Constraint Programming

Lecture 1: Introduction
In This Lecture

- Induction.
- Background.
- An Introduction by Example: The Crystal Maze Sudoku
- Course Overview.
Acknowledgement

• Course Slides due to:
  – Ian Miguel
  – Peter Nightingale
Background
What is this Constraints Business?

• A sub-field of **Artificial Intelligence** called variously:
  • Constraints,
  • Constraint programming,
  • Constraint satisfaction, ...
Who Cares About Constraints?

• IBM recently acquired Ilog, a leading vendor of constraint technology.
  – 1,000+ universities, 1,000+ commercial customers.
  – Clients such as: AT&T, Nissan, Visa, …

• CISCO acquired the ECLiPSe constraint logic programming system.

• The St Andrews Minion solver is used to schedule the CB1000 Nanoproteomic Analysis System.
Constraints: A Natural Means of Knowledge Representation

- $x + y = 30$
- Adjacent countries on map cannot be coloured same.
- The helicopter can carry one passenger.
- University timetabling:
  - No student can attend two lectures at once.
  - Lecture theatre A has a capacity of 100 students.
  - Art History lectures require a slide projector…
Solving Problems with Constraints

- An efficient means of finding solutions to combinatorial problems.
  - Planning, Scheduling, Design, Configuration, …
- Two phases:
  1. **Describe** the problem to be solved as a **constraint model**, a format suitable for input to a constraint solver.
  2. **Search** (automatically) for solutions to the model with a constraint solver.
A constraint model maps the features of a combinatorial problem onto the features of a constraint satisfaction problem (CSP).
The (finite-domain) Constraint Satisfaction Problem

• Given:
  1. A finite set of decision variables.
  2. For each decision variable, a finite domain of potential values.
  3. A finite set of constraints on the decision variables.

• Find:
  • An assignment of values to variables such that all constraints are satisfied.
1. Decision Variables

- A decision variable corresponds to a choice that must be made in solving a problem.
- In university timetabling we must decide, for example:
  - The time for each lecture.
  - The venue for each lecture.
  - The lecturer for each lecture.
  - ...
2. Domains

- **Values** in the domain of a decision variable correspond to the **options** for a particular choice.
- E.g. Decide lecture time.
  - Values in this domain: 9am, 10am, …, 5pm
- E.g. lecture venue.
  - Values in this domain: theatre A, theatre B, …
- A decision variable is **assigned a single** value from its domain.
  - Equivalently: the choice associated with that variable is made.
3. Constraints

- **scope**: subset of the decision variables a constraint involves.
- Of the possible combinations of assignments to the variables in its scope, a constraint specifies:
  - Which are allowed. Assignments that **satisfy** the constraint.
  - Which are disallowed. Assignments that **violate** the constraint.
  - I.e. can think of a constraint as a relation.
- E.g. if variables \( t_A, t_B \), represent time for lectures A, B, both taken by student S:
  - \( t_A \neq t_B \) (student S can’t be in two places at once!)
The CSP is input to a constraint solver, which produces a solution (or solutions).

The model is used to map the solution(s) back onto the original problem.
Constraint Solving

Typically interleaves 2 components:

1. Systematic **Search** through a space of partial assignments.
   - Extend an assignment to a subset of the variables incrementally.
   - Backtrack if establish that current partial assignment **cannot** be extended to a solution.

2. Constraint **Propagation**.
   - Deduction based on constraints, current domains.
   - Usually recorded as reductions in domains.
The Crystal Maze Puzzle

Thanks to Patrick Prosser.
Crystal Maze: Definition

• Given the network:

You have 5 Minutes!

Think about how you are solving it

• Place numbers 1 to 8 on nodes such that:
  • Each number appears exactly once.
  • Consecutive numbers do not appear on adjacent nodes.
Crystal Maze as a CSP

• Now we will see how the Crystal Maze Puzzle can be **modelled** as a CSP.
• Having done so, a constraint solver can make the inferences you just performed by hand automatically.
Crystal Maze as a CSP: Variables

- The nodes are the variables: $x_1$ to $x_8$
- NB The indexing scheme does not matter.
Crystal Maze as a CSP: Domains

- All variables have the same domain: \{1,2,3,4,5,6,7,8\}
Crystal Maze as a CSP: Constraints (1)

• All values used: AllDifferent($x_1$, $\ldots$, $x_8$).
Crystal Maze as a CSP: Constraints (2)

- No consecutive numbers on adjacent nodes:
  \[ |x_1 - x_2 | > 1 \]
  \[ |x_1 - x_3 | > 1 \ldots \]
Solving the Crystal Maze Puzzle

• Now you’ve had a go.
• Let’s see how constraint-based reasoning can lead us to a solution quickly.
Which nodes are hardest to label?
Which nodes are hardest to label?
Which are the least constraining values to use?

