Automated Verification of Pruning Rules for Runway Sequencing

Toby Benjamin Clark, Jason Atkin, and Geert De Maere

Computational Optimisation and Learning Lab,
School of Computer Science, University of Nottingham, UK
{toby.clark, jason.atkin, geert.de_maere}@nottingham.ac.uk

Abstract. Exact approaches to the Runway Sequencing Problem (RSP) rely on pruning rules to make optimisation tractable. However, these rules are typically proven manually, making it difficult to evaluate new candidates or reuse rules across problem variants. We propose the use of Satisfiability Modulo Theories to automatically verify pruning rules for the RSP. Our approach verifies several published pruning rules within a symbolic sequence abstraction, where correctness is established by refuting all counterexamples. This enables a more systematic workflow for pruning rule development, and provides a basis for future rule synthesis.

Keywords: Runway Sequencing · Pruning Rules · Verification · Exact Optimisation · Satisfiability Modulo Theories (SMT) · Formal Methods

1 Introduction

Runway Sequencing is an important online timetabling problem at congested airports, where the efficient use of runways has a direct impact on delay propagation, emissions, and operational costs. Despite the NP-hardness of the general formulation, practical runway sequencing instances possess structural characteristics in their separation constraints and objective functions that can be systematically exploited, enabling the application of exact solution methods [8].

One approach to solving these defines a set of pruning rules [5], which eliminate partial sequences that are provably dominated by other partial sequences. These leverage structural characteristics of the problem constraints to reduce average time complexity, but rely on manual proofs to certify they preserve optimal solutions. However, manual reasoning is tightly coupled to a specific formulation, making rules difficult to reuse across variants with differing constraints or assumptions. Even small model changes can invalidate a pruning rule, necessitating a human review to re-prove rule correctness under the new model.

We propose the use of Satisfiability Modulo Theories (SMT) for verifying the correctness of such rules. SMT extends Boolean satisfiability with constructs such as linear arithmetic to enable automated verification of pruning rules, including those introduced in [5]. Specifically, we use Z3 Theorem Prover [7] to provide the first machine-executable formalisation that such rules preserve correctness, and demonstrate how operational runway sequencing constraints can be translated into a logically verifiable representation, and checked programmatically.

Pruning Rules in Optimisation. Many optimisation algorithms (e.g. dynamic programming, branch-and-bound, tree search) incrementally construct partial solutions and extend them until a complete solution is found. Pruning rules define dominance relations over partial solutions, enabling the early elimination of inferior search branches without compromising optimality.

Whilst pruning rules are not optimisers in themselves, a single correctness proof is orthogonal to the choice of search strategy, enabling their safe reuse across many search procedures that operate over partial sequences.

Pruning Runway Sequencing. De Maere et al. [5] introduced a set of pruning rules for runway sequencing, reducing average time complexity from $O(n!)$ to $O(N^2(n+1)^N)$ without loss of optimality, where N denotes the distinct types of aircraft. Such rules have been applied to instances based on sequencing systems at London Heathrow Airport [5], highlighting their real-world relevance.

These rules are derived from structural properties of wake-separation constraints and time window feasibility, and induce a dominance relation over partial sequences. More broadly, pruning rules have received considerable attention in machine scheduling literature, often used in exact methodologies [1, 2].

Logical Structure & Correctness. Pruning rules can be viewed as logical implications: their preconditions specify when the rule may be applied to a partial solution, and their dominance claim states that the pruned branch cannot yield a better feasible completion with respect to some objective component.

Rules are correct if this implication holds for all feasible instances. Equivalently, whenever the preconditions are satisfied, every feasible completion of the pruned branch is dominated by some feasible completion of the retained branch.

Contributions. We propose the use of Satisfiability Modulo Theories to automatically verify pruning rules for the Runway Sequencing Problem (RSP), including several rules introduced in [5]. In addition to the theoretical contributions outlined below, we implement the approach using the Z3 Theorem Prover and provide executable notebooks that reproduce all verification results.

- We formalise the implicit and explicit semantics of the RSP in SMT.
- We develop a refutation-based SMT approach for verifying pruning rules.
- We automatically verify several of the pruning rules introduced in [5].
- We propose a counterexample-preserving abstraction for runway sequencing.

Our contribution is therefore not a new pruning rule, but a reusable SMT formulation for automatically verifying complete and conditional order pruning rules in the context of the Runway Sequencing Problem. The approach is intended as an offline tool rather than an operational runtime component.

