Globalizing Constraint Models
Using Model Structure to discover Global Patterns

Kevin Leo, Christopher Mears, Guido Tack, Maria Garcia de la Banda

September 20, 2013
Model Improvement
Progressive Party: MiniZinc Example
Model Improvement
Progressive Party: MiniZinc Example

Goal: Organize several parties for boat crews at a yacht club
Model Improvement
Progressive Party: MiniZinc Example

Goal: Organize several parties for boat crews at a yacht club

\textbf{GuestCrews}: Set of guest crews \quad \textbf{crew}: Number of members

\textbf{HostBoats}: Set of party hosts \quad \textbf{capacity}: Space on host

\textbf{Time}: Set of time periods
Model Improvement
Progressive Party: MiniZinc Example

\((Guest\text{Crews}, \text{Time}) \rightarrow \text{HostBoats}\)
array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;
Model Improvement
Progressive Party: MiniZinc Example

array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint forall (h in HostBoats, g in GuestCrews, t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint forall (h in HostBoats, g in GuestCrews, t in Time)
    visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (h in HostBoats)
    forall (g in GuestCrews)
        sum (t in Time) (visits[g,h,t]) <= 1;
Model Improvement
Progressive Party: MiniZinc Example

array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint forall (h in HostBoats, g in GuestCrews, t in Time)
   visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (h in HostBoats)
   forall (g in GuestCrews)
      sum (t in Time) (visits[g,h,t]) <= 1
/
   forall (t in Time)
      sum (g in GuestCrews)
         (crew[g]*visits[g,h,t]) <= capacity[h];
array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint forall (h in HostBoats, g in GuestCrews, t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (h in HostBoats)
forall (g in GuestCrews)

sum (t in Time) (visits[g,h,t]) <= 1

/\ forall (t in Time)

sum (g in GuestCrews)

(crew[g]*visits[g,h,t]) <= capacity[h];
Model Improvement
Progressive Party: MiniZinc Example

array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint forall (h in HostBoats, g in GuestCrews, t in Time)
  visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (g in GuestCrews)
  alldifferent([hostedBy[g,t] | t in Time]);

constraint forall (h in HostBoats)
  forall (t in Time)
    sum (g in GuestCrews)
      (crew[g]*visits[g,h,t]) <= capacity[h];
array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint forall (h in HostBoats, g in GuestCrews, t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (g in GuestCrews)
alldifferent([[hostedBy[g,t] | t in Time]]);

constraint forall (h in HostBoats)
forall (t in Time)
sum (g in GuestCrews)
(crew[g]*visits[g,h,t]) <= capacity[h];
Model Improvement
Progressive Party: MiniZinc Example

array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint forall (h in HostBoats, g in GuestCrews, t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (g in GuestCrews)
alldifferent([hostedBy[g,t] | t in Time]);

constraint forall (t in Time)
bin_packing(capacity,
[hostedBy[g,t] | g in GuestCrews],
crew);
Model Improvement
Progressive Party: MiniZinc Example

array [GuestCrews, Time] of var HostBoats : hostedBy;

constraint forall (g in GuestCrews)
   alldifferent([hostedBy[g,t] | t in Time]);

constraint forall (t in Time)
   bin_packing(capacity,
               [hostedBy[g,t] | g in GuestCrews],
               crew);
Model Improvement
Progressive Party: MiniZinc Example

```mini
array [GuestCrews, Time] of var HostBoats : hostedBy;

constraint forall (g in GuestCrews)
    alldifferent([hostedBy[g,t] | t in Time]);

constraint forall (t in Time)
    bin_packing(capacity,
        [hostedBy[g,t] | t in GuestCrews],
        crew);
```

<table>
<thead>
<tr>
<th>Original</th>
<th>Reformulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Vars:</td>
<td>#Vars:</td>
</tr>
<tr>
<td>GuestCrews</td>
<td>HostBoats</td>
</tr>
<tr>
<td>GuestCrews</td>
<td>HostBoats</td>
</tr>
<tr>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>GuestCrews</td>
<td>HostBoats</td>
</tr>
<tr>
<td></td>
<td>Time</td>
</tr>
<tr>
<td>HostBoats</td>
<td>GuestCrews</td>
</tr>
<tr>
<td>HostBoats</td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td>Time</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>#Cons:</td>
<td>#Cons:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Outline

1. Motivation
2. Related Work
3. The Globalizer
4. Future Direction
Motivation

Global Constraints
Motivation

*Global Constraints*

- Dedicated Algorithms
Motivation

Global Constraints

- Dedicated Algorithms
- Good Decompositions (MiniZinc/Conjure)
Motivation

Global Constraints
- Dedicated Algorithms
- Good Decompositions (MiniZinc/Conjure)
- Hidden Structure
Motivation

Global Constraints
- Dedicated Algorithms
- Good Decompositions (MiniZinc/Conjure)
- Hidden Structure

Modelling Patterns
- Encodings
  - Integers
Motivation

Global Constraints

- Dedicated Algorithms
- Good Decompositions (MiniZinc/Conjure)
- Hidden Structure

Modelling Patterns

- Encodings
  - Integers
  - Sets
  - Paths
Motivation

Global Constraints
- Dedicated Algorithms
- Good Decompositions (MiniZinc/Conjure)
- Hidden Structure

