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

CSP example: 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.** Assignment (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 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: Q_1 = 1
```
```
Unassigned: Q_2, Q_3, Q_4
Assigned: Q_1 = 2
```
```
Unassigned: Q_2, Q_3
Assigned: Q_1 = 2, Q_2 = 4
```
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: $Q_1 = 2$

Unassigned: $Q_1, Q_4$
Assigned: $Q_1 = 2, Q_2 = 4$

Solving a CSP through standard search

- **What search algorithm to use**: ?
  Depth of the tree = Depth of the solution = number of vars

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

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

Unassigned: $Q_1, Q_4$
Assigned: $Q_1 = 2, Q_2 = 4$
Solving a CSP through standard search

- **What search algorithm to use:** Depth first search !!!
  - Since we know the depth of the solution
  - We do not have to keep large number of nodes in queues

Backtracking

**Depth-first search for CSP** is also referred to as **backtracking**

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)
Constraint propagation

A state (more broadly) is defined by a set of variables, their values and a list of legal and illegal assignments for unassigned variables. Legal and illegal assignments can be represented via equations (list of value assignments) and disequations (list of invalid assignments).

Example: map coloring

<table>
<thead>
<tr>
<th>Equation</th>
<th>$A = \text{Red}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disequation</td>
<td>$C \neq \text{Red}$</td>
</tr>
</tbody>
</table>

Constraints + assignments can entail new equations and disequations:

$A = \text{Red} \implies B \neq \text{Red}$

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

<table>
<thead>
<tr>
<th>Red</th>
<th>Blue</th>
<th>Green</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</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>X</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>X</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>X</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>X</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 $E=\text{Blue}$

<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 $F=\text{Green}$

<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 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.

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)
Example
Map coloring of Australia territories

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>
### 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><strong>domain</strong></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><strong>WA=Red</strong></td>
<td>R</td>
<td>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>

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><strong>domain</strong></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: forward checking**

**Map coloring**

```
<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>
```

**Set:** Q=Green

Infer: Exhaustions of alternatives

```
Infer NT:  
```

---

**Example: forward checking**

**Map coloring**

```
Infer NT:  
```

```
Infer: 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: 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>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>
Example: arc consistency

Map coloring

Set: WA=Red
Set: Q=Green

vars | WA | NT | Q | NSW | V | SA | T
--- | --- | --- | --- | --- | --- | --- | ---
domain | R G B | R G B | R G B | R G B | R G B | R G B | R G B
WA=Red | R | G B | R G B | R G B | R G B | G B | R G B
Q=Green | R | B | G | R B | R G B | B | R G B

Consistent assignment

SA=B
NSW=R
Example: arc consistency

Map coloring

Set: WA=Red
Set: Q=Green

vars | WA | NT | Q | NSW | V | SA | T
---|---|---|---|---|---|---|---

Consistent assignment

NSW=B
SA=!
**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>

NSW=B
SA=! 

Inconsistent assignment

---

**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?
Heuristics for CSP

Examples: map coloring

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

• Least constraining value
  – ?
Heuristics for CSP

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
  - What color is the least constraining color?
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

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**

In the following we cover combinatorial optimization problems. Parametric optimization will be covered next lecture.

**Example: Traveling salesman problem**

Problem:
- A graph with distances

  ![Graph Diagram]

  - **Goal:** find the shortest tour which visits every city once and returns to the start

  An example of a valid tour: ABCDEF
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)
  - Reallocation its position

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
**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
  ↓
ABCD | EF |
  ↓
ABCDFE
```

---

**Example: Traveling salesman problem**

“Local” operator:
- generates the next configuration (state)
Searching the configuration space

Search algorithms
- keep only one configuration (the current configuration)

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

Search algorithms

Two 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
  \[ \min f(X) \Leftrightarrow \max [-f(X)] \]
Hill climbing

- Look around at states in the local neighborhood and choose the one with the best value
- Assume: we want to maximize the

```
function Hill-Climbing(problem) returns a solution state
inputs: problem, a problem
state: current, a node
   next, a node

current ← MAKE-NODE(INITIAL-STATE[problem])
loop do
   next ← a highest-valued successor of current
   if VALUE[next] < VALUE[current] then return current
   current ← next
end
```

Hill climbing

- Always choose the next best successor state
- Stop when no improvement possible
Hill climbing

• Look around at states in the local neighborhood and choose the one with the best value

• What can go wrong?

Hill climbing

• Hill climbing can get trapped in the local optimum

• What can go wrong?
Hill climbing

- Hill climbing can get clueless on plateaus

Hill climbing and n-queens

- The quality of a configuration is given by the number of constraints violated
- Then: Hill climbing reduces the number of constraints
- Min-conflict strategy (heuristic):
  - Choose randomly a variable with conflicts
  - Choose its value such that it violates the fewest constraints

Success !! But not always!!! The local optima problem!!!