- Place numbers 1 to 8 on nodes.
Which are the least constraining values to use?
Exploiting Symmetry

• We do not need to consider:
• Eliminate many values for other nodes using constraint propagation:
  • Deduction from constraints and variable domains.
Constraint Propagation: 1 – 8 Appear Exactly Once

\{1,2,3,4,5,6,7,8\}
Constraint Propagation: 1 – 8 appear exactly once

\{2,3,4,5,6,7\}
Constraint Propagation: Adjacency

\{2,3,4,5,6,7\}
Constraint Propagation: Adjacency

\{3,4,5,6\}
Exploiting Symmetry

{3,4,5,6}

{3,4,5,6}
Constraint Propagation: 1 – 8 Appear Exactly Once

{3,4,5,6}  {1,2,3,4,5,6,7,8}
Constraint Propagation: 1 – 8 Appear Exactly Once

\{3,4,5,6\} \quad \{2,3,4,5,6,7\}

\begin{align*}
\text{?} & \quad \text{?} & \quad \text{?} \\
\text{?} & \quad 1 & \quad \text{8} \\
\text{?} & \quad \text{?} & \quad \text{?} \\
\text{?} & \quad \text{?} & \quad \text{?} \\
\end{align*}
Constraint Propagation: Adjacency

\{3,4,5,6\} \quad \{2,3,4,5,6,7\}
Constraint Propagation: Adjacency

\{3,4,5,6\}  \{3,4,5,6\}

\begin{itemize}
  \item \text{Node 1}
  \item \text{Node 2}
  \item \text{Node 3}
  \item \text{Node 4}
  \item \text{Node 5}
  \item \text{Node 6}
\end{itemize}
Exploiting Symmetry

\{3,4,5,6\} \{3,4,5,6\}

\begin{align*}
? & \quad ? \\
1 & \quad 8 \\
? & \quad ? \\
\{3,4,5,6\} & \{3,4,5,6\}
\end{align*}
Constraint Propagation: 1 – 8 Appear Exactly Once

{1,2,3,4,5,6,7,8} {1,2,3,4,5,6,7,8}

{3,4,5,6} {3,4,5,6}

? 1 8

? 3,4,5,6

? 3,4,5,6

? 3,4,5,6

? 3,4,5,6
Constraint Propagation: 1 – 8 Appear Exactly Once
Constraint Propagation: Adjacency
Constraint Propagation: Adjacency

![Graph with nodes and sets]

- Node 1:
  - Adjacencies: {3,4,5,6}
  - Possible values: {3,4,5,6}

- Node 8:
  - Adjacencies: {3,4,5,6}
  - Possible values: {3,4,5,6}

- Node 2, 3, 4, 5, 6, 7:
  - Adjacencies: {3,4,5,6,7}
  - Possible values: {2,3,4,5,6,7}

- Node ?:
  - Adjacencies: {3,4,5,6}
  - Possible values: {3,4,5,6}
Constraint Propagation: 1 – 8 Appear Exactly Once

{3,4,5,6}  {3,4,5,6}

{3,4,5,6,7}  {2,3,4,5,6}

{3,4,5,6}  {3,4,5,6}
Constraint Propagation: 1 – 8 Appear Exactly Once

{3,4,5,6} {3,4,5,6}

7 1 8 2

{3,4,5,6} {3,4,5,6}
Constraint Propagation: Adjacency

{3,4,5,6}  {3,4,5,6}

7 1 8 2

{3,4,5,6}  {3,4,5,6}
Constraint Propagation: Adjacency

{3,4,5} {4,5,6}

7 ? 1

? 8

? ?

{3,4,5} {4,5,6}

2
Search:

- Guess, but be prepared to backtrack…

{3,4,5} {4,5,6}
Constraint Propagation: 1 – 8 Appear Exactly Once

{3,4,5}  {4,5,6}

{4,5,6}
Constraint Propagation: 1 – 8 Appear Exactly Once

Diagram:

- 7
- 1
- 3
- 2
- 8
- ??

