The argmax constraint

Graeme Gange[0000−0002−1354−431X] and Peter J. Stuckey[0000−0003−2186−0459]

Monash University, Australia, {graeme.gange,peter.stuckey}@monash.edu

Abstract. The ARGMAX function returns the index of the (first copy of the) maximum value occuring in a list of values. ARGMAX is important in models where we choose a characteristic value based on a seperate criteria, and for modelling neural networks which make use of ARGMAX in their definition. The ARGMAX constraint has been studied for the special case of its use to find the index of the first `true` in a list of Booleans, since this is useful for encoding if-then-else chains. Here we examine the general ARGMAX constraint for the first time. We define an efficient domain consistent propagator and a decomposition for integers which implements domain consistency.

1 Introduction

The ARGMAX function returns the index of the maximum value occuring in a list of values. Importantly, if two or more entries in the list take the maximum value it returns the index of the *first* such value. While ARGMAX is a well understood mathematical operator, it not nearly as widely used as, for example, MAX. In that sense it has not been carefully considered previously. Recently, a propagation algorithm for ARGMAX was developed for the restricted case of a Boolean array with known upper bound [12]. However, the general case of non-fixed numeric arguments has never been considered in detail.

ARGMAX is important as a constraint since it allows us to embed optimization problems as subproblems of another problem, where we want have access to the solution that leads to the optimal result, rather than just the optimal value.

Example 1. ARGMAX is used to model selecting a value based on a separate criteria. Let (x_i, y_i) be pairs of possible values to be selected x_i and the worthiness criteria for that selection y_i, then a model to select the best value v is $v = x_{argmax\ y}$. More concretely, in MiniZinc this is modelled as

```
array[1..n] of var int: x;
array[1..n] of var int: y;
var int: v = x[arg_max(y)];
```

ARGMAX is increasingly important for discrete optimization because of its use in machine learning models. Many forms of machine learning rely on ARGMAX to define the result of a machine learning algorithm. If we want to use these models inside a discrete optimization problem then we need to be able to model the ARGMAX behaviour.

Example 2. An important modern use of ARGMAX is in the output layer of classification style neural nets, where the last layer is typically an argmax returning the class which is most highly likely to be present. A simple MiniZinc model for such a neural net layer is

```
array[CLASS] of var float: likelihood; % likelihood from NN of each class
var CLASS: x = arg_max(likelihood);    % most likely class
```

This is particularly important given the trend towards embedding classifiers as part of optimization models (e.g. [2]). But also given recent interest in techniques for *explainable AI* (e.g. [4, 1]). Given a classifier K, observations (or *hypotheses*) H and output class c, an *abductive explanation* [7] is a (minimal) subset of H which still guarantees output class c. For multi-class classification problems, this requires querying whether $H' \wedge c \neq \text{ARGMAX}(\textit{likelehood})$ is satisfiable for various subsets H' of H. Hence the need to reason about classifiers.

In this paper we investigate the ARGMAX constraint from a constraint programming perspective. The contributions of this paper are:

- A complete characterization of the domain consistent propagation possible from an ARGMAX constraint;
- An efficient domain consistent propagator for ARGMAX;
- A new decomposition for ARGMAX over integers which maintains domain consistency; and
- Experimental results showing the benefit of the new propagator and decomposition

The remainder of this paper is organized as follows. In the next section we introduce notation. In Section 3 we examine the current default decomposition for ARGMAX in the MiniZinc [9] library, which is the only existing ARGMAX implementation we are aware of, and highlight its shortcomings. In Section 4 we define all the propagation rules that are possible for an ARGMAX constraint, and indeed prove that applying them results in a domain consistent propagator. In Section 5 we define a new decomposition for ARGMAX which is domain consistent (under some easy to satisfy conditions about the constraints that implement it). In Section 6 we discuss a simpler variant of ARGMAX. In Section 7 we provide experiments to demonstrate the effectiveness of the new propagator and decomposition, on both unit tests and complete examples that make use of ARGMAX. Finally in Section 8 we conclude.

2 Preliminaries

A constraint problem $\mathcal{P} = \langle \mathcal{V}, D_{init}, \mathcal{C}, o \rangle$ consists of a set of variables $\mathcal{V}$, an initial domain D_{init}, a set of propagators for constraints $\mathcal{C}$ and an objective o w.l.o.g. to be minimized. A *domain* D is a mapping from each $v \in \mathcal{V}$ variables to a set of integer values $D(v)$, which defined the possible values that v can take. We will use *range notation* $l..u$ to represent the set of integer $R = \{d \in \mathbb{Z} | l \leq d \wedge d \leq u\}$ to define domains.

An assignment θ is a mapping from $\mathcal{V}$ to integers. We say $\theta \in D$ iff $\forall v \in \mathcal{V}, \theta(v) \in D(v)$. An assignment satisfies a constraint $c \in \mathcal{C}$ if $\theta(c)$ is true. A *solution* of problem $\mathcal{P} = \langle \mathcal{V}, D_{init}, \mathcal{C}, o \rangle$ is an assignment $\theta \in D_{init}$ that satisfies all constraints $c \in \mathcal{C}$. An *optimal solution* θ of $\mathcal{P}$ is a solution of $\mathcal{P}$ such that for all other solutions θ' of $\mathcal{P}$, $\theta(o) \leq \theta'(o)$.

