An optimal data-splitting algorithm for aircraft scheduling on a single runway to maximize throughput

Prakash, Rakesh; Piplani, Rajesh; Desai, Jitamitra

2018

Prakash, R., Piplani, R., & Desai, J. (2018). An optimal data-splitting algorithm for aircraft scheduling on a single runway to maximize throughput. *Transportation Research Part C: Emerging Technologies*, 95, 570-581. doi:10.1016/j.trc.2018.07.031

<https://hdl.handle.net/10356/136894>

<https://doi.org/10.1016/j.trc.2018.07.031>

© 2018 Elsevier Ltd. All rights reserved. This paper was published in *Transportation Research Part C: Emerging Technologies* and is made available with permission of Elsevier Ltd.

Downloaded on 26 Apr 2023 22:29:55 SGT

1 An Optimal Data-Splitting Algorithm for Aircraft Scheduling
2 on a Single Runway to Maximize Throughput

3 **Rakesh Prakash, Rajesh Piplani*, Jitamitra Desai**

4 School of Mechanical and Aerospace Engineering

5 Nanyang Technological University

6 50 Nanyang Avenue, Singapore 639798

7 *Corresponding Author: piplani@pmail.ntu.edu.sg

9 In this research, we present a data-splitting algorithm to optimally solve the aircraft
10 sequencing problem (ASP) on a single runway under both segregated and mixed-mode of
11 operation. This problem is formulated as a 0–1 mixed-integer program (MIP), taking into
12 account several realistic constraints, including safety separation standards, wide time-
13 windows, and constrained position shifting, with the objective of maximizing the total
14 throughput. Varied scenarios of large scale realistic instances of this problem, which is
15 NP-hard in general, are computationally difficult to solve with the direct use of commercial
16 solver as well as existing state-of-the-art dynamic programming method. The design
17 of the algorithm is based on a recently introduced data-splitting algorithm which uses
18 the divide-and-conquer paradigm, wherein the given set of flights is divided into several
19 disjoint subsets, each of which is optimized using 0–1 MIP while ensuring the optimality
20 of the entire set. Computational results show that the difficult instances can be solved in
21 real-time and the solution is efficient in comparison to the commercial solver and dynamic
22 programming, using both sequential, as well as parallel, implementation of this pleasingly
23 parallel algorithm.

24 **Keywords**

25 Aircraft sequencing problem; 0–1 mixed-integer programming; asymmetric traveling salesman
26 problem with time-windows; data-splitting algorithm; runway throughput; runway optimiza-
27 tion

1 Introduction

Rising air traffic demand and the resulting stress on the entire ATM system (acronyms' meaning are listed in Table 1) is costing airlines, passengers, and the overall economy several billions of dollars each year (Airlines for America 2017). While injecting additional capacity into the ATM system through infrastructure development can relieve the stress placed on the system, over the short run the existing resources can be more efficiently operated by managing the bottleneck operations related to arrivals, departures, runways, and taxiways (Soomer and Franx 2008, Anagnostakis et al. 2001). Commonly referred to in the literature as the ASP, it entails optimally sequencing and scheduling flight operations on runways which constitute a key bottleneck in the TMA of an airport. Specifically, in the static version of this problem, given a set of aircraft, along with information on the earliest/latest operation time for each aircraft (be it an arrival or a departure), and the minimum safety regulations to protect trailing aircraft from wake vortices generated by the leading aircraft, the objective is to determine a sequence (that optimizes a predefined objective) while simultaneously achieving safety, efficiency, and equity in the ATM system. Passenger safety is achieved by maintaining the required separation between aircraft; runway efficiency is equivalent to achieving low average delay or high throughput; and airline equity is modeled by implementing a CPS strategy wherein an aircraft cannot be shifted by more than k positions (the so-called MPS parameter) from its initial FCFS-based position.

(Place Table 1 here.)

In literature there exist many instances of the ASP (Ghoniem et al. 2014, Beasley et al. 2000, Ghoniem et al. 2015, Briskorn and Stolletz 2014, Lieder et al. 2015, Benlic et al. 2016, Sam et al. 2017, Sölveling and Clarke 2014, Ahmed et al. 2017) which vary depending

51 upon the number of runways (single or multiple), mode of runway operations (segregated or
 52 mixed), problem objective (minimizing delay, maximizing throughput, or minimizing cost),
 53 constraints, such as inclusion of time windows, width of time-windows (narrow or wide), per-
 54 missibility of early landings, CPS inclusion, and solution methods (exact or heuristics). An
 55 extensive literature survey on the problem ASP can be found in Prakash et al. (2017) and Ben-
 56 nell et al. (2011). In this work, which considers the static ASP on a single runway, we formulate
 57 and *exactly* solve a particular version of ASP defined in Balakrishnan and Chandran (2010)
 58 as that is the only work to incorporate all practical constraints including CPS, wide time-
 59 windows, precedence constraints, and early landings/departures. Further, they also consider
 60 the very important objective function of maximizing throughput (or equivalently minimizing
 61 the makespan). Being a potential candidate for deployment in arrival-departure management
 62 systems, this problem has great practical implications. Nonetheless being equivalent to NP-
 63 hard ATSP (Psaraftis 1980), obtaining optimal solutions in real-time of large-scale realistic
 64 instances opens it for the investigation.

65 However, exploiting one or the other problem characteristics of the ASP, *dynamic program-*
 66 *ming* provides an efficient solution for a specific set of scenarios. Assuming that the aircraft
 67 from the same category are identical, all aircraft are ready to land together, and ignoring the
 68 time-window considerations, Psaraftis (1980) provides the first efficient dynamic programming
 69 implementation building on the fact that the number of aircraft categories is limited. His algo-
 70 rithm is exponential in number of categories and polynomial in the number of aircraft and can
 71 provide a *tractable* solution with or without CPS requirements, under the aforementioned sce-
 72 nario. However, considering that the above formulation cannot accommodate time-windows
 73 and precedence restrictions—the two important practical constraints—another DP algorithm
 74 is offered by Balakrishnan and Chandran (2010) whose implementation is exponential in k

(MPS parameter) and polynomial in n (instance size). Even though their formulation includes most of the practical constraints and is *tractable* under the segregated mode of operations for values of k up to 3, it may face the curse of dimensionality in mixed-mode of operations as violation of triangular inequality for some of the aircraft triplets requires time to be included in the state-space. Furthermore, increasing the value of k above 3 renders it intractable. In addition, of all possible optima, DP approach cannot provide a specific minimum makespan schedule that leads to associated minimum delay¹; this requires total delay to be included in the objective function which renders DP computationally prohibitive. Table 2 compares the total delay (averaged over ten randomly generated instances of size 30) for the optimal schedule when maximizing throughput is the objective, where the first column represents the delay when total delay is part of the objective function, and second column represents the delay when it is not; notably, without any loss of makespan, around 2 minutes of delay is saved in the former case. The above factors motivate the development of a *tractable* approach which is more *flexible*. Note that tractability here refers to the ability to solve problems of realistic size in solution times that are appropriate for the given application—not usual polynomial solvability; whereas flexibility refers to tractability over different problem scenarios.