Modelling Patterns
- Encodings
  - Integers
  - Sets
  - Paths
- Common formulations
  - Cliques of not equal
Related Work
Related Work
Related Work

- Sample the solution space of problem
Related Work

- Sample the solution space of problem
- Candidate constraints must accept positive samples
Related Work

- Sample the solution space of problem
- Candidate constraints must accept positive samples
- Hard to generate samples for models
Related Work

- Sample the solution space of problem
- Candidate constraints must accept positive samples
- Hard to generate samples for models
- Sample solutions to sub-problems

Samples

GLOBALIZER
Sample solutions to sub-problems
Related Work

Sub-Problems

Sample solutions to sub-problems

GLOBALIZER
Related Work

- Relaxations of the problem

Sub-Problems

Sample solutions to sub-problems

GLOBALIZER
Related Work

- Relaxations of the problem
- Sub-problems often hand picked
Related Work

- Relaxations of the problem
- Sub-problems often hand picked
- Use syntactic structure of model to find sub-problems
Related Work

Generalizing to Models

- Use Syntactic Structure of Model to find **Sub-Problems**
- **Sample** solutions to sub-problems

GLOBALIZER
Related Work

**Generalizing to Models**

- Constraints discovered must be valid for instances of any size

**Sample solutions to sub-problems**

Use Syntactic Structure of Model to find Sub-Problems

---

GLOBALIZER
Related Work

**Generalizing to Models**

- Constraints discovered must be valid for instances of any size
- Implied constraints often found for individual instances

Use Syntactic Structure of Model to find **Sub-Problems**

**Sample** solutions to sub-problems

GLOBALIZER
Related Work

**Generalizing to Models**

- Constraints discovered must be valid for instances of any size
- Implied constraints often found for individual instances
- We aim to find parametric constraints

**Use Syntactic Structure of Model to find Sub-Problems**

*Sample* solutions to sub-problems

**GLOBALIZER**
Related Work

Find **Parametric** Constraints

- Use Syntactic Structure of Model to find **Sub-Problems**
- **Sample** solutions to sub-problems

GLOBALIZER

**Argument Construction**
Related Work

- Find valid arguments to a constraint

GLOBALIZER

Find **Parametric** Constraints

**Argument Construction**
Related Work

- Find valid arguments to a constraint
- Concentrate on single constraint

Find **Parametric** Constraints

**Argument Construction**

GLOBALIZER
Related Work

- Find valid arguments to a constraint
- Concentrate on single constraint
- Brute-force argument generation
Related Work

- Find valid arguments to a constraint
- Concentrate on single constraint
- Brute-force argument generation
- Use model structure and annotated constraint library to construct arguments
Related Work

- Use model structure and annotated constraint library to construct arguments

```python
bin_packing(capacity,
    [hostedBy[g,t] | g in GuestCrews],
    crew);
```
Related Work

Find **Parametric** Constraints

- Use Syntactic Structure of Model to find **Sub-Problems**
- **Sample** solutions to sub-problems
- Use Model Structure and Annotated Constraint Library to Construct **Arguments**

**GLOBALIZER**

**Proof**
Related Work

Find **Parametric**
Constraints

- Proof for implied constraints admirable

Use Model Structure
and Annotated
Constraint Library to
Construct **Arguments**

**Proof**

**GLOBALIZER**
Related Work

- Proof for implied constraints admirable
- Only possible for simple models

Use Model Structure and Annotated Constraint Library to Construct Arguments

Proof

GLOBALIZER
Related Work

- Proof for implied constraints admirable
- Only possible for simple models
- Very tough to prove equivalence

Find **Parametric** Constraints

Use Model Structure and Annotated Constraint Library to Construct **Arguments**

**Proof**

**GLOBALIZER**
Related Work

- Proof for implied constraints admirable
- Only possible for simple models
- Very tough to prove equivalence
- Even harder without a model

Use Model Structure and Annotated Constraint Library to Construct Arguments

Find Parametric Constraints
Related Work

Find **Parametric** Constraints

- Proof for implied constraints admirable
- Only possible for simple models
- Very tough to prove equivalence
- Even harder without a model
- Use sub-problems to improve **confidence** of suggestions

Use Model Structure and Annotated Constraint Library to Construct **Arguments**

**Proof**
Related Work

- Proof for implied constraints admirable
- Only possible for simple models
- Very tough to prove equivalence
- Even harder without a model
- Use sub-problems to improve confidence of suggestions
- Users can judge equivalence well
Related Work

- Proof for implied constraints admirable
- Only possible for simple models
- Very tough to prove equivalence
- Even harder without a model
- Use sub-problems to improve confidence of suggestions
- Users can judge equivalence well
- Not the only tool to sacrifice proof
Related Work

Model Seeker

Find Parametric Constraints

Use Model Structure and Annotated Constraint Library to Construct Arguments

Proof

GLOBALIZER
Related Work

**Model Seeker**
- Takes a sample of problem solutions

**GLOBALIZER**
- Find *Parametric* Constraints
- Use Model Structure and Annotated Constraint Library to Construct *Arguments*

