# Interval Constraints with Learning: Application to Air Traffic Control

Thibaut Feydy[1(✉)] and Peter J. Stuckey[1,2]

[1] Data61, CSIRO, Melbourne, Australia
{thibaut.feydy,peter.stuckey}@data61.csiro.au
[2] Department of Computing and Information Systems, University of Melbourne, Melbourne, Australia

**Abstract.** Lazy Clause Generation (LCG) is a learning extension of Constraint Programming that combines the power of SAT and CP. In this paper we present an extension of Lazy Clause Generation from finite domain constraints to interval constraints, that is: non-linear constraints over the reals. Because LCG solvers must be able to negate literals involved in computation, LCG for intervals must represent both open and closed intervals. This makes LCG for intervals quite different from LCG for integers. We illustrate the advantage of the technology by solving a mixed integer non-linear Air Traffic Control problem .

## 1 Introduction

The capacities of European en-route Air Traffic Control (ATC) centers are far exceeded by a constant growth in air traffic demand, resulting in ever increasing flight delays. To overcome this issue, novel Air Traffic Management (ATM) schemes are designed while keeping the hard constraint of a minimal 5 nautical mile horizontal safety separation between every pair of aircraft. Nowadays, solutions to avoid conflicts are empirical, and human controllers rely on standard routes and traffic organization to devise them. However, the complexity of conflicts could grow tremendously within future ATM systems, should the aircraft fly on direct routes, from take-off airport to destination. Human controllers would no longer be able to solve them efficiently on their own, thus requiring automated solvers. Former approaches like [6] use local search (namely genetic algorithms) to solve the conflict problem. These meta-heuristics are well suited to solve large scale and difficult problems when no other relevant techniques are known, but stochastic search inherently lacks existence and optimality proofs. An interval constraint approach was offered in [8]. While the method allows proof of optimality and the existence of solutions, it does not scale to the size of a general air traffic sector. The difficulty in handling the required constraints is related to the fact that the separation must be kept at any time. In this paper we propose to solve this problem by extending an Interval Constraint Solver with Learning. Learning methods such as lazy clause generation [15] can exponentially reduce the search complexity and are particularly well suited to such a problem where some of the variables can be discretized. After presenting interval constraint


M. Rueher (Ed.): CP 2016, LNCS 9892, pp. 224–232, 2016.
DOI: 10.1007/978-3-319-44953-1_15

methods and its extensions to learning, we present the air traffic control models and their implementation and provide results validating the approach.

## 2 Preliminaries

### 2.1 Finite Domain Constraint Programming

A *valuation*, $\theta$, is a mapping of variables to values, denoted $\{x_1 \mapsto d_1, \ldots, x_n \mapsto d_n\}$. Define $vars(\theta) = \{x_1, \ldots, x_n\}$. A *primitive constraint*, $c$, is a set of valuations over a set of variables $vars(c)$. A valuation $\theta$ is a *solution* of $c$ if $\{x \mapsto \theta(x) \mid x \in vars(c)\} \in c$. A *constraint* $C$ is a conjunction of primitive constraints, which we often treat as a set. A valuation $\theta$ is a solution of constraint $C$ if it is a solution for each $c \in C$. We write $C_1 \models C_2$ if every solution of $C_1$ is a solution of $c_2$.

An *atomic constraint* is a unary constraint of the form $\langle x = d \rangle$, $\langle x \neq d \rangle$, $\langle x \geq d \rangle$, $\langle x \leq d \rangle$, or *false*. We write atomic constraints in angle brackets to emphasize their special status. A *domain* $D$ is a conjunction of atomic constraints. $D$ is a *false domain* if it has no solutions. We use notation $D(x) = \{\theta(x) \mid \theta \text{ is a solution of } D\}$. A *singleton domain* is one where $|D(x)| = 1, x \in vars(D)$, and we let $\theta_D = \{x \mapsto d_x \mid x \in vars(D), D(x) = \{d_x\}\}$ in this case.