(Place Table 2 here.)

Next, we revisit the formulation of this version of ASP as a 0–1 MIP, an adaptation of Beasley et al. (2000)’s model, enhanced with the inclusion of the CPS constraints. Recognizing that a more flexible approach over current state-of-the-art is needed, this paper proposes an extension to *data-splitting algorithm* (**DS-ASP**) originally proposed in Prakash et al. (2017), that optimizes the flight sequences by a repeated application of the 0–1 MIP on smaller data sets while ensuring the global optimality, and demonstrate the efficacy of the approach.

¹There can be several optima with the same optimal makespan but different values of delay.

(See Figure 1 for the schematic representation of DS-ASP.) Another important and intrinsic feature of the proposed data-splitting framework is that it is pleasingly parallelizable thereby making it amenable to rapidly scalable computations. Note that DS-ASP uses LP-based branch-and-cut approach to solve MIPs.

(Place Figure 1 here)

With the aim to reduce the solution search space, DS-ASP algorithm is based on splitting the given flight data-set into several pairs of leading and following sets and independently optimizing these pairs of sets, while preserving the global optimality. In the process of ensuring global optimality, we have to capture the effect of optimization of the leading set onto the following set, which is not straightforward. The intelligence to do this is developed from analysis of the optimal solution of several instances of leading and following sets, leading to some key insights that help in formulating the optimization problems of the leading and the following sets. Further, we also prove that the resulting problem formulation leads to optimality.

There are four significant contributions that stem from this work:

1. The proposed data-splitting algorithm is more flexible than state-of-the-art DP approach for the real-sized instances.
2. To the best of our knowledge, this is the first LP-based branch-and-cut approach to provide a tractable solution of the problem under consideration. The branch-and-cut solutions for the ASP have been provided by Beasley et al. (2000), Ghoniem et al. (2015); however they are intractable for $k \geq 2$. (Note that CPS is incorporated separately as original formulations ignore it.)
3. It is a potential candidate algorithm to be included in the arrival-departure management systems used in practice since the LP-based branch-and-cut algorithm is able to account for

most of the operational constraints.

4. It motivates the use of DS-ASP for various scheduling problems to obtain the high quality solution in real time where CPS constraints may not be present.

The remainder of this paper is organized as follows. In Section 2, we present the 0–1 MIP formulation for the ASP along with some preliminary computations that serve to highlight the inherent difficulty in solving ASP via a straightforward MIP approach. Next, in Section 3, the details of the data-splitting algorithm, and its pseudo-code are described, alongwith algorithmic insights and routines to enhance the computational performance. Then, Section 4 presents detailed computational results and analysis pertaining to large-scale (realistic) instances, and finally, Section 5 summarizes the contributions of this work and suggests extensions for future research.

2 ASP Optimization

The ASP is formulated as a 0–1 mixed integer program, where the index sets and parameters are as defined below.

Description of Index Sets and Parameters

- $\mathcal{F}$: Set of all arriving and departing flights.
- E_i : Earliest time of arrival (departure) of aircraft i .
- L_i : Latest time of arrival (departure) of aircraft i , where $L_i > E_i$.
- T_i : Target time of arrival (departure) of aircraft i , where $E_i \leq T_i \leq L_i$.
- Δ_{ij} : Mandatory minimum safety separation (in seconds) enforced at the runway threshold, if flight i is ahead of flight j (see Table 3).

143 • $seq_S(i)$: Position of flight i in any feasible sequence S .

144 • k : Maximum position shifting (MPS) parameter.

145 **Decision Variables**

$$x_{ij} = \begin{cases} 1, & \text{if flight } i \text{ is ahead of flight } j \text{ in sequence} \\ 0, & \text{otherwise} \end{cases}$$

146 t_i = Scheduled time of arrival/departure of flight i .

147 z = Runway makespan.

$$\text{ASP :} \quad \text{Minimize} \quad z \quad (1)$$

$$\text{subject to:} \quad x_{ij} + x_{ji} = 1, \quad \forall i < j, (i, j) \in \mathcal{F} \quad (2)$$

$$t_j \geq t_i + \Delta_{ij} - M(1 - x_{ij}), \quad \forall (i, j) \in \mathcal{F} \quad (3)$$

$$i - k \leq (F - \sum_{j \in \mathcal{F}, i \neq j} x_{ij}) \leq i + k, \quad \forall i \in \mathcal{F} \quad (4)$$

$$z \geq t_i \quad \forall i \in \mathcal{F} \quad (5)$$

$$E_i \leq t_i \leq L_i, \quad \forall i \in \mathcal{F} \quad (6)$$

$$x_{ij} \in \{0, 1\} \quad \forall (i, j) \in \mathcal{F}; \quad t_i \geq 0 \quad \forall i \in \mathcal{F}; \quad z \geq 0. \quad (7)$$

148

149 *(Place Table 3 here.)*

150 In the above formulation, the objective function (1) seeks to minimize the makespan (comple-
 151 tion time); constraint (2) enforces the order precedence relationship between flights i and j ;
 152 constraint (3) ensures the time-based separation requirements between flights at the runway
 153 (as mandated by the FAA for various flight classes (see Table 3), where $M \equiv (L_i + \Delta_{ij} - E_j)$;
 154 constraint (4) imposes the CPS constraint that an aircraft cannot be shifted by more than
 155 ‘ k ’ positions from its initial (FCFS) position, where for the sake of notational convenience,
 156 we let $seq_{\text{FCFS}}(i) \equiv i, \forall i \in \mathcal{F}$, i.e., aircraft are labeled based on their FCFS priority order,

and $F \equiv |\mathcal{F}|$; constraint (5) ensures that makespan z must be greater or equal to scheduled time of last aircraft; constraint (6) maintains the scheduled time of arrival (departure) at the runway to be between the earliest and latest times for each aircraft; and finally constraint (7) imposes binary and non-negativity restrictions on the x -, t -, and z variables, respectively.