Paper Overview This paper proceeds as follows: Section 2 introduces SMT; Section 3 introduces the operational constraints of the RSP; Section 4 introduces the verification encodings and Section 5 evaluates the approach.

2 SMT for Pruning Rule Verification

Given that pruning rules are correctness properties rather than performance or approximation claims, validating them amounts to proving a safety property of the search, namely an optimality-preserving invariant over the search space.

However, as correctness is universally quantified, empirical testing can at best reveal counterexamples; it cannot establish validity across all possible configurations. In the RSP, the state space is infinite, as many decision variables range over the real numbers, making exhaustive enumeration impossible.

Therefore, the correctness of pruning rules must be derived logically (or inductively) from the problem formulation. SMT solvers enable this by combining SAT-based reasoning with specialised theory solvers (e.g. linear arithmetic, typically handled via simplex-style methods), allowing them to reason over highly expressive constraint systems without the need for explicit enumeration.

Manual Proof. Correctness arguments in [5] are established manually, typically appealing inductively to the monotonic structure (with respect to takeoff times) of the delay cost, makespan, and CTOT penalties defined in Section 3.

Limitations. However, manual reasoning is brittle under model evolution, as correctness arguments are formulation-specific and lack the structural invariance required to adapt across variants of the Runway Sequencing Problem.

Moreover, the arguments are non-executable, differ rule-by-rule, and offer no generic mechanism for verification, reuse, or quickly testing new rule candidates. Consequently, embedding these rules into new problem variants requires time-consuming rederivation, which can be considered a barrier to their adoption.

SMT. Satisfiability Modulo Theories enables automated, symbolic reasoning over constraints in rich background theories[1], and has become a standard tool for machine-checked verification of correctness properties in programs [7]. Given a set of constraints expressed in first-order logic over background theories, an SMT solver decides their satisfiability and, when satisfiable, can construct a corresponding instance, i.e. a finite assignment of values to variables.

The Z3 Theorem Prover [7] is an SMT solver, with applications in both program verification [3] and operational research [6]. In this work, SMT is not used to compute or optimise runway sequences, only to verify pruning rules.

Pragmatic Benefits. Automated verification of pruning rules supports trustworthy exact search, enables quick rechecking when the problem formulation changes, and facilitates iterative development and refinement of new rules.

Furthermore, SMT encodings are lightweight to develop. Modern SMT solvers, such as Z3, provide bindings for several programming languages, allowing verification constraints to be constructed and executed programmatically.

[1] Solvers for SMT support reasoning over highly expressive extensions of first-order logic, such as linear real/integer arithmetic, nonlinear arithmetic (sometimes undecidable), bit-vectors, arrays, algebraic datatypes, and uninterpreted functions [7].

3 Runway Sequencing Problem

Given a set of aircraft $\mathcal{A}$, De Maere et al. [5] define the Runway Sequencing Problem as the task of determining a total order of landings and takeoffs that satisfies asymmetric minimum separations between aircraft (due to wake vortices and airspace restrictions), and strict feasibility windows, while minimising operational delay, makespan, and regulatory penalties. Formally, a runway sequencing instance $\mathcal{P}$ is defined by the tuple of vectors/matrices shown by Equation (1),

$$\mathcal{P} = \big(\mathcal{A},\ \delta,\ \langle ec, lc\rangle,\ \langle et, lt\rangle,\ b,\ c\big) \tag{1}$$

where for each aircraft $i \in \mathcal{A}$, b_i denotes the earliest time at which i can enter the runway queue, c_i is a fixed queuing delay, $\langle et_i, lt_i\rangle$ is a hard feasibility window on takeoff time, and $\langle ec_i, lc_i\rangle$ is a CTOT[2] window. For any ordered pair of aircraft $(i, j) \in \mathcal{A} \times \mathcal{A}$, δ_{ij} denotes the minimum temporal wake separation required if aircraft i precedes j, and is generally asymmetric. All times are assumed to be continuous and measured in seconds.