A *propagator* $p(c)$ for constraint $c$ is an inference algorithm, it maps a domain $D$ to a set of atomic constraints $p(c)(D)$, where $D \wedge c \models p(c)(D)$. We assume each propagator is *checking*, that is if $\forall x \in vars(c).|D(x)| = 1$ then $p(c)(D) = \emptyset$ if $\theta_D$ is a solution of $c$ and $\{false\}$ otherwise. A propagation solver $prop(P, D)$ applied to a set of propagators $P$ and a domain $D$ repeatedly applies the propagators $p \in P$ until $p(D') = \emptyset$ for $p \in P$, and returns $D'$.

A *constraint satisfaction problem (CSP)* $P = (V, D, C)$ is a constraint $C$ and domain constraint $D$ over variables $V = vars(C) \cup vars(D)$. A CP solver solves the CSP by applying the propagation solver $prop(\{p(c) \mid c \in C\}, D)$ to obtain a new domain $D'$, then if this is not a false domain or singleton domain, guessing an atomic constraint *decision* $a$, and solving the two problems $(V, D' \wedge a, C)$ and $(V, D' \wedge \neg a, C)$.

### 2.2 Lazy Clause Generation for Integers

*Lazy clause generation (LCG)* solvers [15] are hybrid CP and SAT solvers that combine CP propagation based solving with SAT nogood learning. An LCG solver represents an integer variable with initial domain $[\,l\,..\,u\,]$ by the Boolean variables $[\![x = d]\!], l \leq d \leq u$ (*equality variables*) and $[\![x \geq d]\!], l < d \leq u$ (*bounds variables*). Note that each atomic constraint defined earlier, is exactly a Boolean literal using this representation: $\langle x = d \rangle$ is $[\![x = d]\!]$, $\langle x \neq d \rangle$ is $\neg[\![x = d]\!]$, $\langle x \geq d \rangle$ is $[\![x \geq d]\!]$ and $\langle x \leq d \rangle$ is $\neg[\![x \geq d + 1]\!]$.

The Boolean variables are connected to an integer domain propagator which ensures that they maintain a consistent representation of an integer variable, that is $[\![x \geq d + 1]\!] \rightarrow [\![x \geq d]\!], l < d < u$, and $[\![x = d]\!] \leftrightarrow [\![x \geq d]\!] \wedge \neg[\![x \geq d + 1]\!], l < d < u$, $[\![x = l]\!] \leftrightarrow \neg[\![x \geq l + 1]\!]$, and $[\![x = u]\!] \leftrightarrow [\![x \geq u]\!]$.

In LCG solvers propagators are also required to return explanations for each new consequence $l \in p(c)(D)$, that is an explanation clause $e \equiv l_1 \wedge \cdots l_n \rightarrow l$ where $\forall 1 \leq i \leq n, D \models l_i$ and $c \models e$. In LCG solvers during propagation [7,15], a *trail* of newly inferred literals representing atomic constraints is created, each of which has an explanation clause showing which previously true literals made it true.

When an LCG solver infers *false* it, like a SAT solver, repeatedly replaces literals in the explanation for the failure until only one literal that became true since the last decision remains. The resulting explanation of failure is the so called 1UIP nogood [14]. This nogood is then stored in the system as a new constraint (propagator), and the solver backjumps to the second last decision level in the nogood. At this point the nogood is guaranteed to propagate new information. See [15] for more details.

