Optimising Quantified Expressions in Constraint Models

Ian P. Gent, Ian Miguel and Andrea Rendl

University of St Andrews, UK
AIT Austrian Institute of Technology, Austria

September 2010
Workshop on Modelling and Reformulation

Context of this Work

- **Quantified expressions** in solver-independent constraint modelling languages

Context of this Work

- **Quantified expressions** in solver-independent constraint modelling languages
- Example:

  forall i,j:int(1..n) .
  (i≠j) ⇒ (q[i]-i ≠ q[j]-j)

Context of this Work

- **Quantified expressions** in solver-independent constraint modelling languages
- Example:

  forall i,j:int(1..n) .
  (i≠j) ⇒ (q[i]-i ≠ q[j]-j)

- powerful means to compactly represent a set of expressions

Context of this Work

- **Quantified expressions** in solver-independent constraint modelling languages
- Example:

  forall i,j:int(1..n) .
  $(i \neq j) \Rightarrow (q[i]-i \neq q[j]-j)$
- powerful means to compactly represent a set of expressions
- same structure in all constraint modelling languages

Context of this Work

- **Quantified expressions** in solver-independent constraint modelling languages
- Example:

  forall i,j:int(1..n) .
  (i≠j) ⇒ (q[i]-i ≠ q[j]-j)

- powerful means to compactly represent a set of expressions
- same structure in all constraint modelling languages
  - **restriction**: no decision variables in $i_1, \ldots, i_m$ and int(lb..ub)

Goal and Contributions

- **Our Observation**:
  quantified expressions can contain **redundancies**, often when formulated by **novices**

Goal and Contributions

- **Our Observation**:
  quantified expressions can contain **redundancies**, often when formulated by **novices**
- **Our Goal**:
  automatically **improve** poorly formulated quantified expressions

Goal and Contributions

- **Our Observation**:
  quantified expressions can contain **redundancies**, often when formulated by **novices**
- **Our Goal**:
  automatically **improve** poorly formulated quantified expressions
- **Our Contributions**:
  - we consider **2 kinds of redundancies**

Goal and Contributions

- **Our Observation**:
  quantified expressions can contain **redundancies**, often when formulated by **novices**
- **Our Goal**:
  automatically **improve** poorly formulated quantified expressions
- **Our Contributions**:
  - we consider **2 kinds of redundancies**
  - we propose means to **detect** and **address** those redundancies

Loop-invariant Expressions

- **Idea**: analyse equivalent representations of quantified expressions

Loop-invariant Expressions

- **Idea**: analyse equivalent representations of quantified expressions
- Example: $(\mathbf{x} = \mathbf{0}) \Rightarrow \forall_{i \in D}.\ (x[i] = i)$

Loop-invariant Expressions

- **Idea**: analyse equivalent representations of quantified expressions
- Example:

$$(\mathbf{x} = \mathbf{0}) \Rightarrow \forall_{i \in D}.\ (x[i] = i)$$
$$\equiv$$
$$\forall_{i \in D}.\ (\mathbf{x} = \mathbf{0}) \Rightarrow (x[i] = i)$$

Loop-invariant Expressions

- **Idea**: analyse equivalent representations of quantified expressions
- Example: $$(\mathbf{x} = \mathbf{0}) \Rightarrow \forall_{i \in D}.\ (x[i] = i)$$ $$\equiv$$ $$\forall_{i \in D}.\ (\mathbf{x} = \mathbf{0}) \Rightarrow (x[i] = i)$$
- we call '$(x = 0)$' **loop-invariant**

Loop-invariant Expressions

- **Idea**: analyse equivalent representations of quantified expressions
- Example: $$(\mathbf{x} = \mathbf{0}) \;\Rightarrow\; \forall_{i \in D}.\ (x[i] = i)$$ $$\equiv$$ $$\forall_{i \in D}.\ (\mathbf{x} = \mathbf{0}) \Rightarrow (x[i] = i)$$
- we call '$(x = 0)$' **loop-invariant**
- **Question:** which representation is better?

