Constraint Acquisition: Learning Constraint Models from Data

Dimos Tsouros
dimos.tsouros@kuleuven.be
Introduction

❖ Constraint programming (CP)
  ❑ Solving combinatorial problems in AI

❖ Model + Solve paradigm
Introduction

Modelling is not always trivial

- Requires expertise
- Bottleneck for the wider use of CP

Constraint Acquisition

Structure Learning: learn the constraints of the problem, not parameters
Passive acquisition: Using existing data

(Inter)active acquisition: Interact with the user
**Terminology**

- **X**: Variables, \( X = \{x_1, x_2, x_3, \ldots, x_{|X|}\} \)
- **\( D^X \)**: Domains
- **\( \Gamma \)**: Constraint Language, \( \Gamma = \{r_1, r_2, r_3, \ldots, r_{|\Gamma|}\} \)
- **B**: Set of candidate constraints, \( B = \{c_1, c_2, c_3, \ldots, c_{|B|}\} \)
- **\( C_L \)**: Learned constraint set
- **E**: set of Examples, \( E = \{e_1, e_2, e_3, \ldots, e_{|E|}\} \)

**Goal:**

\( C_L \) is equivalent to the (unknown) target set \( C_T \)

*Constraint Acquisition, C. Bessiere et al., AIJ, 2017*
Inputs and Outputs

Running Example: 4x4 Sudoku

$X$: Variables of the problem, $X = \{x_{1,1}, x_{1,2}, x_{1,3}, \ldots, x_{4,4}\}$

$D^X$: Domains of the variables, $D^X_i = \{1, 2, 3, 4\}$

$\Gamma$: Constraint Language, $\Gamma = \{=, \neq, \leq, \geq\}$

$B$: Set of candidate constraints, $B = \{x_{1,1} \neq x_{1,2}, x_{1,1} = x_{1,2}, \ldots\}$

$C_L$: learned constraints, ideally all the $\neq$ constraints in rows, cols and blocks

$E$: set of Examples, $E = \{e_1, e_2, e_3, \ldots, e_{|E|}\}$

The user may not know what constraint relations appear in the model
Inputs and Outputs

Alternative: Model with global constraints

$X$: Variables of the problem, $X = \{x_{1,1}, x_{1,2}, x_{1,3}, \ldots, x_{4,4}\}$

$D^X$: Domains of the variables, $D^X_i = \{1, 2, 3, 4\}$

$\Gamma$: Constraint Language, $\Gamma = \{\text{AllDifferent}\}$

$B$: Set of candidate constraints, $B = \{\text{AllDifferent}(x_{1,1}, x_{1,2}, \ldots), \text{AllDifferent}(x_{1,2}, x_{3,5}, \ldots)\}$

$C_L$: learned constraints, ideally all the \textit{AllDifferent constraints in rows, cols and blocks}

$E$: set of Examples, $E = \{e_1, e_2, e_3, \ldots, e_{|E|}\}$

Problem Formulation

<table>
<thead>
<tr>
<th></th>
<th>$X_{1,1}$</th>
<th>$X_{1,2}$</th>
<th>$X_{1,3}$</th>
<th>$X_{1,4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_{2,1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_{2,2}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_{3,1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_{3,2}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_{4,1}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X_{4,2}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example $e_1$

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Constraint Acquisition system

Vocabulary $(X, D)$

Examples $E$

Queries

Answers
Adapting Candidate Elimination

- $B$: set of (remaining) candidate constraints
- $C_T$: target set of constraints
- $C_L$: learned set of constraints
During the learning process:
- Constraints are removed from $B$
- Constraints are added to $C_L$

- $B$: set of (remaining) candidate constraints
- $C_T$: target set of constraints
- $C_L$: learned set of constraints
Adapting Candidate Elimination

During the learning process:
- Constraints are removed from $B$
- Constraints are added to $C_L$

- $B$: set of (remaining) candidate constraints
- $C_T$: target set of constraints
- $C_L$: learned set of constraints
Adapting Candidate Elimination

If $C_T$ is representable by $B$ the CA system will eventually converge to a $C_L$ equivalent to $C_T$. 