We will be interested in discussing propagation behaviour so we introduce some finer grained notation for reasoning about the current state of a CP solver. The least value a variable x can take in domain D is $\mathsf{lb}_D(x) = \min D(x)$, similarly the greatest value variable x can take in domain D is $\mathsf{ub}_D(x) = \max D(x)$. The *atomic constraints* for problem $\mathcal{P}$ are defined as

$$\{\mathit{false}\} \cup \{\langle x = d \rangle \mid x \in \mathcal{V}, d \in D_{init}(x)\} \cup \{\langle x \geq d \rangle \mid x \in \mathcal{V}, d \in D_{init}(x) - \{\mathsf{lb}_{D_{init}}(x)\}$$

We treat atomic constraints as propositions that reason about the current domain D. For example, $\langle x = d \rangle$ is true in the current domain D if $D(x) = \{d\}$, false if $d \notin D(x)$ and unknown otherwise. Similarly $\langle x \geq d \rangle$ is true if $\mathsf{lb}_D(x) \geq d$, false if $\mathsf{ub}_D(x) < d$ and unknown otherwise.

For real variables[1] we need to introduce the additional open atomic constraints $\{\langle x > d \rangle \mid d \in D_{init}(x) - \{lb_{D_{init}(x)}(x)\}$. For integers $\langle x > d \rangle$ is just shorthand for $\langle x \geq d + 1 \rangle$.

We use the notation $\langle x \neq d \rangle$ as shorthand for $\neg \langle x = d \rangle$, $\langle x \leq d \rangle$ as shorthand for $\neg \langle x > d \rangle$, and $\langle x < d \rangle$ as shorthand for $\neg \langle x \geq d \rangle$.

Finally, we introduce the notation $x \ll^b y$ where x and y are numeric and b is Boolean, which is shorthand for $x \leq y$ if $b = 0$ and $x < y$ where $b = 1$. That is b is an indicator of the *strictness* of the comparison. $x \gg^b y$ is similarly defined. Note that for integers $x \ll^b y$ is equivalent to $x \leq y - b$, and $x \gg^b y$ is equivalent to $x \geq y + b$. The importance of this notation is that given an array $[x_1, \ldots, x_n]$ if $x_i \ll^{i<j} x_j, i \neq j$ then position i cannot be the argmax of the array since x_j is either equal and earlier or greater.

3 Current argmax decomposition

The standard MiniZinc encoding (as of `2.4.3`) of $\mathtt{z} = \mathit{argmax}([\mathtt{x_1}, \ldots, \mathtt{x_n}])$ is reasonably straightforward: it introduces auxiliary variables m_i and p_i respectively indicating the value and position of the maximum over the first i elements, with constraints:

$$\begin{aligned} m_1 &= x_1 \wedge p_1 = 1 \\ m_{i+1} &= \max(m_i, x_{i+1}) \\ p_{i+1} &= \textbf{if } m_i < x_{i+1} \textbf{ then } i+1 \textbf{ else } p_i \\ z &= p_n \end{aligned}$$

This encoding is sound, but unsatisfactory. Assuming an appropriate encoding of the conditional [12] it can propagate forwards to z, but neither enforces domain consistency on z nor effectively incorporates knowledge of z back into the bounds.

[1] Implemented by floating point ranges.

Example 3. Consider $\mathtt{z} = \mathit{argmax}([\mathtt{x_1}, \mathtt{x_2}, \mathtt{x_3}, \mathtt{x_4}])$, with current domains $\{z \mapsto \{1, 3, 4\}, x_1 \in 1..3, x_2 \in 2..10, x_3 \in 3..5, x_4 \in 4..6\}$. Using the decomposition above, we find:

i	1	2	3	4
$D(m_i)$	1..3	2..10	3..10	4..10
$D(p_i)$	$\{1\}$	1..2	1..3	$\{1, 3, 4\}$

At this point, no further pruning can be performed. However, we miss several inferences. First, since $\mathsf{lb}_D(x_4) > \mathsf{ub}_D(x_1)$, we can infer $1 \notin D(z)$. And for x_2 *not* to be the max, it can be at most 5. □

The underlying problem with the decomposition is that it doesnt propagate information backwards about p_i. We know that $p_4 \neq 2$ which means $p_2 \neq 2$ which would allow the pruning of x_2.

4 Propagation behaviour of argmax

A domain consistent propagator for ARGMAX is the strongest possible implementation we can hope for in a constraint programming solver. The constraint $z = \mathit{argmax}([x_1, \ldots, x_n])$ can correctly propagate as follows:

– If a value j is known not to be the maximum index value z, then it cannot take a value greater than (or equal to if it occurs earlier than the index of the max ubi) to the maximum possible value of the maximum of $x_1, \ldots, x_n$, ub:

$$\begin{aligned} \forall j. \mathit{ubi} = \mathit{argmax}_{i \in D(z)}\, \mathsf{ub}_D(x_i) \wedge \mathit{ub} = \mathsf{ub}_D(x_{\mathit{ubi}}) \wedge \langle z \neq j \rangle \\ \Rightarrow \left\langle x_j \ll^{j < \mathit{ubi}} \mathit{ub} \right\rangle \end{aligned} \quad (1)$$

– If a variable x_j's maximum value is less than the least possible maximum value of $x_1, \ldots, x_n$, lb, (or equal to and it occurs at a later position than where this value first occurs lbi) then it cannot be the argmax position z:

$$\begin{aligned} \forall j. \mathit{lbi} = \mathit{argmax}_{i \in 1..n}\, \mathsf{lb}_D(x_i) \wedge \mathit{lb} = \mathsf{lb}_D(x_{\mathit{lbi}}) \\ \wedge \left\langle x_j \ll^{j < \mathit{lbi}} \mathit{lb} \right\rangle \Rightarrow \langle z \neq j \rangle \end{aligned} \quad (2)$$