**Proof**
Related Work

**Model Seeker**
- Takes a sample of problem solutions
- Partitions samples into collections

Find **Parametric** Constraints

Use Model Structure and Annotated Constraint Library to Construct **Arguments**

**Proof**

**GLOBALIZER**
Related Work

Model Seeker
- Takes a sample of problem solutions
- Partitions samples into collections
- Arguments to Constraint Seeker
- Receives ranked globals from GCC

Find Parameteric Constraints

Use Model Structure and Annotated Constraint Library to Construct Arguments

Proof

GLOBALIZER
Related Work

Model Seeker
- Takes a sample of problem solutions
- Partitions samples into collections
- Arguments to Constraint Seeker
  - Receives ranked globals from GCC
  - Suggests models valid for samples

Use Model Structure and Annotated Constraint Library to Construct Arguments

Find Parametric
Constraints

Proof

GLOBALIZER
Related Work

- Use Syntactic Structure of Model to find **Sub-Problems**
- **Sample** solutions to sub-problems

Find **Parametric** Constraints

- Use Model Structure and Annotated Constraint Library to Construct **Arguments**
- **Proof** Isn't Necessary: Use Sub-problems to Improve **Confidence**

GLOBALIZER
Globalizer: Overview

Model

Data sets

normalisation

sub-problem group generation

sub-problem group

Constraint library

candidate generation

ranking / filtering

Web interface
Globalizer: Overview

Model

Data sets

normalisation

sub-problem group generation

Constraint library

candidate generation

ranking / filtering

sub-problem group
Globalizer: Overview
Globalizer: Overview

Model

Data sets

normalisation

sub-problem group generation

Constraint library

sub-problem group

candidate generation

ranking / filtering

Web interface
Globalizer: Overview

- Model
- Data sets
- Constraint library
- Web interface

1. Normalisation
2. Sub-problem group generation
3. Candidate generation
4. Ranking / filtering

Sub-problem group
Normalisation

\[
\text{constraint } \forall (h \in \text{HostBoats}) \\
\quad \forall (g \in \text{GuestCrews}) \\
\quad \sum (t \in \text{Time}) \\
\quad \quad (\text{visits}[g,h,t]) \leq 1 \\
\land \quad \forall (t \in \text{Time}) \\
\quad \sum (g \in \text{GuestCrews}) \\
\quad \quad (\text{crew}[g] \times \text{visits}[g,h,t]) \leq \text{capacity}[h];
\]
Normalisation

\[
\text{constraint } \forall (h \in \text{HostBoats}) \\forall (g \in \text{GuestCrews}) \\
\quad \sum (t \in \text{Time}) (\text{visits}[g,h,t]) \leq 1; \\
\text{constraint } \forall (h \in \text{HostBoats}) \\forall (t \in \text{Time}) \\
\quad \sum (g \in \text{GuestCrews}) (\text{crew}[g] \times \text{visits}[g,h,t]) \leq \text{capacity}[h];
\]
Globalizer: Overview
Globalizer: Overview
Sub-Problem Generation

Powerset of constraint items

\[ a < b, \ b < c, \ c < d \]

\[
\{a < b\}, \ \{a < b, b < c\}, \ \{a < b, b < c, c < d\}, \\
\{a < b, c < d\}, \ \{b < c\}, \ \{b < c, c < d\}, \ \{c < d\}
\]
Sub-Problem Generation

Powerset of constraint items

\[ a < b, \ b < c, \ c < d \]

\{a<b\}, \ \{a<b, b<c\}, \ \{a<b, b<c, c<d\}, \ \{a<b, c<d\}, \ \{a<b,c<d\}, \ \{b<c\}, \ \{b<c, c<d\}, \ \{c<d\}
Sub-Problem Generation

Powerset of connected constraint items

\[ a < b, \; b < c, \; c < d \]

\{a<b\}, \; \{a<b, b<c\}, \; \{a<b, b<c, c<d\}, \]

\{a<b, c<d\}, \; \{b<c\}, \; \{b<c, c<d\}, \; \{c<d\}
Globalizer: Overview

Model

Data sets

Web interface

normalisation

sub-problem group generation

candidate generation

ranking / filtering

Constraint library
Sub-Problem Generation
Groups

\begin{verbatim}
constraint forall (g in GuestCrews,
               h in HostBoats,
               t in Time)
    visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (g in GuestCrews)
    forall (h in HostBoats)
    sum (t in Time) (visits[g,h,t]) <= 1;
\end{verbatim}
Sub-Problem Generation

Groups

```
constraint forall (g in GuestCrews,
    h in HostBoats,
    t in Time)
    visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (g in GuestCrews)
forall (h in HostBoats)
sum (t in Time) (visits[g,h,t]) <= 1;
```
Sub-Problem Generation

Groups

\[
\text{constraint } \forall (g \in \text{GuestCrews}, h \in \text{HostBoats}, t \in \text{Time}) \quad \text{visits}[g,h,t] = 1 \iff \text{hostedBy}[g,t] = h;
\]

\[
\text{constraint } \forall (g \in \text{GuestCrews}) \quad \forall (h \in \text{HostBoats}) \quad \sum_{t \in \text{Time}} (\text{visits}[g,h,t]) \leq 1;
\]
int: \ g = \text{min}(\text{GuestCrews});

\textbf{constraint} \ \text{forall} \ (\ h \ \text{in} \ \text{HostBoats}, \ t \ \text{in} \ \text{Time})

