Constraint Acquisition

Nadjib Lazaar

U. Montpellier, France
LIRMM - COCONUT team

24-11-17
CAVIAR - Jussieu

lazaar@lirmm.fr
Motivations

CSP

Problem

solution

Computer
Motivations

Question: How does the user write down the constraints of a problem?

Limitations: modelling constraint networks require a fair expertise

[Freuder99, Frisch et al.05, Smith06]

Need: Simple way to build constraint model ➔ Modeller-assistant
Motivations

**Question:** How does the user write down the constraints of a problem?
**Limitations:** modelling constraint networks require a fair expertise

[Freuder99, Frisch et al.05, Smith06]

**Need:** Simple way to build constraint model ➔ Modeller-assistant
**How:** In a Machine Learning way (passive/active, offline/online, by reinforcement...)

Learning process

- solutions
- non-solutions

CSP

Problem

solution
Motivations

• **Question:** How does the user write down the constraints of a problem?
• **Limitations:** modelling constraint networks require a fair expertise
  [Freuder99, Frisch et al.05, Smith06]

• **Need:** Simple way to build constraint model ➔ Modeller-assistant
• **How:** In a Machine Learning way (passive/active, offline/online, by reinforcement...)
Let \( X = x_1, \ldots, x_n \) a set of attributes of domains \( D = D_1, \ldots, D_n \)

A concept is a Boolean function \( f: X \rightarrow \{0, 1\} \)

- \( f(x_i) = 0 \Rightarrow x_i \) is a negative instance
- \( f(x_j) = 1 \Rightarrow x_j \) is a positive instance

Given a set of hypothesis \( H \), any subset of \( H \) represents a version space

A concept to learn is the set of positive instances that can be represented by a version space
Version Space Learning (Overview) [Mitchell82]

Concept to learn:
\[ f : \ (\forall x_i \in E^+ : f(x_i) = 1) \land (\forall x_i \in E^- : f(x_i) = 0) \]

\[ f \equiv h_2 \land h_6 \land h_9 \]
Constraint Acquisition as Version Space Learning

Concept Learning

Constraint Programming:

Constraint Acquisition
Constraint Acquisition Problem

- Inputs:
  - (X,D): Vocabulary
  - \(\Gamma\): Constraint language
  - \(B\): Bias (constraints/hypothesis)
  - \(C_T\): Target Network (concept to learn)
  - \((E^+,E^-)\): training set

- Output:
  - \(C_L\): Learned network such that:

Convergence Pb:

\[
(C_L \subset B) \land (\forall e_i \in E^+ : e_i \in sol(C_L)) \land (\forall e_i \in E^- : e_i \notin sol(C_L))
\]

\textbf{coNP-complete} [Constraint Acquisition, AIJ17]
Example

- $\Gamma = \{<, =\}$
- $B = \{x_i < x_j, x_i = x_j, \forall i, j\}$
- $C_T = \{x_1 = x_3, x_1 < x_2\}$
- $C_L = \{x_1 = x_3, x_3 < x_2\}$
State of the art

- **Matchmaker agents** [Freuder and Wallace wAAAI97]

- **CONACQ**
  - SAT-Based constraint acquisition
  - Bidirectional search using Membership queries
  - Conacq1.0 (passive learning) [Bessiere et al. ECML05]
  - Conacq2.0 (active learning) [Bessiere et al. IJCAI07]

- Argument based CONACQ [Friedrich et al.09]

\[
\mathcal{K} = (\neg x_1 \land \neg x_2 \land \neg x_3) \land (x_4 \lor x_5 \lor x_6 \lor x_7) \ldots
\]

- No-learnability using Membership queries [Constraint Acquisition, AIJ17]
State of the art

- **ModelSeeker** [Beldiceanu and Simonis, CP11’12]
  - A passive learning
  - Based on global constraint catalogue (≈1000)
  - Bottom-up search
QUACQ: Quick Acquisition