*Example 1.* Consider a CSP with constraints $x \geq y$, $t \geq 2 \rightarrow b$, $b \rightarrow x \leq 3z$, $b \rightarrow y \geq 2$, over integers $x$, $y$, $z$ and $t$, and Boolean $b$ and initial domain $D = \langle x \geq 0 \rangle \wedge \langle x \leq 10 \rangle \wedge \langle y \geq 0 \rangle \wedge \langle y \leq 10 \rangle \wedge \langle z \geq 0 \rangle \wedge \langle z \leq 10 \rangle \wedge \langle t \geq 0 \rangle \wedge \langle t \leq 10 \rangle$. An initial decision $\langle z \leq 5 \rangle$ ($\neg[\![z \geq 6]\!]$) causes no propagation. The next decision $\langle t \geq 6 \rangle$ ($[\![t \geq 6]\!]$) causes $b$ which in turn causes $[\![y \geq 2]\!]$ and (with $\neg[\![z \geq 6]\!]$) $\neg[\![x \geq 2]\!]$, and these two propagate to *false*. The initial nogood is $[\![y \geq 2]\!] \wedge \neg[\![x \geq 2]\!] \rightarrow \mathit{false}$, replacing $\neg[\![x \geq 2]\!]$ by its reasons gives $\neg[\![z \geq 6]\!] \wedge b \wedge [\![y \geq 2]\!] \rightarrow \mathit{false}$, then replacing $[\![y \geq 2]\!]$ gives $\neg[\![z \geq 6]\!] \wedge b \rightarrow \mathit{false}$. The resulting 1UIP nogood is $[\![z \geq 6]\!] \vee \neg b$. □

### 2.3 Interval Arithmetic

Given the discrete representation of numbers by computers it is impossible to solve continuous problems exactly. Interval constraint solvers use interval arithmetic [13] to compute sound approximations of the constraint system, through a combination of local consistencies and search.

Let $\mathbb{R}$ be the set of real numbers, and let $\mathbb{R}^\infty$ be $\mathbb{R} \cup \{+\infty, -\infty\}$. Let $\mathbb{F}$ be the subset of $\mathbb{R}$ of the representable floating-point numbers in a given format, and let $\mathbb{F}^\infty$ be $\mathbb{F} \cup \{+\infty, -\infty\}$. Let $\downarrow(r)$ (resp. $\uparrow(r)$) be the downward (resp. upward) roundings to $\mathbb{F}^\infty$ of a real number $r$. Given two numbers $a \in \mathbb{F} \cup \{-\infty\}$ and $b \in \mathbb{F} \cup \{+\infty\}$ the *closed interval* $[a, b]$ is the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$.

Less usual for interval arithmetic we will also consider open and semi-open intervals. The *open interval* $(a, b)$ is the set $\{x \in \mathbb{R} \mid a < x < b\}$, while the two forms of *semi-open intervals* $(a, b]$ and $[a, b)$ represent the sets $\{x \in \mathbb{R} \mid a < x \leq b\}$ and $\{x \in \mathbb{R} \mid a \leq x < b\}$ respectively.

We will use $\mathbb{I}$ to represent the set of *closed intervals*, which is closed under intersection. We will use $\mathbb{I}^+$ to represent the set of (all) *intervals*, including open and semi-open intervals, which is also closed under intersection.

Given a closed interval $I$ we define $\lfloor I \rfloor$ (resp. $\lceil I \rceil$) as the smallest (resp. largest) element of $I$.

Given a real operator $*$, the associated interval operator $\circledast$ is defined by $X \circledast Y = \bigcap_{\mathbb{I}} \{Z \mid \forall x \in X, \forall y \in Y, x * y \in Z\}$, e.g. $[a, b] \ominus [c, d] = [\downarrow(a - d), \uparrow(b - c)]$.

### 2.4 Interval Constraints Solving

A real (resp. interval) constraint is an atomic formula arising from a relation over real (resp. interval) expressions and variables. In practice interval constraint propagators enforce approximate consistencies, often *hull consistency* [3] or *box consistency* [2]. The original hull consistency algorithm *hc*3 decomposes constraints into primitives constraints implemented each by a corresponding propagator.

