Constraint satisfaction search.
Combinatorial optimization search.

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Constraint satisfaction problem (CSP)

Objective:
• Find a configuration satisfying goal conditions

• Constraint satisfaction problem (CSP) is a configuration search problem where:
  – A state is defined by a set of variables and their values
  – Goal condition is represented by a set constraints on possible variable values
**CSP example: N-queens**

**Goal:** n queens placed in non-attacking positions on the board

**Variables:**
- Represent queens, one for each column:
  - $Q_1, Q_2, Q_3, Q_4$
- Values:
  - Row placement of each queen on the board
  
  $\{1, 2, 3, 4\}$

**Constraints:**
- $Q_i \neq Q_j$ Two queens not in the same row
- $|Q_i - Q_j| \neq |i - j|$ Two queens not on the same diagonal

---

**Map coloring**

Color a map using k different colors such that no adjacent countries have the same color

**Variables:**
- Represent countries
  - $A, B, C, D, E$
- Values:
  - k different colors
  
  $\{$Red, Blue, Green,$\ldots\}$

**Constraints:**
- $A \neq B, A \neq C, C \neq E, \ldots$ etc
  
  An example of a problem with **binary constraints**
Solving a CSP through standard search

- **Maximum depth of the tree:** Number of variables in the CSP
- **Depth of the solution:** Number of variables in the CSP
- **Branching factor:** if we fix the order of variable assignments the branch factor depends on the number of their values

Unassigned: $Q_1, Q_2, Q_3, Q_4$
Assigned: $Q_1 = 1$

Unassigned: $Q_2, Q_3, Q_4$
Assigned: $Q_1 = 2$

...
Constraint consistency

Question:
- When to check the constraints defining the goal condition?
- The violation of constraints can be checked:
  - at the end (for the leaf nodes)
  - for each node of the search tree during its generation or before its expansion

Checking the constraints for intermediate nodes:
- More efficient: cuts branches of the search tree early
Constraint consistency

Checking the constraints for intermediate nodes:
• More efficient: cuts branches of the search tree early

Another way to cut the search space and tree exploration
• Current variable assignments together with constraints restrict remaining legal values of unassigned variables
• The remaining legal and illegal values of variables may be inferred (effect of constraints propagates)
Constraint consistency

Another way to cut the search space and tree exploration

- Current variable assignments together with constraints restrict remaining legal values of unassigned variables
- The remaining legal and illegal values of variables may be inferred (effect of constraints propagates)

Constraint propagation

A state (more broadly) is defined:
- by a set of assigned variables, their values and
- a list of legal and illegal assignments for unassigned variables

Legal and illegal assignments can be represented:
- equations (value assignments) and
- disequations (list of invalid assignments)

\[ A = \text{Red}, \ \text{Blue} \quad C \neq \text{Red} \]

Constraints + assignments can entail new equations and disequations

\[ A = \text{Red} \quad \rightarrow \quad B \neq \text{Red} \]

Constraint propagation: the process of inferring of new equations and disequations from existing equations and disequations
Constraint propagation

- Assign A=Red

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Blue</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

✓ - Assignments (equations)  × - Invalid assignments (disequations)

Constraint propagation

- Assign A=Red

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Blue</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td>×</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

✓ - Assignments (equations)  × - Invalid assignments (disequations)
Constraint propagation

• Assign E=Blue

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Blue</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Constraint propagation

• Assign E=Blue

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Blue</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CS 1571 Intro to AI
M. Hauskrecht
Constraint propagation

- Assign F=Green

<table>
<thead>
<tr>
<th></th>
<th>Red</th>
<th>Blue</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Constraint propagation

Three techniques for propagating the effects of past assignments and constraints:

- **Node propagation**
- **Arc consistency**
- **Forward checking**

**Difference:**
- Completeness of inferences
- Time complexity of inferences.
Constraint propagation

1. **Node consistency.** Infers:
   - **equations** (valid assignments) or **disequations** (invalid assignments) for an individual variable by applying **a unary constraint**