Set of all candidates

Removed candidates

Learned constraints

$B_{\overline{C_T}}$

$C_T$

$C_L$
Adapting Candidate Elimination

• Examples: Assignments to the variables of the problem

• Learning from **positive** examples (Solutions):
  - Violated constraints cannot be part of the model
  - Otherwise, it could not be a solution

  Shrink bias

  Set of all candidates
  
  Removed candidates
  
  Learned constraints

• Learning from **negative** examples (Non-solutions):
  - One (or more) violated constraint is a constraint of the problem
  - Otherwise, it would be a solution

  Learning Constraints

  Set of all candidates
  
  Removed candidates
  
  Learned constraints
We discussed:
- Why Constraint Acquisition
- Passive and active acquisition
- Inputs and outputs of CA
- Candidate Elimination
Passive Acquisition

Assignment to all variables of the problem, labelled as:

• a solution

• or a non-solution

Constraint Acquisition, C. Bessiere et al., AIJ, 2017
Passive Acquisition

**Version Space Learning**
- ConAcq  [C. Bessiere et al., AIJ, 2017]

**Generate and Aggregate**
- COUNT-CP  [M. Kumar et al., CP, 2022]
- MineAcq  [S. Prestwich., CP, 2021]
- BAYESACQ  [S. Prestwich et al., AMAI, 2021]
- EML  [M. Lombardi et al., 2017]
- SEQACQ  [S. Prestwich., ECAI, 2020]
- BAYESACQ  [S. Prestwich et al., AMAI, 2021]

**Data Mining**
- MineAcq  [S. Prestwich., CP, 2021]

**Training Classifiers**
- ModelSeeker  [N. Beldiceanu et al., CP, 2012]
- CABSC  [C. Coulombe et al., CP, 2012]

**Statistical Approaches**

**Other**
ModelSeeker – Learning Global Constraints from data

- Domain knowledge on which patterns to apply $\Gamma$
- Generate candidates
- $\Gamma$ $P$: patterns
- Constraint Seeker: Test candidates on $E'$
- $B$ $B_{sat}$ Dominance check on $B_{sat}$
- Transform $E_+$ to convenient form
- Only positive examples
- Actually, a bit more than just a violation test

A Model Seeker: Extracting Global Constraint Models from Positive Examples, N. Beldiceanu et al., CP, 2012
ConAcq – Sat-based version space approach

∀e ∈ E

Test constraints in e

B_{viol}

Yes

e ∈ E_{+}

No

Remove B_{viol}

Add ∀c ∈ B_{viol} c to C_L

Also from disjunctions in C_L

Constraint Acquisition, C. Bessiere et al., AIJ, 2017
Statistical approaches – BayesAcq

Create $E_c$, 

Train Bernoulli NB in $E_c$

Derive $P(e \notin sol(c) | E_-)$ and $P(e \notin sol(c) | E_+)$ from NB for all $c \in B$

Learn constraints with probability higher than a threshold

$C_L = \left\{ c \mid \frac{P(e \notin sol(c) | E_-)}{P(e \notin sol(c) | E_+)} > \text{threshold}, \forall c \in B \right\}$

Robust approach: Can handle noise in training set

transforming $E$ using candidate violations as Boolean features, $F_e = \{ e \in sol(c) \mid \forall c \in B \}, \forall e \in E$
Robust approach: Can handle noise in training set

Use of a Sequential Probability Ratio Test (SPRT) based approach in $E$, $\forall c \in B$

Not representing the whole set of candidates at the same time

Test the $c$ in $e$

$$\begin{align*}
N_{E+}(c) &\, += \, [e \notin \text{sol}(c) \land e \in E_+] \\
N_E(c) &\, += \, [e \notin \text{sol}(c)]
\end{align*}$$

Dismiss $c$ if $N_{E+}(c) > R$
Add $c$ to $C_L$ if $N_E(c) > A$

Remove from $c$ candidates

Robust Constraint Acquisition by Sequential Analysis, S. Prestwich., ECAI, 2020
Learns constraints of the following form: \( lb \leq expr \leq ub \)

Aggregates the bounds of the expressions based on \( E_+ \)

Language \( \Gamma \) contains **numerical** expressions and not constraints, e.g. \( x_{i,j} - x_{k,l}, x_{i,j} + x_{k,l} \) etc.