\text{visits}[g,h,t] = 1 \iff \text{hostedBy}[g,t] = h;

\textbf{constraint} \ \text{forall} \ (h \ \text{in} \ \text{HostBoats})

\text{sum} \ (t \ \text{in} \ \text{Time}) (\text{visits}[g,h,t]) \leq 1;
Sub-Problem Generation
Groups

int: g = min(GuestCrews);

int: g = max(GuestCrews);

constraint forall (h in HostBoats, t in Time)
    visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (h in HostBoats)
    sum (t in Time) (visits[g,h,t]) <= 1;
Sub-Problem Generation

Groups

int: g = min(GuestCrews);
constraint forall (g in GuestCrews, h in HostBoats, t in Time)
   visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

int: g = max(GuestCrews);
constraint forall (h in HostBoats, t in Time)
   visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (g in GuestCrews, h in HostBoats, t in Time)
   sum (t in Time) (visits[g,h,t]) <= 1;

constraint forall (h in HostBoats, t in Time)
   sum (g in GuestCrews) (visits[g,h,t]) <= 1;
constraint forall (g in GuestCrews,
    h in HostBoats,
    t in Time)
  visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (g in GuestCrews)
  forall (h in HostBoats)
  sum (t in Time) (visits[g,h,t]) <= 1;
**Sub-Problem Generation**

**Groups**

```plaintext
int: h = min(HostBoats);

constraint forall (g in GuestCrews, t in Time)
    visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (g in GuestCrews)
    sum (t in Time) (visits[g,h,t]) <= 1;
```
Sub-Problem Generation

Groups

\[
\text{int: } h = \min(\text{HostBoats});
\]

\[
\text{int: } h = \max(\text{HostBoats});
\]

\[
\text{constraint } \text{forall} (g \text{ in } \text{GuestCrews}, t \text{ in } \text{Time})
\]
\[
\text{visits}[g,h,t] = 1 \leftrightarrow \text{hostedBy}[g,t] = h;
\]

\[
\text{constraint } \text{forall} (g \text{ in } \text{GuestCrews})
\]
\[
\text{sum} (t \text{ in } \text{Time}) (\text{visits}[g,h,t]) \leq 1;
\]
Sub-Problem Generation
Groups

\[
\text{constraint} \ \forall (g \in \text{GuestCrews}, \ h \in \text{HostBoats}, \ t \in \text{Time}) \quad \text{visits}[g,h,t] = 1 \iff \text{hostedBy}[g,t] = h;
\]

\[
\text{constraint} \ \forall (g \in \text{GuestCrews}) \quad \sum (t \in \text{Time}) (\text{visits}[g,h,t]) \leq 1;
\]

\[
\text{int: } g = \min(\text{GuestCrews});
\]

\[
\text{constraint} \ \forall (h \in \text{HostBoats}, \ t \in \text{Time}) \quad \text{visits}[g,h,t] = 1 \iff \text{hostedBy}[g,t] = h;
\]

\[
\text{constraint} \ \forall (h \in \text{HostBoats}) \quad \sum (t \in \text{Time}) (\text{visits}[g,h,t]) \leq 1;
\]

\[
\text{int: } h = \min(\text{HostBoats});
\]

\[
\text{constraint} \ \forall (g \in \text{GuestCrews}, \ h \in \text{HostBoats}, \ t \in \text{Time}) \quad \text{visits}[g,h,t] = 1 \iff \text{hostedBy}[g,t] = h;
\]

\[
\text{constraint} \ \forall (g \in \text{GuestCrews}) \quad \sum (t \in \text{Time}) (\text{visits}[g,h,t]) \leq 1;
\]

\[
\text{int: } g = \max(\text{GuestCrews});
\]

\[
\text{constraint} \ \forall (h \in \text{HostBoats}, \ t \in \text{Time}) \quad \text{visits}[g,h,t] = 1 \iff \text{hostedBy}[g,t] = h;
\]

\[
\text{constraint} \ \forall (h \in \text{HostBoats}) \quad \sum (t \in \text{Time}) (\text{visits}[g,h,t]) \leq 1;
\]

\[
\text{int: } h = \max(\text{HostBoats});
\]
Sub-Problem Generation
Groups

\[
\text{constraint } \forall (g \in \text{GuestCrews, h in HostBoats, t in Time}) \\
\quad \text{visits}[g,h,t] = 1 \iff \text{hostedBy}[g,t] = h;
\]

\[
\text{constraint } \forall (g \in \text{GuestCrews}) \\
\quad \forall (h \in \text{HostBoats}) \\
\quad \sum (t \in \text{Time}) (\text{visits}[g,h,t]) \leq 1;
\]
Sub-Problem Generation
Groups

\[
\text{int: } g = \min(GuestCrews); \\
\text{int: } h = \min(HostBoats); \\
\text{constraint forall (} \\
\text{t in Time} \\
\text{visits}[g,h,t] = 1 \leftrightarrow \text{hostedBy}[g,t] = h; \\
\text{constraint} \\
\text{sum (} t \text{ in Time) (visits}[g,h,t]) \leq 1;
\]
Sub-Problem Generation
Groups

```
int: g = min(GuestCrews);
int: h = min(HostBoats);
int: g = min(GuestCrews);
int: h = max(HostBoats);
constraint forall (t in Time)
    visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint
    sum (t in Time) (visits[g,h,t]) <= 1;
```
Sub-Problem Generation
Groups