– If the index of the max z is known to be j then x_j cannot take values that would be too low to be consistent with the least possible value lb of the max of $x_1, \ldots, x_n$:

$$\begin{aligned} D(z) = \{j\} \wedge \mathit{lbi} = \mathit{argmax}_{i \in 1..n}\, \mathsf{lb}_D(x_i) \wedge \mathit{lb} = \mathsf{lb}_D(x_{\mathit{lbi}}) \\ \Rightarrow \left\langle x_j \gg^{j > \mathit{lbi}} \mathit{lb} \right\rangle \end{aligned} \quad (3)$$

Example 4. Consider Example 3. Figure 1 illustrates the application of the propagation rules. Equation (1) finds $\mathit{ubi} = 4, \mathit{ub} = 6$ and prunes any values which would set z to a removed value (marked in green), inferring $x_2 < 6$. Equation (2) finds $\mathit{lbi} = 4, \mathit{lb} = 4$ and removes any indices which can no longer become lbi, discovering $z \neq 1$ since $x_1 < 4$.

Suppose instead that also $\mathcal{D}(z) = 1$ then Equation 3 will apply enforcing that $x_1 \geq 4$ and thus causing failure. □

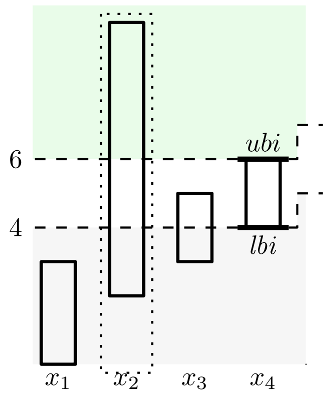


Fig. 1: Propagation of ARGMAX from domains in Example 3. 2 has already been removed from $\mathcal{D}(z)$, so cannot be chosen as *ubi*.

We can indeed show that these three rules define all the propagation behaviour we can correctly define for the constraint $z = \mathit{argmax}([x_1, \ldots, x_n])$.

Theorem 1. *Exhaustive application of the rules of Equations (1), (2) and (3) enforces domain consistency for* $z = \mathit{argmax}([x_1, \ldots, x_n])$.

Proof. Given domain D which results from the exhaustive application of the rules of Equations (1), (2) and (3), we show that it provides a supporting solution for each value remaining in the domain.

Suppose $j \in D(z)$. We construct a solution $\theta \in D$ of the constraint where $z = j$. We claim that $\theta = \{z \mapsto j, x_j \mapsto \mathsf{ub}_D(x_j), x_i \mapsto \mathsf{lb}_D(x_i), i \neq j\}$ is a solution. Suppose to the contrary, then there exists $i \neq j$ where either $\mathsf{lb}_D(x_i) > \mathsf{ub}_D(x_j)$ or $\mathsf{lb}_D(x_i) = \mathsf{ub}_D(x_j) \wedge i < j$. Let lbi and lb be defined as in Equation (2), then clearly $\mathit{lb} \geq \mathsf{lb}_D(x_i)$ and the conditions for the Equation (2) hold, so $j \notin D(z)$. Contradiction.

Suppose $v \in D(x_j)$. We construct a solution $\theta \in D$ of the constraint where $x_j = v$. Let ubi and ub be defined as in (Equation 1).

Consider the case where $j \neq \mathit{ubi}$. Consider the valuation $\theta = \{z \mapsto \mathit{ubi}, x_j \mapsto v, x_i \mapsto \mathsf{ub}_D(x_i), i \neq j\}$. We claim this is a solution. Suppose to the contrary then $v > \mathsf{ub}_D(x_{\mathit{ubi}})$ or $v = \mathsf{ub}_D(x_{\mathit{ubi}}) \wedge j < \mathit{ubi}$. For this to occur $j \notin D(z)$ otherwise its impossible by the definition of ubi and ub. But then all the conditions of Equation (1) apply and we would have imposed $x_j \ll^{j<\mathit{ubi}} \mathit{ub}$, meaning $v \notin D(x_j)$. Contradiction.

Consider the case where $j = \mathit{ubi}$. Consider the valuation $\theta = \{z \mapsto j, x_j \mapsto v, x_i \mapsto \mathsf{lb}_D(x_i)), i \neq j\}$. We claim this is a solution or we can construct an alternate. Note that $j = \mathit{ubi} \in D(z)$ by definition.

Suppose to the contrary θ is not a solution. Then there exists $i \neq j$ where either $\mathsf{lb}_D(x_i) > v$ or $\mathsf{lb}_D(x_i) = v \wedge i < j$. Let lbi and lb be defined as in Equation (3), then clearly $\mathit{lb} \geq v$. There are two cases. If $D(z) = \{j\}$ then

```
ARGMAX($[x_1, \dots, x_n], z$)
    $ubi \leftarrow argmax(\mathsf{ub}_D(x_i) | i \in \mathcal{D}(z))$
    $ub \leftarrow \mathsf{ub}_D(x_{ubi})$
    for($j \in 1..n - \mathcal{D}(z)$)
        post($\langle x_j \ll^{j<ubi} ub \rangle$)
    $lbi \leftarrow argmax(\mathsf{lb}_D(x_i) | i \in 1..n)$
    $lb \leftarrow \mathsf{lb}_D(x_{lbi})$
    for($j \in \mathcal{D}(z)$)
        if($\mathsf{ub}_D(x_j) \ll^{j<lbi} lb$):
            post($\langle z \neq j \rangle$)
    if($\mathcal{D}(z) = \{j\}$)
        post($\langle x_j \gg^{lbi<j} \mathsf{lb}_D(x_{lbi}) \rangle$)
```

Fig. 2: Basic propagator for $\mathtt{z} = argmax([\mathtt{x_1}, ..., \mathtt{x_n}])$.

the conditions of Equation (3) hold and we would have imposed $x_j \gg^{j>lbi} lb$, meaning $v \notin D(x_j)$. Contradiction.