∀ \( expr \in \Gamma \)

\( E_+ \) ➔ **Aggregate Bounds** based on \( E_+ \) (min and max value)

∀ generated expression ➔ Generalization to first-order constraints

Express bounds using symbolic expressions ➔ Constraint Filtering

Pattern-based grouping like in ModelSeeker

Learning Constraint Programming Models from Data using Generate-and-Aggregate, M. Kumar et al., CP, 2022
We discussed:
- Passive Constraint Acquisition
- Different approaches
- Learning global and fixed arity constraints
- Learning first-order constraints
Interactive Constraint Acquisition

Oracle

Unlabelled examples

Labels

CSP Model

Membership query

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Answer: Negative in both of them (a constraint is violated)

Partial query

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>-</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Learning Constraints through Partial Queries, C. Bessiere et al., AIJ, 2023
Challenges for interactive CA

Minimum number of queries

Too many questions

Minimum waiting time for the user

I need to wait too much for each query

Learning constraint models from data, D Tsouros et al., 2023
Interactive Constraint Acquisition

- **Generate example**
- **Ask query to the user**
- **Answer**
  - **Learn violated constraints**
  - **Eliminate violated candidates**
  - **Update Version space**
- **Learn constraints:**
  - zoom in violated constraints’ scopes using partial queries and add them to $C_L$
- **Shrink version space:** remove violated constraints from $B$

- **Converged**

- **Example found**
- **No example found**

---

Learning Constraints through Partial Queries, C. Bessiere et al., AIJ, 2023
Interactive Constraint Acquisition

Positive answer: Eliminate candidates

Negative answer: Learn constraints

Query 0: is this a solution?
[[1 3 2 2]]
[1 1 1 1]
[2 4 3 2]]
Answer: No
Query generation

Informative query

- Generate “informative” examples

Quality of query

- Get the maximum amount of information

Convergence
Query generation

Find an Informative ("irredundant") query

- Not violating any learned constraint in $C_L$
- Violating at least one constraint from $B$

Find $e \in sol(C_L \land \forall_{c \in B} \neg c)$

Learning Constraints through Partial Queries, C. Bessiere et al., AIJ, 2023
What we can learn from examples

• Learning from positive examples (Solutions):
  - Violated constraints cannot be part of the model
  - Otherwise, it could not be a solution

• Learning from negative examples (Non-solutions):
  - One (or more) violated constraint is a constraint of the problem
  - Otherwise, it would be a solution

- Violated constraints cannot be part of the model
- Otherwise, it could not be a solution

• In both cases, we get information only about the violated candidates and not about the satisfied ones!
Query generation

Find an Informative ("irredundant") query

- Not violating any learned constraint in $C_L$
- Violating at least one constraint from $B$

Find $e \in sol(C_L \land \forall c \in B \sim c)$

- We know that the violated constraints cannot be part of the model
- Otherwise, it could not be a solution

- We know that (at least) one of the violated constraints is a constraint of the problem
- Otherwise, it would be a solution
Query generation

Informative query

Quality of query
- Get the maximum amount of information

Convergence
Query generation

Quality of query

• Better generated examples lead to faster convergence
• More information per query -> less queries needed

Typically: maximize candidate violations

\[ \max \sum_{c \in B} \sim c \]

Not fully aligning with the goal!! We will discuss this later

Constraint acquisition via partial queries, C. Bessiere et al., IJCAI, 2013
Finding an Irredundant query

- Finding an informative query is not always easy…

B can be huge!!

B can contain indirectly implied constraints

Assume a simple 9x9 Sudoku puzzle.
- Combinations of $\neq$ constraints imply others
- 648 of them imply the rest 162

When the 648 constraints have been learned and must be satisfied, the rest cannot be violated!