\[
\text{int: } g = \min(GuestCrews); \\
\text{int: } h = \min(HostBoats); \\
\text{int: } g = \min(GuestCrews); \\
\text{int: } h = \max(HostBoats); \\
\text{int: } g = \max(GuestCrews); \\
\text{int: } h = \max(HostBoats); \\
\text{constraint} \quad \forall (t \in \text{Time}) \\
\quad \text{visits}[g,h,t] = 1 \leftrightarrow \text{hostedBy}[g,t] = h; \\
\text{constraint} \\
\quad \sum (t \in \text{Time}) (\text{visits}[g,h,t]) \leq 1;
\]
Sub-Problem Generation
Groups

int: g = min(GuestCrews);
int: h = min(HostBoats);

int: g = min(GuestCrews);
int: h = max(HostBoats);

int: g = max(GuestCrews);
int: h = max(HostBoats);

int: g = max(GuestCrews);
int: h = min(HostBoats);
constraint forall (t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint
Sub-Problem Generation

Groups

\[
\text{int: } g = \min(\text{GuestCrews});
\]
\[
\text{int: } h = \min(\text{HostBoats});
\]
\[
\text{constraint forall (g in GuestCrews,}
\text{h in HostBoats,}
\text{t in Time)}
\]
\[
\text{visits}[g,h,t] = 1 \leftrightarrow\text{hostedBy}[g,t] = h;
\]
\[
\text{constraint forall (g in GuestCrews)}
\]
\[
\text{forall (h in HostBoats)}
\]
\[
\text{sum (t in Time) (visits[g,h,t])} \leq 1;
\]

\[
\text{int: } g = \max(\text{GuestCrews});
\]
\[
\text{int: } h = \max(\text{HostBoats});
\]
\[
\text{constraint forall (g in GuestCrews,}
\text{t in Time)}
\]
\[
\text{visits}[g,h,t] = 1 \leftrightarrow\text{hostedBy}[g,t] = h;
\]
\[
\text{constraint forall (h in HostBoats)}
\]
\[
\text{forall (g in GuestCrews)}
\]
\[
\text{sum (t in Time) (visits[g,h,t])} \leq 1;
\]

\[
\text{int: } g = \min(\text{GuestCrews});
\]
\[
\text{int: } h = \min(\text{HostBoats});
\]
\[
\text{constraint forall (g in GuestCrews,}
\text{t in Time)}
\]
\[
\text{visits}[g,h,t] = 1 \leftrightarrow\text{hostedBy}[g,t] = h;
\]
\[
\text{constraint forall (h in HostBoats)}
\]
\[
\text{forall (g in GuestCrews)}
\]
\[
\text{sum (t in Time) (visits[g,h,t])} \leq 1;
\]

\[
\text{int: } g = \max(\text{GuestCrews});
\]
\[
\text{int: } h = \max(\text{HostBoats});
\]
\[
\text{constraint forall (g in GuestCrews,}
\text{t in Time)}
\]
\[
\text{visits}[g,h,t] = 1 \leftrightarrow\text{hostedBy}[g,t] = h;
\]
\[
\text{constraint forall (h in HostBoats)}
\]
\[
\text{forall (g in GuestCrews)}
\]
\[
\text{sum (t in Time) (visits[g,h,t])} \leq 1;
\]
Globalizer: Overview

Model

Data sets

normalisation

sub-problem group generation

sub-problem group

candidate generation

ranking / filtering

Constraint library
Globalizer: Overview
int: g = min(GuestCrews);

c constraint forall (h in HostBoats, t in Time)
    visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

c constraint forall (h in HostBoats)
    sum (t in Time) (visits[g,h,t]) <= 1;
Candidate Generation

**Individual Sub-Problem**

```plaintext
int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
    visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats)
    sum (t in Time) (visits[g,h,t]) <= 1;
```

```
int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
    visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats)
    sum (t in Time) (visits[g,h,t]) <= 1;
```
Candidate Generation

Individual Sub-Problem

int: g = min(GuestCrews);

constraint forall (h in HostBoats, t in Time)
    visits[g, h, t] = 1 <-> hostedBy[g, t] = h;

constraint forall (h in HostBoats)
    sum (t in Time) (visits[g, h, t]) <= 1;

Possible Arguments
Candidate Generation

\[ g = \min(\text{GuestCrews}) \]

**Constraint**

\[
\forall (h \in \text{HostBoats}, t \in \text{Time}) \quad \text{visits}[g, h, t] = 1 \leftrightarrow \text{hostedBy}[g, t] = h;
\]

\[
\forall (h \in \text{HostBoats}) \quad \sum_{t \in \text{Time}} (\text{visits}[g, h, t]) \leq 1;
\]

**Possible Arguments**

- `int g = min(GuestCrews);`
- `constraint forall (h in HostBoats) (visits[g, h, t]) <= 1;`
- `constraint forall (h in HostBoats, t in Time) (visits[g, h, t]) = 1 <-> hostedBy[g, t] = h;`
- `int nh; int ng; set of int HostBoats = 1..nh;
set of int GuestCrews = 1..ng;
set of int Time = 1..p;
array [GuestCrews, HostBoats, Time] of var HostBoats: hostedBy;
array [GuestCrews, Time] of var 0..1: visits;`
Candidate Generation