Note that ASP formulation described above can be further improved by using variable-fixing strategies based on CPS (Proposition 1; Prakash et al. (2017)) and ELW rule (Proposition 2; Briskorn and Stolletz (2014)) and the addition of valid inequalities that serve to tighten the underlying linear programming representation of Desai and Prakash (2016), and Ghoniem et al. (2015). However, since these valid inequalities are found to require more computational resources on average, in our discussion below we apply only variable fixing strategies and ignore other model enhancements.

PROPOSITION 1: *For a pair of aircraft $(i, j) \in \mathcal{F}$, if the FCFS sequence positions of i and j satisfy the condition: $j - i \geq 2k$, where k is the MPS parameter, then set $x_{ij} = 1$.*

PROPOSITION 2: (ELW Rule) *Consider a pair of aircraft $(i, j) \in \mathcal{F}$ that belong to the same category. If the time-window restrictions satisfy the following conditions: (i) $E_i \leq E_j$; and (ii) $L_i \leq L_j$; then set $x_{ij} = 1$.*

Precedence constraints which fix the precedence order of aircraft arriving from the same jet-routes are not explicitly included in the formulation as they are airport specific and can be easily accounted for by fixing the variables based on input data. Further, if an airport administers precedence constraints, then ELW rule should not be implemented as it may lead to sub-optimality.

To gain an understanding of the computational difficulties involved in solving the ASP, we tested the optimization model, given by (1)–(7), enhanced with the variable fixing strategies outlined in Propositions 1-2, on several randomly generated large-scale (realistic) instances.

From the results (Table 4), it can clearly be seen that the computational time increases exponentially and the MIP model fails to converge to optimality, particularly for cases where the maximum position shifting parameter $k \geq 2$.

(Place Table 4 here.)

3 Data-Splitting Algorithm

DS-ASP is implemented in two stages. The first stage is instance generation, in which the original flight data-set is divided into several possible pairs of disjoint subsets of leading and following aircraft. The second stage is the solution stage, in which each resulting subset pair is independently solved to determine its optimal solution. The obtained solutions are finally compared to get the overall optimum. The primary motivations behind this algorithm are:

- (i) solving smaller data-set is computationally more efficient as compared to larger data-sets;
- (ii) as a result of the CPS constraint, the number of pairs of leading and following data-sets is limited to the order of $\binom{2k}{k}$, where k is value of the MPS parameter, and is *independent* of the total number of flights n ;
- (iii) search space is significantly reduced using this strategy (Prakash et al. 2017);
- (iv) prowess of commercial solver in converging to optimality for smaller-sized problems; and
- (v) the algorithm is pleasingly parallel.

Even though the basic framework of DS-ASP stays the same, the procedure varies from application to application. In the case of minimizing makespan of a single runway, the instance generation scheme is the same as in minimizing delay on a single runway (Prakash et al. 2017), however, the solution procedure would be different as described in Section 3.2. Moreover, an effective algorithmic enhancement routine and multiple splitting of the data-set (in order to handle large sized instances) are new features of DS-ASP discussed below; these

routines are designed to exploit a characteristic of minimizing makespan function that it is a *separable* objective function (Psaraftis 1978), i.e., the optimal makespan of a given sequence can be determined from the optimal makespan of its subsequences, and this property remains unaffected in presence of the given problem-constraints (2)-(7), as proved in Proposition 3.

A detailed description of different stages of the algorithm is given below.

3.1 Preparation of Instance Pairs

Let n denote the number of aircraft positioned at $\{1, \dots, m-1, m, m+1, \dots, n\}$ in FCFS order. After splitting this flight data into two (equal when n is even) subsets, the leading set A and the following set B are composed of aircraft at positions $\{1, \dots, m\}$ and $\{m+1, \dots, n\}$, respectively. Now due to the CPS constraint, only aircraft between $(m-k+1)$ and $(m+k)$ positions can crossover between the leading and the following sets; thus, there are only $\binom{2k}{k}$ such pairs of leading and following aircraft sets. Note that aircraft in the following set have to maintain the required safety separation from the last aircraft in the leading set, hence both the *aircraft type* and the *arrival time* of the last scheduled flight in the leading set affect the makespan of the following set. Therefore, the number of pairs is further expanded by the combinations denoted as $(\dot{A}, B)$ resulting after fixing all the aircraft in the leading set A that can possibly occurs at the last position, i.e., the aircraft between positions $m-k$ to m ; these $(\dot{A}, B)$ pairs will finally be used by the algorithm. (Effect of arrival time is accounted in the solution stage.)

The (A, B) pairing scheme is illustrated in Figure 2, wherein a data-set consisting of eight flights is split at $m = 4$ resulting in two sets of size four each. The k is set to two and hence the aircraft between Position 3 to Position 6 can crossover resulting in six (A, B) pairs, e.g., $A = \{H1, H2, L3, L4\}$, $B = \{L5, L6, H7, S8\}$; $A = \{H1, H2, L3, L5\}$, $B =$

226 $\{L4, L6, H7, S8\}$; $A = \{H1, H2, L4, L6\}$, $B = \{L3, L5, H7, S8\}$. Now consider an (A, B)
 227 pair $A = \{H1, H2, L3, L4\}$, $B = \{L5, L6, H7, S8\}$; the resulting $(\dot{A}, B)$ pairs after fixing
 228 the aircraft between positions 2 to 4 at the last position (denoted as $\bullet$) in set A of this
 229 example pair are: $\dot{A} = \{H1, H2, L3, \dot{L4}\}$, $B = \{L5, L6, H7, S8\}$; $\dot{A} = \{H1, H2, L4, \dot{L3}\}$, $B =$
 230 $\{L5, L6, H7, S8\}$; and, $\dot{A} = \{H1, L3, L4, \dot{H2}\}$, $B = \{L5, L6, H7, S8\}$.

231 *(Place Fig 2 here.)*

232 Furthermore, if ELW rule is applicable and in case pair (A, B) or $(\dot{A}, B)$ is found to violate
 233 it, then those pairs can be discarded, e.g., pair $A = \{H1, H2, L3, L5\}$, $B = \{L4, L6, H7, S8\}$
 234 can be discarded as flight $L4$ is scheduled behind $L5$.