- **QUACQ** [Bessiere et al. IJCAI13]
  - Active learning approach
  - Bidirectional search
  - But it can be top-down search only if no positive example
  - Based on partial queries to elucidate the scope of the constraint to learn
Membership Queries

ask(2, 8, 4, 2, 6, 5, 1, 6)
Partial Queries

ask(2, 8, 4, 2, 6, 5, 1, 6) = No
Partial Queries

ask(2, 8, 4, 2, -, -, -, -) = No
Partial Queries

\[ \text{ask}(2, 8, -, -, -, -, -, -, -) = \text{Yes} \]
Partial Queries

ask(2, 8, 4, -, -, -, -, -, -) = No
QUACQ: Quick Acquisition

ask(e) -> Gen-query

yes -> reduce(B)
QUACQ: Quick Acquisition

- yes: reduce(B)
- ask(e): Gen-query
- No: partial-ask(e)
- partial-ask(e): FindScope
QUACQ: Quick Acquisition

Yes

ask(e)

reduce(B)

Gen-query

No

partial-ask(e)

FindScope

FindC

scope
QUACQ: Quick Acquisition

QUACQ

reduce(B)

Gen-query

Update(C_L)

FindScope

FindC

yes

ask(e)

No

partial-ask(e)
QUACQ: Quick Acquisition

- yes: reduce(B)
- ask(e): Gen-query
- B=∅: Update(C_L)
- partial-ask(e): FindScope, FindC

QL

---

**QUACQ**

- reduce(B)
- Gen-query
- B=∅: Update(C_L)
- FindScope, FindC

QL
Algorithm 1: QUACQ: Acquiring a constraint network $C_T$ with partial queries

1. $C_L \leftarrow \emptyset$
2. while true do
3.   if sol($C_L$) = $\emptyset$ then return “collapse”;
4.   choose $e$ in $D^X$ accepted by $C_L$ and rejected by $B$;
5.   if $e = \text{nil}$ then return “convergence on $C_L$”;
6.   if $ASK(e) = \text{yes}$ then $B \leftarrow B \setminus \kappa_B(e)$;
7.   else
8.     $c \leftarrow \text{FindC}(e, \text{FindScope}(e, \emptyset, X, \text{false}))$;
9.     if $c = \text{nil}$ then return “collapse”;
10.    else $C_L \leftarrow C_L \cup \{c\}$;
The number of queries required to find the target concept is in:

\[ O(|C_T| \cdot (\log |X| + |\Gamma|)) \]

The number of queries required to converge is in:

\[ O(|B|) \]
Some Results

- **Sudoku**

A target network on 81 variables with 810 constraints

Bias of 19440 binary constraints

| $|C_L|$ | $\#q$ | $\#q_c$ | $\bar{q}$ | time |
|------|-------|---------|----------|------|
| **Sudoku 9 × 9** | 810   | 8645    | 821      | 20.58| 0.16 |
Experiments

- Zebra puzzle
- QUACQ behavior on different bias sizes
Conclusions

- QUACQ: new constraint acquisition approach based on partial queries
  - Active learning approach
  - Learning a constraint in a log scale of #queries
  - Queries are often much shorter than membership ones
  - Can follow a top-down search to learn a constraint network
Constraint Acquisition

Nadjib Lazaar

U. Montpellier, France
LIRMM - COCONUT team

24-11-17
Jussieu
In practice?

Limitation:
• Hard to put in practice:
  • QUACQ needs more than 8000 queries to learn the Sudoku model

Need:
• Reduce the dialogue with the user to make constraint acquisition more efficient in practice

How:
• Eliciting more information on why a complete instantiation is classified as negative by the user
QUACQ: Quick Acquisition

- **Gen-query**: 
  - **reduce(B)**
  - **B=∅**
  - **Update(C_L)**
    - **One constraint**
    - **FindC**
    - **One scope**
    - **FindScope**
    - **partial-ask(e)**
    - **ask(e)**

- **Yes**
  - **C_L**
MULTIACQ: Multiple Acquisition [IJCAI-W15]
MULTIACQ: Multiple Acquisition

Q: Why the user said No?