Loop-invariant Expressions

- Many different cases....
  1. $\mathbf{A} \wedge \forall_I E_I \quad \equiv \quad \forall_I \mathbf{A} \wedge E_I$

Loop-invariant Expressions

- Many different cases....
  1. $\mathbf{A} \wedge \forall_I E_I \equiv \forall_I \mathbf{A} \wedge E_I$
  2. $\mathbf{A} \vee \exists_I E_I \equiv \exists_I \mathbf{A} \vee E_I$
  3. $m\mathbf{A} + \sum_I E_I \equiv \sum_I \mathbf{A} + E_I$ where $m = |I|$
  4. $\mathbf{A} \vee (\forall_I E_I)) \equiv \forall_I \mathbf{A} \vee E_I$
  5. etc

Loop-invariant Expressions

- Many different cases....
  1. $\mathbf{A} \wedge \forall_I E_I \quad \equiv \quad \forall_I \mathbf{A} \wedge E_I$
  2. $\mathbf{A} \vee \exists_I E_I \quad \equiv \quad \exists_I \mathbf{A} \vee E_I$
  3. $m\mathbf{A} + \sum_I E_I \quad \equiv \quad \sum_I \mathbf{A} + E_I$ where $m = |I|$
  4. $\mathbf{A} \vee (\forall_I E_I)) \quad \equiv \quad \forall_I \mathbf{A} \vee E_I$
  5. etc
- Intuitively, we expect the outside-representation to be better...

Loop-invariant Expressions

- Many different cases....
  1. $\mathbf{A} \wedge \forall_I E_I \quad \equiv \quad \forall_I \mathbf{A} \wedge E_I$
  2. $\mathbf{A} \vee \exists_I E_I \quad \equiv \quad \exists_I \mathbf{A} \vee E_I$
  3. $m\mathbf{A} + \sum_I E_I \quad \equiv \quad \sum_I \mathbf{A} + E_I$ where $m = |I|$
  4. $\mathbf{A} \vee (\forall_I E_I)) \quad \equiv \quad \forall_I \mathbf{A} \vee E_I$
  5. etc
- Intuitively, we expect the outside-representation to be better... is this true for all cases?

Comparing Representations

- We compare representations at **solver level** (flat representation)

Comparing Representations

- We compare representations at **solver level** (flat representation)
- We assume the solver provides:
  - (reifyable) n-ary conjunction ($\forall$)
  - (reifyable) n-ary disjunction ($\exists$)
  - n-ary sum ($\sum$)

Comparing Representations

- We compare representations at **solver level** (flat representation)
- We assume the solver provides:
  - (reifyable) n-ary conjunction ($\forall$)
  - (reifyable) n-ary disjunction ($\exists$)
  - n-ary sum ($\sum$)
- Let's look at one case (see paper for other cases):
  $\mathbf{A} \Rightarrow (\forall_I E_I) \quad \equiv \quad \forall_I \mathbf{A} \Rightarrow E_I$

Comparing Representations

	Inside-Representation	Outside-Representation
Original	$(\forall_I A \Rightarrow E_I)$	$A \Rightarrow (\forall_I E_I)$

Comparing Representations

	Inside-Representation	Outside-Representation
Original	$(\forall_I A \Rightarrow E_I)$	$A \Rightarrow (\forall_I E_I)$
Unrolled	$(A \Rightarrow E_1) \wedge$ $\ldots$ $(A \Rightarrow E_k)$	$A \Rightarrow (E_1 \wedge \cdots \wedge E_k)$

Comparing Representations

	Inside-Representation	Outside-Representation
Original	$(\forall_I A \Rightarrow E_I)$	$A \Rightarrow (\forall_I E_I)$
Unrolled	$(A \Rightarrow E_1) \wedge$ $\ldots$ $(A \Rightarrow E_k)$	$A \Rightarrow (E_1 \wedge \cdots \wedge E_k)$
Flat (unnested)	$a \Rightarrow e_1$ $\ldots$ $a \Rightarrow e_k$	$aux \Leftrightarrow (e_1 \wedge \cdots \wedge e_k)$ $a \Rightarrow aux$

