Constraint-satisfaction search

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

Search problem

A search problem:
• Search space (or state space): a set of objects among which we conduct the search;
• Initial state: an object we start to search from;
• Operators (actions): transform one state in the search space to the other;
• Goal condition: describes the object we search for

• Possible metric on a search space:
  – measures the quality of the object with regard to the goal

Search problems occur in planning, optimizations, learning
Constraint satisfaction problem (CSP)

Constraint satisfaction problem (CSP) is a configuration search problem where:
• A state is defined by a set of variables
• Goal condition is represented by a set constraints on possible variable values

Special properties of the CSP allow more specific procedures to be designed and applied for solving them

Example of a CSP: 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
**Satisfiability (SAT) problem**

Determine whether a sentence in the conjunctive normal form (CNF) is satisfiable (can evaluate to true)
- Used in the propositional logic (covered later)

\[(P \lor Q \lor \neg R) \land (\neg P \lor \neg R \lor S) \land (\neg P \lor Q \lor \neg T)\]...

**Variables:**
- Propositional symbols (P, R, T, S)
- Values: True, False

**Constraints:**
- Every conjunct must evaluate to true, at least one of the literals must evaluate to true

\[(P \lor Q \lor \neg R) \equiv True , (\neg P \lor \neg R \lor S) \equiv True ,\ldots\]

---

**Other real world CSP problems**

**Scheduling problems:**
- E.g. telescope scheduling
- High-school class schedule

**Design problems:**
- Hardware configurations
- VLSI design

**More complex problems may involve:**
- **real-valued variables**
- **additional preferences on variable assignments** – the optimal configuration is sought
Map coloring

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

Variables:

- Variable values:

Constraints:
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,..\}

**Constraints:** \( A \neq B, A \neq C, C \neq E, \) etc
An example of a problem with **binary constraints**

---

Constraint satisfaction as a search problem

**Formulation of a CSP as a search problem:**
- **States.** Assignments(partial, complete) of values to variables.
- **Initial state.** No variable is assigned a value.
- **Operators.** Assign a value to one of the unassigned variables.
- **Goal condition.** All variables are assigned, no constraints are violated.

- **Constraints** can be represented:
  - **Explicitly** by a set of allowable values
  - **Implicitly** by a function that tests for the satisfaction of constraints
Solving CSP as a standard search

Unassigned: \( Q_1, Q_2, Q_3, Q_4 \)
Assigned:

Unassigned: \( Q_2, Q_1, Q_4 \)
Assigned: \( Q_1 = 1 \)

Unassigned: \( Q_2, Q_1, Q_4 \)
Assigned: \( Q_1 = 2 \)

Unassigned: \( Q_3, Q_4 \)
Assigned: \( Q_1 = 2, Q_2 = 4 \)

---

Solving a CSP through standard search

- Maximum depth of the tree (m): ?
- Depth of the solution (d): ?
- Branching factor (b): ?

Unassigned: \( Q_1, Q_2, Q_3, Q_4 \)
Assigned:

Unassigned: \( Q_2, Q_3, Q_4 \)
Assigned: \( Q_1 = 1 \)

Unassigned: \( Q_2, Q_3, Q_4 \)
Assigned: \( Q_1 = 2 \)

Unassigned: \( Q_3, Q_4 \)
Assigned: \( Q_1 = 2, Q_2 = 4 \)

---
Solving a CSP through standard search

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

Depth of the tree $= \text{Depth of the solution} = \text{number of vars}$

---

Solving a CSP through standard search

- **What search algorithm to use?**
  Depth of the tree $= \text{Depth of the solution} = \text{number of vars}$
Solving a CSP through standard search

- **What search algorithm to use:** Depth first search
  - Since we know the depth of the solution
  - **DFS in context of CSP is also referred to as backtracking**

Checking constraint consistency

The violation of constraints needs to be checked for each node, either during its generation or before its expansion

**Consistency of constraints:**
- 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)
- To prevent “blind” search space exploration it is necessary to keep track of the remaining legal values, so we know when the constraints are violated and when to terminate the search
**Constraint propagation**

A **state** (more broadly) is defined by a set of variables and their legal and illegal assignments.

Legal and illegal assignments can be represented through variable **equations** and variable **disequations**.

**Example: map coloring**

Equation \( A = \text{Red} \)

Disequation \( C \neq \text{Red} \)

**Constraints + assignments**

Can entail new equations and disequations:

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

**Constraint propagation**: the process of inferring 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>

- equations ✓

- 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>

✓ - equations  ✗ - 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>

CS 2710 Foundations of AI
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>x</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>x</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

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>x</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>
### 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>

Conflict !!! No legal assignments available for B and C
Constraint propagation

- We can derive remaining legal values through propagation

<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>

B=Green
C=Green

Constraint propagation

- We can derive remaining legal values through propagation

<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>

B=Green  ➔  F=Red
C=Green

CS 2710 Foundations of AI
Constraint propagation

• We can derive remaining legal values through propagation

<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>

B=Green  F=Red
C=Green

Constraint propagation

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

• **Value propagation**
• **Arc consistency**
• **Forward checking**

• **Difference:**
  – Completeness of inferences
  – Time complexity of inferences.
**Constraint propagation**

1. **Value propagation.** Infers:
   - equations from the set of equations defining the partial assignment, and a 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)

**Value propagation: analysis**

**Value propagation.** Infers:
- equations from the set of equations defining the partial assignment, and a constraint

**Procedure:**
- At every step, after a new variable gets assigned a value we check if a constrain does not imply a new equation on yet to be assigned variable.
  - A set of equations may be inferred
Forward checking: analysis

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

**Procedure:**
- At every step, after a new variable gets assigned a value (a new equation is created) we check a constraint if it does not imply a new disequation on yet to be assigned variable.
- After all possible disequations are derived we check if a new equation is not implied by a set of disequations.
- And repeat till no new derivations can be made.

- A set of disequations and equations may be inferred

---

Arc consistency: analysis

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

**Procedure:**
- At every step, after a new equation or a disequation is generated we need to check constraints if a new disequation is not implied on yet to be assigned variable.
- After all possible disequations are derived we check if a new equation is not implied by a set of disequations.
- And repeat till no new derivations are possible.

- A set of disequations and equations may be inferred
Heuristics for CSPs

Backtracking 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