2. **Arc consistency.** Infers:
   - **disequations** from the set of **equations and disequations** defining the partial assignment, and **a constraint**
   - **equations through the exhaustion of alternatives**

3. **Forward checking.** Infers:
   - **disequations** from a set of **equations** defining the partial assignment, and a constraint
   - **Equations through the exhaustion of alternatives**

   **Restricted forward checking:**
   - uses only active constraints (active constraint – only one variable unassigned in the constraint)

---

Example

**Map coloring of Australia territories**

Note: problem with binary constraints
Example: node consistency

Map coloring

Assume a constraint:
WA ≠ Green

<table>
<thead>
<tr>
<th>vars</th>
<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>domain inferred</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
</tbody>
</table>

Example: node consistency

Map coloring

Assume a constraint:
WA ≠ Green

Infer: invalid assignments from WA ≠ Green constraint

<table>
<thead>
<tr>
<th>vars</th>
<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>domain inferred</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>inferred</td>
<td>R B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
</tbody>
</table>
Example: forward checking

Map coloring

Set: WA=Red

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
</tbody>
</table>

Infer: invalid assignments from WA=Red + constraints
### Example: forward checking

**Map coloring**

Set: Q=Green

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>WA=Red</td>
<td>R</td>
<td>G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>G B</td>
<td>R G B</td>
</tr>
</tbody>
</table>

Infer: invalid assignments from Q=Green + constraints

### Example: forward checking

**Map coloring**

Infer: invalid assignments from Q=Green + constraints

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>WA=Red</td>
<td>R</td>
<td>G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>G B</td>
<td>R G B</td>
</tr>
<tr>
<td>Q=Green</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td>R B</td>
<td>R G B</td>
<td>B</td>
<td>R G B</td>
</tr>
</tbody>
</table>
Example: forward checking

Map coloring

Infer: NT=B
Exhaustions of alternatives

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>WA=Red</td>
<td>R</td>
<td>G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>G B</td>
<td>R G B</td>
</tr>
<tr>
<td>Q=Green</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td>R B</td>
<td>R G B</td>
<td>B</td>
<td>R G B</td>
</tr>
<tr>
<td>Infer NT</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Example: forward checking

Map coloring

Infer: invalid assignments from NT=B

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>WA=Red</td>
<td>R</td>
<td>G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>G B</td>
<td>R G B</td>
</tr>
<tr>
<td>Q=Green</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td>R B</td>
<td>R G B</td>
<td>B</td>
<td>R G B</td>
</tr>
<tr>
<td>Infer NT</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td>R B</td>
<td>R G B</td>
<td>!</td>
<td>R G B</td>
</tr>
</tbody>
</table>
Example: arc consistency

Map coloring

Set: WA=Red
Set: Q=Green

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>WA=Red</td>
<td>R</td>
<td>G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>G B</td>
<td>R G B</td>
</tr>
<tr>
<td>Q=Green</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td>R B</td>
<td>R G B</td>
<td>B</td>
<td>R G B</td>
</tr>
</tbody>
</table>

Infer: invalid assignments from valid and invalid assignments

Example: arc consistency

Map coloring

Infer: invalid assignments from valid and invalid assignments

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>WA=Red</td>
<td>R</td>
<td>G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>G B</td>
<td>R G B</td>
</tr>
<tr>
<td>Q=Green</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td>R B</td>
<td>R G B</td>
<td>B</td>
<td>R G B</td>
</tr>
</tbody>
</table>
Example: arc consistency

Map coloring

Infer: invalid assignments from valid and invalid assignments

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>WA=Red</td>
<td>R</td>
<td>G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>G B</td>
<td>R G B</td>
</tr>
<tr>
<td>Q=Green</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td>R B</td>
<td>R G B</td>
<td>B</td>
<td>R G B</td>
</tr>
</tbody>
</table>

Consistent assignment

Infer: invalid assignments from valid and invalid assignments

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>NSW=R</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>SA=B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
</tbody>
</table>
Example: arc consistency