Otherwise, there is some other value $j' \neq j, j' \in D(z)$. We construct $\theta' = \{z \mapsto j', x_j \mapsto v, x_{j'} \mapsto \mathsf{ub}_D(x'_j), x_i \mapsto \mathsf{lb}_D(x_i), i \notin \{j, j'\}\}$. Suppose to the contrary that θ' is not a solution, then there exists $i \neq j'$ where either $\mathsf{lb}_D(x_i) > \mathsf{ub}_D(x_{j'})$ or $\mathsf{lb}_D(x_i) = \mathsf{ub}_D(x_{j'}) \wedge i < j'$. Note that $v \leq lb$ so it cannot be the unique cause why $\mathsf{ub}_D(x_{j'})$ does not represent the max value assigned to an x variable in θ'. But this means the conditions of Equation 2 apply for $j = j'$ which would mean we would propagate $z \neq j'$. Contradiction.

Since every value in the domain D is supported by a solution θ of the constraint in D the propagation enforces domain consistency. □

Theorem 1 shows that the propagation rules enforce domain consistency, but we need to enforce them efficiently. A straightforward propagator implementation is shown in Figure 2, more or less a direct implementation of rules (1)-(3). Note that **post**(a) make the domain change given by the atomic constraint a. The propagator is clearly $O(n)$.

Table 1 implicitly defines an incremental propagator for the ARGMAX constraint. It shows how Equations (1–3) are applied on relevant domain change events. *lbi* and *ubi* are maintained between calls. Whenever the current *ubi* is (potentially) invalidated, we call filter-ub to compute an updated *ubi*, then apply Equation (1) to tighten upper bonds. When the maximum lower bound increases, filter-z is called to apply Equation (2) to prune any invalidated indices. Equation (2) is also applied upon $\langle x_i \leq k \rangle$, but only for the changed index. The complexity of the incremental propagator is still $O(n)$ since it may need to update *ubi* or filter multiple domains.

Example 5. Consider Example 3 where the initial domains are instead $\{z \in \{3, 4\}, x_1 \in 1..3, x_2 \in 2..5, x_3 \in 3..5, x_4 \in 4..6\}$ which is a fixpoint for the rules. The propagator state has $ubi = 4$, and $lbi = 4$. If we have new event $\langle x_2 \geq 3 \rangle$ then the test immediately fails and nothing happens. If we have new event $\langle x_1 \leq 2 \rangle$

Table 1: Propagation events for $\mathtt{z} = \mathtt{argmax}([\mathtt{x_1}, ..., \mathtt{x_n}])$.

```
on $\langle \mathbf{z \neq j} \rangle$:
if($j = ubi$):
    filter-ub($\mathcal{D}$)
else:
    post($x_i \ll^{j<lbi} \mathsf{ub}_D(x_{ubi})$)
if($\mathcal{D}(z) = \{j'\}$):
    filter-lb($j', \mathcal{D}$)
-----------------------------------------
on $\langle \mathbf{x_j \leq k} \rangle$:
if($j = ubi$):
    filter-ub($\mathcal{D}$)
if($k \ll^{j<lbi} \mathsf{lb}_D(x_{lbi})$):
    post($z \neq j$)
-----------------------------------------
on $\langle \mathbf{x_j \geq k} \rangle$:
if($\neg(k \ll^{j \leq lbi} \mathsf{lb}_D(x_{lbi}))$)
    $lbi \leftarrow j$
    if($\mathcal{D}(z) = \{j'\}$)
        filter-lb($j', \mathcal{D}$)
    else
        filter-z($\mathcal{D}$)
```

```
filter-ub($\mathcal{D}$):
    $ubi \leftarrow argmax(\mathsf{ub}_D(x_i) | i \in \mathcal{D}(z))$
    $ub \leftarrow \mathsf{ub}_D(x_{ubi})$
    for($j \in 1..n - \mathcal{D}(z)$)
        post($\langle x_j \ll^{j<ubi} ub \rangle$)

filter-z($\mathcal{D}$):
    for($j \in \mathcal{D}(z)$):
        if($\mathsf{ub}_D(x_j) \ll^{j<lbi} lb$):
            post($\langle z \neq j \rangle$)

filter-lb($j, \mathcal{D}$):
    post($\langle x_j \gg^{lbi<j} \mathsf{lb}_D(x_{lbi}) \rangle$)
    $lbi \leftarrow j$
```

similarly both test fails and nothing happens. On $\langle z \neq 4 \rangle$ we determine the new upper bound $ub = 5$ and $ubi = 3$ and post $\langle x_4 \leq 5 \rangle$ and since $D(z)$ is now a singleton we also post $\langle x_3 \geq 4 \rangle$. □

As mentioned above, *lbi* and *ubi* are perserved between propagator calls. In practice, we also persistently track the set $I_r = \{i \in 1..n{-}D(z) \mid \neg(ub(x_i) \ll^{i<lbi} lb(x_{lbi}))\}$. This is the set of indices which may be updated by FILTER-UB: they have been removed from $\mathcal{D}(z)$, but would otherwise be candidates for the maximum index. Then in FILTER-UB we iterate over I_r directly, skipping irrelevant indices.