235 **Modifying for Mixed Traffic:** In case of arriving traffic only, the satisfaction of trian-
 236 gular inequality with respect to the safety separation distance requires that the first aircraft
 237 in set B be separated only from the last aircraft in $\dot{A}$; however, triangular inequality in case
 238 of mixed-traffic may be violated for the following instances:

239 Case 1: Of the last two flights i and j from set $\dot{A}$, the second last is a heavy arrival, followed
 240 by a departure of any aircraft type. That is then followed by a large arrival k (from set B).

241 Case 2: Of the last two flights i and j from set $\dot{A}$, the second last is a heavy arrival, followed
 242 by a departure of any aircraft type. That is then followed by a small arrival k (from set B).

243 Based on the given safety separation matrix (Table 3)), since $\Delta_{ij} + \Delta_{jk} = 135$ seconds and
 244 $\Delta_{ik} = 157$, violation (for 22 seconds) takes place in the first case, and 61 seconds in the latter.

245 In order to prevent these violations, the original pair $(\dot{A}, B)$ consisting of a departure at last
 246 position in $\dot{A}$ is replaced with two new pairs, along with the necessary separation constraints.

247 Heavy aircraft at the second last position in $\dot{A}$ is fixed in the first pair, and the second pair
 248 does not allow any heavy aircraft at this position; there is no change for flights in set B .

3.2 Solving the instance pairs

Here, first the optimization problems for sets $\dot{A}$ and B are framed and then it is proved in

Proposition 3 that solving these problems lead to optima.

$$Opt_{\dot{A}} \equiv \min \left\{ z, \text{ s.t. } (2) - (7), \forall i \in \dot{A} \right\}.$$

$$Opt_B \equiv \min \left\{ z, \text{ s.t. } (2) - (7), t_i \geq t_{\dot{l}} + \Delta_{\dot{l}i}, \forall i \in B \right\}, \text{ where } \dot{l} \text{ is the last aircraft in } \dot{A} \text{ from}$$

which aircraft in the following set have to maintain the minimum separation. In case of

mixed-traffic, extra set of constraints $(t_i \geq t_{\dot{l}} + \Delta_{\dot{l}i}, \forall i \in B)$ for separation from second last

aircraft must also be included when a heavy arrival is fixed at the second last position in $\dot{A}$.

PROPOSITION 3: *Solving $Opt_{\dot{A}}$ and Opt_B optimizes the given pair $(\dot{A}, B)$.*

PROOF. Consider the constraint $t_i \geq t_{\dot{l}} + \Delta_{\dot{l}i}, \forall i \in B$ which captures the effect of last aircraft

and makespan of leading set on the following set. It should be noted that this is the only

constraint dependent on $Opt_{\dot{A}}$; CPS and time-windows requirements are independent of $Opt_{\dot{A}}$.

Further, $t_{\dot{l}}$ is the only quantity that controls the solution space of Opt_B as $\dot{l}$ is fixed. Since,

minimizing $t_{\dot{l}}$ relaxes the solution space of Opt_B , and vice-versa, (see Fig 3 for illustration),

optimizing $Opt_{\dot{A}}$ and subsequently optimizing Opt_B leads to optima of the given pair $(\dot{A}, B)$.

Hence proved. (Note that optima of $(\dot{A}, B)$ is same as optima of B).

(Place Fig 3 here.)

The pseudo-code of the above data-splitting algorithm (splitting at single point) is listed

next.

Algorithm 1: Pseudo-code for single-split

1: Set incumbent $\leftarrow +\infty$, $\Pi \leftarrow \emptyset$, and $\mathcal{P} = \{\text{set of all pairs } (\dot{A}, B)\}$.

2: If $\mathcal{P} = \emptyset$, stop; return incumbent and Π as the best solution. Else, go to Step 3.

3: Arbitrarily select one instance pair $(\dot{A}, B)$ from $\mathcal{P}$.

- 273 4: Solve $Opt_{\dot{A}}$. Let $\dot{l}$ and t_i be the last scheduled aircraft and its time of arrival/departure.
- 274 5: Solve Opt_B . Let $\hat{z}$ be the optimal solution.
- 275 6: Update $\mathcal{P} \leftarrow \mathcal{P} \setminus \{\dot{A}, B\}$. If $\hat{z} < \text{incumbent}$, set $\text{incumbent} \leftarrow \hat{z}$ and update Π with the
- 276 optimal solution of pair $(\dot{A}, B)$; go to Step 2.
-

277 3.3 Algorithmic Enhancements by fathoming “sibling” instance pairs

Consider two instance pairs $(\dot{A}_1, B)$ and $(\dot{A}_2, B)$, resulting from the same parent pair (A, B) . Let $\hat{\mathcal{F}}$ represent the set of flights in set B between positions $(m+1)$ to $(m+k+1)$, and let $(\dot{z}_1, \dot{l}_1)$ and $(\dot{z}_2, \dot{l}_2)$ denote the completion time, and last flight in the optimal solutions for set $\dot{A}_1$ and $\dot{A}_2$, respectively. Furthermore, suppose that Set B has been solved to optimality in pair $(\dot{A}_1, B)$, with flight $f \in \hat{\mathcal{F}}$ placed at the first position in the optimal solution of set B , and $\hat{z}$ is the optimal throughput of pair $(\dot{A}_1, B)$. Now, consider the following cases:

Case 1. Suppose that flight f is bound to the first position in any optimal solution of set B . Calculate,

$$s = (\dot{z}_2 - \dot{z}_1) + (\Delta_{i_2 f} - \Delta_{i_1 f}) \quad (8)$$

278 where s measures the difference in start times for the first flight f in set B between pairs $(\dot{A}_1,$
 279 $B)$ and $(\dot{A}_2, B)$.

Case 2. If flight f is not bound to first position in any optimal solution of set B , then we need to determine the minimum difference in the start times with respect to all the flights which can come at the first position in set B . Towards this end, calculate,

$$s = (\dot{z}_2 - \dot{z}_1) + \min_{\bar{f} \in \hat{\mathcal{F}}} (\Delta_{i_2 \bar{f}} - \Delta_{i_1 \bar{f}}) \quad (9)$$

Now, consider the following conditions on s :

1. $s \geq 0$. Under this scenario, we can forgo optimizing set B in the second (sibling) pair, since the optimal value of the second pair is at least equal to the optimal value of the first pair.

