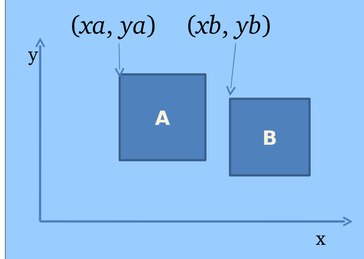
Constraints – What Are They?

A constraint is a relation among a set of variables. We consider only *finite-domain* variables – *ie* each variable has a finite set of values.

Consider the square packing problem in case study 3. Each square is represented with two variables for the position of a corner, and we have a non-overlap constraint:



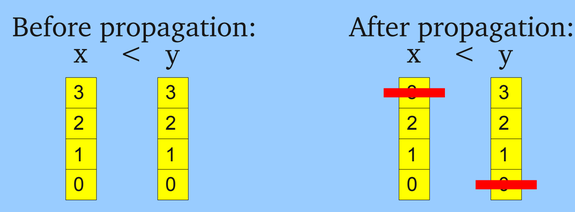
Constraint satisfied because:
$xa + Widtha \leq xb$

We are interested in *solving* constraint satisfaction problems. This involves finding an assignment to each variable such that all constraints are satisfied. This is typically done by:

- Search – trying out assignments and backtracking if they do not lead to a solution
- Propagation – reasoning on the constraints to remove values from variable domains, when they cannot take part in any solution

Propagation and Support

Constraint *propagation algorithms* filter values out of variable domains when the values cannot be part of a global solution. For example, consider the following less-than constraint:

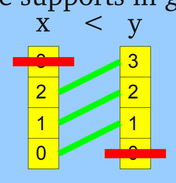


Because of the constraint, value 3 of x and value 0 of y cannot take part in any solution, so they are deleted.

Support

If a value is contained in a satisfying assignment for the constraint (*eg* x=2, y=3 for <) then it is not deleted, and we call the satisfying assignment a *support* for the value. This concept of support is pervasive in propagation algorithms.

Some supports in green:



Short Supports

The key concept for this paper is *short supports*.
Some constraints can be satisfied by assigning only a few of their variables – after the assignment, the constraint doesn't care about the values of the rest. A short assignment that satisfies the constraint is called a *short support*.

A conventional support will only support the values contained in it. A short support will support *all* values of any variable not mentioned in it. For example:

Domains $x1$:{1,..,11}, $x2$, $x3$:{1,..,10}
Constraint: ($x1 = x2$ OR $x1 = x3$)
Short support S: ($x1 \rightarrow 1$, $x2 \rightarrow 1$)

S supports $x1 \rightarrow 1$, $x2 \rightarrow 1$ and *all values* of $x3$

We use short supports to develop a new, much more efficient propagation algorithm called ShortGAC.

Explicit and Implicit Support

S supports $x1 \rightarrow 1$, $x2 \rightarrow 1$ *explicitly*
S supports all values of $x3$ *implicitly*

ShortGAC is designed to handle implicit support very efficiently.

University of St Andrews
School of Computer Science

Exploiting Short Supports for Generalised Arc Consistency for Arbitrary Constraints

Peter Nightingale, Ian P. Gent, Chris Jefferson and Ian Miguel

Case Study 1: BIBDs with Element

We applied the ShortGAC propagator to Element constraints in a set of quasigroup existence problems. Quasigroups are a combinatorial structure, and the problem is to find one with particular properties.

The following table reports the node rate of the constraint solver, where ShortGAC with Element is normalised to 1.

Propagator	Speed
Watched Element (state-of-the-art special-purpose propagator)	8.2
ShortGAC with Element instantiation	1
ShortGAC with List	0.57
Constructive Or	0.0042
GAC-Schema	0.0011
ShortGAC with full-length supports	0.0004

ShortGAC was not able to match the special-purpose propagator, but it does do extremely well against the other general-purpose methods GAC-Schema and Constructive Or.

Just Another Table Constraint?

There is already a lot of research on propagating table constraints efficiently. Is this just another table propagator?

No – ShortGAC is *much* more efficient than a generic table constraint when short supports are available.

The aim is to compete with special-purpose propagators, and other approaches such as Constructive Or.

For all three of the case studies, the constraint can be naturally represented as a disjunction of simpler constraints. Hence it is natural to compare ShortGAC with Constructive Or. In each case, ShortGAC is much faster.

Case Study 3: Square Packing

ShortGAC was applied to non-overlap constraints when packing squares into a rectangle.

Propagator	Speed
ShortGAC with Square Packing	1
GAC-Schema	0.11
ShortGAC with List	0.11
ShortGAC with full-length supports	0.035
Constructive Or	0.023

In this case we did not have a special-purpose propagator, and ShortGAC is clearly the fastest method.

Case Study 2: Lex-ordering

ShortGAC was applied to Lex-ordering constraints that arise in BIBD problems (a combinatorial design problem).

Propagator	Speed
GACLex (special-purpose propagator)	1.74
ShortGAC with Lex instantiation	1
Constructive Or	0.0045
GAC-Schema	0.0036
ShortGAC with full-length supports	0.0033

ShortGAC almost matched the special-purpose propagator, and is much faster than general-purpose methods GAC-Schema and Constructive Or.

The ShortGAC Algorithm

Consider, for example, the Element constraint
Element([x[0], x[1], x[2]], y, z)
With the variables x[0], x[1], x[2], $y \in$ {0,..,2}, $z \in$ {0,..,3}.
This constraint is satisfied iff $x[y]=z$, the value of the x variable indexed by y is equal to the value of z.

Suppose we have found one short support, A. The major data structures are as follows:

Supports:	A:	$x_0 \mapsto 1, y \mapsto 0, z \mapsto 1$			
supportListPerLit:	Variable				
Value	x_0	x_1	x_2	y	z
0	{}	{}	{}	$\{A\}$	{}
1	$\{A\}$	{}	{}	{}	$\{A\}$
2	{}	{}	{}	{}	{}
3	X	X	X	X	{}
supportsPerVar:	1	0	0	1	1
numSupports:	1				

supportListPerLit has a linked list of short supports for each variable and value.
supportsPerVar has a counter for each variable, indicating how many short supports mention the variable.
numSupports is the number of short supports currently known to the algorithm.

If supportsPerVar[w]<numSupports, then:

- there is a short support not containing w
- all values of w are implicitly supported
- variable w can be *completely ignored*

Variables x[1] and x[2] can be ignored in the current state.

There is no short support that supports $z \rightarrow 4$, so this value is deleted. When the algorithm has a full set of short supports (*ie* all remaining values are supported) the data structures look like this:

Supports:	A:	$x_0 \mapsto 1, y \mapsto 0, z \mapsto 1$			
	B:	$x_1 \mapsto 0, y \mapsto 1, z \mapsto 0$			
	C:	$x_0 \mapsto 2, y \mapsto 0, z \mapsto 2$			
	D:	$x_2 \mapsto 0, y \mapsto 2, z \mapsto 0$			
supportListPerLit:	Variable				
Value	x_0	x_1	x_2	y	z
0	{}	$\{B\}$	$\{D\}$	$\{A, C\}$	$\{B, D\}$
1	$\{A\}$	{}	{}	$\{B\}$	$\{A\}$
2	$\{C\}$	{}	{}	$\{D\}$	$\{C\}$
3	X	X	X	X	X
supportsPerVar:	2	1	1	4	4
numSupports:	4				

Acknowledgements

We would like to thank anonymous reviewers for their comments, and EPSRC for funding this work through grants EP/H004092/1 and EP/E030394/1.