Individual Sub-Problem

int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats)
sum (t in Time) (visits[g,h,t]) <= 1;

Constraint Library

Possible Arguments

alldifferent alldifferent_except_0 all_equal atleast atmost bin_packing bin_packing_capa bin_packing_load channel circuit count cumulative decreasing diffn distribute element exactly gcc global_cardinality increasing inverse lex_less lex_lesseq lex2 maximum minimum member nvalue sliding_sum sort strict_lex2 subcircuit unary value_precede
Candidate Generation

**Individual Sub-Problem**

```plaintext
int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
    visits[g, h, t] = 1 <-> hostedBy[g, t] = h;
constraint forall (h in HostBoats)
    sum (t in Time) (visits[g, h, t]) <= 1;
```

**Constraint Library**

```
alldifferent  alldifferent_except_0  all_equal  atleast
atmost  bin_packing  bin_packing_capa  bin_packing_load
channel  circuit  count  cumulative  decreasing  diffn
distribute  element  exactly  gcc  global_cardinality
increasing  inverse  lex_less  lex_leq  lex2  maximum
minimum  member  nvalue  sliding_sum  sort  strict_lex2
subcircuit  unary  value_precede
```
Candidate Generation

```
int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
   visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats)
   sum (t in Time) (visits[g,h,t]) <= 1;
```

Individual Sub-Problem

```
int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
   visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats)
   sum (t in Time) (visits[g,h,t]) <= 1;
```

Constraint Library

```
int: p; int: nh; int: ng;
set of int: HostBoats = 1..nh;
set of int: GuestCrews = 1..ng;
set of int: Time = 1..p;
array [GuestCrews, Time] of var HostBoats:
   hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1:
   visits;
```

Predicate Signature

```
predicate all_different(array[int] of var int: x)
```
Candidate Generation

Individual Sub-Problem

Group

Constraint Library

Possible Arguments

Constraint Signature

Generated Constraints

Constraint Library

Possible Arguments

Constraint Signature

Generated Constraints

int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats)
sum (t in Time) (visits[g,h,t]) <= 1;

int: p; int: nh; int: ng;
set of int: HostBoats = 1..nh;
set of int: GuestCrews = 1..ng;
set of int: Time = 1..p;
array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint all_differerent(hostedBy_used);
constraint all_differerent(visits_used);
...
Candidate Generation

**Group**

```
int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats)
sum (t in Time) (visits[g,h,t]) <= 1;
```

**Individual Sub-Problem**

```
int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats)
sum (t in Time) (visits[g,h,t]) <= 1;
```

**Samples**

```
hostedBy1 = [4, 2, 3, 5, 4, 3, 5, ...];
visits1 = [0, 0, 0, 0, 0, 1, 0, ...];
hostedBy2 = [4, 5, 3, 2, 4, 3, 5, ...];
visits2 = [0, 0, 0, 0, 0, 0, 0, ...];
hostedBy3 = [4, 5, 3, 1, 3, 1, 5, ...];
visits3 = [0, 0, 0, 1, 0, 0, 0, ...];
```

**Generated Constraints**

```
constraint all_different(hostedBy_used);
```

**Possible Arguments**

```
int: p; int: nh; int: ng;
set of int: HostBoats = 1..nh;
set of int: GuestCrews = 1..ng;
set of int: Time = 1..p;
array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;
```

**Generated Constraints**

```
constraint all_different(hostedBy_used);
```

**Constraint Signature**

```
predicate all_different(array[int] of var int: x)
```
Candidate Generation

Individual Sub-Problem

Group

Constraint Library

Possible Arguments

Constraint Signature

Generated Constraints

Constraint accepts Samples?

Samples

Constraints

Constraint Signature

Predicate all_different(array[int] of var int: x)

int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time)
visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats)
sum (t in Time) (visits[g,h,t]) <= 1;

hostedBy1 = [4, 2, 3, 5, 4, 3, 5,...];
visits1 = [0, 0, 0, 0, 0, 1, 0,...];
hostedBy2 = [4, 5, 3, 2, 4, 3, 5,...];
visits2 = [0, 0, 0, 0, 0, 0, 0,...];
hostedBy3 = [4, 5, 3, 1, 3, 1, 5,...];
visits3 = [0, 0, 0, 1, 0, 0, 0,...];

int: p; int: nh; int: ng;
set of int: HostBoats = 1..nh;
set of int: GuestCrews = 1..ng;
set of int: Time = 1..p;
array [GuestCrews, Time] of var HostBoats : hostedBy;
array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint all_different(hostedBy_used)
Candidate Generation

```
constraint forall (h in HostBoats, t in Time) 
  visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats) 
  sum (t in Time) (visits[g,h,t]) <= 1;
```

```
int: g = min(GuestCrews);
constraint forall (h in HostBoats, t in Time) 
  visits[g,h,t] = 1 <-> hostedBy[g,t] = h;
constraint forall (h in HostBoats) 
  sum (t in Time) (visits[g,h,t]) <= 1;
```

```
constraint all_different(hostedBy_used);
constraint all_different_except_zero(hostedBy_used);
...'
```
Globalizer: Overview