2. $s < 0$. Careful examination of optimal makespan $\hat{z}$ reveals that

$$\hat{z} = \dot{z}_1 + \Delta_{i_1 f} + \bar{z}_f \quad (10)$$

where $\bar{z}_f$ is the makespan length corresponding to set B with flight f at the first position. Since, flight f corresponds to the optimal makespan, clearly for other flight $\bar{f}$ which can occupy first position in set B ,

$$\begin{aligned} \hat{z} \leq \dot{z}_1 + \Delta_{i_1 \bar{f}} + \bar{z}_{\bar{f}} &\Rightarrow \bar{z}_{\bar{f}} \geq \hat{z} - \dot{z}_1 - \Delta_{i_1 \bar{f}}. \\ &\Rightarrow \bar{z}_{\bar{f}} \geq \bar{z}_f + \Delta_{i_1 f} - \Delta_{i_1 \bar{f}}. \\ &\Rightarrow \bar{z}_{\bar{f}}^{min} = \bar{z}_f + \Delta_{i_1 f} - \Delta_{i_1 \bar{f}}. \end{aligned}$$

Therefore, the lower bound on optimal makespan of pair $(\dot{A}_2, B)$ is given as,

$$\min_{\bar{f} \in \bar{\mathcal{F}}} (\dot{z}_2 + \Delta_{i_2 \bar{f}} + \bar{z}_{\bar{f}}^{min}) \quad (11)$$

280 These lower bounds are compared at the end with minimum of all the makespan for which
 281 optimality has been concluded. If the former is smaller than the latter, optimization is run
 282 for the corresponding pair, else, it is aborted. Note that this algorithmic enhancement is only
 283 applicable in the scenario where separation of flights in the following set is only needed from
 284 the last flight in the leading set which may not be true for some instance-pairs in case of
 285 mixed traffic.

The pseudo-code of enhanced data-splitting algorithm is listed next.

Algorithm 2: Enhanced data-splitting algorithm

- 1: Set incumbent $\leftarrow +\infty$, $\Pi \leftarrow \emptyset$, and $\mathcal{S} = \{\text{set of all pairs } (A, B)\}$.
 - 2: If $\mathcal{S} = \emptyset$, go to final Step; else, go to Step 3.
 - 3: Arbitrarily select one instance pair (A, B) from $\mathcal{S}$, set $flag \leftarrow 0$. Let $\dot{\mathcal{S}} = \{\text{set of all pairs } (\dot{A}, B)\}$ corresponding to (A, B) .
 - 4: If $\dot{\mathcal{S}} = \emptyset$, update $\mathcal{S} \leftarrow \mathcal{S} \setminus \{A, B\}$, go to Step 2. Else, arbitrarily select one instance pair $(\dot{A}, B)$ from $\dot{\mathcal{S}}$.
 - 5: Solve $Opt_{\dot{A}}$. Let $\dot{l}$ and $t_{\dot{l}}$ be the last scheduled aircraft and its arrival/departure time. If $flag = 0$ or separation is also needed from second last aircraft in $\dot{A}$, set $flag = 1$ if only former condition is satisfied and go to Step 6; else, go to Step 7.
 - 6: Solve Opt_B . Let $\hat{z}$ be the optimal solution. Update $\dot{\mathcal{S}} \leftarrow \dot{\mathcal{S}} \setminus \{\dot{A}, B\}$. If $\hat{z} < \text{incumbent}$, set incumbent $\leftarrow \hat{z}$ and update Π with the optimal solution of pair $(\dot{A}, B)$, go to Step 4.
 - 7: Calculate s . If $s \geq 0$, set $\hat{z} \leftarrow +\infty$; else, calculate the lower bound and save it in memory. Update $\dot{\mathcal{S}} \leftarrow \dot{\mathcal{S}} \setminus \{\dot{A}, B\}$, go to Step 4.
 - 8: For each saved entry: If lower bound is $\geq$ incumbent, discard it; else solve Opt_B and update incumbent and Π in case $\hat{z} < \text{incumbent}$. Return incumbent and Π as the best solution.
-

3.4 Multiple Splits

When handling very large sized instances, such as $n = 60$, splitting the data set once may not lead to computationally efficient solutions as each resulting set is still large. One way to address this is multiple splitting wherein a data-set is split into groups of more than two subsets; however, the challenge then is to obtain optimality through independent solutions

of several subsets. Fortunately, the independent solutions with the objective of minimizing makespan automatically lead to optima as minimizing the makespan of a preceding set also maximizes the search space of the following set.

In case of multiple splits, data-splitting algorithm is applied recursively: first a data-set is split into two set resulting in several instance pairs; subsequently each subset of a pair is split again if its cardinality is more than a user specified minimum, say ϕ . A comparatively higher degree of parallelization can be achieved with multiple splits, thus solving any instance, however large, in real time. For the sake of comprehension, we ignore the algorithmic enhancements of Section 3.3 in the listing of pseudo-code of the multiple splitting algorithm that follows.

Algorithm 3: Pseudo-Code for multiple-split

Initialize: $\mathcal{S} = \mathcal{F}, t_i = 0, \dot{l} | \Delta s_{i\dot{l}} = 0 \forall i \in \mathcal{F}$. Optimal Decision = DATA-SPLITTING ($\mathcal{S}, t_i, \dot{l}$)

function DATA-SPLITTING ($\mathcal{S}, t_i, \dot{l}$)

- 1: Set incumbent $\leftarrow +\infty$, $\Pi \leftarrow \emptyset$, and split $\mathcal{S}$ into $\mathcal{P} = \{\text{set of all pairs } (\dot{A}, B)\}$ with instance sizes n_1 and n_2 .
- 2: If $\mathcal{P} = \emptyset$, stop; return incumbent as the best solution. Else, go to Step 3.
- 3: Arbitrarily select one instance pair $(\dot{A}, B)$ from $\mathcal{P}$.
- 4: If $n_1 \geq \phi$, call DATA-SPLITTING ($\dot{A}, t_i, \dot{l}$); else, solve $Opt_{\dot{A}}$ with extra constraints $t_i \geq t_i + \Delta_{i\dot{l}} \forall i \in \dot{A}$. Let $\dot{l}$ and t_i be the last scheduled aircraft and its time of arrival/departure.
- 5: If $n_2 \geq \phi$, call DATA-SPLITTING ($B, t_i, \dot{l}$); else, solve Opt_B . Let $\hat{z}$ be the optimal solution.
- 6: Update $\mathcal{P} \leftarrow \mathcal{P} \setminus \{\dot{A}, B\}$. If $\hat{z} < \text{incumbent}$, set incumbent $\leftarrow \hat{z}$ and update Π with the optimal solution of pair $(\dot{A}, B)$; go to Step 2.

