EO Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td>airland8</td>
<td>2000</td>
<td>2915</td>
<td>2710</td>
<td>2320</td>
</tr>
<tr>
<td>airland9</td>
<td>7848</td>
<td>13555</td>
<td>12554</td>
<td>6110</td>
</tr>
<tr>
<td>airland10</td>
<td>17726</td>
<td>31945</td>
<td>31034</td>
<td>16386</td>
</tr>
<tr>
<td>airland11</td>
<td>19327</td>
<td>27417</td>
<td>23963</td>
<td>14559</td>
</tr>
<tr>
<td>airland12</td>
<td>25049</td>
<td>34246</td>
<td>31440</td>
<td>19342</td>
</tr>
<tr>
<td>airland13</td>
<td>58393</td>
<td>78008</td>
<td>58392</td>
<td>44708</td>
</tr>
</tbody>
</table>

The percentages given in Table 3 reference the outcomes of DALP-OPT listed in Table 2. In a dynamic environment, it is not impossible for a stochastic approach like EO (hybridised with a deterministic scheduler) to outperform an exact algorithm (equally hybridised with a different deterministic scheduler). This phenomenon was already observed in [5]: as reproduced in Table 2, the GA-based DALP-H2 produces a better result on airland13 than DALP-OPT. To confirm this, trials were run on some problems using an exhaustive search over each time window. The experiments consistently returned solutions with higher costs than using the EO solver. Apparently, a perfectly optimised time window does not represent an optimal prerequisite for the optimisation of the following time window.

<table>
<thead>
<tr>
<th>Instance</th>
<th>Min</th>
<th>Average</th>
<th>Std. Dev.</th>
</tr>
</thead>
<tbody>
<tr>
<td>airland8</td>
<td>116%</td>
<td>117%</td>
<td>1.4%</td>
</tr>
<tr>
<td>airland9</td>
<td>77%</td>
<td>77%</td>
<td>0%</td>
</tr>
<tr>
<td>airland10</td>
<td>92%</td>
<td>96%</td>
<td>1.5%</td>
</tr>
<tr>
<td>airland11</td>
<td>75%</td>
<td>75%</td>
<td>0%</td>
</tr>
<tr>
<td>airland12</td>
<td>77%</td>
<td>77%</td>
<td>0%</td>
</tr>
<tr>
<td>airland13</td>
<td>76%</td>
<td>77%</td>
<td>0.2%</td>
</tr>
</tbody>
</table>
The supposition that EO performs very well because it does not supply the optimal solution for each of the time windows is supported by the minimal variation in the results as listed in Table 3. The limited range of the pairwise swaps of the neighbourhood (only adjacent aircraft swap positions) may easily lead to a suboptimal time window. To verify this, a different neighbourhood model was designed and explored in Section 6.

In the literature, the larger problems from the OR-lib [3] are only solved as complete datasets in one approach [21]. Most algorithms used to solve the ALP were not designed to optimise hundreds of aircraft in combination. While in the current work, the subdivision of the problem emerged as a by-product of the development of the scheduling procedure, Jung and Laguna [20] explicitly attempted to subdivide the problem instances into smaller time segments with fewer aircraft. Based on sequences with equal numbers of time units, the approach optimises a few aircraft at a time using the mixed-integer linear program introduced in [4]. Although the borders are later adjusted to the time lines of the aircraft, no recursion is used to adjust preceding landing times after subsequent aircraft have been scheduled.

Although an appropriate method for subdivision has been developed in this work, using the principle expressed in Equation 2, the approach is based on the characteristics of the specific datasets [3], as explained in Section 2.1. The idea depends on the fact that between larger clusters of aircraft approaching in close sequence, we can expect ‘breaks’ at irregular intervals, allowing for longer distances between aircraft than the minimum separation time prescribes. These breaks allow for separate optimisation of groups of aircraft without compromising the global optimum. Judging by the datasets used by [18] (Sydney airport), this may well be a realistic assumption.

It is tempting to experiment with this method on the complete datasets in a ‘static’ setting (because for once, there would be an unequivocal quality measure due to the absence of time windows), but in reality, airports are hardly likely to optimise a whole day’s schedule all at once.

Comments from professionals in this area have corroborated the suspicion that the ALP often presents itself as a more ad-hoc problem, requiring repeated and fast reoptimisation of few aircraft at a time to account for unexpected events like the need for a second runway approach in bad weather. The tests carried out in the following section were therefore aimed at exploring the behaviour and reliability of the EO solver, not to solve realistic ALP problems.
6 Designing a ‘Hard’ Problem

The dynamic hybrid implementation presented in the current work has shown above-average performance. However, the problem characteristics give rise to the suspicion that the permutation problem left to the stochastic solver is not very demanding (Section 2.1). Moreover, the results suggest the solver is successful because of its modest changes to the solution (Section 5). The number of aircraft partaking in the permutation along with the rather modest length of moves from target-time ordering have inspired the current EO implementation (Section 4.2) with a very ‘cautious’ neighbourhood which swaps adjacent aircraft only. The limitations in complexity are well founded in real situations: No airline pilot would be content to be overtaken by a great many other aircraft before being allowed to land.