*Example 2.* The constraint c: $(x+y)+2*b=0$ can be decomposed into $c_1 : z = x+y$ and $c_2 : z+2*b=0$. The *hull consistent* propagator for the constraint $c_1$, with the domains $X$, $Y$, and $Z$ computes the common fixed-point of the projection operators: $X \leftarrow X \cap (Z \ominus Y)$, $Y \leftarrow Y \cap (Z \ominus X)$, and $Z \leftarrow Z \cap (X \oplus Y)$. □

A refinement of the *hc*3 algorithm is *hc*4 [12] which avoids decomposing the constraints by working directly on a tree-like representation of constraints where each node is either a variable, a constant or a primitive function operator. The variables domains pruning is done through a forward evaluation of the tree followed by a backward top-down projection narrowing operation. During the top-down pruning the algorithm may prematurely end by the computation of an empty interval, in which case the constraint is inconsistent w.r.t the current domain.

*Example 3.* The constraint $c : (x+y)+(2*b)=0$ has the tree representation $c : (e_1 : (e_2 : x+y)+(e_3 : 2*b))=0$. Given the domain $X$, $Y$, $B$, the evaluation phases computes $e_2.f = X \oplus Y$, $e_3.f = [2,2] \otimes B$, $e_1.f = e_2.f \oplus e_3.f$. The top-down pruning phases enforces the projection $e_1.b \leftarrow e_1.f \cap [0,0]$, $e_2.b \leftarrow e_2.f \cap (e_1.b \ominus e_3.f)$, $X \leftarrow X \cap (e_2.b \ominus Y)$, $Y \leftarrow Y \cap (e_2.b \ominus X)$, $e_3.b \leftarrow e_3.f \cap (e_1.b \ominus e_2.f)$, $B \leftarrow B \cap (e_3.b \oslash [2,2])$. □

## 3 Lazy Clause Generation for Intervals

The critical question in defining a learning solver is how to represent the changes in variables. A natural representation for interval variable $x$ would be using atomic constraints of the form $\langle x \in I \rangle$, which record the entire interval $I$ attached to the variable. Indeed there are finite domain learning solvers which take this approach [16]. The disadvantages of this approach is that resulting nogoods are unlikely to be very reusable, and the atomic constraints themselves interact in complex ways. A stronger disadvantage is that atomic constraints will be negated, and the negative form of these constraints is hard to reason about.

The obvious choice, analogous to the integer case is to use the atomic constraints $\langle x \geq a \rangle$, $\langle x \leq a \rangle$, $a \in \mathbb{F}$. This allows us to represent all closed intervals. Unlike the integer case we cannot get away with a single set of bounds variables since $\neg \langle x \geq a \rangle \not\leftrightarrow \langle x \leq a \rangle$. Hence we need 2 sets of Boolean variables $[\![x \geq a]\!]$ and $[\![x \leq a]\!]$. Since $\langle x < a \rangle \leftrightarrow \neg [\![x \geq a]\!]$ and $\langle x > a \rangle \leftrightarrow \neg [\![x \leq a]\!]$, we will be able to represent open and semi-open intervals.

Clearly we cannot create a Boolean variable for each possible atomic constraint $\langle x \geq a \rangle$, $\langle x \leq a \rangle$, $a \in \mathbb{F}$ for variable $x$ *apriori*, there are far too many. Indeed even during propagation far too many atomic constraints will appear for us to represent them each by a Boolean variable. In an LCG (and SAT) solver each Boolean variable is a non-trivial data structure storing watch lists, activity counts, and any associated atomic constraint.

To avoid the cost of creating many Boolean variables during propagation we make use of a *stateless* atomic constraint representation (tagged pointer), which carries its meaning with it, and use this for propagation, and recording the implication graph in the trail, and the explanations of propagation, and for building explanations. Most atomic constraints will appear on the trail, and simply be removed by backtracking/backjumping. We will only create Boolean variables corresponding to atomic constraints that end up in the nogoods that are created.

