Motivation

Constraint programming

- Declarative paradigm to problem solving
- Specifying problems as variables, domains, and constraints
- A versatile tool for combinatorial and optimisation problems (resource allocation, job-shop scheduling etc)
Motivation

Constraint programming

- Declarative paradigm to problem solving
- Specifying problems as *variables*, *domains*, and *constraints*
- A *versatile* tool for combinatorial and optimisation problems (resource allocation, job-shop scheduling etc)

Self-organizing, adaptive Systems:

- Claim to achieve higher robustness by local rules that lead to re-structuring
- *Fault-tolerance* by added redundancies
- Typically require a feedback loop along with reconfigurations (e.g., MAPE cycle, Observer-Controller, . . .) Ramirez and Cheng (2010)
- Intended system behaviour can be described in terms of first-order predicates over a system’s variables
- → Use constraints from *specification* for the implementation of *reconfiguration*
Two worlds

Constraints for Specification and Analysis

- Derived from Requirements Engineering processes
- Formal description in, e.g., OCL
- Models available for formal analysis
Two worlds

Constraints for Specification and Analysis

- Derived from Requirements Engineering processes
- Formal description in, e.g., OCL
- Models available for formal analysis

Constraints for Engineering and Optimization

- Decision making for reconfigurations
- Self-optimization using constraint models of agents that are combined automatically
A first example
A bit more formally

The **Restore-Invariant-Approach**

Nafz et al. (2013)
General Idea

1. Notice change in the system that violates the invariant
   1. Assume, a driller breaks $\rightarrow \text{drill} \not\in \text{availableRoles}$
   2. Assume further, this robot is assigned to drill $\text{assignedRole} = \text{drill}$
   3. e.g., $\text{assignedRole} \not\in \text{availableRoles}$

2. Search for new assignment using constraint programming

3. Restore the invariant

4. Provable correctness as long as the system is within the corridor (using, e.g., theorem provers such as KIV or model checking tools, hybrid automata, . . .)
A second example . . .

Unit Commitment in Power Management
Time-dependent resource allocation

Variation of general resource allocation

- Demand is given for a time frame $\mathcal{T}$
- Agents are subject to constraints such as minimal or maximal rates and inertia
- Therefore, proactive allocation necessary!
- Limited rates of change between consecutive time steps
Time-dependent resource allocation

Variation of general resource allocation

- Demand is given for a time frame $\mathcal{T}$
- Agents are subject to constraints such as minimal or maximal rates and *inertia*
- Therefore, *proactive* allocation necessary!
- Limited rates of change between consecutive time steps

$$\text{minimise} \quad \sum_{t \in \mathcal{T}} |D_t - \sum_{a \in A} P^a_t|$$

subject to

$$\exists [x, y] \in L^a : x \leq P^a_t \leq y, \forall a \in A, t \in \mathcal{T}$$

$$P^a_{t+1} \geq c_{\min}^a (\Sigma^a_t), \forall a \in A, t \in \mathcal{T}$$

$$P^a_{t+1} \leq c_{\max}^a (\Sigma^a_t), \forall a \in A, t \in \mathcal{T}$$
Time-dependent resource allocation

Variation of general resource allocation

- Demand is given for a time frame $\mathcal{T}$
- Agents are subject to constraints such as minimal or maximal rates and *inertia*
- Therefore, *proactive* allocation necessary!
- Limited rates of change between consecutive time steps

\[
\text{minimise } P \sum_{t \in \mathcal{T}} |D_t - \sum_{a \in \mathcal{A}} P^a_t|
\]

subject to

\[
\exists [x, y] \in L^a: x \leq P^a_t \leq y, \forall a \in \mathcal{A}, t \in \mathcal{T}
\]

\[
P^a_{t+1} \geq c^a_{\text{min}}(\Sigma_t^a), \forall a \in \mathcal{A}, t \in \mathcal{T}
\]

\[
P^a_{t+1} \leq c^a_{\text{max}}(\Sigma_t^a), \forall a \in \mathcal{A}, t \in \mathcal{T}
\]

For instance, if $P^a_{t+1} \leq P^a_t + 30$, then $\langle 50, 80, 110 \rangle \checkmark$ but $\langle 50, 90, 125 \rangle \times$
Roadmap

Within this realm

- A formalism to deal with over-constrained problems and specify *preferences*
- A framework to *synthesize* a set of constraint models into one optimization problem
- Algorithms to *abstract* a constraint model from a set of constraint models
- Learning-based techniques to *build* models
- Tools to support the *elicitation* of preferences
Constraint Relationships – An example

Schiendorfer et al. (2013); Knapp and Schiendorfer (2014)

- Variables: $X$, $Y$, $Z$
- Domains: $\{0, 1, 2\}$
- Constraints:
  - $c_1 : x + 1 = y$
  - $c_2 : z = y + 2$
  - $c_3 : x + y \leq 3$