- FindScope function
- QuickXplain like function [Junker 04]
- Returns one scope (explanation)

\[ \text{FindScope}(e) = (X3, x5) \]

\[ \#\text{learned\_constraint} = 1 \]

- FindAllScopes function
- CAMUS like function [Liffiton et al. 07]
- Returns all Minimal No Scopes (MUS in SAT)

\[ \text{FindAllScope}(e) = \{(X1, x3), (X1, x4), (X3, x5), (X5, x6), (X5, x8), (X4, x8)\} \]

\[ \#\text{learned\_constraint} = 6 \]
Given a negative example e, an MNS is a subset of variables $U \subseteq X$ such that:

$$\text{ASK}(e_U) = no \text{ and } \forall x_i \in U : \text{ASK}(e_{U \setminus x_i}) = yes$$

- **Lemmas:**
  
  ![Diagram]

  \[\text{ASK}(Y') = \text{NO} \]
  \[\text{ASK}(Y) = \text{YES} \]
  \[\text{ASK}(Y'') = \text{YES} \]

  **LEMMMA 1**

  **LEMMMA 2**
FindAllScopes function

**INPUT:** example $e$ on ($X_1$, $X_2$, $X_3$, $X_4$) variables

**OUTPUT:** $MNS = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** $\text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** $\text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>Recursive calls</th>
<th>ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example e on (X1, X2, X3, X4) variables

**OUTPUT:** MNS = (X1, X2), (X1, X3), (X2, X3, X4)

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>
**FindAllScopes** function

**INPUT:** example e on (X1, X2, X3, X4) variables  
**OUTPUT:** MNS = (X1, X2), (X1, X3), (X2, X3, X4)

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** $\text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

# Recursive calls | # ask
--- | ---
5 | 5

is an MNS!
FindAllScopes function

**INPUT:** example e on (X1, X2, X3, X4) variables

**OUTPUT:** MNS = (X1, X2), (X1, X3), (X2, X3, X4)

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** $\text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** $\text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>
**FindAllScopes function**

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** $\text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>7</td>
</tr>
</tbody>
</table>
**FindAllScopes function**

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** $\text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** MNS = $(X_1, X_2)$, $(X_1, X_3)$, $(X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8</td>
</tr>
</tbody>
</table>

**LEMMA 2**
**FindAllScopes function**

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables  
**OUTPUT:** $\text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>8</td>
</tr>
</tbody>
</table>

Diagonalized decision tree:

- $X_1 X_2 X_3 X_4$
- $X_1 X_2 X_3$
- $X_1 X_2 X_4$
- $X_1 X_2$
- $X_1 X_3$
- $X_2 X_3$
- $X_2 X_4$
- $X_1$
- $X_2$
- $\text{LEMMA 2}$
FindAllScopes function

**INPUT:** example e on (X1, X2, X3, X4) variables

**OUTPUT:** MNS = (X1, X2), (X1, X3), (X2, X3, X4)

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>10</td>
</tr>
</tbody>
</table>
**FindAllScopes function**

**INPUT:** example \( e \) on \((X_1, X_2, X_3, X_4)\) variables

**OUTPUT:** \( \text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4) \)

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>10</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example e on (X1, X2, X3, X4) variables

**OUTPUT:** MNS = (X1, X2), (X1, X3), (X2, X3, X4)

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>10</td>
</tr>
</tbody>
</table>

**LEMMA 1**
FindAllScopes function

**INPUT:** example $e$ on $(X1, X2, X3, X4)$ variables

**OUTPUT:** $\text{MNS} = (X1, X2), (X1, X3), (X2, X3, X4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>11</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** $MNS = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>12</td>
</tr>
</tbody>
</table>
FindAllScopes function

**INPUT:** example e on (X1, X2, X3, X4) variables

**OUTPUT:**  MNS = (X1, X2), (X1, X3), (X2, X3, X4)

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>12</td>
</tr>
</tbody>
</table>