*Example 4.* Reconsider the CSP of Example 1 where now $x$, $y$, $z$ and $t$ are interval variables. An initial decision $\langle z \leq 5 \rangle$ causes no propagation. The next decision $\langle t \geq 6 \rangle$ causes $b$ which in turn causes $\langle y \geq 2 \rangle$ and (with $\langle z \leq 5 \rangle$) $\left\langle x \leq \uparrow \frac{5}{3} \right\rangle$, and these two propagate to *false*. The initial nogood is $\langle y \geq 2 \rangle \wedge \left\langle x \leq \uparrow \frac{5}{3} \right\rangle \rightarrow \mathit{false}$, replacing $\left\langle x \leq \uparrow \frac{5}{3} \right\rangle$ by its reasons gives $\langle z \leq 5 \rangle \wedge b \wedge \langle y \geq 2 \rangle \rightarrow \mathit{false}$, then replacing $\langle y \geq 2 \rangle$ gives $\langle z \leq 5 \rangle \wedge b \rightarrow \mathit{false}$. The resulting 1UIP nogood is $\neg [\![ z \leq 5 ]\!] \vee \neg b$. Note how the entire process uses atomic constraints, except the final stored nogood which uses literals. □

A critical component of the interval learning solver is the *interval domain propagator* which is responsible for mapping interval domain information to atomic constraints and any associated Boolean literals, and vice versa.

The domain of interval variable $x$ is implemented as a sorted map from float values $a$ to atomic constraints $\langle x < a \rangle$, $\langle x \leq a \rangle$, $\langle x \geq a \rangle$, and $\langle x > a \rangle$. We cache the current upper and lower bounds for $x$, but not their positions in the map. Changes to $D(x)$ require walking the map to determine which atomic constraints become *true* or *false*. Note that in this way the domain propagator for $x$ also maintains the consistency of the Boolean literals associated with $x$, which will be added to the queue for propagation.

When a new atomic constraint is created, it is inserted appropriately in the map. Note usually a new atomic constraint is only created by propagation which makes it true, so we can implement this simply by walking the map from the current bound to the position of the new bound and inserting it, since we have to walk the map setting the other atomic constraints in the path *true* or *false* appropriately.

### 3.1 Propagation with Learning

Note that although we must represent open, semi-open and closed intervals, in order to have the representation of intervals closed under negation, the interval propagation almost always relaxes intervals to be closed. The only cases where

this does not occur is when no floating point operations occur on the interval bounds, for example in equality, min and max. Clearly the resulting computation is still safe.

In order to provide explanations for variable domain updates, interval operations are *augmented* to maintain the reasons for their results, in the form of a set of atoms per bound. For example the augmentation $\boxplus$ of the operator $\oplus$ is defined as $(X, l_x, u_x) \boxplus (Y, l_y, u_y) = (X \oplus Y, l_x \cup l_y, u_x \cup u_y)$. Given a variable $x$ with a domain $X$ let $\Delta(x) = (X, \{\langle x \geq \lfloor X \rfloor \rangle\}, \{\langle x \leq \lceil X \rceil \rangle\})$. These augmented operators are used in the implementation of propagators, to derive reasons for failure or variables bounds updates,

*Example 5.* Reconsiding Example 3 in the context of learning, the bottom-up evaluation now computes $e_2.f = \Delta(x) \boxplus \Delta(y)$, $e_3.f = ([2,2], \{\}, \{\}) \boxtimes \Delta(b)$, $e_1.f = e_2.f \boxplus e_3.f$. The top-down pruning phases enforces the projection $e_1.b \leftarrow e_1.f \cap ([0,0], \{\}, \{\})$, $e_2.b \leftarrow e_2.f \cap (e_1.b \boxminus e_3.f)$, $e_3.b \leftarrow e_3.f \cap (e_1.b \boxminus e_2.f)$ and the following potential updates augmented with explanations for $x$, $y$, and $b$ : $\Delta(x) \boxed{\cap} (e_2.b \boxminus \Delta(y))$, $\Delta(y) \boxed{\cap} (e_2.b \boxminus \Delta(x))$ and $\Delta(b) \boxed{\cap} (e_3.b \boxslash [2,2])$.

