Exploiting Short Supports for Generalised Arc Consistency for Arbitrary Constraints

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

Constraints, GAC

- Suppose we have finite-domain variables x_1, x_2, x_3 with domains x_1:{1,..,11}, x_2, x_3:{1,..,10}
- Constraint: ($x_1 = x_2$ OR $x_1 = x_3$)
- *Generalised Arc-Consistency* (GAC) requires that each value of each variable is contained in a satisfying tuple of the constraint
- To *establish* GAC: $x_1 \neq 11$

Support

- Suppose we have finite-domain variables x_1, x_2, x_3 with domains x_1:{1,..,11}, x_2, x_3:{1,..,10}
- Constraint: ($x_1 = x_2$ OR $x_1 = x_3$)
- Traditional definition of GAC *support*: a satisfying tuple of the constraint
- Value $x_1 \rightarrow 11$ has no support, and is deleted
- Value $x_1 \rightarrow 1$ is *not* deleted because it has support $\langle 1, 1, 3 \rangle$ (for example).

Short Support

- *The key idea of this paper:*
- A constraint may be satisfied by an assignment to a small subset of its variables
- We re-define support as *short support*
- Exploit these *short supports* to maintain GAC faster
  - The novel aspect is not really the idea of short support, but the algorithm

Short Support – Example

- Consider the running example again
- Domains x_1:{1,..,11}, x_2, x_3:{1,..,10}
- Constraint: ($x_1 = x_2$ OR $x_1 = x_3$)
- Short support: ($x_1 \rightarrow 1$, $x_2 \rightarrow 1$)
- Any *extension* of this short support to cover x_3 is a full-length support
  - Assuming we always use values in the domain
- Supports $x_1 \rightarrow 1$, $x_2 \rightarrow 1$, and all values of x_3

Short Support – Explicit and Implicit

- Consider the running example again
- Domains x_1:{1,..,11}, x_2, x_3:{1,..,10}
- Constraint: ($x_1 = x_2$ OR $x_1 = x_3$)
- Short support: ($x_1 \rightarrow 1$, $x_2 \rightarrow 1$)
  - *Explicitly* supports $x_1 \rightarrow 1$, $x_2 \rightarrow 1$
  - *Implicitly* supports all values of x_3

SHORTGAC

- A new GAC algorithm
- *Designed around implicit support:*
  - Never spends any time finding new supports for a variable that is implicitly supported
- *Exploits watched literals:*
  - The algorithm is not invoked unless a short support has been lost
  - O(o) for other value deletions
  - Can simulate watched-literal unit propagation (not very quickly)

ShortGAC

- Like GAC-Schema, can be instantiated in different ways
- Generic:
  - *List* – a list of short supports is given
- Constraint specific:
  - Lex-ordering
  - Element
  - Square Packing

Just Another Table Constraint?

- **¡No! (translation: No!)**
  - SHORTGAC competes with:
    - Special-purpose propagators such as Element, Lex
    - Constructive Or
  - Aim to be orders of magnitude faster than table constraints
    - When short supports are available

The Natural Habitat of a Short Support

- Short supports are *disjunctive*
  - Each literal may be supported by S_1 or S_2 or ….
- They arise typically from a disjunction
  - The running example: ($x_1 = x_2$ or $x_1 = x_3$)
  - 2d short supports of size 2
- Therefore we compare ShortGAC to *Constructive Or*
  - A careful incremental implementation similar to Würtz and Müller

Case Study 1: Element

- Element constraint $X[y]=z$ for a vector X, index variable y and result variable z
- $(y=0$ and $X[0]=z)$ or $(y=1$ and $X[1]=z)$ or …
- d^2 short supports of size 3

Case Study 1: Element

- SHORTGAC closely mimics the state-of-the-art *Watched Element* propagator
  - Because of watched literals, SHORTGAC is called only when Watched Element is called
  - Same short supports as Watched Element uses implicitly
  - However, SHORTGAC does more work when called
    - Does not go straight to value that lost explicit support – see future work