**LEMMA 1**
**FindAllScopes function**

**INPUT:** example $e$ on $(X_1, X_2, X_3, X_4)$ variables

**OUTPUT:** $\text{MNS} = (X_1, X_2), (X_1, X_3), (X_2, X_3, X_4)$

<table>
<thead>
<tr>
<th>#Recursive calls</th>
<th>#ask</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>12</td>
</tr>
</tbody>
</table>
Some Results

Sudoku

A target network on 81 variables with 810 constraints

Bias of 19440 binary constraints

| $|C_L|$ | $\# q$ | $\# q_c$ | $\bar{q}$ | time |
|---|---|---|---|---|
| **Sudoku 9 \times 9** | 810 | 8645 | 821 | 20.58 | 0.16 |

MultiAcq $\Rightarrow$ **3821** (gain 60%)
Conclusions

- QUACQ focuses on the scope of one constraint each time we give it a negative example.

- MULTIACQ with its FindAllScopes function aims to report all minimal scopes of violated constraints.

- The results show:
  - MULTIACQ dramatically improves the basic version of QUACQ in terms of #queries.
  - The queries are often much shorter.
  - MULTIACQ can be time-consuming.

Still time left?
Constraint Acquisition

Nadjib Lazaar

U. Montpellier, France
LIRMM - COCONUT team

24-05-17
Nantes
In practice?

**Limitation:**
- Hard to put in practice:
  - QUACQ needs more than 8000 queries to learn the Sudoku model

**Need:**
- Reduce the dialogue with the user to make constraint acquisition more efficient in practice

**How:**
- Eliciting more information on why a complete instantiation is classified as negative by the user  ➔ MULTIACQ [IJCAI16]
- Eliciting more information by asking complex queries to the user [ECAI14]
A type is a subset of variables defined by the user as having a common property.

Example (School Timetabling Problem)

- **Teachers**
  - $T_1$, $T_2$, $T_3$, $T_4$, $T_5$, $T_6$, $T_7$, $T_8$

- **Students**

- **Rooms**

- **Courses**
A type is a subset of variables defined by the user as having a common property.

Example (School Timetabling Problem)

Can C1 be generalized to all Teachers, Rooms and Courses?
Let \( c(x, y) \) a learned constraint and \( X, Y \) are types of \( x, y \):

Generalization Query: \( \text{AskGen}((X, Y), c) \)

The user says yes iff the constraint \( c \) holds on all possible scope

\[
(x_i, y_i) \in (X, Y)
\]

Properties

\[
\begin{array}{c}
X' \\
\downarrow \\
X \\
\downarrow \\
X''
\end{array}
\quad
\begin{array}{c}
Y' \\
\downarrow \\
Y \\
\downarrow \\
Y''
\end{array}
\]

- super-types
- types
- Sub-types
Let \( c(x, y) \) a learned constraint and \( X, Y \) are types of \( x, y \):

Generalization Query: \( \text{AskGen}((X, Y), c) \)

The user says \textbf{yes} iff the constraint \( c \) holds on all possible scope

\[
(x_i, y_i) \in (X, Y)
\]

Properties

\[
\begin{align*}
\text{AskGen}(X', Y', c) &= \text{YES} \\
\text{AskGen}(X'', Y'', c) &= \text{YES}
\end{align*}
\]
Let $c(x, y)$ a learned constraint and $X, Y$ are types of $x, y$:

- **Generalization Query:** $AskGen((X, Y), c)$

The user says yes iff the constraint $c$ holds on all possible scope

$$(x_i, y_i) \in (X, Y)$$

- **Properties**

  \[
  \begin{align*}
  &AskGen( X', Y', c) = NO \\
  &AskGen( X, Y, c) = NO \\
  &AskGen( X'', Y'', c) = NO
  \end{align*}
  \]
**GENACQ**

