Constraint Satisfaction
Constraint Satisfaction
State

\[ h(n) \]
public class Board {
    protected Player[][] grid;
    public Board(int n) {
        grid = new Player[n][n];
    }
    ...
}

public class State {
    protected Board board;
    protected Player nextPlayer;
    ...
}

public Player getWinner() {  
    Player p;
    p = checkHorizontals();
    if (p != null) {  
        return p;
    }
    p = checkVerticals();
    if (p != null) {  
        return p;
    }
    p = checkDiagonals();
    if (p != null) {  
        return p;
    }
    return null;
}

protected Player checkHorizontals() {  
    for (int y=0; y < size; y++) {  
        Player p = checkHorizontal(y);
        if (p != null) {  
            return p;
        }
    }
    return null;
}

protected Player checkHorizontal(int y) {  
    return checkLine(0, y, 1, 0);
}

...
The Problem With States

- State representation is specific to a given problem (or domain of problems)
- Functions on states (successor generation, goal test) are specific to the state representation
- Heuristic functions are both problem-specific and dependent on the state representation
- Many design choices, many opportunities for coding errors
Approach

• Impose a structure on the representation of states

• Using that representation, successor generation and goal tests are problem- and domain-independent

• Can also develop effective problem- and domain-independent heuristics
Bottom Line

Represent State This Way → Write No Code! → No Bugs!
Assign a color to each region such that no two neighboring regions have the same color
Color WA, NT, Q, NSW, V, SA, T

def enum Color = red, green, blue
Color WA, NT, Q, NSW, V, SA, T

```c
enum Color = red, green, blue

WA=red, NT=red, Q=green, NSW=blue
V=red, SA=blue, T=green
```
Color WA, NT, Q, NSW, V, SA, T

enum Color = red, green, blue

State: assignment of colors to regions

Action: pick an unassigned region and assign it a color

\[ 7 \times 3 \times 6 \times 3 \times 5 \times 3 \times 4 \times 3 \times 3 \times 2 \times 3 \times 1 \times 3 = 7! \times 3^7 = 11,022,480 \]

\[ n! \times k^n \]
Color WA, NT, Q, NSW, V, SA, T

```c
enum Color = red, green, blue
```

State: assignment of colors to regions
Successor function: pick an unassigned region and assign it a color
Goal test: All regions assigned and no adjacent regions have the same color
Color WA, NT, Q, NSW, V, SA, T

def isGoal(State s) {
    if (s.WA != s.SA && s.WA != s.NT && s.NT != s.Q && s.NT != s.SA &&
        s.SA != s.Q && s.SA != s.NSW && s.SA != s.V && s.Q != s.NSW &&
        s.NSW != s.V && s.SA != s.V && s.Q != s.NSW &&
        s.NSW != s.V) {
        return true;
    } else {
        return false;
    }
}
Color WA, NT, Q, NSW, V, SA, T

```java
enum Color = red, green, blue
WA=red, NT=red, Q=green, NSW=blue
V=red, SA=blue, T=green
```
Color WA, NT, Q, NSW, V, SA, T

```c
enum Color = red, green, blue

WA=red, NT=green, Q=red, NSW=green
V=red, SA=blue, T=red
```
Constraint Satisfaction Problem (CSP)

X: Set of variables \{ X_1, ..., X_n \}

D: Set of domains \{ D_1, ..., D_n \}

Each domain D_i = set of values \{ v_1, ..., v_k \}

C: Set of constraints \{ C_1, ..., C_m \}
Factored Representation

- Splits a state into variables (or attributes) that can have values
- Factored states can be more or less similar (unlike atomic states)
- Can also represent uncertainty (don’t know value of some attribute)
Australia Map CSP

X: \{ X_i \} = \{ WA, NT, Q, NSW, V, SA, T \}

D: Each \( D_i \) = \{ red, green, blue \}

C: \{ SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, \\
SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, \\
VSW \neq V \}
More CSP Terminology

• Assignment: \{ X_i = v_i, X_j = v_j, \ldots \}

• Consistent: does not violate any constraints

• Partial: some variables are unassigned

• Complete: every variable is assigned

• Solution: consistent, complete assignment
Constraints

- Unary constraint: one variable
  - e.g., NSW ≠ red, \(X_i\) is even, \(X_i = 2\)

- Binary constraint: two variables
  - e.g., NSW ≠ WA, \(X_i > X_j\), \(X_i + X_j = 2\)

- “Global” constraint: more than two vars
  - e.g., \(X_i\) is between \(X_j\) and \(X_k\), \(\text{AllDiff}(X_i, X_j, X_k)\)

- Can be reduced to set of binary constraints (possibly inefficiently)
Constraint Satisfaction Problem (CSP)