- Not all three constraints can be satisfied simultaneously
- E.g. $c_2$ forces $z$ to be 2 and $y$ to be 0, conflicting with $c_1$
- We can choose between solutions satisfying $\{c_1, c_3\}$ or $\{c_2, c_3\}$
- How to settle this conflict?
Dealing with Heterogeneity

Table: Different cold and hot start-up times in h for power plant types Jarass and Obermair (2012); Mayer et al. (2013)

<table>
<thead>
<tr>
<th>Plant type</th>
<th>Cold start-up (down &gt; 48h)</th>
<th>Hot start-up (down &lt; 8h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black coal</td>
<td>4 – 5</td>
<td>2</td>
</tr>
<tr>
<td>Brown coal</td>
<td>6 – 8</td>
<td>2 – 4</td>
</tr>
<tr>
<td>Gas turbine</td>
<td>0.5</td>
<td>0.25</td>
</tr>
<tr>
<td>Photothermal</td>
<td>4 – 5</td>
<td>2</td>
</tr>
</tbody>
</table>
Dealing with Heterogeneity

Table: Different cold and hot start-up times in h for power plant types Jarass and Obermair (2012); Mayer et al. (2013)

<table>
<thead>
<tr>
<th>Plant type</th>
<th>Cold start-up (down &gt; 48h)</th>
<th>Hot start-up (down &lt; 8h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Black coal</td>
<td>4 – 5</td>
<td>2</td>
</tr>
<tr>
<td>Brown coal</td>
<td>6 – 8</td>
<td>2 – 4</td>
</tr>
<tr>
<td>Gas turbine</td>
<td>0.5</td>
<td>0.25</td>
</tr>
<tr>
<td>Photothermal</td>
<td>4 – 5</td>
<td>2</td>
</tr>
</tbody>
</table>
Modelling variable start-up times

\[
\begin{align*}
[\text{cd} > 0] & \quad / \quad \text{cd} \leftarrow \text{cd} - 1 \\
[\text{sig} = 1] & \quad / \quad \text{cd} \leftarrow \text{ic} \\
[\text{sig} = -1] & \quad / \quad \text{on} \\
[\text{cd} = 0] & \quad / \quad \text{off}
\end{align*}
\]
Modelling variable start-up times

\[
\begin{align*}
[\text{cd} > 0] & \quad / \quad \text{cd} \leftarrow \text{cd} - 1 \\
[\text{sig} = 1] & \quad / \quad \text{cd} \leftarrow \text{ic}
\end{align*}
\]

How to manage both kinds of plants (and others)?
Modelling variable start-up times

\[ \text{sig} = 1 \] / \text{cd} \leftarrow \text{cd} - 1

\[ \text{sig} = -1 \] / \text{cd} \leftarrow \text{ic}

\[ \text{cd} > 0 \] / \text{cd} \leftarrow \text{cd} - 1

\[ \text{cd} = 0 \]

How to manage both kinds of plants (and others)?

→ Separation of shared or *public* interface variables (production) and *private* variables modelling internal aspects (e.g., for the current down-time, start-up duration at this particular down-time, etc.)

Schiendorfer et al. (2014)
Learning Models from Data

First Approach with Support Vector Data Description

\[ \gamma = 3.9 \quad "\text{good" fit} \]

Training data

Classified inside SVDD

\[ A_{a[t+1]} \]

\[ A_{a[t]} \]

\[ I_1 \]

\[ I_2 \]

\[ \Delta A_{a_{\text{max}}} \]
Preference Elicitation

• Claim: Use simplicity of constraint relationships to offer a more intuitive way of expressing preferences to users

• Show users solutions (in terms of violated constraints) and let them state “I like solution A more than B”.

• Find a constraint relationship that is consistent with a user’s preference decisions

• Tool: Abductive Logical Programming using CHR
Preference Elicitation

• Claim: Use simplicity of constraint relationships to offer a more intuitive way of expressing preferences to users

• Show users solutions (in terms of violated constraints) and let them state “I like solution A more than B”.

• Find a constraint relationship that is consistent with a user’s preference decisions

• → Tool: Abductive Logical Programming using CHR

?- worsens([a,b], [b,c]), worsens([b], [a]).
crbetter(c,b)
crbetter(a,b)
crbetter(c,a)
Future work

Theoretical

• Theoretical relationships of constraint relationships (and partial valuation structures) to other frameworks
• Abstraction algorithms for other application domains (e.g., energy consumers)
• Statistical model checking to obtain design-time guarantees

Practical

• Use automata-based models for simulation and engineering of constraint models
• Implementation of a reachability checker for hybrid automata
• Learning/abductive reasoning for preference elicitation
• Investigate alternative machine learning algorithms for model building
• Combine multiple paradigms: constraint-based models, automatas, machine learning models and optimization problem


References III