Takeoff and Release. Given a sequence S over $\mathcal{A}$, let $S_i \subset \mathcal{A}$ denote the set of aircraft preceding i in S. Assuming aircraft depart as early as operationally feasible (a reasonable assumption at busy airports with takeoff queues), the takeoff time (2) of aircraft i is given as the latest of its release time, which is the earliest possible time an aircraft can be sequenced (3), and the minimum δ-separation from all aircraft preceding i in sequence S.

$$t_i = \max\big(r_i,\ \max_{x \in S_i}(t_x + \delta_{xi})\big) \tag{2}$$

$$r_i = \max(b_i + c_i,\ et_i,\ ec_i) \tag{3}$$

Objective Function. Orderings are ranked in [5] using a multi-objective function (4). The first objective measures makespan, defined as the maximum takeoff time in sequence S. The second objective measures delay, computed as the sum over aircraft of weighted deviations between base times and scheduled takeoff times (where α penalises greater delays more, promoting equity), together with a linear penalty for aircraft that miss their CTOT slots (5).

$$F(S) = \left(\max_{\ell \in S} t_\ell,\ \sum_{i \in S}\big(W_1\,(t_i - b_i)^\alpha + W_2\,C(t_i, lc_i)\big)\right) \tag{4}$$

CTOT Penalty. Violations of CTOT windows are penalised using a weighted piecewise linear function (5) with operator-defined parameters (ω_1 to ω_4).

$$C(t_i, lc_i) = \begin{cases} 0, & t_i \le lc_i, \\ \omega_1\,(t_i - lc_i) + \omega_2, & lc_i < t_i \le lc_i + 300, \\ \omega_3\,(t_i - lc_i) + \omega_4, & t_i > lc_i + 300. \end{cases} \tag{5}$$

[2] CTOT (Calculated takeoff Time) is a time slot allocated by air traffic flow management to balance network demand, with penalties applied for late departure.

4 SMT Formalisation and Verification

We formalise the core feasibility constraints of the RSP as SMT-solvable encodings. These are expressed in quantifier-free linear real arithmetic[3] with if-then-else expressions. Polynomial objective terms introduced later induce a local theory extension to quantifier-free nonlinear real arithmetic (`QF_NRA`).

Timing Variables. All timing parameters are continuous and are therefore encoded over $\mathbb{R}$. We use the notation $x : \tau$ to denote that variable x has type τ, since in SMT every variable must be declared with a fixed type. For each $i \in \mathcal{A}$, timing parameters are non-negative real-valued variables (6) (7), with additional constraints enforcing ordering over time windows and CTOT slots (8).

$$r_i, b_i, c_i, ec_i, lc_i, et_i, lt_i \ : \mathbb{R},\ \forall i \in \mathcal{A} \qquad \delta_{xy} : \mathbb{R},\ \forall \langle x, y \rangle \in \mathcal{A} \times \mathcal{A} \tag{6}$$

$$r_i, b_i, c_i, ec_i, lc_i, et_i, lt_i \ \geq 0,\ \forall i \in \mathcal{A} \qquad \delta_{xy} \geq 0,\ \forall \langle x, y \rangle \in \mathcal{A} \times \mathcal{A} \tag{7}$$

$$et_i < lt_i \ \wedge \ ec_i < lc_i,\ \forall i \in \mathcal{A} \tag{8}$$

Release Times. The release time of aircraft i, defined in (3), is the maximum of its queue-adjusted base time, the start of its time window, and the start of its CTOT window. We encode this in SMT as (9). Whilst the max operator is non-native to SMT, it can be defined using nested if-then-else expressions.

$$r_i = \max(b_i + c_i,\ et_i,\ ec_i), \quad \forall i \in \mathcal{A} \tag{9}$$

Sequence Evaluation. Given a sequence S and aircraft $i \in S$, where S_i denotes the set of aircraft preceding i in S, the takeoff time is defined recursively by (10), which directly implements the operational takeoff semantics of (2).

$$t_i = \max\Big(r_i,\ \max_{x \in S_i}\big(t_x + \delta_{xi}\big)\Big) \tag{10}$$

Separation Identicality. Several rules in [5] apply to pairs of separation-identical aircraft. We introduce the binary relation $i =_\delta j$ to denote that aircraft i and j have identical separation requirements with respect to all aircraft in $\mathcal{A}$ (11).

$$i =_\delta j \ \underset{\text{DEF}}{\iff} (\delta_{ix} = \delta_{jx} \wedge \delta_{xi} = \delta_{xj}, \quad \forall x \in \mathcal{A}) \tag{11}$$

Weights & Constants. All operator weights and constants are non-negative real-valued parameters (12). Since α is an exponent, we constrain $\alpha \geq 1$. Several pruning rules rely on the monotonicity of the CTOT function (with respect to takeoff time), to ensure weightings preserve this property, we require (13).

$$W_1, W_2, \omega_1, \omega_2, \omega_3, \omega_4, \alpha : \mathbb{R} \qquad W_1, W_2, \omega_1, \omega_2, \omega_3, \omega_4 \ \geq 0 \qquad \alpha \geq 1 \tag{12}$$

$$(\omega_1 \leq \omega_3) \ \wedge \ (\omega_2 \leq \omega_4) \tag{13}$$

We fix α to a constant, since arbitrary exponentiation is generally undecidable in SMT. Similarly, $\omega_1 \ldots \omega_4$ and W_1, W_2 are treated as constants.