Model

Data sets

normalisation

sub-problem group generation

Constraint library

sub-problem group

candidate generation

ranking / filtering

Web interface
Globalizer: Overview

Model

Data sets

normalisation

sub-problem group generation

sub-problem group

candidate generation

ranking / filtering

Constraint library

Web interface
Constraint all_different(hostedBy_used)

Candidate Constraint
Ranking / Filtering

```
constraint all_different(hostedBy_used)
```

Candidate Constraint

```
hostedBy1 = [4, 2, 3, 5, 4, 3, 5,...];
visits1   = [0, 0, 0, 0, 0, 1, 0,...];
hostedBy2 = [4, 5, 3, 2, 4, 3, 5,...];
visits2   = [0, 0, 0, 0, 0, 0, 0,...];
hostedBy3 = [4, 5, 3, 1, 3, 1, 5,...];
visits3   = [0, 0, 0, 1, 0, 0, 0,...];
```

Candidate Samples
**Candidate Constraint**

```
constraint all_different(hostedBy_used)
```

**Candidate Samples**

```
hostedBy1 = [4, 2, 3, 5, 4, 3, 5,...]);
visits1   = [0, 0, 0, 0, 0, 1, 0,...]);
hostedBy2 = [4, 5, 3, 2, 4, 3, 5,...]);
visits2   = [0, 0, 0, 0, 0, 0, 0,...]);
hostedBy3 = [4, 5, 3, 1, 3, 1, 5,...]);
visits3   = [0, 0, 0, 1, 0, 0, 0,...]);
```

**Sub-Problem**

```
int: g = min(GuestCrews);

constraint forall (h in HostBoats,
                  t in Time)
   visits[g,h,t] = 1 <-> hostedBy[g,t] = h;

constraint forall (h in HostBoats)
   sum (t in Time) (visits[g,h,t]) <= 1;
```
**Candidate Constraint**

\[
\text{constraint all_different(hostedBy_used)}
\]

**Sub-Problem**

\[
\text{int: } g = \min(\text{GuestCrews});
\]

\[
\text{constraint forall (h in HostBoats, t in Time)}\]
\[
\text{visits}[g,h,t] = 1 \leftrightarrow \text{hostedBy}[g,t] = h;
\]

\[
\text{constraint forall (h in HostBoats)}\]
\[
\sum (t \in \text{Time}) (\text{visits}[g,h,t]) \leq 1;
\]

**Candidate Samples**

- \( \text{hostedBy1} = [4, 2, 3, 5, 4, 3, 5, \ldots] \);
- \( \text{visits1} = [0, 0, 0, 0, 0, 1, 0, \ldots] \);
- \( \text{hostedBy2} = [4, 5, 3, 2, 4, 3, 5, \ldots] \);
- \( \text{visits2} = [0, 0, 0, 0, 0, 0, 0, \ldots] \);
- \( \text{hostedBy3} = [4, 5, 3, 1, 3, 1, 5, \ldots] \);
- \( \text{visits3} = [0, 0, 0, 1, 0, 0, 0, \ldots] \);

**Sub-Problem Accepts Candidate Samples?**
**Candidate Constraint**

\[ \text{int: } g = \min(\text{GuestCrews}); \]

\[
\text{constraint } \forall (h \in \text{HostBoats}, t \in \text{Time}) \text{ visits}[g, h, t] = 1 \iff \text{hostedBy}[g, t] = h; \\
\text{constraint } \forall (h \in \text{HostBoats}) \sum (t \in \text{Time}) (\text{visits}[g, h, t]) \leq 1; \\
\]

---

**Fraction of accepted samples**

\[
\frac{30}{30} \text{ bin\_packing\_capa}(\ldots); \\
\frac{30}{30} \text{ alldifferent}(\ldots) \\
\frac{30}{30} \text{ channel}(\ldots) \\
\frac{17}{30} \text{ atleast}(\ldots) \\
\ldots
\]

---

**Sub-Problem Accepts Candidate Samples?**

```
hostedBy1 = [4, 2, 3, 5, 4, 3, 5,...]);
visits1 = [0, 0, 0, 0, 0, 1, 0,...]);
hostedBy2 = [4, 5, 3, 2, 4, 3, 5,...]);
visits2 = [0, 0, 0, 0, 0, 0, 0,...]);
hostedBy3 = [4, 5, 3, 1, 3, 1, 5,...]);
visits3 = [0, 0, 1, 0, 0, 0, 0,...]);
```
Globalizer