Case Study 1: Element

- Order by speed:
  - Watched Element 8.2
  - SHORTGAC with Element 1 (normalised)
  - SHORTGAC with List 0.57
  - Constructive Or 0.0042
  - GAC-Schema 0.0011
  - SHORTGAC with full-length 0.0004
- *Huge separation between short supports and the rest*
- *All methods explore the same search tree*

Case Study 2: Lex

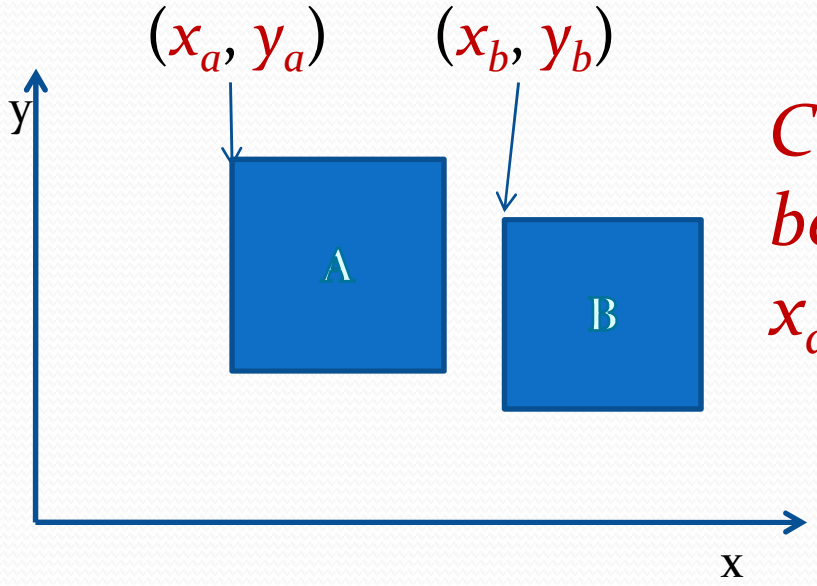
- Lex-ordering constraint between two vectors of variables X and Y
- Ensures that X comes before Y in dictionary order
  - aardvark<apple
- $X[1]<Y[1]$ or ($X[1]=Y[1]$ and $X[2]<Y[2]$) or …

Case Study 2: Lex

- Order by speed:
  - GACLex propagator 1.74
  - ShortGAC with Lex 1
  - Constructive Or 0.0045
  - GAC-Schema 0.0036
  - ShortGAC with full-length 0.0033
- *Again, huge separation between short supports / special-purpose propagator and the rest*
- ShortGAC *very close to special-purpose state-of-the-art propagator*

Case Study 3: Square Packing

- Squares to be packed into rectangle
- Squares represented by coordinates of corner
- Non-overlap (arity 4) constraint: does not overlap in *x*- or *y*-dimension



Constraint satisfied because:

$x_a + Width_a \leq x_b$

Case Study 3: Square Packing

- Order by speed:
  - SHORTGAC with SquarePacking 1
  - GAC-Schema 0.11
  - SHORTGAC with List 0.11
  - SHORTGAC with full-length 0.035
  - Constructive Or 0.023
- *Smaller separation between short supports and the rest*
  - Possibly because constraint only arity 4
  - Implicit support covers at most 2 variables

Future (Current) Work

- SHORTGAC poor against GAC-Schema on full-length supports
  - SHORTGAC very poor at finding a literal that lost its *last explicit support*
  - We have improved this in a new algorithm
- SHORTGAC backtracks all its data structures
  - In many cases short supports are *backtrack-stable*
  - This is exploited in a second new algorithm

Conclusions

- SHORTGAC, a new generic GAC algorithm
- Exploits idea of *short support*, implicit in some special-purpose propagators
- Faster than Constructive Or, GAC-Schema in our experiments, approaches special-purpose propagators