X: Set of variables \{ X_1, ..., X_n \}
D: Set of domains \{ D_1, ..., D_n \}
Each domain \( D_i = \) set of values \{ v_1, ..., v_k \}
C: Set of constraints \{ C_1, ..., C_m \}
• Faster to search (solve)
• Problem-independent (no code!)
• Heuristics
<table>
<thead>
<tr>
<th>State</th>
<th>R, G, B</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NT</td>
<td>R, G, B</td>
</tr>
<tr>
<td>SA</td>
<td>R, G, B</td>
</tr>
<tr>
<td>Q</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NSW</td>
<td>R, G, B</td>
</tr>
<tr>
<td>V</td>
<td>R, G, B</td>
</tr>
<tr>
<td>T</td>
<td>R, G, B</td>
</tr>
<tr>
<td>State</td>
<td>Color Code</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
</tr>
<tr>
<td>WA</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NT</td>
<td>R, G, B</td>
</tr>
<tr>
<td>SA</td>
<td>B</td>
</tr>
<tr>
<td>Q</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NSW</td>
<td>R, G, B</td>
</tr>
<tr>
<td>V</td>
<td>R, G, B</td>
</tr>
<tr>
<td>T</td>
<td>R, G, B</td>
</tr>
<tr>
<td>State</td>
<td>Colors</td>
</tr>
<tr>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td>WA</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NT</td>
<td>R, G, B</td>
</tr>
<tr>
<td>SA</td>
<td>B</td>
</tr>
<tr>
<td>Q</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NSW</td>
<td>R, G, B</td>
</tr>
<tr>
<td>V</td>
<td>R, G, B</td>
</tr>
<tr>
<td>T</td>
<td>R, G, B</td>
</tr>
</tbody>
</table>

Remaining possibilities: $3^5 = 243$
<table>
<thead>
<tr>
<th>State</th>
<th>Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>R, G</td>
</tr>
<tr>
<td>NT</td>
<td>R, G</td>
</tr>
<tr>
<td>SA</td>
<td>B</td>
</tr>
<tr>
<td>Q</td>
<td>R, G</td>
</tr>
<tr>
<td>NSW</td>
<td>R, G</td>
</tr>
<tr>
<td>V</td>
<td>R, G</td>
</tr>
<tr>
<td>T</td>
<td>R, G, B</td>
</tr>
</tbody>
</table>

Remaining possibilities: \(2^5 = 32\)
Constraint Propagation

- Using the constraints to reduce the set of legal values of a variable, which can in turn reduce the legal values of another variable, and so on
- Not a search process!
- Part of state update in state-space search
- A type of inference: making implicit information explicit
Pros & Cons

• Good:
  • Can significantly reduce the space of assignments left to search

• Bad:
  • How long does it take to do the propagation?
Constraint Propagation ↔ State-Space Search
Node Consistency

• Apply all unary constraints
<table>
<thead>
<tr>
<th>State</th>
<th>Colour</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NT</td>
<td>R, G, B</td>
</tr>
<tr>
<td>SA</td>
<td>R, G, B</td>
</tr>
<tr>
<td>Q</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NSW</td>
<td>R, G, B</td>
</tr>
<tr>
<td>V</td>
<td>R, G, B</td>
</tr>
<tr>
<td>T</td>
<td>R, G, B</td>
</tr>
</tbody>
</table>

SA ≠ green
<table>
<thead>
<tr>
<th>State</th>
<th>Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>WA</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NT</td>
<td>R, G, B</td>
</tr>
<tr>
<td>SA</td>
<td>R, B</td>
</tr>
<tr>
<td>Q</td>
<td>R, G, B</td>
</tr>
<tr>
<td>NSW</td>
<td>R, G, B</td>
</tr>
<tr>
<td>V</td>
<td>R, G, B</td>
</tr>
<tr>
<td>T</td>
<td>R, G, B</td>
</tr>
</tbody>
</table>

SA ≠ green
<table>
<thead>
<tr>
<th></th>
<th>WA</th>
<th>NT</th>
<th>SA</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>T</th>
</tr>
</thead>
</table>

**INCONSISTENT**

SA ≠ green
SA ≠ red
SA ≠ blue
Inconsistency

- Empty domain for any variable
- No possible values for that variable
- No possible assignment including that variable
- No possible solution!
Node Consistency

• Apply all unary constraints

• If problem is not inconsistent, then we can always propagate unary constraints at the start

• And then we can ignore them

• Complexity: $O(nd)$ for $n$ variables, domains at most $d$
Arc Consistency
Arc Consistency

$X_i$ is arc-consistent w.r.t. $X_j$ if
for every value in the domain $D_i$,
there is some value in the domain $D_j$
that satisfies the binary constraint on the arc $(X_i, X_j)$
Arc Consistency

\[ Y = X^2 \]

\{ 0,1,2,3,4,5,6,7,8,9 \} \quad \{ 0,1,2,3,4,5,6,7,8,9 \}
Arc Consistency
Arc Consistency

\[ X \text{ arc-consistent with respect to } Y \]

\[ \{0, 1, 2, 3\} \quad Y = X^2 \quad \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \]
Arc Consistency

\[ Y^2 = X \]

\{ 0, 1, 2, 3 \} \quad \{ 0, 1, 4, 9 \}

\( Y \) arc-consistent with respect to \( X \)