**Inputs**
- A learned constraint
- Combination of possible types (i.e., table)

**Output**
- Set of constraints

![Tree Diagram]

- Zebra Problem
- Variables: $X_1$, $X_2$, $X_3$, $X_4$, $X_5$
- Types: drink, cigaret, color, pet, nationality
**Inputs**
- A learned constraint
- Combination of possible types (i.e., table)

**Output**
- Set of constraints

---

**Zebra Problem**

- **Inputs**
  - Learned constraint: \( x_2 \neq x_5 \)
  - Table:
    - #q = 0
    - askGen

- **Diagram**
  - Zebra Problem
  - drink, cigaret, color, pet, nationality
  - \( x_1, x_2, x_3, x_4, x_5 \)
  - \( \neq \)
**Inputs**
- A learned constraint
- Combination of possible types (i.e., table)

**Output**
- Set of constraints

**Zebra Problem**

**INPUTS**

- Learned constraint: \( x_2 \neq x_5 \)
- Table:

<table>
<thead>
<tr>
<th>( #q = 1 )</th>
<th>( x_2 )</th>
<th>( x_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 )</td>
<td>color</td>
<td>✓</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>X</td>
<td>x</td>
</tr>
<tr>
<td>color</td>
<td>( x_5 )</td>
<td></td>
</tr>
<tr>
<td>color</td>
<td>color</td>
<td></td>
</tr>
<tr>
<td>color</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>( x_5 )</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>color</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>
Inputs

- A learned constraint
- Combination of possible types (i.e., table)

Output

- Set of constraints

INPUTS

- Learned constraint: $x_2 \neq x_5$
- Table:

<table>
<thead>
<tr>
<th>#q = 2</th>
<th>$x_2$</th>
<th>$x_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_2$</td>
<td>color</td>
<td>✔</td>
</tr>
<tr>
<td>color</td>
<td>$x_5$</td>
<td></td>
</tr>
<tr>
<td>color</td>
<td>color</td>
<td></td>
</tr>
<tr>
<td>color</td>
<td>$x_5$</td>
<td></td>
</tr>
<tr>
<td>$x$</td>
<td>$x_5$</td>
<td></td>
</tr>
<tr>
<td>$x$</td>
<td>color</td>
<td></td>
</tr>
<tr>
<td>$x$</td>
<td>$x$</td>
<td></td>
</tr>
</tbody>
</table>

Zebra Problem
**Inputs**
- A learned constraint
- Combination of possible types (i.e., table)

**Output**
- Set of constraints

### Zebra Problem

- **INPUTS**
  - Learned constraint: $x_2 \neq x_5$
  - Table:
    - $#q = 3$
    - | $x_2$ | $x_5$ |
      |-------|-------|
      | $x_2$ | color |
      | $x_2$  | X     |
      | color  | $x_5$ |
      | color  | color |
      | color  | X     |
      | X      | $x_5$ |
      | X      | color |
      | X      | X     |
**Inputs**
- A learned constraint
- Combination of possible types (i.e., table)

**Output**
- Set of constraints

---

**GENACQ**

**Zebra Problem**

**INPUTS**

- Learned constraint: \( x_2 \neq x_5 \)
- Table:

<table>
<thead>
<tr>
<th>#q = 4</th>
<th>( x_2 )</th>
<th>( x_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_2 )</td>
<td>color</td>
<td>✔️</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( x )</td>
<td>✗️</td>
</tr>
<tr>
<td>color</td>
<td>( x_5 )</td>
<td>✔️</td>
</tr>
<tr>
<td>color</td>
<td>color</td>
<td>✔️</td>
</tr>
<tr>
<td>color</td>
<td>( x )</td>
<td>✗️</td>
</tr>
<tr>
<td>( x )</td>
<td>( x_5 )</td>
<td>✗️</td>
</tr>
<tr>
<td>( x )</td>
<td>color</td>
<td>✗️</td>
</tr>
<tr>
<td>( x )</td>
<td>( x )</td>
<td>✗️</td>
</tr>
</tbody>
</table>
Inputs
- A learned constraint
- Combination of possible types (i.e., table)

