Constraint Satisfaction

CS 486/686: Introduction to Artificial Intelligence
Outline

• What are Constraint Satisfaction Problems (CSPs)?
• Standard Search and CSPs
• Improvements
  - Backtracking
  - Backtracking + heuristics
  - Forward Checking
Introduction

Standard search

**State** is a “black box”: arbitrary data structure

**Goal test**: any function over states

**Successor function**: anything that lets you move from one state to another

Constraint satisfaction problems (CSPs)

A special subset of search problems

**States** are defined by *variables* $X_i$ with values from *domains* $D_i$

**Goal test** is a *set of constraints* specifying allowable combinations of values for subsets of variables
Example: Map Colouring

- **Variables**
  - \( V = \{ T, V, NSW, Q, NT, WA, SA \} \)

- **Domains**
  - \( D = \{ \text{red, blue, green} \} \)

- **Constraints**: adjacent regions must have different colours
  - Implicit: \( WA \neq NT \)
  - Explicit: \( (WA, NT) \in \{(\text{red, blue}), (\text{red, green}), (\text{blue, red})\} \ldots \)

- **Solution** is an assignment satisfying all constraints
  - \( \{WA=\text{red}, NT=\text{green}, Q=\text{red}, NSW=\text{green}, V=\text{red}, SA=\text{blue}, T=\text{green} \} \)
N Queens Problem

• Variables: \(X_{i,j}\)

• Domains: \(\{0,1\}\)

• Constraints:

\[
\forall i, j, k \ (X_{i,j}, X_{i,k}) \in \{(0,0), (0,1), (1,0)\}
\]
\[
\forall i, j, k \ (X_{i,j}, X_{k,j}) \in \{(0,0), (0,1), (1,0)\}
\]
\[
\forall i, j, k \ (X_{i,j}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}
\]
\[
\forall i, j, k \ (X_{i,j}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}
\]
N Queens Problem

• Variables: $Q_i$

• Domains: \{1,2,…,N\}

• Constraints:
  • Implicit:
    \[
    \forall i, j \text{ non-threatening}(Q_i, Q_j)
    \]
  • Explicit:
    \[
    (Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}
    \]
    
    \[
    \ldots
    \]
3 Sat

- **Variables:** $V_1, \ldots, V_n$
- **Domains:** $\{0,1\}$
- **Constraints:**
  - $K$ constraints of the form $V_i^* \lor V_j^* \lor V_k^* \lor \neg V_i$ where $V_i^*$ is either $V_i$ or $\neg V_i$

$$
A \lor \neg B \lor \neg C \\
\neg A \lor B \lor D \\
D \lor B \lor E \\
\neg A \lor \neg B \lor C
$$

A canonical NP-complete problem
Types of CSPs

- **Discrete Variables**
  - **Finite domains**
    - If domain has size $d$, then there are $O(d^n)$ complete assignments
    - Boolean CSPs (including 3-SAT)
  - **Infinite domains** (e.g. integers)
    - Constraint languages
    - Linear constraints are solvable but non-linear are undecidable

- **Continuous Variables**
  - Linear programming (linear constraints solvable in polynomial time)
Types of CSPs

- **Varieties of Constraints**
  - **Unary constraints**: involve a single variable
    - NSW ≠ red
  - **Binary constraints**: involve a pair of variables
    - NSW ≠ Q
  - **Higher-order constraints**: involve more than two variables
    - AllDiff(V₁, ..., Vₙ)

- **Soft Constraints (preferences)**
  - red “is better than” green
  - Constrained optimization problems
Constraint Graphs

You can represent binary constraints with a constraint graph

Nodes are variables

Edges are constraints
We can use standard search to solve CSPs

States:
Initial State:
Successor Function:
Goal Test:
CSPs and Search

States:
Initial State:
Successor Function:
Goal Test:

What happens if we run something like BFS?
Commutativity

Key Insight:

- CSPs are **commutative**
  - Order of actions taken does not effect outcome
  - Can assign variables in any order
- CSP algorithms take advantage of this
  - Consider possible assignments for a **single variable at each node** in the search tree

\[ \{\text{WA=red, NT=blue}\} \text{ is equivalent to } \{\text{NT=blue, WA=red}\} \]
Backtracking Search

Backtracking search is the basic algorithm for CSPs

- Select unassigned variable X
- For each value \( \{x_1, \ldots, x_n\} \) in domain of X
  - If value satisfies constraints, assign \( X = x_i \) and exit loop
- If an assignment is found
  - Move to next variable
- If no assignment found
  - Back up to preceding variable and try a different assignment for it
Backtracking Example
Backtracking Example
Backtracking Example
Backtracking Example
Backtracking and Efficiency

Note that backtracking search is basically DFS with some small improvements. Can we improve on it further?