Set constraints:
- {4,5,6}
- {4,5}
- {4,5,6}
Constraint Propagation: Adjacency
Constraint Propagation: Adjacency

{4,5}  {4,5,6}  {5,6}
Search

• Guess, but be prepared to backtrack…

{5,6}

{4,5}

{4,5,6}
Search

- Guess, but be prepared to backtrack…
Constraint Propagation: 1 – 8 appear exactly once.
Constraint Propagation: 1 – 8 Appear Exactly Once

\[
\begin{array}{c}
7 & 3 & 5 \\
1 & ? & 8 \\
? & ? & 2 \\
\{4\} & \{4,6\} \\
\end{array}
\]
Constraint Propagation: 1 – 8 appear exactly once

\{4, 6\}
Constraint Propagation: 1 – 8 Appear Exactly Once

{6}
Solution
Remember

- Generally, it doesn’t go as well as this.
- Search will often involve backtracking.
Example 2: Sudoku
Example: Sudoku

- Has a very neat constraint model.
- Example sudoku taken from:
  - H. Simonis “Sudoku as a Constraint Problem”,

```
  2 6  
  3 7 8 6
  4 5  
  5 1 7 9
  3 9 5 1
  4 3 2 5
  1 3 2  
  5 2 4 9
  3 8 4 6
```
The Sudoku Problem

- Given: a $9 \times 9$ grid, with some entries blank, some containing a digit.
- Find: a complete grid.
The Sudoku Problem: Constraints

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>6</th>
<th></th>
<th>8</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td>7</td>
<td>8</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>5</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>7</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>3</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>2</td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
<td>4</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

- Such that:
  - On any row, all entries are distinct.
The Sudoku Problem: Constraints

- Such that:
  - On any column, all entries are distinct.
The Sudoku Problem: Constraints

- Such that:
  - These (the red & white) 3 × 3 squares contain distinct entries.
Sudoku: Constraint Model

- 81 variables, one for each grid entry.
- Domain: \{1, \ldots, 9\}
  - For simplicity we’ll assume that pre-filled entries are represented by variables with singleton domains.
- All-different constraints on rows, cols, 3 × 3 squares.
### Sudoku Model: Variables

<table>
<thead>
<tr>
<th>1,2,3,4,5,6,7,8,9</th>
<th>2</th>
<th>6</th>
<th>1,2,3,4,5,6,7,8,9</th>
<th>7</th>
<th>8</th>
<th>1,2,3,4,5,6,7,8,9</th>
</tr>
</thead>
<tbody>
<tr>
<td>3,1,2,3,4,5,6,7,8,9</td>
<td>7</td>
<td>8</td>
<td>1,2,3,4,5,6,7,8,9</td>
<td>6</td>
<td>7</td>
<td>1,2,3,4,5,6,7,8,9</td>
</tr>
<tr>
<td>4,1,2,3,4,5,6,7,8,9</td>
<td>5</td>
<td>7</td>
<td>1,2,3,4,5,6,7,8,9</td>
<td>9</td>
<td>7</td>
<td>1,2,3,4,5,6,7,8,9</td>
</tr>
<tr>
<td>1,2,3,4,5,6,7,8,9</td>
<td>3</td>
<td>9</td>
<td>1,2,3,4,5,6,7,8,9</td>
<td>5</td>
<td>1</td>
<td>1,2,3,4,5,6,7,8,9</td>
</tr>
<tr>
<td>1,2,3,4,5,6,7,8,9</td>
<td>4</td>
<td>3</td>
<td>1,2,3,4,5,6,7,8,9</td>
<td>2</td>
<td>5</td>
<td>1,2,3,4,5,6,7,8,9</td>
</tr>
<tr>
<td>1,2,3,4,5,6,7,8,9</td>
<td>3</td>
<td>6</td>
<td>1,2,3,4,5,6,7,8,9</td>
<td>2</td>
<td>4</td>
<td>1,2,3,4,5,6,7,8,9</td>
</tr>
<tr>
<td>1,2,3,4,5,6,7,8,9</td>
<td>4</td>
<td>6</td>
<td>4,1,2,3,4,5,6,7,8,9</td>
<td>2</td>
<td>9</td>
<td>1,2,3,4,5,6,7,8,9</td>
</tr>
<tr>
<td>1,2,3,4,5,6,7,8,9</td>
<td>8</td>
<td>4</td>
<td>1,2,3,4,5,6,7,8,9</td>
<td>6</td>
<td>6</td>
<td>1,2,3,4,5,6,7,8,9</td>
</tr>
</tbody>
</table>
Sudoku: Constraint Propagation

• The all-different constraints in the Sudoku model propagate well, leading to lots of useful deductions.