Thursday, February 28, 13
Arc Consistency

\{ 0,1,2,3 \} \quad Y = X^2 \quad \{ 0,1,4,9 \}

Thursday, February 28, 13
boolean AC3(csp) {
    Set queue = all arcs in csp
    while (queue is not empty) {
        <i,j> = queue.removeFirst()
        if (revise(csp, i, j)) {
            if Di is empty {
                return false
            }
            foreach k in neighbors(i) {
                add <k,i> to queue
            }
        }
    }
    return true
}

boolean revise(csp, i, j) {
    boolean changed = false
    boolean ok = false
    foreach vj in Dj {
        if (<vi,vj> satisfies Cij )
            ok = true
    }
    foreach vi in Di {
        if (!ok) {
            delete vi from Di
            changed = true
        }
    }
    return changed
}
AC-3 Analysis

- CSP with $n$ variables, domain size $\leq d$, $c$ constraints (arcs)
- Each arc can be inserted in the queue at most $d$ times
- Checking a single arc takes $O(d^2)$ time
- Total time: $O(cd^3)$
More Constraint Propagation

- Path consistency
- $k$-consistency
  - Generalization of node- (1-), arc (2-), and path (3-) consistency
  - Establishing $k$-consistency is exponential in $k$
  - Typically use arc-consistency and rarely path-consistency
Constraint Propagation

• Bottom line: “After constraint propagation, we are left with a CSP that is equivalent to the original CSP—they both have the same solutions—but the new CSP will in most cases be faster to search because its variables have smaller domains.”
Constraint Propagation  State-Space Search
State-Space Search for CSPs

- **State**: assignment of values to variables
- **Initial state**: all variables unassigned
- **Action**: Assign value to variable
\[ X_1 = \emptyset, X_2 = \emptyset, \ldots, X_n = \emptyset \]

\[ X_1 = v_1, X_2 = \emptyset, \ldots, X_n = \emptyset \]
\[ X_1 = \emptyset, X_2 = v_1, \ldots, X_n = \emptyset \]
\[ X_1 = \emptyset, X_2 = \emptyset, \ldots, X_n = v_1 \]
\[ X_1 = v_2, X_2 = \emptyset, \ldots, X_n = \emptyset \]
\[ X_1 = \emptyset, X_2 = v_2, \ldots, X_n = \emptyset \]
\[ X_1 = \emptyset, X_2 = \emptyset, \ldots, X_n = v_2 \]
\[ \ldots \]

Total number of nodes searched:
\[ n \cdot d \times (n - 1) \cdot d \times \ldots \times 1 \cdot d = n! \cdot d^n \]

\# of possible complete assignments: \( d^n \)
Commutativity

commutative  |ˈkämyəˌtætiv%;  kəˈmyoʊətətiv|

adjective Mathematics
involving the condition that a group of quantities connected by operators gives the same result whatever the order of the quantities involved, e.g., $a \times b = b \times a$. 
CSPs are Commutative

- CSPs are commutative because we reach the same partial assignment regardless of order
- Need only consider assignment to a single variable at each node in the search tree
$n$ levels (one per variable), at most $d$ nodes per level:

$$d^n$$
• No legal choice
• Empty domain
• Inconsistent partial assignment
• Cannot be extended to a complete, consistent assignment
function BT(csp)
    return backtrack({}, csp)

function backtrack(assignment, csp)
    if (assignment is complete)
        return assignment
    var = SelectUnassignedVar(csp)
    foreach value in OrderDomainValues(var, assignment, csp)
        if (value is consistent with assignment)
            add <var, value> to assignment
            result = backtrack(assignment, csp)
            if (result != failure)
                return result
            else
                remove <var, value> from assignment
    return failure
Backtracking Search

• DFS search through the space of assignments

• Assign one variable at a time

• Because the representation of CSPs is standardized, no need to supply initial state, action function, transition model, or goal test!
Heuristics for CSPs

- Minimum-remaining values (most constrained variable)
- Degree heuristic (variable involved in most constraints with unassigned variables)
- Least constraining value (if we only want to find one solution)
Interleaving Search and Inference

• After each choice during search, we can perform inference to reduce future search

• Forward checking

• MAC: Maintaining Arc Consistency

• Bottom line: Cost of inference is subsumed by cost of search, so do it
Interleaving Search and Inference

• Search through space of assignments

• Commutativity => Only have to consider assignment to one variable at a time

• Constraint propagation to reduce domains of variables for subsequent search
Other CSP Topics

- Intelligent backtracking
- Local search
CSPs Summary

• Impose a structure on the representation of states: Variables, Domains, Constraints

• Backtracking (DFS) search for complete, consistent assignment of values to variables

• Inference (constraint propagation) can reduce the domains of variables

• Preprocessing or interleaved with search

• Useful problem-independent heuristics
CSP Secret Sauce

- Factored representation of state:
  - Variables, Domains, Constraints

- Allows:
  - Early pruning of inconsistent states
  - Inference during update to reduce alternatives
For next time:
AIMA Chapter 7.0-7.4