Comparing Representations

	Inside-Representation	Outside-Representation
Original	$(\forall_I A \Rightarrow E_I)$	$A \Rightarrow (\forall_I E_I)$
Unrolled	$(A \Rightarrow E_1) \wedge$ $\ldots$ $(A \Rightarrow E_k)$	$A \Rightarrow (E_1 \wedge \cdots \wedge E_k)$
Flat (unnested)	$a \Rightarrow e_1$ $\ldots$ $a \Rightarrow e_k$	$aux \Leftrightarrow (e_1 \wedge \cdots \wedge e_k)$ $a \Rightarrow aux$
	0 *auxiliary variables* k *constraints*	1 *auxiliary variable* 2 *constraints*

Comparing Representations

- **Inside-Representation**: more constraints (increasing with k), no additional variables

Comparing Representations

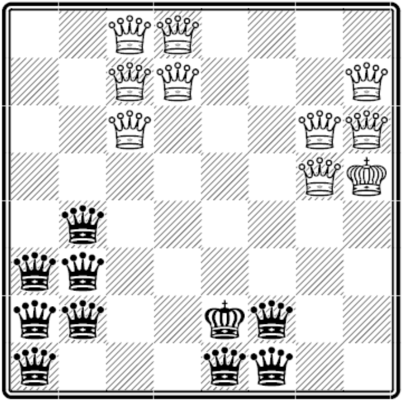
- **Inside-Representation**: more constraints (increasing with k), no additional variables
- **Outside-Representation**: only two constraints but 1 additional variable

Comparing Representations

- **Inside-Representation**: more constraints (increasing with k), no additional variables
- **Outside-Representation**: only two constraints but 1 additional variable
- Let's compare the representations in an example!

Example: Peaceful Army of Queens

Place two equally-sized armies of queens on a chess board such that they do not attack another, maximising the army size



Peaceful Army of Queens: Outside Representation

Non-attacking Constraints in model based on Smith et al (2004):

forall *fields(i,j) on the chess board.*

Peaceful Army of Queens: Outside Representation

Non-attacking Constraints in model based on Smith et al (2004):

forall *fields(i,j) on the chess board.*
white queen at field(i,j) $\Rightarrow$

Peaceful Army of Queens: Outside Representation

Non-attacking Constraints in model based on Smith et al (2004):

forall *fields(i,j) on the chess board.*
white queen at field(i,j) $\Rightarrow$
forall k.
no black queen at field(i,k) (same column)

Peaceful Army of Queens: Outside Representation

Non-attacking Constraints in model based on Smith et al (2004):

forall *fields(i,j) on the chess board.*
  white queen at field(i,j) $\Rightarrow$
    forall k.
      no black queen at field(i,k) (same column)
     $\wedge$ *no black queen at field(k,j) (same row)*

Peaceful Army of Queens: Outside Representation

Non-attacking Constraints in model based on Smith et al (2004):

forall *fields(i,j) on the chess board.*
white queen at field(i,j) $\Rightarrow$
forall *k*.
no black queen at field(i,k) (same column)
$\wedge$ *no black queen at field(k,j) (same row)*
$\wedge$ *no black queen at field(i+k,j+k) (NW-diagonal)*
$\wedge$ *no black queen at field(i-k,j+k) (SW-diagonal)*
$\wedge$ *no black queen at field(i+k,j-k) (NE-diagonal)*
$\wedge$ *no black queen at field(i-k,j-k) (SE-diagonal)*

Peaceful Army of Queens: Inside Representation

Alternatively, moving loop-invariant expression **inside**:

forall *fields(i,j) on the chess board.*

Peaceful Army of Queens: Inside Representation

Alternatively, moving loop-invariant expression **inside**:

forall *fields(i,j) on the chess board.*
  forall k.
    white queen at field(i,j) $\Rightarrow$
        no black queen at field(i,k) (column)