• As we will see these (probably) correspond to the way in which you make deductions when solving sudoku.
**Sudoku: Propagation**

Propagate AllDiff on $3 \times 3$ square.
### Sudoku: Propagation

<table>
<thead>
<tr>
<th>{1,5,7,8,9}</th>
<th>2</th>
<th>6</th>
<th>{1,2,3,4,5,6,7,8,9}</th>
<th>{1,2,3,4,5,6,7,8,9}</th>
<th>{1,2,3,4,5,6,7,8,9}</th>
<th>8</th>
<th>1</th>
<th>{1,2,3,4,5,6,7,8,9}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td>6,7,8,9</td>
<td>6,7,8,9</td>
<td></td>
<td></td>
<td>6,7,8,9</td>
</tr>
<tr>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td></td>
<td>5</td>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td>7</td>
<td>9</td>
<td></td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>{1,5,7,8,9}</td>
<td>{1,5,7,8,9}</td>
<td>1</td>
<td>5</td>
<td></td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td></td>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td>5</td>
<td>6,7,8,9</td>
<td>9</td>
<td></td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{1,2,3,4,5}</td>
<td>{1,2,3,4,5}</td>
<td>1</td>
<td>5</td>
<td></td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{1,2,3,4,5}</td>
<td>{1,2,3,4,5}</td>
<td>1</td>
<td>5</td>
<td></td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
</tbody>
</table>

This overlaps with the top-left 3 x 3 square we just looked at.

This is typical of how constraints communicate – through the domains of variables.

Domain modifications trigger propagation for constraints that constrain that variable.
**Sudoku: Propagation**

<table>
<thead>
<tr>
<th>{5,7,9}</th>
<th>{3,4,5,7,9}</th>
<th>8</th>
<th>1</th>
<th>{3,4,5,7,9}</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>{5,7,8}</td>
<td>7</td>
<td>8</td>
<td>{1,2,3,4,5}</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>{1,2,3,4,5}</td>
</tr>
<tr>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td>{1,2,3,4,5}</td>
<td>8</td>
<td>1</td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
<tr>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td>2</td>
<td>5</td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
<tr>
<td>1</td>
<td>{5,6}</td>
<td>3</td>
<td>9</td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
<tr>
<td>5</td>
<td>{1,2,3,4,5}</td>
<td>6</td>
<td>6</td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
<tr>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td>{1,2,3,4,5,6,7,8,9}</td>
<td>4</td>
<td>6</td>
<td>{1,2,3,4,5,6,7,8,9}</td>
</tr>
</tbody>
</table>

Propagate AllDiff on col 1.

We have made several new deductions in the top-left 3 x 3 square since we first considered it.

Generally, we would need to go back to the all-diff constraint on that 3x3 square to determine whether this can trigger yet more deductions.

Constraint queue controls propagation order.