Output
- Set of constraints

Zebra Problem

Learned constraint: \( x_2 \neq x_5 \)

Table:

<table>
<thead>
<tr>
<th>( x_2 )</th>
<th>( x_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( x_2 )</td>
</tr>
<tr>
<td>color</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>color</td>
<td>color</td>
</tr>
<tr>
<td>( x_2 )</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>X</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>X</td>
<td>color</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

\#q = 5
**Inputs**
- A learned constraint
- Combination of possible types (i.e., table)

**Output**
- Set of constraints

Zebra Problem

- **INPUTS**
  - Learned constraint: $x_2 \neq x_5$
  - Table
- **OUTPUT**
  - 9 constraints:

```
#q = 5
```
## Results

<table>
<thead>
<tr>
<th></th>
<th>QUACQ</th>
<th>G-QUACQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Ask</td>
<td>#Ask</td>
</tr>
<tr>
<td>Zebra</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sudoku</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Latin square</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RFLAP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Purdey</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

50%  
95%  
84%  
88%  
34%
Conclusions

- Generalization query based on types of variables
- GENACQ algorithm
- Several heuristics and strategies to select the good candidate generalization query
- Can be plugged in any active constraint acquisition system
- Results by plugging GENACQ in the QUACQ acquisition System

Next step

- Detecting Types of Variables for Generalization [ICTAI15]
Constraint Acquisition

Nadjib Lazaar

U. Montpellier, France
LIRMM - COCONUT team

24-11-17
CAVIAR - Jussieu
Motivations

Types

+ a constraint

**Examples**

Type T1

**Set of generalized constraints**
Motivations

Types

+ a constraint

**EXAMPLE**

Type T1

**Limitation:** Hidden types

**The need:** Detecting types before generalization

**How:** Reasoning on and mining the partial constraint graph
Detecting types of variables

- Variables of the same type are often tightly connected with similar constraints
- Variables of different types are connected in a weaker way
- Detecting sub-graphs arose in the study of networks:
  - Social networks [Wasserman and Faust, 94]
  - Biochemical networks [Ito et al. 01]

- Detecting community structures (types in our context)
Mine&Ask algorithm
Mine&Ask algorithm

Target Network

Current Network

Projection Network

Diagram showing nodes X1 to X11 with connections R1.
Mine&Ask algorithm

Target Network

Current Network

Mining step

T1

T2

T3
Mine&Ask algorithm

Target Network

Current Network

Generalization step

AskGen(T1,R1) = YES
AskGen(T2,R1) = YES
AskGen(T3,R1) = NO

3 questions ➔ 9 constraints
Mining the graph of learned constraints

- Modularity optimization for communities detection [Newman and Girvan, 04]

\[ Q = \sum_{i,j} \left[ \frac{A_{i,j}}{2m} - \frac{\text{deg}(i) \times \text{deg}(j)}{4m^2} \right] (c(i) = c(j)) \]

A high value of modularity Q correspond to a good partition
Mining the graph of learned constraints

- Modularity optimization for communities detection  [Newman and Girvan, 04]
- Edge betweenness centrality  [Girvan and Newman 02]

\[ B(e) = \sum_{i,j} \frac{\sigma_{ij}(e)}{\sigma_{ij}} \]

#paths through the edge e

# shortest paths between i and j
Mining the graph of learned constraints

- Modularity optimization for communities detection  [Newman and Girvan, 04]
- Edge betweenness centrality  [Girvan and Newman 02]
- Quasi-cliques detection based on Bron Kerbosch’s algorithm  
  [Bron and Kerbosch 73]
Experimental evaluation

- Mine&Ask is implemented and plugged in QUACQ system, leading to M-QUACQ version.
- M-QUACQ is compared to the basic version of QUACQ and the G-QUACQ version including GENACQ Algorithm.
- We evaluate the three different extracting types methods:
  - Modularity
  - Betweenness
  - $\gamma$-clique
### Some Results