Consider the constraint with domains $x \in [-2, 0]$, $y \in [-1, 0]$ and $b \in [0, 1]$ when $b$ changes to [1,1], ignoring any rounding problems for simplicity. We recalculate $e_3.f = ([2,2], \{\langle b \geq 1 \rangle\}, \{\})$, $e_2.b = ([-2,-2], \{\}, \{\langle b \geq 1 \rangle\})$, $e_2.b \boxminus \Delta(y) = ([-2,-1], \{\langle y \leq 0 \rangle\}, \{\langle b \geq 1 \rangle, \langle y \geq -1 \rangle\})$, $x = ([-2,-1], \{\langle x \geq -2 \rangle\}, \{\langle b \geq 1 \rangle, \langle y \geq -1 \rangle\})$. The explanation for the change in $x$ is $\langle b \geq 1 \rangle \wedge \langle y \geq -1 \rangle \rightarrow \langle x \leq -1 \rangle$. □

In practice it is possible, during forward evaluation, to simply flag bits indicating which of an expression children bounds are used during evaluation of its $f$ field to avoid the systematic creation and union of sets of atoms. A reason will be then reconstructed, if needed, when a variable bound is updated.

## 4 Mixed Models

In this section we present the models first introduced in [8]. An aircraft $i$ is characterized by an initial position $p_i(0) = (x_i(0), y_i(0))$, a speed $v_i$, a heading $\theta_i$ and a waypoint or destination $w_i$ along its path (see Fig. 1). We consider horizontal maneuvers between aircraft at the same altitude. At any given time, two aircraft are in conflict when the distance between them is less than a safety separation $d$. The considered maneuvers for maintaining separation involve deviations of the aircraft headings. Given that these maneuvers are orders for pilots, the starting time and deviation angle of a maneuver are discrete variables, indeed arbitrarily precise orders would be unrealistic.

### 4.1 Horizontal TCAS Model

This simple model is for emergency situations and could be used for a real-time Traffic Collision Avoidance System (TCAS): at the initial time, deviations are

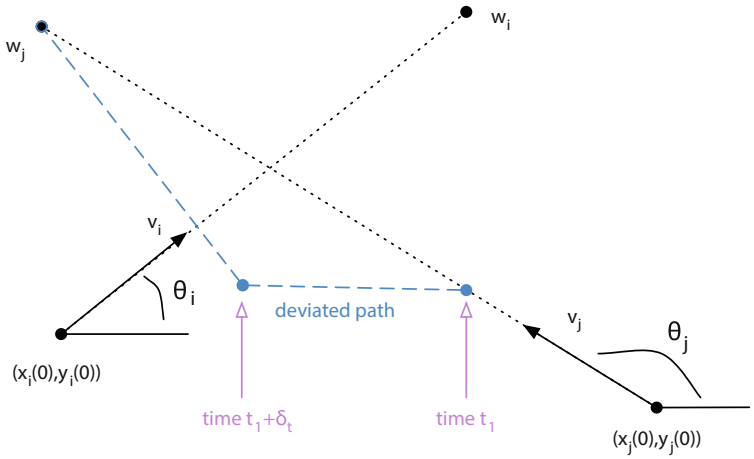


**Fig. 1.** Illustration of a deviated path to avoid conflict in the human controller model.

applied to the aircraft headings to avoid conflicts. It has one discrete decision variable $\alpha_i$ per aircraft $i$.