Peaceful Army of Queens: Inside Representation

Alternatively, moving loop-invariant expression **inside**:

forall *fields(i,j) on the chess board.*
  forall k.
    white queen at field(i,j) $\Rightarrow$
          no black queen at field(i,k) (column)
$\wedge$ **forall** k.
    white queen at field(i,j) $\Rightarrow$
          $\wedge$ *no black queen at field(k,j) (row)*

Peaceful Army of Queens: Inside Representation

Alternatively, moving loop-invariant expression **inside**:

forall *fields(i,j) on the chess board.*
forall k.
white queen at field(i,j) $\Rightarrow$
no black queen at field(i,k) (column)
$\wedge$ **forall** k.
white queen at field(i,j) $\Rightarrow$
$\wedge$ *no black queen at field(k,j) (row)*
$\wedge$ **forall** k.
white queen at field(i,j) $\Rightarrow$
$\wedge$ *no black queen at field(i+k,j+k) (NW-diagonal)*
...

Comparing Inside- and Outside-Representation

What did we do?

1. We modelled two different PAQ models (in Essence')

Comparing Inside- and Outside-Representation

What did we do?

1. We modelled two different PAQ models (in Essence')
2. We translated both models to solvers Gecode and Minion (using Tailor), generating:

Comparing Inside- and Outside-Representation

What did we do?

1. We modelled two different PAQ models (in Essence')
2. We translated both models to solvers Gecode and Minion (using Tailor), generating:
   - outside-representation
   - inside-representation

   for both models

Comparing Inside- and Outside-Representation

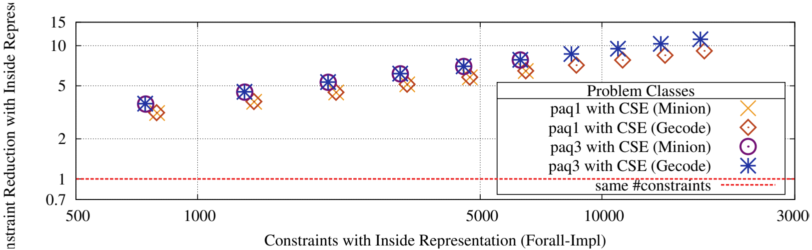
What did we do?

1. We modelled two different PAQ models (in Essence')
2. We translated both models to solvers Gecode and Minion (using Tailor), generating:
   - outside-representation
   - inside-representation

   for both models
3. We solved both representations using the same solving setup

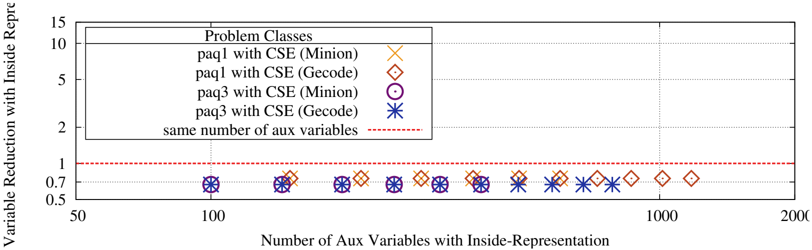
Comparing Number of Constraints

Inside-Representation has far **more** constraints than **Outside**-Representation

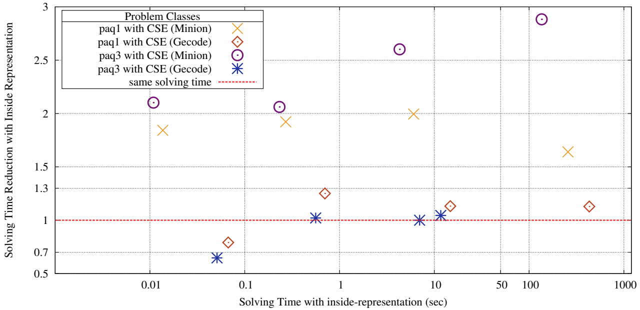


Comparing Number of Auxiliary Variables

Inside-Representation has **30% less** auxiliary variables than **Outside**-Representation



Comparing Number Solving Performance



- **Inside**-Rep. better in Minion (speedup of max. 300%)
- **Inside**-Rep. slightly better in Gecode (speedup of max. 30%)