Explaining Propagation In any lazy clause generation [10] solver (as we shall use in Section 7), a propagator for constraint c must provide an *explanation* for any inference φ: a conjunction E of atomic constraints such that $D \rightarrow E$, and $E \wedge c \rightarrow \varphi$. Fortunately, for ARGMAX the explanations are readily extracted from equations (1 – 3). The explanation for Equation (1) for $\langle x_j \ll^{j<ubi} ub \rangle$ is the most involved, as we need to justify *why* $ub(x_{ubi})$ takes its value. That is, for each index, why it does not support a greater upper bound. The explanation becomes:

$$\forall_{k \in \mathcal{D}(z)} \left\langle x_k \ll^{k<ubi} ub \right\rangle \wedge \forall_{k \notin \mathcal{D}(z)} \langle z \neq k \rangle \rightarrow \left\langle x_j \ll^{j<ubi} ub \right\rangle.$$

The explanation from Equation (2) as to why $\langle z \neq j \rangle$ is that there was some other index k such that $lb(x_k) \gg^{k>j} \mathsf{ub}_D(x_j)$. At the time of propagation, *lbi* is always such an index, so our explanation is

$$\left\langle x_j \ll^{j<lbi} lb \right\rangle \wedge \langle x_{lbi} \geq lb \rangle \rightarrow \langle z \neq j \rangle\,.$$

Similarly, the explanation from Equation (3) for $\left\langle x_j \gg^{lbi>j} c \right\rangle$ is always

$$\langle z = j \rangle \wedge \neg \left\langle x_{lbi} \ll^{lbi<j} c \right\rangle \rightarrow \left\langle x_j \gg^{lbi>j} c \right\rangle.$$

Example 6. Examining the propagation in Example 4, the explanation for $\langle x_2 \leq 5 \rangle$ is $\langle x_1 < 6 \rangle \wedge \langle x_3 \leq 6 \rangle \wedge \langle x_4 \leq 6 \rangle \wedge \langle z \neq 2 \rangle \rightarrow \langle x_2 < 6 \rangle$, while the explanation for $\langle z \neq 1 \rangle$ is $\langle x_1 < 4 \rangle \wedge \langle x_4 \geq 4 \rangle \rightarrow \langle z \neq 1 \rangle$. □

5 Decomposition

For the case of integer values, can produce a better decomposition of the ARGMAX propagator by reasoning lexicographically about the pairs $(x_j, -j)$. The maximum of these pairs defines the position and value of the max. Using this idea results in the decomposition below. We introduce auxiliary variables q_j to represent the value of the pair $(x_j, -j)$ as a single integer $n \times x_j + n - j$. We introduce M to represent the maximum value of the pairs. The decomposition is:

$$\begin{aligned} q_j &= n \times x_j + n - j, \quad 1 \leq j \leq n \\ M &= max([q_1, \ldots, q_n]) \\ z \neq j \leftrightarrow M &> q_j, \quad 1 \leq j \leq n \end{aligned}$$

Note that $q_i < q_j$ iff $x_i \ll^{i<j} x_j$. (We cannot do the same for real or floating point values, as we cannot readily define a view T s.t. $T(x_i, i) < T(x_j, j) \equiv x_i \ll^{i<j} x_j$). Note that the q_j variables can be defined by variable views [11] and hence dont exist as separate variables, thus the main cost of the decomposition is the global `max` constraint and the reified inequalities $b_j \leftrightarrow M > q_j$, where b_j is equated to $\langle z \neq j \rangle$.

Theorem 2. *The decomposition defined above implements domain consistency for the constraint* $z = \text{ARGMAX}([x_1, \ldots, x_n]$*, assuming bounds consistent propagation of* `max` *and the reified inequalities, and domain consistent propagation of* $q_j = n \times x_j + (u - j)$.

Proof. We show that it implements all the propagation behaviour discussed in Section 4. Note that $q_j = n \times x_j + (n - j)$ is often implemented by a view [11] which is equivalent to domain consistent propagation.

Examining Equation (1) and give a domain D where ubi and ub are defined as shown in the equation. Suppose the value $\mathsf{ub}_D(M)$ arises from position i. If $i \in D(z)$ then $ubi = i$ by definition and bounds consistency of the max and $\mathsf{ub}_D(M) = n \times ub - (n - ubi)$. The fact that $z \neq j$ will immediately assert $q_j < n \times ub + (n - ubi)$ which is equivalent to $x_j \leq ub - (j < ubi)$. This holds since the propagation of $q_j = n \times x_j + (n - j)$ is domain consistent. Now consider when $i \notin D(z)$. Then by bounds consistency of the max $\mathsf{ub}_D(M) = n \times \mathsf{ub}_D(x_i) - (n - i)$.

But since $z \neq i$ the decomposition enforces $q_i < n \times \mathsf{ub}_D(x_i) - (n - i)$ or equivalently $x_i < \mathsf{ub}_D(x_i)$ by the domain consistent propagation of $q_j = n \times x_j + (n-j)$. This reduces the upper bound of x_i. This process will continue until the (new) index i giving the max value for M is in $D(z)$ otherwise its upper bound will also be reduced. So eventually we hit the first case above. Hence Equation (1) is implemented by the decomposition.