Given two aircraft $i$ and $j$, let $\boldsymbol{v_{ij}}$ be the relative speed and $\boldsymbol{p_{ij}}(t)$ the relative position between them at time $t$, and $d$ the safety distance. We have $\boldsymbol{p_{ij}}(t) = \boldsymbol{p_{ij}}(0) + \boldsymbol{v_{ij}}(t-0)$. These two aircraft are not in conflict at a given time $t$ if the distance separating them is greater than $d$: $P(i,j) = \boldsymbol{p_{ij}}(t)^2 - d^2 > 0$. If the discriminant $\Delta_{P(i,j)}$ of $P(i,j)$ is negative, these two aircraft will not enter into conflict, hence the inequality constraint per pair of aircraft is:

$$(\boldsymbol{p_{ij}}(0)\boldsymbol{v_{ij}})^2 - (\boldsymbol{p_{ij}}(0)^2 - d^2)v_{ij}^2 < 0$$

with :

$$\boldsymbol{p_{ij}}(0) = \begin{pmatrix} x_i(0) - x_j(0) \\ y_i(0) - y_j(0) \end{pmatrix} \qquad \boldsymbol{v_{ij}} = \begin{pmatrix} v_i cos(\theta_i + \alpha_i) - v_j cos(\theta_j + \alpha_j) \\ v_i sin(\theta_i + \alpha_i) - v_j sin(\theta_j + \alpha_j) \end{pmatrix}$$

### 4.2 Horizontal Human Controller Model

In this model we consider that the aircraft is initially heading toward a waypoint. To avoid a conflict, it is possible to deviate the aircraft from its original heading, at some time $t_1$. After an amount of time $\delta_t$ it will then head back toward its original destination. The path of an aircraft $p$ is then composed of three segments $s_{p1}$, $s_{p2}$ and $s_{p3}$. Given a pair of aircraft $i$ and $j$, a conflict can arise for each pair of segments $(s_{ix}, s_{jy})$, resulting in 9 avoidance constraints per pair of aircraft.

Given two segments $s_{ix}$, $s_{jy}$, let $P(s_{ix}, s_{jy})$ be the associated distance polynomial as defined in the previous model. The avoidance constraint is defined by the following disjunction :

- there is no common time segment during which the aircraft $i$ flies over $s_{ix}$ and aircraft $j$ flies over $s_{jy}$, or
- the discriminant of $P(s_{ix}, s_{jy})$ is negative, or
- the roots of $P(s_{ix}, s_{jy})$ are outside the common flight time.

## 5 Experiments

The lazy clause generation solver *Chuffed* [5] was extended with interval constraint support, which can be used both with or without learning. The optimization strategy chosen was the minimisation of the sum of the absolute deviations $|\alpha_i|$ which is a good approximation of how disruptive the solution is.

Other possible optimization strategies would be the minimization of number of deviated aircraft, which is a relevant criterion for a human controller, or to minimize the total lengthening of the paths, which better captures the airline operators' concerns.

In Table 1, we compare the performance of Interval Constraints without learning (IC) and with Lazy Clause Generation for simple avoidance problems (tcas$x$) and human controller model (hmc$x$) involving 4, 8 and 12 aircraft with a 90 s timeout. We obtain an exponential search space reduction from learning, with IC only solving the smaller human controller model problem. Since the aircrafts are constrained pairwise, it is likely that the nogoods transpose well to different parts of the search space. The benchmarks are available in MiniZinc format at people.unimelb.edu.au/pstuckey/atc.

**Table 1.** Comparison of Interval Constraints with and without learning.

| Problem | IC | | LCG | | |
|---|---|---|---|---|---|
| | #bts | t(s) | #lits | #bts | t(s) |
| tcas4 | 13 | 0.1 | 65 | 26 | 0.1 |
| tcas8 | 1001 | 0.2 | 237 | 128 | 0.1 |
| tcas12 | 52863 | 4.65 | 2107 | 1655 | 0.3 |

| Problem | IC | | LCG | | |
|---|---|---|---|---|---|
| | #bts | t(s) | #lits | #bts | t(s) |
| hcm4 | 513342 | 60.1 | 21206 | 27621 | 4.4 |
| hcm8 | — | >90 | 28521 | 42372 | 21.2 |
| hcm12 | — | >90 | 43234 | 64890 | 62.1 |

## 6 Related Work and Conclusion

Constraint systems such as $ECL^iPS^e$ [4] support both integers and interval constraints. The framework presented in [11] and the SMT solver HySAT [9], based on the iSAT algorithm [10], combine interval constraint propagation with the learning framework of SMT to solve real constraints, implementing a form of *hc*3 augmented by explanations (as opposed to *hc*4 that we implement).