Conclusion on Loop-Invariant Expressions

- **Against our expectations**: it can be beneficial to move loop-invariant expressions into quantifications

Conclusion on Loop-Invariant Expressions

- **Against our expectations**: it can be beneficial to move loop-invariant expressions into quantifications
- Difficult to make a **general** statement
  - depends on solver (provided propagators, architecture, etc)
  - depends on problem structure

Conclusion on Loop-Invariant Expressions

- **Against our expectations**: it can be beneficial to move loop-invariant expressions into quantifications
- Difficult to make a **general** statement
  - depends on solver (provided propagators, architecture, etc)
  - depends on problem structure
- Tailor can **automatically reformulate** quantifications to inside/outside-representation
  - user can choose preferable representation (for each case) in translation settings

Weak Guards

- A **guard** B for an expression E has to hold to enforce E
  - $B \Rightarrow E$

Weak Guards

- A **guard** B for an expression E has to hold to enforce E
  - $B \Rightarrow E$
- Often used in modelling, mostly to restrict quantifying variables

Weak Guards

- A **guard** B for an expression E has to hold to enforce E
  - $B \Rightarrow E$
- Often used in modelling, mostly to restrict quantifying variables
- **Example:**

forall i, j in $(1..n)$.
$(i \neq j) \quad \Rightarrow \quad \text{queen[i]} + \text{i} \neq \text{queen[j]} + \text{j}$

Weak Guards

- If guards are **weak** they yield duplicate constraints

Weak Guards

- If guards are **weak** they yield duplicate constraints
- **forall** i, j in $(1..n)$.
  $(i \neq j) \quad \Rightarrow \quad \text{queen[i]} + \text{i} \neq \text{queen[j]} + \text{j}$

Weak Guards

- If guards are **weak** they yield duplicate constraints
- **forall** i, j in $(1..n)$.
  $(i \neq j) \quad \Rightarrow \quad$ queen[i] + i $\neq$ queen[j] + j
- is unrolled to:
  queen[1]+1 != queen[2]+2, queen[1]+1 != queen[3]+3,
  queen[2]+2 != queen[1]+1, queen[2]+2 != queen[3]+3,
  queen[3]+3 != queen[2]+2, queen[3]+3 != queen[1]+1,
  etc

Weak Guards

- If guards are **weak** they yield duplicate constraints
- **forall** i, j in $(1..n)$.
  $(i \neq j) \quad \Rightarrow \quad$ queen[i] + i $\neq$ queen[j] + j
- is unrolled to:
  queen[1]+1 != queen[2]+2, queen[1]+1 != queen[3]+3,
  queen[2]+2 != queen[1]+1, queen[2]+2 != queen[3]+3,
  queen[3]+3 != queen[2]+2, queen[3]+3 != queen[1]+1,
  etc

Weak Guards

- If guards are **weak** they yield duplicate constraints
- **forall** i, j in $(1..n)$.
  $(i \neq j) \quad \Rightarrow \quad \text{queen[i]} + \text{i} \neq \text{queen[j]} + \text{j}$
- is unrolled to:
  queen[1]+1 != queen[2]+2, queen[1]+1 != queen[3]+3,
  queen[2]+2 != queen[1]+1, queen[2]+2 != queen[3]+3,
  queen[3]+3 != queen[2]+2, queen[3]+3 != queen[1]+1,
  etc

Addressing Weak Guards

- **Option1**: remove duplicate constraints after quantification is unrolled

Addressing Weak Guards

- **Option1**: remove duplicate constraints after quantification is unrolled
  - **problem**: only possible when quantification can be unrolled, i.e. all parameters are known

Addressing Weak Guards

- **Option1**: remove duplicate constraints after quantification is unrolled
  - **problem**: only possible when quantification can be unrolled, i.e. all parameters are known
- **Option2**: strengthen the guard!

Strengthening Guards

- **Our Idea**: use **unification** to strengthen guards

Strengthening Guards

- **Our Idea**: use **unification** to strengthen guards
- **Unification Example**:
  - What is the unifier for '$x + i$' and '$x + 3$'?