However, to find the limitations of this hybrid solver, it is useful to create highly unrealistic though very complex datasets with a chaotic order as an optimal solution. Target-time ordering is working against this aim. Consequently, the ‘harder’ datasets contain groups of aircraft that wish to land at exactly the same moment. In combination with a sufficiently long landing range, this will guarantee that the complexity is not inadvertently reduced by the length of the time window.

![Figure 6. AirlandX, an artificially created instance.](image)

The sequence of 3 x 12 aircraft\(^1\) shown in Fig.6 has equal penalties and randomly chosen separation times. The separation times between the adjacent planes of the ideal sequence are only half as long as the shortest random sep-

\(^1\)Available at: http://irene.moser.googlepages.com/airlandX.txt; http://irene.moser.googlepages.com/airlandY.txt.
aration time. The ideal solution lands all aircraft in each group of 12 before any aircraft in the next group lands. If the algorithm proceeds according to the optimal sequence, there are at most 16 aircraft in the active solver window, due to the overlap of the landing times of the preceding group and the appearance times of the following group. The number of active aircraft increases if the solver assigns the wrong position to an aircraft. Note that unlike in the datasets from [3], the separation times are only enforced between subsequent aircraft.

To match the increasing complexity, a neighbourhood with larger steps was designed for the EO solver. The power-law-distributed choice of next move typically finds long slopes in the fitness space hard to master. If the ideal position is far away, the algorithm is unlikely to perform the numerous steps in the same direction, unless the search space is rather small. To allow for larger jumps, each aircraft in turn was matched with a partner from a normal distribution, with numbers exceeding the sequence starting from the beginning (‘wrapping around’). The small-step neighbourhood explained in Section 4.2 still performed better on this problem. It seems that this neighbourhood is still capable of reaching a position as far away as 12 aircraft, even if the intermediate moves worsen the quality at first.

Fig. 6 demonstrates the ideal path through this problem with the white lines depicting aircraft in the active solver window. In an attempt to corroborate the assumption that the ‘small-step’ neighbourhood will eventually, with growing complexity, be outperformed by a design incorporating larger jumps, a problem with 24 consecutive aircraft planning to land at the same time was created. Fig. 7 shows the instance airlandY. Here, the maximum range the aircraft can shift is 24, the same as the number of aircraft present.

The results over 50 trials (Table 4, percentages reference the best-known solutions) show that more drastic changes in the neighbourhood are indeed beneficial when optimising a sequence of 24 aircraft with a maximum shift of 24. While the airlandX problem is still solved to better average quality with the small-step neighbourhood, the balance is clearly tipped in favour of more drastic changes when as many as 24 aircraft are present. The exact number of aircraft that constitutes the border of this phase transition is not very important in a factual situation. Making an aircraft wait while approximately 20 aircraft pass it by to land earlier is a very unreal situation, as are the datasets airlandX and airlandY.

The rather high values for the standard deviations are partly explained by the drastic rise in cost caused by a single misplaced aircraft. The least possible cost of a wrongly placed aircraft is at least four times the cost of a correctly
placed aircraft (since at least two aircraft have the wrong successor), because the minimum separation times for adjacent pairs that are not part of the optimal solution are at least twice as long.

As was expected, the alternative neighbourhood with drastic steps also performed worse on the OR-lib instances airland8–13. While this restricts the general applicability of neighbourhoods, it is of little consequence to the ALP, as a small-step neighbourhood has proved sufficient to solve all realistic datasets—and beyond—to satisfactory quality in a very short time.

### 7 Conclusion

A new hybrid algorithm for the single-runway dynamic ALP was introduced. The solver consists of a deterministic (exact) and a stochastic component. The deterministic component can be used to optimise time lines of tasks which use the same resource, provided the order of the tasks is known. The stochastic part is based on a relatively new heuristic which uses a power-law distribution to determine its next move, enabling it to perform well on multi-modal fitness landscapes. The chosen neighbourhood proved to scale
surprisingly well with the number of aircraft, solving unrealistically chaotic
instances of up to 24 aircraft. Safety reasons will dictate that realistic sched-
ules as encountered at airports will be less complex, if somewhat more ad-hoc.

The datasets airland6 and airland7 are good examples for the limitations
of this approach. Providing few aircraft to the active window, they are un-
suitable for combinatorial optimisation in the dynamic context. The situations
describe, however, do not seem altogether unrealistic. To accommodate
them, the Controller would ideally hand active windows of very small size to a
strategy-based solver, as it is likely that the ATC would prescribe a policy for
these cases.

Acknowledgments

Special thanks go to my supervisor Professor Tim Hendtlass for not only
supporting my efforts in many ways but giving me a free hand in directing my
research.

References

simple model of evolution, Physics Review Letters, Vol. 74, pp. 4083–
4086.

[2] Balakrishnan H., Chandran B., 2006 Scheduling Aircraft Landings un-
der Constrained Position Shifting, Proceedings of the AIAA Guidance,

uk/~mastjjb/jeb/orlib/)

Scheduling Aircraft Landings - the Static Case, Transportation Science,
Vol. 34, pp. 180–197.

Displacement Problem and Dynamically Scheduling Aircraft Landings,