The SMT approaches do not tightly integrate the handling of integer and interval variables which is a distinct disadvantage for applications such as ATC. They both elide the issue of too many literals appearing in the trail, which may be because the benchmarks they use are quite distinct from those appearing in typical CP interval problems where interval propagation can take many iterations to quiesce. Hence it appears the implementation issues for LCG and SMT for intervals are quite different.

The domain of Air Traffic Management is very complex and contains many hard combinatorial problems. Although there is little existing work regarding

continuous or mixed problems, CP approaches has been developed for many of the combinatorial problems in this area such as arrival management, runway allocation, workload management. See [1] for a survey.

## References


1. Allignol, C., Barnier, N., Flener, P., Pearson, J.: Constraint programming for air traffic management: a survey. Knowl. Eng. Rev. **27**(03), 361–392 (2012)
2. Benhamou, F., McAllester, D., Van Hentenryck, P.: Clp (intervals) revisited. Rapport technique, p. 30. Citeseer (1994)
3. Benhamou, F.: Interval constraint logic programming. In: Podelski, A. (ed.) Constraint Programming: Basics and Trends. LNCS, vol. 910, pp. 1–21. Springer, Heidelberg (1995)
4. Brisset, P., Sakkout, H.E., Fruhwirth, T., Gervet, C., Harvey, W., Meier, M., Novello, S., Le Provost, T., Schimpf, J., Shen, K., Wallace, M.: ECLiPSe Constraint Library Manual, October 2005
5. Chu, G.G.: Improving combinatorial optimization. Ph.d. thesis, The University of Melbourne (2011)
6. Durand, N., Alliot, J.M., Noailles, J.: Automatic aircraft conflict resolution using genetic algorithms. In: Proceedings of the 1996 ACM Symposium on Applied Computing, pp. 289–298. ACM (1996)
7. Feydy, T., Stuckey, P.J.: Lazy clause generation reengineered. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 352–366. Springer, Heidelberg (2009)
8. Feydy, T., Barnier, N., Brisset, P., Durand, N.: Mixed conflict model for air traffic control. In: IntCp 2005, Workshop on Interval analysis, constraint propagation, applications (2005)
9. Fränzle, M., Herde, C.: Hysat: an efficient proof engine for bounded model checking of hybrid systems. Formal Methods Syst. Des. **30**(3), 179–198 (2007)
10. Franzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure. J. Satisfiability, Boolean Model. Comput. **1**, 209–236 (2007)
11. Haller, L., Griggio, A., Brain, M., Kroening, D.: Deciding floating-point logic with systematic abstraction. In: Formal Methods in Computer-Aided Design (FMCAD 2012), pp. 131–140. IEEE (2012)
12. Ilog, S.: Revising hull and box consistency. In: Logic Programming: Proceedings of the 1999 International Conference on Logic Programming, p. 230. MIT press (1999)
13. Moore, R.E.: Interval Analysis, vol. 4. Prentice-Hall, Englewood Cliffs (1966)
14. Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proceedings of the 39th Design Automation Conference (DAC 2001) (2001)
15. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
16. Veksler, M., Strichman, O.: Learning general constraints in CSP. In: Michel, L. (ed.) CPAIOR 2015. LNCS, vol. 9075, pp. 410–426. Springer, Heidelberg (2015). http://dx.doi.org/10.1007/978-3-319-18008-3_28