Strengthening Guards

- **Our Idea**: use **unification** to strengthen guards
- **Unification Example**:
  - What is the unifier for '$x + i$' and '$x + 3$'?
  - $u = \{3/i\}$ (i substituted with 3)
- We want to demonstrate the algorithm on an example...

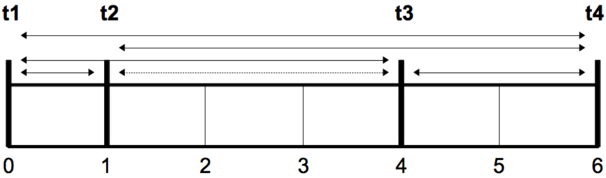
Strengthening the Guard in Golomb Ruler

A Golomb Ruler has *n* **ticks** such that the **distance** between each tick is **different**, minimising the length of the ruler.

Strengthening the Guard in Golomb Ruler

A Golomb Ruler has *n* **ticks** such that the **distance** between each tick is **different**, minimising the length of the ruler.

Sample Golomb Ruler with 4 ticks and length 6:



Strengthening the Guard in Golomb Ruler

‘The distances between all ticks are different’-Constraint:

Strengthening the Guard in Golomb Ruler

'The distances between all ticks are different'-Constraint:

forall $i1, i2, i3, i4$: TICKS.
$((i1>i2) \wedge (i3>i4) \wedge (i2 \neq i4)) \Rightarrow$
$(ruler[i1]-ruler[i2] \neq ruler[i3]-ruler[i4])$

Strengthening the Guard in Golomb Ruler

$$\text{STRENGTHEN\_GUARD}(\forall_I : D.B_I \Rightarrow E_I)$$

Strengthening the Guard in Golomb Ruler

STRENGTHEN_GUARD($\forall_I : D.B_I \Rightarrow E_I$)

- (1) If E_I's root node corresponds to a binary commutative operator then continue, otherwise stop.

Strengthening the Guard in Golomb Ruler

STRENGTHEN_GUARD($\forall_I : D.B_I \Rightarrow E_I$)

- (1) If E_I's root node corresponds to a binary commutative operator then continue, otherwise stop.

forall $i1, i2, i3, i4$: TICKS.
  ((i1>i2) $\wedge$ (i3>i4) $\wedge$ (i2$\neq$i4)) $\Rightarrow$
    (ruler[i1]-ruler[i2] $\neq$ ruler[i3]-ruler[i4])

Strengthening the Guard in Golomb Ruler

STRENGTHEN_GUARD($\forall_I : D.B_I \Rightarrow E_I$)

- (2) Compute the set of unifiers U for the two children of E_I, e_1 and e_2.

UNIFY (ruler[i1]-ruler[i2], ruler[i3]-ruler[i4]):

$$u_1 = \{i_1/i_3 \wedge i_2/i_4\} \qquad u_2 = \{i_3/i_1 \wedge i_4/i_2\}$$
$$u_3 = \{i_3/i_1 \wedge i_2/i_4\} \qquad u_4 = \{i_1/i_3 \wedge i_4/i_2\}$$

Strengthening the Guard in Golomb Ruler

STRENGTHEN_GUARD($\forall_I : D.B_I \Rightarrow E_I$)

- (3) Search U for unifiers from which we can deduce equivalence of the quantifying variables.

UNIFY (ruler[i1]-ruler[i2], ruler[i3]-ruler[i4]):

$$u_1 = \{i_1/i_3 \wedge i_2/i_4\} \qquad u_2 = \{i_3/i_1 \wedge i_4/i_2\}$$
$$u_3 = \{i_3/i_1 \wedge i_2/i_4\} \qquad u_4 = \{i_1/i_3 \wedge i_4/i_2\}$$

we deduce that $(i_1 = i_3) \wedge (i_2 = i_4)$

Strengthening the Guard in Golomb Ruler

STRENGTHEN_GUARD($\forall_I : D.B_I \Rightarrow E_I$)