[3] In SMT, problems are classified by logical theories. `QF_LRA` allows linear constraints over $\mathbb{R}$ without quantifiers, while `QF_NRA` additionally permits nonlinear terms [7].

4.1 Proof by Refutation with SMT

Correctness arguments in [5] rely on showing that a dominance relation between two aircraft propagates inductively to all subsequent aircraft in the sequence, with respect to makespan, delay, time window feasibility, and CTOT penalty.

In contrast, our SMT-based approach does not reason inductively over sequence structure. Instead, each pruning rule is encoded as a global logical property over symbolic timing and separation parameters. Correctness is then established by ruling out the existence of any feasible counterexample.

Satisfiability & Proof. SMT solvers, such as Z3, check the satisfiability of logical formulae expressed over the supported background theories (see Sec. 2). If the formula is satisfiable, the solver returns an instance, i.e. a valid assignment of variables to values. Otherwise, if the formula is unsatisfiable, then the solver has automatically proven that no satisfying instantiation exists.

In SMT-based verification, the correctness of a hypothesis (e.g. that a pruning rule is correct) is established by proving that no counterexample exists. Assumptions are encoded together with the negation of the hypothesis; if this formula is unsatisfiable, then the claim is valid, as there is no instance in which all assumptions hold true and the hypothesis holds false.

In the context of pruning the RSP, we verify a pruning rule by showing that no feasible instance satisfies the rule preconditions whilst falsifying the dominance claim, i.e., satisfying the logical negation of the dominance claim.

Correctness Pattern. Let β be a pruning rule with precondition P and dominance claim Q, and let Φ denote the foundational constraints, defined in (6)–(13). Rule β is correct if, for all models satisfying Φ, P implies Q, (14).

$$(\Phi \wedge P) \;\Rightarrow\; Q. \tag{14}$$

SMT solvers decide satisfiability, rather than validity; directly asserting $\Phi \wedge P \Rightarrow Q$ and checking satisfiability would establish merely the existence of an instance in which β is correct, rather than its universally quantified correctness under all feasible configurations. Instead, we use the standard equivalences (15) and verify correctness by checking that $\Phi \wedge P \wedge \neg Q$ is unsatisfiable.

$$(\Phi \wedge P) \Rightarrow Q \;\equiv\; \neg(\Phi \wedge P) \vee Q \;\equiv\; \neg\big((\Phi \wedge P) \wedge \neg Q\big) \tag{15}$$

If $\Phi \wedge P \wedge \neg Q$ is unsatisfiable, then no configuration satisfies Φ and P while violating Q, therefore the pruning rule β is correct with respect to Φ. If satisfiable, a concrete counterexample was found, and can be used for debugging.

Non-Vacuity. Rules may appear correct simply because their preconditions are unsatisfiable. Consider $r_i \neq r_i$ as a rule precondition; it is a syntactically correct expression in a decidable theory, yet it is inherently unsatisfiable; thus any paired dominance claim would appear correct under naïve refutation alone.

We also require rules to be non-vacuous; verifying that $\Phi \wedge P$ is satisfiable guarantees the rule is able to prune in at least one feasible configuration.

4.2 Formulating Complete Order Pruning Rules

We demonstrate how various complete order pruning rules from [5] can be formulated and verified using SMT. The first rule is presented in detail, with the remaining rules following an analogous swapped-sequence formulation.

Complete order pruning rules state that aircraft i and j form a complete order if the objective value and feasibility of any sequence S including i and j cannot, under any circumstances, be improved by reversing i and j. Consequently, any sequence in which j precedes i can be safely pruned from the search.

Makespan Objective. In [5], it is shown that objectives to minimize makespan can induce a complete order upon two aircraft i and j if $r_i \leq r_j$ and $i =_\delta j$. We verify that, under these conditions, for any pair of sequences S, S' that differ only by swapping i and j, the makespan component in S is no worse than in S'.