Map coloring

Infer: invalid assignments from valid and invalid assignments

<table>
<thead>
<tr>
<th>vars</th>
<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>domain</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
</tr>
<tr>
<td>WA=Red</td>
<td>R</td>
<td>G B</td>
<td>R G B</td>
<td>R G B</td>
<td>R G B</td>
<td>G B</td>
<td>R G B</td>
</tr>
<tr>
<td>Q=Green</td>
<td>R</td>
<td>B</td>
<td>G</td>
<td>R B</td>
<td>R G B</td>
<td>B</td>
<td>R G B</td>
</tr>
</tbody>
</table>

Inconsistent assignment

Infer: invalid assignments from valid and invalid assignments
Heuristics for CSPs

CSP searches the space in the depth-first manner. But we still can choose:

• Which variable to assign next?
• Which value to choose first?

Heuristics

• Most constrained variable
  – Which variable is likely to become a bottleneck?

• Least constraining value
  – Which value gives us more flexibility later?

Examples: map coloring

Heuristics

• Most constrained variable
  – Country E is the most constrained one (cannot use Red, Green)

• Least constraining value
  – Assume we have chosen variable C
  – Red is the least constraining valid color for the future
Finding optimal configurations

Search for the optimal configuration

Constrain satisfaction problem:
Objective: find a configuration that satisfies all constraints

Optimal configuration problem:
Objective: find the best configuration
The quality of a configuration: is defined by some quality measure that reflects our preference towards each configuration (or state)

Our goal: optimize the configuration according to the quality measure also referred to as objective function
Search for the optimal configuration

**Optimal configuration search:**
- Configurations are described in terms of variables and their values
- Each configuration has a quality measure
- **Goal:** find the configuration with the best value

If the space of configurations we search among is
- **Discrete or finite**
  - then it is a combinatorial optimization problem
- **Continuous**
  - then it is a parametric optimization problem

Example: Traveling salesman problem

**Problem:**
- A graph with distances
- A tour – a path that visits every city once and returns to the start e.g. ABCDEF
- **Goal:** find the shortest tour
Example: N queens

- A CSP problem
- Is it possible to formulate the problem as an optimal configuration search problem? Yes.
- The quality of a configuration in a CSP can be measured by the number of violated constraints
- Solving: minimize the number of constraint violations
Iterative optimization methods

• Searching systematically for the best configuration with the DFS may not be the best solution
• Worst case running time:
  – Exponential in the number of variables
• Solutions to large ‘optimal’ configuration problems are often found using iterative optimization methods

• Methods:
  – Hill climbing
  – Simulated Annealing
  – Genetic algorithms

Properties:

– Search the space of “complete” configurations
– Take advantage of local moves
  • Operators make “local” changes to “complete” configurations
– Keep track of just one state (the current state)
  • no memory of past states
  • !!! No search tree is necessary !!!
Example: N-queens

- “Local” operators for generating the next state:
  - Select a variable (a queen)
  - Reallocate its position

![Chessboard with queen placements](image)

Example: Traveling salesman problem

“Local” operator for generating the next state:
- divide the existing tour into two parts,
- reconnect the two parts in the opposite order

Example:

```
ABCDEF
```

![Traveling salesman problem example](image)
Example: Traveling salesman problem

“Local” operator for generating the next state:
- divide the existing tour into two parts,
- reconnect the two parts in the opposite order

Example:

\[
\begin{align*}
\text{ABCDEF} & \downarrow \\
\text{ABCD} & \downarrow \\
\text{EF} & \\
\text{ABCD} & \downarrow \\
\text{EF} & \\
\text{ABCDFE} & \\
\end{align*}
\]
Searching the configuration space

Search algorithms

- keep only one configuration (the current configuration)

Problem:
- How to decide about which operator to apply?

Search algorithms

Strategies to choose the configuration (state) to be visited next:
- Hill climbing
- Simulated annealing

- Later: Extensions to multiple current states:
  - Genetic algorithms

- Note: Maximization is inverse of the minimization
  \[ \min f(X) \equiv \max \left[-f(X)\right] \]