Indirect implications are not detected with simple propagation!!
Query generation

Custom solvers

- Custom solvers are often employed to deal with this
  - Looking for $e_Y \in sol(C_L[Y] \land \forall c \in B[Y] \sim c)$
  - Arbitrary $Y \subseteq X$

Projection-based query generation

- Project down to the relevant variables
  \[ Y = \bigcup_{c \in B} \text{var}(c) \]
  - We can only get information on variables of constraints in $B$
  - Avoiding indirect implications
- Find $e_Y \in sol(C_L[Y] \land \forall c \in B[Y] \sim c)$
Interactive Constraint Acquisition

What happens after query generation?

Generate example

- Example found
  - Ask query to the user
  - Answer
    - No: Learn violated constraints
    - Yes: Eliminate violated candidates

Update Version space

Converged

- No example found
  - Converged

Learn constraints: zoom in violated constraints’ scopes using partial queries and add them to $C_L$

How to find which of the violated constraint(s) to learn?

Shrink version space: remove violated constraints from $B$

$e$: example generated

$\kappa B \leftarrow \{ c \mid c \notin sol(e) \}$

$B \leftarrow B \setminus \kappa B(e)$
Learning a constraint

2-step process:

1. **FindScope**: exploit partial (sub)queries to find the problematic part of the assignment
2. **FindC**: Try different assignments to find the specific constraint in the scope
Finding the scope of a constraint

Exploit partial (sub)queries to find the conflicting part

- Find a minimal conflicting scope
- Find the constraint

In each step, post the example as a query to the user.

Decompose

Learning Constraints through Partial Queries, C. Bessiere et al., AIJ, 2023
Finding the scope of a constraint

Exploit partial (sub)queries to find the conflicting part

Find a minimal conflicting scope

Find the constraint

Learning Constraints through Partial Queries, C. Bessiere et al., AIJ, 2023
Finding the scope of a constraint

Find a minimal conflicting scope

Find the constraint

Finding the scope of a constraint
Finding the relation of a constraint

We have found the scope: \( \{x_{1,1}, x_{2,2}\} \)
Assume that the candidate constraints for this scope are:
\( \{x_{1,1} \neq x_{2,2}, x_{1,1} > x_{2,2}, x_{1,1} < x_{2,2}\} \)

What is the real conflict?

Try different assignments to find the specific constraint in the scope

```
1 - - -
- 2 - -
- - - -
- - - -
```

Positive
Remove \( x_{1,1} > x_{2,2} \)

```
2 - - -
- 1 - -
- - - -
- - - -
```

Positive
Remove \( x_{1,1} < x_{2,2} \)

Only \( x_{1,1} \neq x_{2,2} \) left as candidate: Learn it
\( C_L \leftarrow C_L \cup \{x_{1,1} \neq x_{2,2}\} \)

Learning Constraints through Partial Queries, C. Bessiere et al., AIJ, 2023
**Interactive Constraint Acquisition**

**QuAcq**

**QuAcq:**
- Learning one violated constraint per generated example
- Logarithmic number of queries for each constraint

Learning Constraints through Partial Queries, C. Bessiere et al., AIJ, 2023
Multiple Acquisition:

- Learn multiple constraints in each loop instance
- Don’t generate a new example when a constraint is learnt
  - Instead, get an example in a subset of variables not violating the constraint found

Efficient Multiple Constraint Acquisition, D. Tsouros et al., Constraints, 2020
What if the set of candidates is too large??

- Can’t store all of it at the same time??
- Too slow query generation??
**Bottom-up approach:** Using Interactive CA algorithms, on an (incrementally growing) subset of variables

Start with \( Y_1 \leftarrow \emptyset \), or a small subset of \( X \)

\[ \forall x_i \in X \setminus Y_1 \]

\[ Y_i \leftarrow Y_{i-1} \cup \{x_i\} \]

Generate \( B \) with \( x_i \)

\[ C_L[Y_i-1] \]

Interactive CA algorithm

\[ C_L[Y_i] \]

\[ Y \subset X \]

Yes

Converged

Guided Bottom-up Constraint Acquisition, D. Tsouros et al., CP, 2023
GrowAcq: Growing Acquisition