4 Computational Results

In this section, with a prototype implementation, we showcase the computational behavior of DS-ASP for various large-sized realistic instances under various modes of operations. Furthermore, we demonstrate its prowess over ASP and DP in real-time handling of instances under various scenarios.

The experimental design is similar to Balakrishnan and Chandran (2010), and mimics the realistic scenario of peak traffic condition as follows: (i) aircraft are randomly generated with mixture of 40% Heavy + 40% Large + 20% Small; (ii) considering the scheduling horizon of one hour and peak traffic conditions, instance sizes up to 40 are chosen for all the computational testing and comparisons, while instance size of 60 is chosen to demonstrate the benefit of multiple split; (iii) target times are randomly generated, assuming that each aircraft probabilistically appears *uniformly* once every γ seconds where γ is set to 50 or 60 seconds; (iv) for mixed mode scenarios, 50% are arrivals and 50% are departures; and (v) earliest time of arrival for each flight is set to be 60 seconds before the target times as it is most economic for the arriving flight (Neuman and Erzberger 1991), and latest time is set to be 1800 seconds past its target time due to fuel restrictions limit. The computational experiments are run on a Windows machine, equipped with a Intel(R) Xeon(R) CPU E5-1630 3.70GHz 32GB RAM processor, using MATLAB in conjunction with GuRoBi 6.5.2 (www.gurobi.com) as the underlying MIP-solver. The time-limit (TL) is set as 3600 seconds, and each computation represents the time in seconds averaged over ten randomly generated instances. In the following discussion, we describe the computational results for arrival traffic first, followed by mixed traffic and finally present the scheme to use DS-ASP in scenarios without CPS restrictions.

4.1 Arrival Traffic Only

Computations recorded in Table 5 compare the performance of DS-ASP on arrival traffic with DP and directly solving ASP (1)-(7) (including variable fixing strategies). We can make two key observations from Table 5: (i) DS-ASP outperforms ASP for any k ; (ii) DP outperforms DS-ASP for $k \leq 2$, but not for $k = 3$ and 4; DP faces dramatic increase in computational time for $k = 4$. The reasons behind the DS-ASP's superior performance over ASP are: (i) a significant reduction in search space; (ii) the solver closing the gap more efficiently for smaller instances (verified by the consistency in solution time); and (iii) time taken to solve the LP-relaxation at each node is significantly lower in case of smaller instances. With regards to DP, the smaller state space in case of $k = 1$ or $k = 2$ renders DP remarkably efficient; however, $k \geq 3$ explodes the state space of DP. The computational performance of DS-ASP is further enhanced (Table 6) through a simple parallel implementation.

(Place Table 5 and Table 6 here.)

Table 7 illustrates the computational benefits of multiple splits. For the example scenarios of instance size 60, first set of computations are performed by splitting it into two equal data-sets of size 30, and second set by splitting it into four data-sets of size 15 each. Two important observations can be made from the results of Table 7. Firstly, a solution can be obtained in real time by multiple splits. Secondly, the time savings are remarkable over single split, e.g. over 98% for instance size of 60, making DS-ASP even better than DP for $k = 3$.

(Place Table 7 here.)

4.2 Mixed Traffic

Table 8 records the time taken by DS-ASP in solving instances of mixed traffic for value of k up to 3 in comparison to ASP. We do not compare the performance with DP in this case as DP discretizes the time-interval in handling mixed traffic, whereas DS-ASP handles continuous time. In addition, even after discretization, DP may have to evaluate a rather large number of states in case of mixed traffic, and is expected to perform rather poorly (the computations for mixed traffic scenario is not presented in the Balakrishnan and Chandran (2010)). Notably, ASP is efficient in handling mixed-traffic scenario when $\gamma \geq 60$; the optimal average separation time is close to 60 seconds in case of mixed traffic, therefore $\gamma \geq 60$ provides a very tight lower bound helping branch-and-cut tree to converge quickly. Further, as recorded in Table 8, the computational time can be significantly reduced by virtue of parallelization.

(Place Table 8 and Table 9 here.)

4.3 DS-ASP applied in case of Unconstrained Position Shifting

Aircraft sequencing problem can be viewed as a job shop scheduling problem with release times and sequence-dependent processing times (but zero setup times) or sequence-dependent setup times (but zero processing times) (Beasley et al. 2000, Bianco et al. 2006), where release times in job shop are equivalent to earliest times of arrival in ASP, and sequence-dependent processing/setup times to safety separation requirements. Therefore, there is a promising avenue for the DS-ASP to be applied for various other important scheduling problems in transportation or other areas some of which are also listed in Lahmar and Benjaafar (2007). However, CPS constraints may not always be imposed for these scheduling problems, and solving the DS-ASP to optimality in case of unconstrained position shifting (equivalent to

very large value of k) is not computationally viable.

But, as we show next, we can possibly solve an unconstrained sequence dependent scheduling problem using data-splitting algorithm with imposition of k , say $k = 10$, and achieve an objective function value close to optimality. There are two drivers behind this. First, makespan tends to converge as k increases. Second, empirically, DS-ASP is efficient in handling realistic instances of ASP for k up to 10. To illustrate this, we record the makespan decrease in Fig 4 and computation time increase in Fig 5 with increasing k (averaged for 10 instances of size 30 and 40). For instance size 30, average makespan of 2556 seconds is obtained in just 74 seconds; whereas, directly solving it using solver takes 1156 seconds on average to converge. Notably, 2556 seconds is also the average optimal value. Similarly, for an instance size of 40, average makespan of 3397 seconds is obtained in 544 seconds; whereas in the preset time limit of 3600 seconds, the best upper bound obtained using solver is 3431 (> 3397) seconds.

(Place Fig 4 and Fig 5 here.)