Model

Data sets

normalisation

sub-problem group generation

Constraint library

sub-problem group

candidate generation

ranking / filtering

Web interface
Globalizer

Model

Data sets

normalisation

sub-problem group generation

Constraint library

sub-problem group

candidate generation

ranking / filtering

Web interface
User Interface

minizinc.org/globalizer

```
1.00 bin_packing_capa(capacity, [hostedBy[1,3], hostedBy[2,3], hostedBy[3,3],
    hostedBy[4,3], hostedBy[5,3], hostedBy[6,3], hostedBy[7,3],
    hostedBy[8,3], hostedBy[9,3], hostedBy[10,3]], crew)

1.00 alldifferent([hostedBy[10,1],hostedBy[10,2],hostedBy[10,3]])

1.00 channel([hostedBy[10,3]],
    [visits[10,1,3], visits[10,2,3], visits[10,3,3], visits[10,4,3], visits[10,5]
1.00 unary([hostedBy[10,1], hostedBy[10,2], hostedBy[10,3]],
    [visits[10,5,1], visits[10,5,2], visits[10,5,3]])
```

```
model

set of int : HostBoats = 1..nh;
set of int : GuestCrews = 1..ng;
set of int : Time = 1..p;

array [GuestCrews] of int : crew;
array [HostBoats] of int : capacity;

array [GuestCrews, Time] of var HostBoats : hostedBy;

array [GuestCrews, HostBoats, Time] of var 0..1 : visits;

constraint forall (g in GuestCrews, h in HostBoats, t in Time)
    (visits[g,h,t] = 1 <-> hostedBy[g,h,t]);

constraint forall (h in HostBoats) (forall (g in GuestCrews)
    (sum (t in Time) (visits[g,h,t]) <= 1)
    /
    forall (t in Time)
     (sum (g in GuestCrews) (crew[g]*visits[g,h,t]) <= capacity[h])
    )

array [GuestCrews, GuestCrews, Time] of var 0..1 : meet;

constraint forall (k, l in GuestCrews where k<l) (forall (t in Time)
    (hostedBy[k,t] = hostedBy[l,t] -> meet[k,l,t] = 1)
    /
    sum (t in Time) (meet[k,l,t]) <= 1
    )

solve satisfy;
output [show(hostedBy)];
```
1.00 bin_packing_capa(capacity, [hostedBy[1,3], hostedBy[2,3], hostedBy[3,3], hostedBy[4,3], hostedBy[5,3], hostedBy[6,3], hostedBy[7,3], hostedBy[8,3], hostedBy[9,3], hostedBy[10,3]], crew)

1.00 alldifferent([hostedBy[10,1],hostedBy[10,2],hostedBy[10,3]])

1.00 channel([hostedBy[10,3]], [visits[10,1,3], visits[10,2,3], visits[10,3,3], visits[10,4,3], visits[10,5,3]])

1.00 unary([hostedBy[10,1], hostedBy[10,2], hostedBy[10,3]], [visits[10,5,1], visits[10,5,2], visits[10,5,3]])
User Interface
minizinc.org/globalizer

1.00 bin_packing_capa(capacity, [hostedBy[1,3], hostedBy[2,3], hostedBy[3,3],
    hostedBy[4,3], hostedBy[5,3], hostedBy[6,3], hostedBy[7,3],
    hostedBy[8,3], hostedBy[9,3], hostedBy[10,3]], crew)

array [GuestCrews, HostBoats, Time] of var 0..1 : visits;
constraint forall (g in GuestCrews, h in HostBoats, t in Time)
    (visits[g,h,t] = 1 <-> hostedBy[g,t]=h);

constraint forall (h in HostBoats) (
    forall (g in GuestCrews)
        (sum (t in Time) (visits[g,h,t]) <= 1)
    /
    forall (t in Time)
        (sum (g in GuestCrews) (crew[g]*visits[g,h,t]) <= capacity[h])
);
## Experiments

| problem           | time (s) | |groups| | sat. tests | top candidates         |
|-------------------|----------|----------------|--------|-------------|------------------------|
| Basketball        | 58       | 4              |        | 179         | gcc                    |
| Cars              | 215      | 12             |        | 31335       | gcc count sliding_sum  |
| Jobshop           | 23       | 16             |        | 3620        | unary                  |
| Party             | 691      | 48             |        | 36429       | bin_packing_capa alldifferent channel unary |
| Packing           | 1659     | 29             |        | 114597      | diffn                  |
| Schedule          | 174      | 13             |        | 22835       | gcc                    |
| Sudoku (Linear)   | 3996     | 166            |        | 24465       | alldifferent gcc       |
| Warehouses        | 245      | 24             |        | 69871       | gcc                    |
Future Work
Future Work

- Automatic Channeling
Future Work

- Automatic Channeling
- Syntactic matching
Future Work

- Automatic Channeling
- Syntactic matching
- Symmetry & Dominance Detection
Future Work

- Automatic Channeling
- Syntactic matching
- Symmetry & Dominance Detection
- Integration of Global Constraint Catalog
Future Work

- Automatic Channeling
- Syntactic matching
- Symmetry & Dominance Detection
- Integration of Global Constraint Catalog
- Show adequacy of sampling approach
Summary

Use model structure to guide constraint discovery
- Split model into subproblems
- Extract arguments
- Test equivalence
  - subproblem $\rightarrow$ constraint ?
  - subproblem $\leftarrow$ constraint ?

The MiniZinc Globalizer is available at:
- minizinc.org/globalizer

Implemented with libmzn C++ / Haskell interface:
- minizinc.org/libmzn
Thank You

Use **model structure** to guide constraint discovery

- Split model into **subproblems**
- Extract arguments
- Test equivalence
  - **subproblem → constraint** ?
  - **subproblem ← constraint** ?

The MiniZinc Globalizer is available at:

- [minizinc.org/globalizer](http://minizinc.org/globalizer)

Implemented with **libmzn C++ / Haskell interface**:

- [minizinc.org/libmzn](http://minizinc.org/libmzn)