Examining Equation (2) and give a domain D where lbi and lb are defined as shown in the equation. By bounds consistency of the max $\mathsf{lb}_D(M) = n \times lb + (n - lbi)$. The fact that $x_{lbi} \leq lb - (j < lbi)$ means that $q_{lbi} < n \times lb + (n - lbi)$ is known to be true (by domain consistency of $q_i = n \times x_j + (n-j)$), meaning that the constraint will immediately assert $z \neq j$. Hence Equation (2) is implemented by the decomposition.

Examining Equation (3) and give a domain D where lbi and lb are defined as shown in the equation. By bounds consistency of the max $\mathsf{lb}_D(M) = n \times lb + (n - lbi)$. Making $z = j$ enforces $M \leq q_j$ which given the bound on M bounds consistency enforces (for all q_k and in particular) $q_j \geq n \times lb + (n - lbi)$ which is by domain propagation of $q_i = n \times x_j + (n-j)$ equivalent to $x_j \geq lb + (lbi < j)$. Hence Equation (3) is implemented by the decomposition. □

The decomposition has an advantage over the propagator since the introduced variables M and q allow for concise explanations.

Example 7. Consider Example 3. Using the new decomposition we the definition of the new variables gives domains

i	1	2	3	4	M
$D(q_i)$	7..15	10..42	13..21	16..24	16..42

The fact that $q_1 < M$ must hold enforces $\langle z \neq 1 \rangle$. The explanation is simply $\langle q_1 \leq 15 \rangle \wedge \langle M \geq 16 \rangle \rightarrow \langle z \neq 1 \rangle$. Assuming the q are defined by views this will be $\langle x_1 \leq 3 \rangle \wedge \langle M \geq 16 \rangle \rightarrow \langle z \neq 1 \rangle$. Note that this is more reusable than the explanation by the propagator which commits to a single variable x_4.

Then $\langle z \neq 2 \rangle$ enforces that $\langle q_2 < 42 \rangle$ which removes the top value from its domain, this then changes the domain of x_2 to 2..9 and q_2 to 10..38, the process repeats until the domain of x_2 becomes 2..5 and q_2 is 10..22 and M is 16..24. The explanation chain is long, the final explanation for $\langle x_2 \leq 5 \rangle$ is $\langle z \neq 2 \rangle \wedge \langle M \leq 26 \rangle \rightarrow \langle x_2 \leq 5 \rangle$, which again is agnostic about the reason for the upper bound on M. □

6 Argmax Variant

There is a very similar constraint to ARGMAX in the global constraint catalog, called `max_index` [8]. This relation is defined as

$$max\_index(z, [x_1, \ldots, x_n]) \equiv x_z = max([x_1, \ldots, x_n])$$

so it simply requires z to be *an* index where the maximum value occurs, not the first. This means it is not functional, and cannot thus be used for encoding conditional constructs like ARGMAX. The simple definition by decomposition above is not domain consistent.

Example 8. Consider the problem of Example 3 where the constraint is instead $\texttt{max\_index}(\texttt{z}, [\texttt{x}_1, \texttt{x}_2, \texttt{x}_3, \texttt{x}_4])$. Then the decomposition determines $max(x) \in 4..10$ and propagates $z \neq 1$. It does not determine that $x_2 \leq 6$ which is a consequence of the constraint since if $x_2 = 7$ then z could not be the correct index 2. □

We can define the domain consistent propagator for `max_index` by using rules analogous to rules (1)–(3).

– If a value j is known not to be the maximum index value z, then it cannot take a value greater than the maximum possible value of the maximum of $x_1, \ldots, x_n$, ub:

$$\forall j. ub = \max_{i \in D(z)} \mathsf{ub}_D(x_i) \wedge \langle z \neq j \rangle \Rightarrow \langle x_j \leq ub \rangle \tag{4}$$

– If a variable x_j's maximum value is less than the least possible maximum value of $x_1, \ldots, x_n$, lb, then it cannot be the argmax position z:

$$\forall j. lb = \max_{i \in 1..n} \mathsf{lb}_D(x_i) \wedge \langle x_j < lb \rangle \Rightarrow \langle z \neq j \rangle \tag{5}$$

– If the index of the max z is known to be j then x_j cannot take values that would be too low to be consistent with the least possible value lb of the max of $x_1, \ldots, x_n$:

$$D(z) = \{j\} \wedge lb = \max_{i \in 1..n} \mathsf{lb}_D(x_i) \Rightarrow \langle x_j \geq lb \rangle \tag{6}$$

We can straightforwardly modify the propagator of Figure 1 and incremental update rules of Table 1 to implement this simpler variant.

7 Experimental Evaluation

We have implemented the ARGMAX propagator presented in Section 4 in `geas` [5], a lazy clause generation-based CP solver. We compare the standard MiniZinc decomposition DEC with the new propagator PROP and and the new decomposition NEW presented in Section 5. Experiments were conducted on a Core i7 7820-HQ 2.9Ghz with 32 Gb ram, running Ubuntu 18.04. All experiments were conducted with a 600 second timeout. All times are in seconds; a '—' entry indicates a timeout. Models and instances used in this section are available at `https://gkgange.github.io/papers/2020/argmax`.