**Ordering:**
- Which variables should be tried first?
- In what order should a variable’s values be tried?

**Filtering:**
- Can we detect failure early?

**Structure:**
- Can we exploit the problem structure?
Ordering: Most Constrained Variable

• Choose the variable which has the fewest “legal” moves
  - AKA minimum remaining values (MRV)

\[
\begin{align*}
D_{NT} &= \{\text{green, blue}\} \\
D_{SA} &= \{\text{green, blue}\} \\
D_{\text{others}} &= \{\text{red, green, blue}\} \\
D_{SA} &= \{\text{blue}\} \\
D_{Q} &= \{\text{blue, red}\} \\
D_{\text{others}} &= \{\text{red, green, blue}\}
\end{align*}
\]
Ordering: Most Constraining Variable

• Most constraining variable:
  - Choose variable with most constraints on remaining variables

• Tie-breaker among most constrained variables

SA is involved in 5 constraints
Ordering: Least-Constraining Value

- Given a variable, choose the least constraining value:
  - The one that rules out the fewest values in the remaining variables
Filtering: Forward Checking

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

### Table

<table>
<thead>
<tr>
<th></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>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
</tr>
</tbody>
</table>
Example: Forward Checking

Forward checking removes the value Red of NT and of SA
Example: Forward Checking

<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>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
</tr>
<tr>
<td>R</td>
<td>GB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>GB</td>
<td>RGB</td>
</tr>
<tr>
<td>R</td>
<td>GB</td>
<td>G</td>
<td>RGB</td>
<td>RGB</td>
<td>GB</td>
<td>RGB</td>
</tr>
</tbody>
</table>
Example: Forward Checking

<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>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
</tr>
<tr>
<td>R</td>
<td>GB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>GB</td>
<td>RGB</td>
</tr>
<tr>
<td>R</td>
<td>B</td>
<td>G</td>
<td>RB</td>
<td>RGB</td>
<td>B</td>
<td>RGB</td>
</tr>
<tr>
<td>R</td>
<td>B</td>
<td>G</td>
<td>RB</td>
<td>B</td>
<td>B</td>
<td>RGB</td>
</tr>
</tbody>
</table>
Example: Forward Checking

Empty set: the current assignment \{\text{(WA} \leftarrow \text{R}), \text{(Q} \leftarrow \text{G}), \text{(V} \leftarrow \text{B)}\} does not lead to a solution

<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>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
</tr>
<tr>
<td>R</td>
<td>GB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>GB</td>
<td>RGB</td>
</tr>
<tr>
<td>R</td>
<td>B</td>
<td>G</td>
<td>RB</td>
<td>RGB</td>
<td>B</td>
<td>RGB</td>
</tr>
<tr>
<td>R</td>
<td>B</td>
<td>G</td>
<td>RB</td>
<td>B</td>
<td>B</td>
<td>RGB</td>
</tr>
</tbody>
</table>
Filtering: Arc Consistency

Forward checking propagates information from assigned to unassigned variables, but it can not detect all future failures early.

NT and SA can not both be blue!

Need to reason about constraints
Given domains $D_1$ and $D_2$, an arc is consistent if for all $x$ in $D_1$ there is a $y$ in $D_2$ such that $x$ and $y$ are consistent.

Is the arc from SA to NSW consistent?
Is the arc from NSW to SA consistent?
Tasmania does not interact with the rest of the problem.

**Idea:** Break down the graph into its connected components. Solve each component separately.

**Significant potential savings:**
- Assume $n$ variables with domain size $d$: $O(d^n)$
- Assume each component involves $c$ variables ($n/c$ components) for some constant $c$: $O(d^c n/c)$
CSPs can be solved in $O(nd^2)$ if there are no loops in the constraint graph.

**Step 1:** For $i=n$ to 1, make-consistent($X_i$,parent($X_i$))

**Step 2:** For $i=1$ to $n$, assign value to $X_i$ consistent with parent($X_i$)  [Note: No backtracking!]
Structure: Non-Trees?

If we assign SA a colour and then remove that colour from the domains all other variables, then we have a tree.

**Step 1:** Choose a subset $S$ of variables such that the constraint graph becomes a tree when $S$ is removed ($S$ is the cycle cutset)

**Step 2:** For each possible valid assignment to the variables in $S$
1. Remove from the domains of remaining variables, all values that are inconsistent with $S$
2. If the remaining CSP has a solution, return it
Structure: Cutsets

Running time:

- Let $c$ be the size of the cutset then
  - $d^c$ combinations of variables in $S$
  - For each combination must solve a tree problem of size $n-c$ \( O(n-c)d^2 \)
- Therefore, running time is \( O(d^c(n-c)d^2) \)
- Finding smallest cutset is NP-hard but efficient approximations exist
Structure: Non-Trees?

Tree decompositions

1. Each variable appears in at least one subproblem
2. If two variables are connected by a constraint, then they (and the constraint) must appear together in at least one subproblem
3. If a variable appears in two subproblems in the tree, it must appear in every subproblem along the path connecting those subproblems
Structure: Tree Decompositions

- Solve each subproblem independently
  - e.g \{ (WA=r, NT=g, SA=b), (WA=b, NT=g, SA=r), ... \}

- Solve constraints connecting the subproblems using tree-based algorithm (to make sure that subproblems with shared variables agree)

Want to make the subproblems as small as possible!

**Tree width**: $w = \text{Size of largest subproblem} - 1$

Running time $O(nd^{w+1})$

Finding tree decomposition with min tree-width is NP-hard, but good heuristics exist
Summary

- How to formalize problems as CSPs
- Backtracking search
- Improvements using
  - Ordering
  - Filtering
  - Structure