- (4) Add lex-ordering constraint C on all quantifying variables whose equivalence renders e_1 and e_2 equivalent

C: $i_1, i_2 \leq_{lex} i_3, i_4$
hence $(i_1 \leq i_3) \wedge (i_1 < i_3 \vee i_2 \leq i_4)$

Strengthening the Guard in Golomb Ruler

Yielding the constraint with **strengthend guard:**

forall $i1, i2, i3, i4$: TICKS.
((i1>i2) $\wedge$ (i3>i4) $\wedge$ (i2$\neq$i4) $\wedge$
$(i_1 \leq i_3) \wedge (i_1 < i_3 \vee i_2 \leq i_4))$
$\Rightarrow$
(ruler[i1]-ruler[i2] $\neq$ ruler[i3]-ruler[i4])

Strengthening the Guard in Golomb Ruler

Yielding the constraint with **strengthend guard:**

forall $i1, i2, i3, i4$: TICKS.
((i1>i2) ∧ (i3>i4) ∧ (i2≠i4) ∧
$(i_1 \leq i_3) \wedge (i_1 < i_3 \vee i_2 \leq i_4))$
⇒
(ruler[i1]-ruler[i2] ≠ ruler[i3]-ruler[i4])

However: we have not implemented the algorithm yet!

Effects of Duplicate constraints

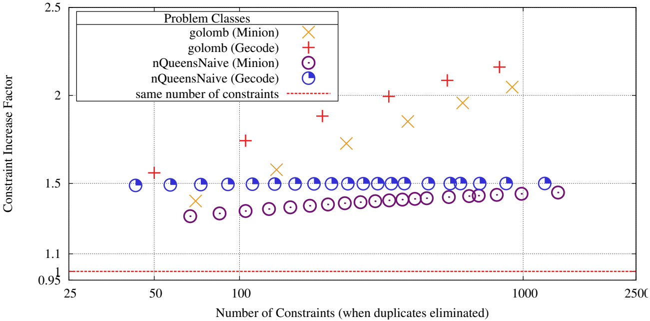
- How bad is the **effect of duplicate constraints** due to weak guards?
    - in other words: is it worth putting energy into strengthening guards?

Effects of Duplicate constraints

- How bad is the **effect of duplicate constraints** due to weak guards?
  - in other words: is it worth putting energy into strengthening guards?
- We analyse the effects on two naive models in solver Minion and Gecode:
  - Naive n-Queens
  - Naive Golomb Ruler

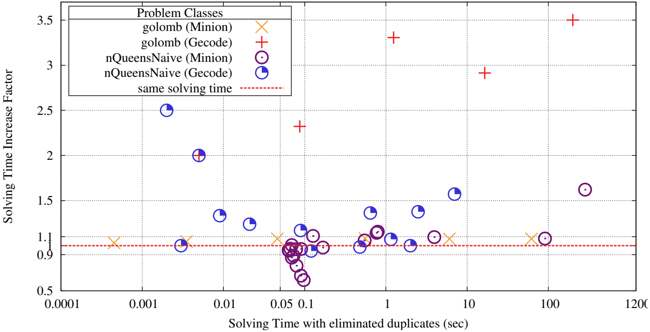
The Number of Duplicate Constraints

For both solvers: constant for n-Queens, linear within Golomb Ruler



Effect on Solving Performance

strong effect in Gecode, mild effect in Minion



Conclusions for Weak Guards

- Duplicate constraints **can impair** the solving performance

Conclusions for Weak Guards

- Duplicate constraints **can impair** the solving performance
- We have an idea on how to **strengthen guards** to address this redundancy

Conclusions for Weak Guards

- Duplicate constraints **can impair** the solving performance
- We have an idea on how to **strengthen guards** to address this redundancy
- We still need to implement/test/refine the algorithm..

Summary

- There is scope for optimisations in quantifications

Summary

- There is scope for optimisations in quantifications
- We can already provide some enhancement

Summary

- There is scope for optimisations in quantifications
- We can already provide some enhancement
- But there is still a lot to investigate!

Thank You.