5 Conclusions

In this paper, a data-splitting algorithm for ASP on a single runway with the objective of maximizing throughput is proposed which provides real-time optimal solutions for the large scale realistic instances; a first LP-based branch-and-cut approach to do so. Existing solution approaches, including dynamic programming, and direct application of commercial solvers are intractable under wide range of realistic scenarios. Under the data-splitting algorithm, the original flight data-set is first decomposed into several potential pairs of equal-sized leading and following aircraft sets. The optimal solution is then developed by application of the

0–1 MIP on the smaller datasets. Various procedures are prescribed to further enhance the performance of the algorithm in form of fathoming the sibling instance pairs and multiple splits. Our computational experiments demonstrate that an LP based branch-and-cut approach, applied on the above model, outperforms dynamic programming-based approaches in handling different problem scenarios, including that of mixed-traffic. The algorithm is easy to implement, and the run times are sufficiently small to enable real-time deployment. Finally, an extension is suggested to solve the various job shop scheduling problems which may not be restricted by CPS.

There can be multiple directions of future research. Application of data-splitting algorithm on other variants of ASP on a single runway and analyzing the run time behavior with respect to standard solvers and existing solutions could be one such direction. The algorithm can also be extended to multiple runway scenario; however, the interaction of runways in both instance generation and solution stages has to be captured. In instance generation stage, now last flight on each runway need to be fixed which results in a larger number of instance-pairs as compared to the single runway scenario where only one flight is fixed. Further, as opposed to the single runway where only one makespan (of leading set) affects the solution space of the following set, in case of multiple runways, there are multiple makespans corresponding to each runway which together determine the solution space of the following set. We have already applied DS-ASP for a dual independent runways system, and the results are forthcoming. Further, the algorithm could be extended to solve various scheduling problems where CPS is imposed, and to obtain high quality solutions to such problems where CPS is not imposed.

Finally, this algorithm can also be applied in dynamic and uncertain environment using *Receding Horizon Control* (**RHC**) approach (Hu and Chen 2005) which considers critical factors such as congestion and level of uncertainty. Under this framework, the planning

443 horizon is divided into a certain number of intervals, and optimization is done on the predicted
444 traffic over these intervals. Only the optimal schedule corresponding to the first interval is
445 implemented; once the the time-horizon rolls over to the second interval, the same process
446 is repeated, and continued until the final interval. Note that time-horizon (or instance size)
447 should neither be too small nor too large as this method performs poorly in both the cases
448 (Hu and Chen 2005).

449 **Acknowledgment:** This research has been partially supported under ATMRI (NTU-CAAS)
450 Grant No. M4061216.

References

- Ahmed, M. S., Alam, S. and Barlow, M. (2017), An evolutionary optimization approach for path planning of arrival aircraft for optimal sequencing, in *Intelligent and Evolutionary Systems*, Springer, pp. 1–16.
- Airlines for America (2017). Accessed December 2017, <http://airlines.org/dataset/per-minute-cost-of-delays-to-u-s-airlines/>.
- Anagnostakis, I., Clarke, J.-P., Böhme, D. and Völcker, U. (2001), Runway operations planning and control: Sequencing and scheduling, *Journal of Aircraft* **38**(6), 988–996.
- Balakrishnan, H. and Chandran, B. G. (2010), Algorithms for scheduling runway operations under constrained position shifting, *Operations Research* **58**(6), 1650–1665.
- Beasley, J. E., Krishnamoorthy, M., Sharaiha, Y. M. and Abramson, D. (2000), Scheduling aircraft landings - The static case, *Transportation Science* **34**(2), 180–197.
- Benlic, U., Brownlee, A. E. and Burke, E. K. (2016), Heuristic search for the coupled runway sequencing and taxiway routing problem, *Transportation Research Part C: Emerging Technologies* **71**, 333 – 355.
- Bennell, J. A., Mesgarpour, M. and Potts, C. N. (2011), Airport runway scheduling, *4OR: A Quarterly Journal of Operations Research* **9**(2), 115–138.
- Bianco, L., Dell’Olmo, P. and Giordani, S. (2006), Scheduling models for air traffic control in terminal areas, *Journal of Scheduling* **9**(3), 223–253.
- Briskorn, D. and Stolletz, R. (2014), Aircraft landing problems with aircraft classes, *Journal of Scheduling* **17**(1), 31–45.

- 472 Desai, J. and Prakash, R. (2016), An optimization framework for terminal sequencing and
473 scheduling: The single runway case, in *Proceedings of the Complex Systems Design &
474 Management 2016 Asia Conference*, Springer, Cham, pp. 195–207.
- 475 Ghoniem, A., Farhadi, F. and Reihaneh, M. (2015), An accelerated branch-and-price algo-
476 rithm for multiple-runway aircraft sequencing problems, *European Journal of Operational
477 Research* **246**(1), 34–43.
- 478 Ghoniem, A., Sherali, H. D. and Baik, H. (2014), Enhanced models for a mixed arrival-
479 departure aircraft sequencing problem, *INFORMS Journal on Computing* **26**(3), 514–530.
- 480 Hu, X.-B. and Chen, W.-H. (2005), Receding horizon control for aircraft arrival sequencing
481 and scheduling, *IEEE Transactions on Intelligent Transportation Systems* **6**(2), 189–197.
- 482 Lahmar, M. and Benjaafar, S. (2007), Sequencing with limited flexibility, *IIE Transactions*
483 **39**(10), 937–955.
- 484 Lieder, A., Briskorn, D. and Stolletz, R. (2015), A dynamic programming approach for the
485 aircraft landing problem with aircraft classes, *European Journal of Operational Research*
486 **243**(1), 61–69.
- 487 Neuman, F. and Erzberger, H. (1991), *Analysis of delay reducing and fuel saving sequencing
488 and spacing algorithms for arrival traffic*, NASA Ames Research Center, Moffett Field, CA.
- 489 Prakash, R., Desai, J. and Piplani, R. (2017), An optimal data-splitting algorithm for air-
490 craft sequencing on a single runway. Working Paper, Nanyang Technological University
491 Singapore.