7.1 Unit tests

Let π be an $n \times n$ matrix of values generated randomly in $[1, n]$. We construct a constraint system:

n	DEC	PROP	NEW
30	4.1	**0.07**	0.11
40	57.16	**0.14**	0.25
50	362.71	**0.86**	1.32
60	—	**3.88**	5.72
70	—	**5.96**	9.89
80	—	**21.82**	31.67
90	—	**43.93**	82.46
100	—	**152.32**	224.06
110	—	**258.95**	345.13
120	—	—	—

Fig. 3: Unit-test performance for ARGMAX on integer variables, for increasing n.

```
forall (i in 1..n) (x[i] = arg_max( [ pi[i, j] * x[j] | j in 1..n ] ));
alldifferent(x);
```

The constraint system is vanishingly unlikely to be satisfiable (and indeed this occurs in none of our tests).

Results comparing the time for the three ARGMAX implementations to prove unsatisfiability are given in Figure 3, for increasing values of n. The standard decomposition is evidently not competitive. For these problems, the faster propagation speed of the propagator is more essential than the search reduction from more expressive nogoods generated by the decomposition. Its often the case that nogood learning is of limited utility for random problems.

7.2 Wireless signal selection

We consider a modified facility location problem, modelling a WLAN planning task. The problem is to assign signal strengths to a set of access points A, maximizing connection quality for a set of clients C, subject to:

- Each client connects to the access point with maximum (observed) signal strength;
- Each client has a desired throughput; and
- Each access point has a maximum total throughput; all connected clients incur a penalty if their combined bandwidth demands are not met.

We generated instances by placing n access points and m clients randomly on a 70×70 grid, for $n \in \{10, 15, 20\}$ and $m \in \{40, 55, 70, 85\}$. Each client is assigned a random throughput chosen uniformly in [1, 100], and each access point has maximum throughput 300. The ARGMAX constraint is used for the first constraint.

The results shown in Figure 4 given the average time on 10 instances of each size, and in parentheses the number of the 10 instances that succeeded. They clearly demonstrate that the new propagator and decomposition are subtantially

APs	clients	DEC	PROP	NEW
10	40	22.81 (10)	1.87 (10)	**1.40** (10)
10	55	264.87 (9)	91.47 (10)	**31.01** (10)
10	70	570.91 (2)	416.46 (7)	**173.07** (10)
10	85	— (0)	564.23 (1)	**461.60** (6)
15	40	49.32 (10)	3.86 (10)	**2.00** (10)
15	55	437.44 (5)	185.84 (8)	**78.14** (10)
15	70	583.80 (1)	403.88 (4)	**310.15** (7)
15	85	— (0)	384.45 (5)	**164.66** (10)
20	40	78.58 (10)	7.03 (10)	**5.81** (10)
20	55	366.93 (7)	96.25 (9)	**55.10** (10)
20	70	545.76 (3)	303.65 (7)	**138.06** (9)
20	85	— (0)	574.85 (1)	**519.16** (2)

Fig. 4: Comparison of methods on wireless signal selection problems of various sizes. Columns give average runtime over instances of each size (timeouts are counted as 600s), and number of instances solved (out of 10).

better than the current decomposition. Here the intermediate variable M provides more reusable nogoods, meaning the new decomposition improves over the propagator.

7.3 Boosted Tree Explanation

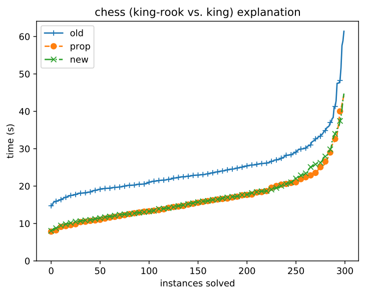
For this experiment, we considered the SOYBEAN and CHESS (KING-ROOK VS. KING) classification tasks from the UCI Machine Learning Repository.[2] These tasks have 35 and 6 attributes respectively, and both have 18 possible output classes. We generated boosted-tree models using XGBOOST [3], consisting of 50 trees per class, and encoded the resulting classifiers into MiniZinc. We then took the first 300 instances from each data-set set, and derived cardinality-minimal explanations for the observed class label using GEAS' core-guided optimization capabilities [5] to emulate the implicit hitting-set method of [6].

These instances contain only a single ARGMAX constraint, at the head of a very large system of ELEMENT and LINEAR constraints. Here, the ARGMAX propagation constitutes a smaller fraction of runtime – the model spends much more time in the score computations – but its *strength* is still important. On these instances, we see that the propagator and new decomposition perform similarly, both of which demonstrate advantage over the old decomposition.

7.4 Boolean arg-max

Given that the decomposition of Section 5 is domain consistent, it is interesting to consider how it compares to the specialised decomposition of [12] which is also domain consistent for the case of a Boolean array with at least one *true* value.

[2] `https://archive.ics.uci.edu/ml/`



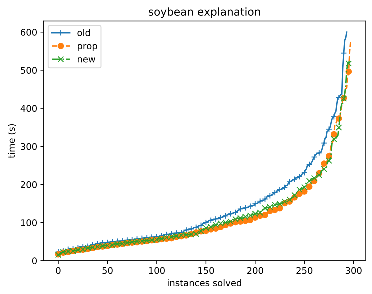


Fig. 5: Time to compute cardinality-minimal explanations for instances of the SOYBEAN and CHESS classification tasks.

n	B-DEC	B-PROP	NEW
25	**0.00**	**0.00**	**0.00**
50	**0.00**	**0.00**	**0.00**
100	**0.00**	**0.00**	0.01
200	0.01	**0.00**	0.01
400	0.01	**0.00**	0.01
800	0.01	**0.00**	0.03
1600	0.01	**0.01**	0.05
3200	0.03	**0.01**	0.10
6400	0.05	**0.02**	0.25

(a) Unit test from [12]

n	B-DEC	B-PROP	NEW
25	**0.00**	**0.00**	**0.00**
50	0.01	**0.00**	0.01
100	0.02	**0.01**	0.02
200	0.04	**0.02**	**0.02**
400	0.13	0.08	**0.04**
800	0.51	0.36	**0.09**
1600	18.69	**2.05**	7.53
3200	121.64	**13.05**	33.08
6400	-	**59.09**	127.88

(b) Alternate unit test.