Stop when we reach a fixpoint.
### Sudoku: Propagation

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>7,9</strong></td>
<td>2</td>
<td><strong>{3,4,5,7,9}</strong></td>
<td><strong>{3,4,5,7,9}</strong></td>
<td><strong>{3,4,5,7,9}</strong></td>
</tr>
<tr>
<td></td>
<td><strong>3</strong></td>
<td><strong>{1,5,7,8,9}</strong></td>
<td><strong>{1,2,3,4,5,6,7,8,9}</strong></td>
<td><strong>6</strong></td>
</tr>
<tr>
<td></td>
<td><strong>4</strong></td>
<td><strong>{1,5,7,8,9}</strong></td>
<td><strong>1,2,3,4,5,6,7</strong></td>
<td><strong>8</strong></td>
</tr>
<tr>
<td><strong>2,6,7,8,9</strong></td>
<td>5</td>
<td><strong>{1,2,3,4,5,6,7,8,9}</strong></td>
<td><strong>5</strong></td>
<td><strong>9</strong></td>
</tr>
<tr>
<td><strong>2,6,7,8,9</strong></td>
<td><strong>{1,2,3,4,5,6,7,8,9}</strong></td>
<td><strong>5</strong></td>
<td><strong>1,2,3,4,5,6,7,8,9</strong></td>
<td></td>
</tr>
<tr>
<td><strong>2,6,7,8,9</strong></td>
<td><strong>{1,2,3,4,5,6,7,8,9}</strong></td>
<td><strong>{1,2,3,4,5,6,7,8,9}</strong></td>
<td><strong>5</strong></td>
<td><strong>1,2,3,4,5,6,7,8,9</strong></td>
</tr>
<tr>
<td><strong>1</strong></td>
<td><strong>{1,2,3,4,5,6,7,8,9}</strong></td>
<td><strong>2</strong></td>
<td><strong>{1,2,3,4,5,6,7,8,9}</strong></td>
<td></td>
</tr>
<tr>
<td><strong>5</strong></td>
<td><strong>{1,2,3,4,5,6,7,8,9}</strong></td>
<td><strong>9</strong></td>
<td><strong>{1,2,3,4,5,6,7,8,9}</strong></td>
<td></td>
</tr>
</tbody>
</table>

**Propagate AllDiff on 3 × 3 square**

**Can you see why 3, 4, 9 can be removed?**

**This is an example of a less obvious deduction**

**In fact alldiff propagator removes all values that cannot participate in a solution to that constraint**
...And so on:

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{7}</td>
<td>2</td>
<td>6</td>
<td>{4}</td>
<td>{9}</td>
<td>{3}</td>
<td>8</td>
<td>1</td>
<td>{5}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>{1}</td>
<td>{5}</td>
<td>7</td>
<td>{2}</td>
<td>8</td>
<td>{9}</td>
<td>{4}</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>{8}</td>
<td>{9}</td>
<td>{6}</td>
<td>5</td>
<td>{1}</td>
<td>{2}</td>
<td>{3}</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>{8}</td>
<td>5</td>
<td>{2}</td>
<td>1</td>
<td>{4}</td>
<td>7</td>
<td>{6}</td>
<td>9</td>
<td>{3}</td>
<td></td>
</tr>
<tr>
<td>{6}</td>
<td>{7}</td>
<td>3</td>
<td>9</td>
<td>{8}</td>
<td>5</td>
<td>1</td>
<td>{2}</td>
<td>{4}</td>
<td></td>
</tr>
<tr>
<td>{9}</td>
<td>4</td>
<td>{1}</td>
<td>3</td>
<td>{6}</td>
<td>2</td>
<td>{7}</td>
<td>5</td>
<td>{8}</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>{9}</td>
<td>{4}</td>
<td>{8}</td>
<td>3</td>
<td>{6}</td>
<td>{5}</td>
<td>{7}</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>{6}</td>
<td>{7}</td>
<td>2</td>
<td>{1}</td>
<td>4</td>
<td>{3}</td>
<td>{8}</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>{2}</td>
<td>3</td>
<td>8</td>
<td>{5}</td>
<td>{7}</td>
<td>{9}</td>
<td>4</td>
<td>6</td>
<td>{1}</td>
<td></td>
</tr>
</tbody>
</table>
Sudoku: Propagation

• As Simonis demonstrated:

• For Sudoku constraint propagation is almost always sufficiently powerful to find the solution.
  – By design, each sudoku has one solution.

• Unfortunately, this is not generally the case…
Course Overview
A Course in Two Parts

• Part I: Inside a constraint solver.
• Part II: Constraint modelling.
• Why this way around?
  • Because we have to model to take advantage of the solver.
  • If we don’t understand the solver, we cannot model effectively.
Part I: Constraint Solving

- Definitions.
- Basic solving techniques.
- Constraint Propagation:
  - Consistency properties.
  - Consistency-enforcing algorithms.
- Heuristics.
- Advanced search.
Part II: Constraint Modelling

- Modelling by Example.
- Channelling.
- Exploiting Symmetry.
- Implied Constraints.
Summary

• Background:
  • Constraint Modelling & Solving.
  • The Constraint Satisfaction Problem (CSP).
• The Crystal Maze Puzzle.
• Sudoku.
• Take a look at: www.csplib.org.