Start with $Y_1 \leftarrow \emptyset$, or a small subset of $X$

$x_i \in X \setminus Y_1$

$Y_i \leftarrow Y_{i-1} \cup \{x_i\}$

$x_i \in Y_i$

$Y \subseteq X$

Algorithm

$C_L[Y_i]$

Yes

No

Converged

Saves time during query generation! We can use more efficiently the time to guide better Constraint Acquisition

Bot

Interactive CA algorithms, on an (incrementally growing) subset of variables

Need only constraints that newly added variable participates

Guided Bottom-up Constraint Acquisition, D. Tsouros et al., CP, 2023
Interactive Constraint Acquisition
Small summary

We discussed:
- How Interactive CA works
- Query generation
- Finding the scope of problem constraints
- Finding the relation of the constraints
- Growing Acquisition
Guiding Interactive Constraint Acquisition
Guiding Query Generation

Quality of query

- Better generated examples lead to faster convergence
- More information per query -> less queries needed

Typically: maximize candidate violations

$$\max \sum_{c \in B} \sim c$$

Not fully aligning with the goal!!
Guiding Query Generation

Better generated examples lead to faster convergence

Positive answers: shrink $B$ fast

The more we have violated the faster $B$ will shrink

- Negative answers: Find the conflict fast

The less candidates we have violated, the less queries we need to find the constraint(s)

\[
\max \sum_{c \in B} \left[ e \notin \text{sol}(c) \right]
\]

\[
\min \sum_{c \in B} \left[ e \notin \text{sol}(c) \right]
\]

Opposite objectives based on the (future) answer

Guided Bottom-up Constraint Acquisition, D. Tsouros et al., CP, 2023
Guiding Query Generation

Better generated examples lead to faster convergence

Positive answers: shrink $B$ fast

Negative answers: Find the conflict fast

We cannot know the answer of the user before we ask the query $\rightarrow$ max violations

But what if we can predict if a candidate is a constraint of the problem or not?

The less candidates we have violated, the faster $B$ will shrink.

The less candidates we have violated, the less queries we need to find the scope (and relation) of the constraint(s)

$$\max \sum_{c \in B} [e \notin \text{sol}(c)]$$

Opposite objectives based on the (future) answer

$$\min \sum_{c \in B} [e \notin \text{sol}(c)]$$
Guiding Query Generation

- Positive answers: shrink $B$ fast $\rightarrow \max \sum_{c \in B} \mathbb{1}[e \notin \text{sol}(c)]$
- Negative answers: Find the conflict fast $\rightarrow \min \sum_{c \in B} \mathbb{1}[e \notin \text{sol}(c)]$

What if we can predict if a candidate is a constraint of the problem or not?

Use of Oracle $O(c) = (c \in CT)$, to guide query generation based on the prediction of the constraint

$$e = \underset{e \in \text{sol}(C_L \land B)}{\text{argmax}} \sum_{c \in B} \mathbb{1}[e \notin \text{sol}(c)] \cdot (1 - |\Gamma| \cdot [O(c)])$$

1. Aim for positive answers first: $\max \sum_{c \in B} \mathbb{1}[e \notin \text{sol}(c)]$
2. When a (probably true) constraint has to be violated, leading to a negative answer $\min \sum_{c \in B} \mathbb{1}[e \notin \text{sol}(c)]$

Guided Bottom-up Constraint Acquisition, D. Tsouros et al., CP, 2023
Guiding Query Generation

- **Positive answers**: shrink $B$ fast → $\max \sum_{c \in B} [e \notin \text{sol}(c)]$
- **Negative answers**: Find the conflict fast → $\min \sum_{c \in B} [e \notin \text{sol}(c)]$

What if we can predict if a candidate is a constraint of the problem or not?

Use of Oracle $O(c) = (c \in CT)$, to guide query generation based on **the prediction of the constraint**

$$e = \arg\max_{e \in \text{Sol}(C_L \land B)} \sum_{c \in B} [e \notin \text{sol}(c)] \cdot (1 - |\Gamma| \cdot [O(c)])$$

1. **Aim for positive answers** first:
   - $\max (\sum_{c \in B} [e \notin \text{sol}(c)])$
2. When a (probably true) constraint has to be violated, leading to a **negative answer**
   - $\min (\sum_{c \in B} [e \notin \text{sol}(c)])$

If the constraint $c$ is violated
- Increase objective value by 1

If it is a constraint predicted to be true: reduce objective value significantly

**Guided Bottom-up Constraint Acquisition**, D. Tsouros et al., CP, 2023
Guiding CA when finding the scope

Exploit partial (sub)queries to find the conflicting part

How are the removed variable assignments decided???

