Constraint Satisfaction
Part II
CSE 473
473 Topics

Blind
Informed (Heuristics)
Constraint Satisfaction (Factored)
Adversarial
What We Covered Last Week

• Constraint Satisfaction as search
• Backtracking methods:
  • Chronological
  • Backjumping
  • Conflict-directed backjumping
Plan For Today

• Improving Efficiency, continued
  • Forward Checking (FC)
  • Constraint Propagation
• Application of CSP
  • Automatically Rendering UIs
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values
Forward checking

Idea: Keep track of remaining legal values for unassigned variables
Terminate search when any variable has no legal values
Forward checking

**Idea:** Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values
**Forward checking**

Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values

```
<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image 1" /></td>
<td><img src="image2.png" alt="Image 2" /></td>
<td><img src="image3.png" alt="Image 3" /></td>
<td><img src="image4.png" alt="Image 4" /></td>
<td><img src="image5.png" alt="Image 5" /></td>
<td><img src="image6.png" alt="Image 6" /></td>
<td><img src="image7.png" alt="Image 7" /></td>
</tr>
<tr>
<td><img src="image8.png" alt="Image 8" /></td>
<td><img src="image9.png" alt="Image 9" /></td>
<td><img src="image10.png" alt="Image 10" /></td>
<td><img src="image11.png" alt="Image 11" /></td>
<td><img src="image12.png" alt="Image 12" /></td>
<td><img src="image13.png" alt="Image 13" /></td>
<td><img src="image14.png" alt="Image 14" /></td>
</tr>
<tr>
<td><img src="image15.png" alt="Image 15" /></td>
<td><img src="image16.png" alt="Image 16" /></td>
<td><img src="image17.png" alt="Image 17" /></td>
<td><img src="image18.png" alt="Image 18" /></td>
<td><img src="image19.png" alt="Image 19" /></td>
<td><img src="image20.png" alt="Image 20" /></td>
<td><img src="image21.png" alt="Image 21" /></td>
</tr>
</tbody>
</table>
```
**Constraint propagation**

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

\[
\begin{array}{cccccccc}
\text{WA} & \text{NT} & \text{Q} & \text{NSW} & \text{V} & \text{SA} & \text{T} \\
\hline
\text{\cellcolor{red}red} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} & \text{\cellcolor{green}green} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} \\
\text{\cellcolor{red}red} & \text{\cellcolor{green}green} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} \\
\text{\cellcolor{red}red} & \text{\cellcolor{red}red} & \text{\cellcolor{green}green} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} & \text{\cellcolor{red}red} \\
\end{array}
\]

\(NT\) and \(SA\) cannot both be blue!

*Constraint propagation* repeatedly enforces constraints locally.
Arc consistency

Simplest form of propagation makes each arc consistent.

$X \rightarrow Y$ is consistent iff

for every value $x$ of $X$ there is some allowed $y$. 
Arc consistency

Simplest form of propagation makes each arc consistent

\[ X \rightarrow Y \text{ is consistent iff} \]

for every value of \( X \) there is some allowed \( Y \)
Arc consistency

Simplest form of propagation makes each arc consistent.

\( X \rightarrow Y \) is consistent iff for every value \( x \) of \( X \) there is some allowed \( y \).

If \( X \) loses a value, neighbors of \( X \) need to be rechecked.
**Arc consistency**

Simplest form of propagation makes each arc **consistent**.

\[ X \rightarrow Y \] is consistent iff 
for every value \( x \) of \( X \) there is some allowed \( y \).

If \( X \) loses a value, neighbors of \( X \) need to be rechecked.

Arc consistency detects failure earlier than forward checking.

Can be run as a preprocessor or after each assignment.
Dynamic variable ordering

• In the N-queens examples we assumed
  • First x1 then x2 then ...
• But this order not required
  Any order ok with respect to completeness
• A good order leads to huge speedup
• A good heuristic (MRV):
  • Choose variable w/ minimum remaining values
• This is easy if one is doing FC
DVO MRV => WOW!!