- 492 Psaraftis, H. N. (1978), A dynamic programming approach to the aircraft sequencing problem,
493 PhD thesis, Massachusetts Institute of Technology, Cambridge.
- 494 Psaraftis, H. N. (1980), A dynamic programming approach for sequencing groups of identical
495 jobs, *Operations Research* **28**(6), 1347–1359.
- 496 Sam, M., DAriano, A., Corman, F. and Pacciarelli, D. (2017), Metaheuristics for efficient
497 aircraft scheduling and re-routing at busy terminal control areas, *Transportation Research*
498 *Part C: Emerging Technologies* **80**, 485 – 511.
- 499 Sölveling, G. and Clarke, J.-P. (2014), Scheduling of airport runway operations using stochas-
500 tic branch and bound methods, *Transportation Research Part C: Emerging Technologies*
501 **45**, 119–137.
- 502 Soomer, M. and Franx, G. J. (2008), Scheduling aircraft landings using airlines preferences,
503 *European Journal of Operational Research* **190**(1), 277–291.

Figure 1: Schematic representation of the data-splitting algorithm

Figure 2: Instance generation framework on eight aircraft, for any aircraft letter and numbers indicates the category and FCFS position

Figure 3: Illustration of search space of following set bounded by $[t_i, L_n]$; it is largest when t_i is lowest as shown in uppermost panel.

Figure 4: Illustration of makespan decrease with increasing k

Figure 5: Illustration of solver time increase with increasing k

Table 1: Table of Acronyms

Acronym	Meaning
ATM	Air Traffic Management
ASP	Aircraft Sequencing Problem
TMA	Terminal Manoeuvring Area
CPS	Constrained Position Shifting
MPS	Maximum Position Shifting
FCFS	First-Come-First-Served
DP	Dynamic Programming
ATSP	Asymmetric Traveling Salesman Problem
MIP	Mixed Integer Program
DS-ASP	Data-Splitting Algorithm for ASP
FAA	Federal Aviation Administration
ELW	Earliest Landing time Windows

Table 2: Comparison of delay with and without minimizing delay (measured in seconds) as an objective function component

k	With Delay	Without Delay
1	14240	14297
2	12935	13084
3	12049	12151

Table 3: Required time-based separations enforced at the runway threshold as per aircraft type and operating mode

Operating Mode		Departure → Departure			Departure → Arrival		
Aircraft Type		Heavy	Large	Small	Heavy	Large	Small
Leading/Following							
Heavy		90	120	120	60	60	60
Large		60	60	60	60	60	60
Small		60	60	60	60	60	60
Operating Mode		Arrival → Departure			Arrival → Arrival		
Aircraft Type		Heavy	Large	Small	Heavy	Large	Small
Leading/Following							
Heavy		75	75	75	96	157	196
Large		75	75	75	60	69	131
Small		75	75	75	60	69	82

Table 4: Computational times based on a direct implementation of Problem ASP on large-scale instances, time-limit is set as 3600 seconds

Instance	# of flights	Mode of Operation	MPS (k)	
			2	3
1	30	Mixed	61	614
2	40	Arrival	551	> 3600
3	40	Arrival	233	> 3600
4	40	Arrival	171	> 3600
5	40	Mixed	2241	> 3600

Table 5: Computational time comparison for DS-ASP, ASP, and DP (– indicates DP is better)

$F \rightarrow$	30			40			% Savings (vs. DP)	% Savings (vs. ASP)
k	DS-ASP	DP	ASP	DS-ASP	DP	ASP		
1	0.3	0.01	0.8	0.82	0.01	2.6	–	67.6
2	0.86	0.34	11.2	3.28	0.43	366	–	99
3	1.8	35	77.6	9.71	43.7	> 3600	85.7	> 99.7
4	4.08	> 3600	360	47.4	> 3600	> 3600	> 99.2	> 98.7

Table 6: Computational time comparison of DS-ASP, its parallel implementation, and DP

$F \rightarrow$	30			40		
k	DS-ASP	Parallel	DP	DS-ASP	Parallel	DP
1	0.5	0.2	0.01	0.76	0.21	0.01
2	1.8	0.38	0.34	3.45	0.95	0.25
3	3.67	0.69	35	12.4	4	58.6

Table 7: Computational time comparison of DS-ASP with one and two splits vs DP

$F \rightarrow$	40			60		
k	1-Split	2-Split	DP	1-Split	2-Split	DP
1	0.7	0.6	0.01	9.2	2.3	0.02
2	3.36	2.7	0.43	102	12.2	0.58
3	12.1	10	43.7	3100	51.2	99

Table 8: Computational time comparison of DS-ASP and ASP

$F \rightarrow$	30				40				% Savings (vs. ASP)
γ	50		60		50		60		
k	DS-ASP	ASP	DS-ASP	ASP	DS-ASP	ASP	DS-ASP	ASP	
1	0.52	0.57	0.47	0.55	1.42	4	1	2.9	57.4
2	1.93	18.4	1.87	3.41	9.73	> 1312	8.85	303	> 98.6
3	9.69	189.63	7.93	12.38	166.87	> 3380	149.2	> 1567	> 94

Table 9: Computational time comparison of DS-ASP and its parallel implementation, in case of mixed traffic

$F \rightarrow$	30		40	
k	DS-ASP	Parallel	DS-ASP	Parallel
1	0.4	0.12	0.96	0.18
2	1.6	0.22	7.1	1
3	10.3	0.88	111.7	14.6

504 List of Figures

505	1	Schematic representation of the data-splitting algorithm	27
506	2	Instance generation framework on eight aircraft, for any aircraft letter and	
507		numbers indicates the category and FCFS position	27
508	3	Illustration of search space of following set bounded by $[t_i, L_n]$; it is largest	
509		when t_i is lowest as shown in uppermost panel.	27
510	4	Illustration of makespan decrease with increasing k	28
511	5	Illustration of solver time increase with increasing k	28

512 List of Tables

513	1	Table of Acronyms	29
514	2	Comparison of delay with and without minimizing delay (measured in seconds)	
515		as an objective function component	29
516	3	Required time-based separations enforced at the runway threshold as per air-	
517		craft type and operating mode	29
518	4	Computational times based on a direct implementation of Problem ASP on	
519		large-scale instances, time-limit is set as 3600 seconds	29
520	5	Computational time comparison for DS-ASP, ASP, and DP (– indicates DP is	
521		better)	30
522	6	Computational time comparison of DS-ASP, its parallel implementation, and	
523		DP	30
524	7	Computational time comparison of DS-ASP with one and two splits vs DP .	30
525	8	Computational time comparison of DS-ASP and ASP	30
526	9	Computational time comparison of DS-ASP and its parallel implementation,	
527		in case of mixed traffic	30