Follow the same logic
- But the new example is a sub-example of previous one
- Instead of deciding variable assignments, decide which variables to keep in the assignment

\[
e_Y = \arg\max_{e_Y \mid Y \subseteq \text{var}(e)} \sum_{c \in B} [e_Y \notin \text{sol}(c)] \cdot (1 - |\Gamma| \cdot \|O(c)\|)
\]

Aim to violate

- No
- True

O(c)

Aim to satisfy
Guiding CA when finding the relation

Try different assignments to find the specific constraint in the scope

How are the assignments decided???

Follow the same logic
- But the new example has to be an assignment only to the scope S found

$$e_S = \arg\max_{e_S \in \text{Sol}(C_L[S] \land B[S])} \sum_{c \in B} \left[ e_S \notin \text{sol}(c) \right] \cdot (1 - |\Gamma| \cdot O(c))$$

Diagram:

- **O(c)**
  - True: Aim to satisfy
  - False: Aim to violate
Do we have an oracle $O(c)$ to guide CA

The oracle $O(c)$ “classifies” a candidate as a problem constraint or not

---

It is a prediction problem

Use Machine Learning!!

---

Query Generation

Learn from user’s answer

---

Aim to violate

---

Aim to satisfy

---

No

---

True

---

Aim to violate

---

Aim to satisfy

---
Do we have an oracle $O(c)$ to guide CA

The oracle $O(c)$ “classifies” a candidate as a problem constraint or not

- **No** → **Aim to violate**
- **True** → **Aim to satisfy**

It is a prediction problem

Use Machine Learning!!

Predictions for constraints not for variable assignments!!

ML → Query Generation → Learn from user’s answer → Update Constraint dataset
Using Machine Learning for the prediction

**Dataset:** Constraint features and class (True or False)
- Constructing during the acquisition process
- Constraints that we know are part of the problem or not
- When a constraint is learned add a positive instance
- When a constraint is removed from B *add a negative instance*
- Use both relation and scope features

<table>
<thead>
<tr>
<th>Relation-based features</th>
<th>Scope-based features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation name (string)</td>
<td>Dim[i] same_val (Bool)</td>
</tr>
<tr>
<td>Has constant (Bool)</td>
<td>Dim[i] avg (float)</td>
</tr>
<tr>
<td>Constant value (int)</td>
<td>Dim[i] distance (int)</td>
</tr>
<tr>
<td>Arity (int)</td>
<td>…</td>
</tr>
</tbody>
</table>
Using Machine Learning for the prediction

**Dataset:** Constraint features and class (True or False)
- Constructing during the acquisition process
- Constraints that we know are part of the problem or not
- When a constraint is learned add a positive instance
- When a constraint is removed from \( B \) add a negative instance
- Use both relation and scope features

Example for constraint \( x_{1,1} \neq x_{1,2} \) in Sudoku

<table>
<thead>
<tr>
<th>Relation-based features</th>
<th>Scope-based features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation name (string)</td>
<td>Dim[i] same_val (Bool)</td>
</tr>
<tr>
<td>Has constant (Bool)</td>
<td>Dim[i] avg (float)</td>
</tr>
<tr>
<td>Constant value (int)</td>
<td>Dim[i] distance (int)</td>
</tr>
<tr>
<td>Arity (int)</td>
<td></td>
</tr>
</tbody>
</table>
Using Machine Learning in the Oracle

The oracle \(O(c)\) “classifies” a candidate as a problem constraint or not.