<table>
<thead>
<tr>
<th>Algo</th>
<th>17 Queens</th>
<th>21 Queens</th>
<th>27 Queens</th>
</tr>
</thead>
<tbody>
<tr>
<td>FC-CBJ(mrv)</td>
<td>1959</td>
<td>2572</td>
<td>5602</td>
</tr>
<tr>
<td>FC-CBJ</td>
<td>67090</td>
<td>114612</td>
<td>737008</td>
</tr>
<tr>
<td>FC</td>
<td>67329</td>
<td>115120</td>
<td>7448781</td>
</tr>
<tr>
<td>CBJ</td>
<td>428645</td>
<td>949128</td>
<td></td>
</tr>
<tr>
<td>BJ</td>
<td>436340</td>
<td>972065</td>
<td></td>
</tr>
<tr>
<td>BT</td>
<td>485597</td>
<td>1156015</td>
<td></td>
</tr>
</tbody>
</table>
Automatic UI Rendering

• Inputs:
  • Interface specification
  • Available widgets
  • Size constraint
  • Cost function ($)
• UI Rendering as search
• UI Rendering as CSP
Automatically Rendered Interfaces for Classroom Controller
Modeling User Interfaces

- **simple types:** \( \text{int} | \text{float} | \text{string} | \text{bool} \)
- **derivative types:** \( \langle \tau, C_\tau \rangle \)
- **vectors:** \( \text{vector}(\tau) \)
- **containers:** \( \{ \tau_i | i \in 1...n \} \)
- **actions:** \( \tau \rightarrow \text{nil} \)
Modeling User Interfaces

- **simple types:** \(\text{int} | \text{float} | \text{string} | \text{bool}\)
- derivative types:
- vectors:
- containers:
- actions:
Modeling User Interfaces

- simple types:
- derivative types: $\langle \tau, C_{\tau} \rangle$
- vectors:
- containers:
- actions:
Modeling User Interfaces

- simple types:
- derivative types:
- vectors:
- containers: \( \{ \tau_i \mid i \in 1 \ldots n \} \)
- actions:

\{bool, string, \{Computer 1, Computer 2, Video\}\}
Modeling User Interfaces: Optional Attributes

- Label
- Set of likely values
- Exact value required
- ...

...
Examples of Available Widgets

Pointer and Keyboard

Touch Screen
Examples of Available Widgets

- Tab View
- Vertical Box
  - Element 1
  - Element 2
- Horizontal Box
  - Element 1
  - Element 2
Cost Function
For Primitive Widgets

\[ F(\text{int}, [0,10], \text{exact} = \text{false}) = 3 \]

\[ F(\text{int}, [0,10], \text{exact} = \text{false}) = 1 \]
UI Rendering As Search

- States
- Operators
- Start state
- Goal test
UI Rendering As CSP

- Variables
- Domains
- Constraints
UI Rendering As CSP

- Variables
  - UI Elements
- Domains
  - Sets of available widgets
- Constraints
  - Total size of the interface
UI Rendering As CSP

Classroom:
  \( \tau: \{ \_ \} \)

Light Bank:
  \( \tau: \{ \_ \} \)

Light:
  \( \tau: \{ \_ \} \)

Power:
  \( \tau: \text{bool} \)

Light ... Light ...

Screen:
  \( \tau: \text{bool} \)

A/V:
  \( \tau: \{ \_ \} \)

Projector:
  \( \tau: \{ \_ \} \)

Input:
  \( \tau: \text{string, \{data1, data2, video\}} \)

Vent:
  \( \tau: \langle \text{int, [0,3]} \rangle \)

Light Level:
  \( \tau: \langle \text{int, [0,10]} \rangle \)
UI Rendering As CSP
Branch And Bound?

- **Gist:** prune branches if they are no better than what you already have
Constraint Satisfaction Recap

- CSP = Factoring a state space
- Chronological Backtracking (BT)
- Backjumping (BJ)
- Conflict-Directed Backjumping (CBJ)
- Forward checking (FC)
- Constraint Propagation
- Dynamic variable ordering heuristics
- Preprocessing Strategies