Swapped Sequences. While concrete instances of S and S' may be arbitrary in length, we model counterexamples in aircraft preceding, between, and following i and j as abstract representative aircraft, later justified in Section 4.4.

Let S and S' be sequences that differ only by swapping the positions of i and j, and let ψ_1, ψ_2, ψ_3 denote abstract aircraft representing counterexamples before, between, and after i and j, with $\mathcal{A}$ defining the set of all aircraft (16).

$$S = \psi_1,\ i,\ \psi_2,\ j,\ \psi_3 \qquad S' = \psi_1,\ j,\ \psi_2,\ i,\ \psi_3 \qquad \mathcal{A} = \{\psi_1, \psi_2, \psi_3, i, j\} \tag{16}$$

Rule Formulation. An objective to minimise makespan is claimed to induce a complete order under the preconditions $r_i \leq r_j$ and $i =_\delta j$. Therefore, the formula $r_i \leq r_j \wedge i =_\delta j$ is encoded in SMT as the rule's precondition.

Makespan is defined in objective function (4) as the maximum takeoff time over a given sequence. Since S denotes the retained ordering, and S' denotes the pruned ordering, the rule claims that the makespan of S is always less than or equal to that of S', explicitly encoded in SMT as $\max_{\ell \in S}(t_\ell) \leq \max_{k \in S'}(t'_k)$.

Verifying Correctness. Following the pattern from Section 4.1, we encode in SMT the foundational constraints (Φ), in conjunction with the rule preconditions and the negation of the makespan dominance claim, shown explicitly in (17).

The SMT solver attempts to find an instance satisfying $r_i \leq r_j \wedge i =_\delta j$ whilst violating the claimed makespan dominance. If no such instance exists, then it follows that no counterexample exists, and the rule is correct.

$$\Phi \ \wedge \ \underbrace{(r_i \leq r_j \ \wedge \ i =_\delta j)}_{\text{Precondition}} \ \wedge \ \neg\underbrace{\big(\max_{\ell \in S}(t_\ell) \leq \max_{k \in S'}(t'_k)\big)}_{\text{Dominance Claim}} \tag{17}$$

When encoded in Z3, formula (17) is proven to be unsatisfiable, indicating that no counterexample exists; thus, the pruning rule is verified as correct[4].

Non-Vacuity. In addition to verifying correctness, we also check that the rule is non-vacuous; here we ensure that $\Phi \wedge (r_i \leq r_j \ \wedge \ i =_\delta j)$ is satisfiable.

[4] Full rule correctness and non-vacuity results are summarised later in Section 5.

Delay, CTOT & Time Windows. Let t_ℓ and t'_ℓ denote the takeoff times of aircraft ℓ in sequences S and S', respectively. For each aircraft $\ell \in S$, we define the delay cost D_ℓ, CTOT penalty C_ℓ, and time window violation V_ℓ in (18).

$$D_\ell = W_1(t_\ell - b_\ell)^\alpha \quad C_\ell = W_2 C(t_\ell,\ lc_\ell) \quad V_\ell = t_\ell > lt_\ell. \tag{18}$$

Here, $D_\ell, C_\ell : \mathbb{R}$ whilst V_ℓ is Boolean. Corresponding quantities for S' are denoted D'_ℓ, C'_ℓ, and V'_ℓ, defined analogously using the takeoff times t'_ℓ in S'.

Delay Objective. Objectives to minimise delay cost similarly induce a complete order between two aircraft i and j under the conditions $r_i \leq r_j$, $i =_\delta j$, and $b_i \leq b_j$. When these preconditions hold, sequencing i before j cannot yield a higher total delay cost than the swapped ordering in which j precedes i.

Rule Formulation. As with the encodings for makespan, we consider sequences S and S' defined in (16), and compare total delay over both sequences. Dominance between sequences S and S' with respect to total delay is defined in (19), where D_ℓ and D'_k denote the per-aircraft delay cost in S and S' respectively.

$$\sum_{\ell \in \mathcal{A}} (D_\ell) \leq \sum_{k \in \mathcal{A}} (D'_k) \tag{19}$$

To verify the rule, we encode the foundational constraints together with the rule preconditions and the negation of the total delay dominance claim (20).

$$\Phi \ \wedge \ (r_i \leq r_j \ \wedge \ i =_\delta j \ \wedge \ b_i \leq b_j) \ \wedge \ \neg\big(\sum_{\ell \in \mathcal{A}} (D_\ell) \leq \sum_{k \in \mathcal{A}} (D'_k)\big) \tag{20}$$

If this formula is unsatisfiable, then no instance satisfying $r_i \leq r_j, i =_\delta j$ and $b_i \leq b_j$ violates the dominance claim, and the rule is correct. We also verify that the rule is non-vacuous, ensuring $\Phi \wedge (r_i \leq r_j \wedge i =_\delta j \wedge b_i \leq b_j)$ is satisfiable.