**Use of any classification technique to simulate the Oracle**

\[ O(c) = \text{Class}(c) \]

**Use of probabilities?**

\[ O(c) = \frac{1}{\log(|Y|)} \leq P(c) \]

**Minimize the expected number of queries**

\(|Y|\): size of the example
\(\log(|Y|)\): number of queries for each constraint when not guided
\(\frac{1}{\log(|Y|)}\): Percentage of queries resulting on a constraint learnt
Using Machine Learning in the Oracle

The oracle $O(c)$ “classifies” a candidate as a problem constraint or not

Use of any classification technique to simulate the Oracle

\[ O(c) = \text{Class}(c) \]

Use of probabilities?

\[ O(c) = \frac{1}{\log(|Y|)} \leq P(c) \]

Minimize the expected number of queries

- Using predicted probabilities instead of class works way better!!
- Assuming that a candidate is a constraint of the problem even if the probability is less than 50% (but above the threshold)
- Threshold defined to minimize number of queries!

\[ |Y|: \text{size of the example} \]
\[ \log(|Y|): \text{number of queries for each constraint when not guided} \]
\[ \frac{1}{\log(|Y|)}: \text{Percentage of queries resulting on a constraint learnt} \]
Open challenges
Challenges

Number of queries

- Number of queries needed to converge is still large.
- Use also more expressive types of queries
- Generalize!
Challenges

Specific classes of constraints

- **Global constraints**: Exploding the set of candidate constraints
- **Linear inequalities** with constants: Need to consider all possible constants -> Exploding the set of candidate constraints

- COUNT
- ALLDIFFERENT
- CUMULATIVE
- SUM
- CIRCUIT

\[ x_1 + 5 < x_2 \]
\[ |x_1 + 12| > x_4 \]
\[ x_1 - x_2 \neq 238 \]
Challenges

Noisy data

• Unlike in machine learning, most constraint acquisition techniques still assume the user always (knows how to) answer correctly
• Tighter integration with modern machine learning techniques
Conclusions

We discussed:
- How to guide constraint acquisition using probabilities for the candidates
- Guiding all layers of constraint acquisition
- How to use Machine Learning for guiding
- Open challenges
Acquire the model from text description!

Recent advances in LLMs and NL4OPT

Can we use this technology to acquire models directly from text description??
Holy grail 2.0

Problem description

- NER
- REL
- Formulate
- Translate

Verify solution
- Refine Model
- Compile and run
- Debug Code

MUS Extraction
I have only 10 hours to spend in Leuven. So I can only see a small number of attractions in the city. Visiting each attraction takes different amounts of time independent of the order in which it is visited, and they will give me different levels of happiness. Is it possible to come up with a list of attractions to visit to get maximum happiness in these 10 hours?

---

Model in CPMpy:

```python
from cpmpy import *

# Parameters
n = 5
T = [2, 3, 1, 4, 2]
H = [5, 7, 3, 8, 4]
max_time = 10

# Variables
x = boolvar(shape=n)

# Constraints
constraints = [
    # Total time spent visiting the attractions cannot exceed max_time
    sum(T[i]*x[i] for i in range(n)) <= max_time,
    # We can only visit each attraction at most once
    sum(x[i] for i in range(n)) <= 1
]

# Objective function
objective = maximize(sum(H[i]*x[i] for i in range(n)))

# Solve the problem
model = Model(constraints + [objective])
stats = model.solve()
```

---

Parameters:

- 'n': the number of attractions
- 'T': a list of 'n' integers representing the time it takes to visit each attraction
- 'H': a list of 'n' integers representing the happiness level of each attraction
- 'max_time': the maximum time available to visit the attractions (in this case, 10 hours)

Variables:

- 'x': a list of 'n' binary variables, where 'x[i] = 1' if attraction 'i' is visited, and 'x[i] = 0' otherwise.

Constraints:

- The total time spent visiting the attractions cannot exceed 'max_time': \( \sum_{i=1}^{n} t_i x_i \leq max_time \)
- We can only visit each attraction at most once: \( \sum_{i=1}^{n} x_i \leq 1 \)

Objective function:

- Maximize the total happiness level of the visited attractions: \( \max \sum_{i=1}^{n} h_i x_i \)
Conclusions

We discussed:
- Background on constraint acquisition
- Passive and interactive constraint acquisition
- Approaches for passive acquisition
- How interactive constraint acquisition works
- Using Machine Learning to guide interactive constraint acquisition
- Open challenges
Thank you for your attention