<table>
<thead>
<tr>
<th>Strategies</th>
<th>QUACQ</th>
<th>G-QUACQ</th>
<th>M-QUACQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Ask</td>
<td>#Ask</td>
<td>#Ask</td>
</tr>
<tr>
<td>Latin Square</td>
<td>#AskGen</td>
<td>#AskGen</td>
<td>#no #yes</td>
</tr>
<tr>
<td>modularity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>2058</td>
<td>129</td>
<td>987</td>
</tr>
<tr>
<td></td>
<td></td>
<td>68</td>
<td>61</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>betweenness</td>
<td>1674</td>
<td>22</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1172</td>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>γ-clique</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PlaceNumPuzzle</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>modularity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3746</td>
<td>351</td>
<td>627</td>
</tr>
<tr>
<td></td>
<td></td>
<td>39</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>betweenness</td>
<td>655</td>
<td>33</td>
<td>31</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>688</td>
<td>33</td>
<td>31</td>
</tr>
<tr>
<td>γ-clique</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Murder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>modularity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>483</td>
<td>230</td>
<td>272</td>
</tr>
<tr>
<td></td>
<td></td>
<td>55</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>betweenness</td>
<td>272</td>
<td>12</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>γ-clique</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>342</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
### Some Results

<table>
<thead>
<tr>
<th>Strategies</th>
<th>QUACQ</th>
<th>G-QUACQ</th>
<th>M-QUACQ</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>#Ask</td>
<td>#Ask</td>
<td>#AskGen</td>
</tr>
</tbody>
</table>

#### Zebra

- **modularity**: 694
- **betweenness**: 257
- **γ-clique**: 67

<table>
<thead>
<tr>
<th></th>
<th>#Ask</th>
<th>#AskGen</th>
<th>#no</th>
<th>#yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUACQ</td>
<td>410</td>
<td>14</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>G-QUACQ</td>
<td>410</td>
<td>14</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>M-QUACQ</td>
<td>410</td>
<td>14</td>
<td>0</td>
<td>14</td>
</tr>
</tbody>
</table>

**40%**

#### Purdey

- **modularity**: 205
- **betweenness**: 93
- **γ-clique**: 39

<table>
<thead>
<tr>
<th></th>
<th>#Ask</th>
<th>#AskGen</th>
<th>#no</th>
<th>#yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUACQ</td>
<td>140</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>G-QUACQ</td>
<td>140</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>M-QUACQ</td>
<td>140</td>
<td>8</td>
<td>0</td>
<td>8</td>
</tr>
</tbody>
</table>

**27%**

#### Sudoku

- **modularity**: 9593
- **betweenness**: 260
- **γ-clique**: 166

<table>
<thead>
<tr>
<th></th>
<th>#Ask</th>
<th>#AskGen</th>
<th>#no</th>
<th>#yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUACQ</td>
<td>7963</td>
<td>57</td>
<td>20</td>
<td>37</td>
</tr>
<tr>
<td>G-QUACQ</td>
<td>8960</td>
<td>50</td>
<td>18</td>
<td>32</td>
</tr>
<tr>
<td>M-QUACQ</td>
<td>9461</td>
<td>117</td>
<td>104</td>
<td>13</td>
</tr>
</tbody>
</table>

**16%**
Conclusions

- Mine&Ask algorithm able to mine partial graphs of constraints and to generalize constraints on potential types
- Used when no knowledge on variable types is provided
- Extracting potential types using:
  - Modularity, betweenness, $\gamma$-clique
- M-QUACQ = Mine&Ask + QUACQ
  - Next?
    - More prediction and mining on partial constraint network for acquisition [IJCAI16]
    - Study on a time-bounded query generation [Ongoing work]
Constraint Acquisition

Nadjib Lazaar

U. Montpellier, France
LIRMM - COCONUT team

24-11-17
CAVIAR - Jussieu