Time Windows. Hard time windows can induce a complete order between two aircraft i and j whenever $r_i \leq r_j$, $i =_\delta j$, and $lt_i \leq lt_j$ hold. Unlike delay and makespan, this rule concerns feasibility rather than optimisation. Specifically, the claim is that if the swapped sequence S' is feasible (i.e. no aircraft violates its time window), then the original sequence S is also feasible.

Rule Formulation. Since V_ℓ and V'_ℓ denote ℓ missing its time window in S and S' respectively, the time window feasibility of sequence S can be modelled as the conjunction of $\neg V_\ell$ for each $\ell \in \mathcal{A}$, with S' defined analogously using V'_ℓ.

Given that the rule applies only when S' is feasible, we include the feasibility of S' into the preconditions. To verify the rule, we conjoin these preconditions with the foundational constraints and the negated dominance claim (21).

$$\Phi \ \wedge \ \Big(r_i \leq r_j \ \wedge \ i =_\delta j \ \wedge \ lt_i \leq lt_j \wedge \ \big(\bigwedge_{k \in \mathcal{A}} (\neg V'_k)\big)\Big) \wedge \ \neg\big(\bigwedge_{\ell \in \mathcal{A}} (\neg V_\ell)\big) \tag{21}$$

Here, Z3 searches for an instance where all preconditions hold, S' is feasible, and S is not. If unsatisfiable, no counterexample exists and the rule is correct.

4.3 Formulating Conditional Order Pruning Rules

Cost-minimising objectives can induce a conditional order between aircraft i and j when $r_i \leq r_j$, $lt_i \leq lt_j$, and $i =_\delta j$. Under these conditions, i should be sequenced before j whenever inequality $\mathcal{I}$ (22) holds, i.e., when the combined local cost of sequencing i before j is no greater than that of the swapped ordering.

$$\begin{aligned}\mathcal{I} = {} & W_1(t_i - b_i)^\alpha + W_2 C(t_i, lc_i) + W_1(t_j - b_j)^\alpha + W_2 C(t_j, lc_j) \\ \leq {} & W_1(t'_i - b_i)^\alpha + W_2 C(t'_i, lc_i) + W_1(t'_j - b_j)^\alpha + W_2 C(t'_j, lc_j)\end{aligned} \tag{22}$$

Unlike complete orders, conditional orders depend on takeoff times directly. We include inequality $\mathcal{I}$ in the rule preconditions and verify that sequencing i before j cannot worsen makespan, delay, or time window feasibility.

Rule Formulation. We present the formulation for makespan in (23). Encodings for delay and time window feasibility follow similarly from their complete order definitions, with the additional term $\mathcal{I}$ included in the preconditions.

$$\Phi \ \wedge \ \big(r_i \leq r_j \ \wedge \ i =_\delta j \ \wedge \ lt_i \leq lt_j \ \wedge \ \mathcal{I}\big) \ \wedge \ \neg\big(\max_{\ell \in S}(t_\ell) \leq \max_{k \in S'}(t'_k)\big) \tag{23}$$

Combined CTOT. Objectives penalising CTOT misses do not induce a complete order, as $C(t_i, lc_i)$ is non-convex. However, a conditional order can be induced when $r_i \leq r_j$, $lt_i \leq lt_j$, $i =_\delta j$, and $\mathcal{I}$ holds, under which sequencing i before j does not increase the combined delay and CTOT cost, encoded as (24).

$$\Phi \wedge \big(r_i \leq r_j \wedge i =_\delta j \wedge lt_i \leq lt_j \wedge \mathcal{I}\big) \wedge \neg\big(\sum_{\ell \in \mathcal{A}}(D_\ell + C_\ell) \leq \sum_{k \in \mathcal{A}}(D'_k + C'_k)\big) \tag{24}$$

4.4 Soundness of the Sequence Abstraction

Verification is performed on fixed-length sequences, whereas real instances admit sequences of arbitrary length. In (16), subsequences of aircraft prefixing, suffixing and in-between i and j are represented by single aircraft rather than arbitrary length sequences; the validity of this representation must be justified.