Fig. 6: Performance of the new decomposition for the specialized Boolean case of ARGMAX.

Figure 6 compares the performance of our new decomposition with the decomposition (B-DEC) and propagator (B-PROP) presented in [12]. Figure 6(a) give results on the unit-test given in [12]. On this, the existing decomposition appears better. However, we note that all solvers prove this problem unsatisfiable without search – so overhead in setting up the constraints plays a major factor. (b) reports performance on the following constraint system:

```
array[1..n] of int: d1 = [ (23*i+2) mod n + 1 | i in 1..n];
array[1..n+1] of int: t1 = [ (37*i) mod n + 1 | i in 1..n+1];

array[1..n] of int: d2 = [ (17*i+3) mod n + 1 | i in 1..n];
array[1..n+1] of int: t2 = [ (31*i) mod n + 1 | i in 1..n+1];
  y = t1[ arg_max([ x = d1[i] | i in 1..n] ++ [true]) ];
  x = t2[ arg_max([ y = d2[i] | i in 1..n] ++ [true]) ];
```

This system is still unsatisfiable, but the unsatisfiability is not discovered at the root. In this case, the results are rather different. The new decomposition shows a $2-4\times$ speedup compared with the existing decomposition. However, the dedicated propagator still has a clear advantage on larger instances.

8 Conclusion

The ARGMAX constraint is a valuable if not well studied constraint. The Boolean version is important for `if-then-else-endif` constructs, and has recently been improved [12]. In this paper we define the domain consistent propagation possible for ARGMAX and then devise a propagator, and domain consistent decomposition for the integer case. We show that these markedly improve on the default decomposition. The decomposition is usually superior for integer problems in learning solvers which can take advantage of the reasoning about the introduced variable M. The propagator is superior in contexts where the new decomposition cannot be used (e.g. for floating point/rational domains), and presumably also for non-learning solvers which cannot take advantage of the richer language of learning of the decomposition.

References

1. Arrieta], A.B., Díaz-Rodríguez, N., Ser], J.D., Bennetot, A., Tabik, S., Barbado, A., Garcia, S., Gil-Lopez, S., Molina, D., Benjamins, R., Chatila, R., Herrera, F.: Explainable artificial intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. Information Fusion **58**, 82 – 115 (2020). https://doi.org/https://doi.org/10.1016/j.inffus.2019.12.012, `http://www.sciencedirect.com/science/article/pii/S1566253519308103`
2. Bartolini, A., Lombardi, M., Milano, M., Benini, L.: Neuron constraints to model complex real-world problems. In: Lee, J.H. (ed.) Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, September 12-16, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6876, pp. 115–129. Springer (2011). https://doi.org/10.1007/978-3-642-23786-7_11, `https://doi.org/10.1007/978-3-642-23786-7\_11`
3. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Krishnapuram, B., Shah, M., Smola, A.J., Aggarwal, C.C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016. pp. 785–794. ACM (2016). https://doi.org/10.1145/2939672.2939785, `https://doi.org/10.1145/2939672.2939785`
4. Explainable artificial intelligence: Darpa program. http://darpa.mil/program/explainable-artificial-intelligence (2020)
5. Gange, G., Berg, J., Demirović, E., Stuckey, P.J.: Core-guided and core-boosted search for constraint programming. In: Hebrard, E., Musliu, N. (eds.) Proceedings of Seventeenth International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming (CPAIOR2020). p. to appear. Springer (2020)
6. Ignatiev, A., Morgado, A., Marques-Silva, J.: Propositional abduction with implicit hitting sets. In: Kaminka, G.A., Fox, M., Bouquet, P., Hüllermeier, E., Dignum, V., Dignum, F., van Harmelen, F. (eds.) ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016). Frontiers in Artificial Intelligence and Applications, vol. 285, pp. 1327–1335. IOS Press (2016). https://doi.org/10.3233/978-1-61499-672-9-1327, `https://doi.org/10.3233/978-1-61499-672-9-1327`

7. Ignatiev, A., Narodytska, N., Marques-Silva, J.: Abduction-based explanations for machine learning models. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. pp. 1511–1519. AAAI Press (2019). https://doi.org/10.1609/aaai.v33i01.33011511, `https://doi.org/10.1609/aaai.v33i01.33011511`
8. Global constraint catalog: `max_index`. https://web.imt-atlantique.fr/x-info/sdemasse/gccat/Cmax_index.html (2020)
9. Nethercote, N., Stuckey, P., Becket, R., Br, S., , Duck, G., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Bessiere, C. (ed.) Proceedings of the 13th International Conference on Pr inciples and Practice of Constraint Programming. LNCS, vol. 4741, pp. 529–543. Springer-Verlag (2007)
10. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
11. Schulte, C., Tack, G.: Views and iterators for generic constraint implementations. In: van Beek, P. (ed.) Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3709, pp. 817–821. Springer (2005). https://doi.org/10.1007/11564751_71, `https://doi.org/10.1007/11564751\_71`
12. Stuckey, P.J., Tack, G.: Compiling conditional constraints. In: de Givry, S., Schiex, T. (eds.) Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming. pp. 384–400. No. 11802 in LNCS (2019)