In all cases, a subsequence is abstracted by a single aircraft in the model. This abstraction must preserve all effects relevant to rule correctness.

Binding Separations & Costs. Representative aircraft must capture the effects of subsequences on subsequent aircraft; thus, any binding separations on later aircraft must be equivalently imposed by the separations of the singular aircraft. This is discussed in the Prefix Subsequences explanation, below.

It is also necessary to capture the effects on the cost of the sequence being replaced, by the cost of this single aircraft replacing it. We ensure that the replacement aircraft is such that when the total cost of the original sequence would increase, so would the cost of the single aircraft. This is further justified in the Counterexamples explanation, below.

Establishing these two properties is sufficient, since all constraints are expressed in terms of aircraft timings (6)-(13), and the objective components (and therefore rule correctness) are also defined over such timings (4), (18).

Prefix Subsequences. For any aircraft k in sequence S, the takeoff time t_k is defined as $\max(r_k,\ \max_{x \in S_k}(t_x + \delta_{xk}))$. Although all predecessors impose separation constraints, t_k is only influenced by the maximum of these constraints; thus, the effect of S_k on t_k may be characterised solely by this maximal value.

Since δ-separations are independent and arbitrary, a single aircraft ψ_1 can be defined to reproduce the binding constraints upon all later aircraft using (25). The range of δ values considered for ψ_1 needs to cover all values that the replaced sequence could generate, which is the case in the result in this paper.

$$\delta_{\psi_1 k} = \max_{x \in S_k}(t_x + \delta_{xk}) - t_{\psi_1} \quad \forall k \in \{i,\ j,\ \psi_2,\ \psi_3\} \tag{25}$$

Counterexamples. If a counterexample exists for an arbitrary subsequence of aircraft after i or j, then swapping i and j must worsen at least one objective component. Since objectives are defined over individual aircraft, this implies the existence of at least one later aircraft ℓ whose contribution to some objective component worsens under the swap. We call this aircraft a witness.

Since any worsening sequence admits a witness, it suffices to represent only the aircraft whose takeoff time captures the worsening effect, rather than the entire subsequence. We now show how a single aircraft ψ_3 can accomplish this by considering the binding separations giving rise to the witness.

Binding Separations. Let ℓ be the witness aircraft that worsens under the swap. Since only i and j change position, any difference in t_ℓ can only be due to constraints originating from i or j. This can occur either directly, through a separation constraint between i or j and ℓ, or indirectly, where the effect of i or j propagates through intermediate aircraft before influencing t_ℓ.

Since separation parameters δ_{xy} are independent across aircraft pairs (separations between any two aircraft can be chosen without affecting others) and the solver is able to freely assign arbitrary non-negative values, the solver can freely choose δ-parameters to instantiate ψ_3 so as to reproduce the witness behaviour.

- If i or j influences t_ℓ directly, it can select ψ_3 with identical δ-parameters to ℓ, so that ψ_3 is subject to exactly the same binding constraints.
- If influence propagates through intermediate aircraft, it can select parameters of δ to impose the same constraint on ℓ as the full propagation chain.

In both cases, ψ_3 is able to reproduce the binding constraint for t_ℓ, and therefore attain the same takeoff time as ℓ. Since objectives depend only on these timing quantities[5], any worsening observed for ℓ under the swap is preserved by ψ_3, therefore the presence of a counterexample is maintained using only ψ_3.

Intermediate Segment. Aircraft between i and j may both influence a witness ℓ and act as a witness themselves. This is possible because separations from earlier aircraft, $\delta_{x\ell}$, are independent of separations to later aircraft, $\delta_{\ell y}$; thus, both effects on cost and on later aircraft can be represented by a single aircraft.

[5] Setting $(t_\ell,\ et_\ell,\ lt_\ell,\ ec_\ell,\ lc_\ell) := (t_\psi,\ et_\psi,\ lt_\psi,\ ec_\psi,\ lc_\psi)$ means that any objective component defined over these quantities evaluates identically for both ψ and ℓ.

5 Evaluation

We presented an SMT-based approach for verifying pruning rules in the Runway Sequencing Problem, showing that the conditional and complete order pruning rules can be encoded and checked automatically with Z3 Theorem Prover.

All verification encodings were checked automatically by Z3. Across all verified pruning rules, solver runtimes were consistently low (sub-one-second). Our full correctness and non-vacuity results are summarised in Table 1. This demonstrates that using SMT for the verification of pruning rules is tractable.

Replicability. All SMT encodings used in this work are freely and publicly available online in a Github repository[6] as executable, annotated notebooks.

In this paper, we omit explicit formulations for conditional orders with unknown takeoff times for brevity; these are given in the accompanying repository. This repository includes the formalisation of the RSP constraints using Z3 Theorem Prover, the encodings of pruning rules, and executable notebooks to reproduce both correctness/non-vacuity checks and show counterexamples.

Counterexamples & Synthesis. In our approach, refuting the correctness of a pruning rule constructs a concrete counterexample. Such instances can be used to guide the iterative refinement and debugging of pruning rules, laying the groundwork for future automated pruning rule synthesis methodologies.

Specifically, this suggests a direction towards counterexample-guided inductive synthesis (CEGIS), in which candidate rules would be iteratively synthesised and checked against a verifier, with counterexamples driving refinement. Such approaches have been successfully applied in the synthesis of programs [4].

Table 1. Summary of verified pruning rules.

Rule Type	Objective	Correct	Non-Vacuous
Complete Orders	*Makespan*	✓	✓
	Delay	✓	✓
	Time Windows	✓	✓
Conditional Orders (with known t_i / $\mathcal{I}$)	*Makespan*	✓	✓
	Delay & CTOT	✓	✓
	Time Windows	✓	✓
Conditional Orders (with unknown t_i)	*Makespan*	✓	✓
	Delay & CTOT	✓	✓
	Time Windows	✓	✓

[6] https://github.com/tobybenjaminclark/rsp-smt

Limitations. Our approach verifies a specific class of pruning rules, namely complete and conditional order rules over swapped sequences. Although this captures several published rules, broader classes of dominance arguments in [5] may require different abstractions or more expressive encodings.

Modifications to objective components which extend outside the decidable fragments of SMT may render automated verification undecidable or less tractable, limiting the range of model variants in which the approach can be used.

While SMT provides automated, push-button verification, it offers limited insight into why the result is correct, placing a significant degree of trust in the solver implementation when it fails to find a counterexample.

Generalisation. While the case study focuses on runway sequencing, the approach is not domain-specific. Dominance relations and pruning rules are used elsewhere in exact optimisation, and are similarly amenable to verification.

Our approach recasts rule correctness as an invariant over the underlying model, decoupled from the optimisation procedure in which the rules are applied.

Conclusion. This paper presented an approach for verifying pruning rules using Z3, enabling automated correctness checking of complete and conditional orders.

The semantics of the RSP were formalised within decidable SMT theories, and verification over a finite sequence abstraction was justified as sufficient to guarantee correctness and non-vacuity for arbitrary length sequences.

More broadly, this work illustrates how formal methods and optimisation can be hybridised to enable the automated verification of pruning rules, and suggests a natural future research direction towards their automated synthesis.

References

1. Allahverdi, A., Gupta, J.N., Aldowaisan, T.: A review of scheduling research involving setup considerations. Omega **27**(2), 219–239 (1999)
2. Allahverdi, A., Soroush, H.M.: The significance of reducing setup times/setup costs. European journal of operational research **187**(3), 978–984 (2008)
3. Bjørner, N.S., McMillan, K.L., Rybalchenko, A.: Program verification as satisfiability modulo theories. SMT@ IJCAR **20**, 3–11 (2012)
4. David, C., Kroening, D.: Program synthesis: challenges and opportunities. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **375**(2104), 20150403 (2017)
5. De Maere, G., Atkin, J.A., Burke, E.K.: Pruning rules for optimal runway sequencing. Transportation Science **52**(4), 898–916 (2018)
6. Karapetyan, D., Gutin, G.: Solving the workflow satisfiability problem using general purpose solvers. IEEE Transactions on Dependable and Secure Computing **20**(6), 4474–4485 (2022)
7. de Moura, L.M., Bjørner, N.S.: Proofs and refutations, and z3. In: LPAR Workshops. vol. 418, pp. 123–132. Doha, Qatar (2008)
8. Shirini, K., Aghdasi, H.S., Saeedvand, S.: A comprehensive survey on multiple-runway aircraft landing optimization problem. International Journal of Aeronautical and Space Sciences **25**(4), 1574–1602 (2024)