The story up to now...

- Uninformed Search
  - BFS, DFS, IDS

- Informed Search
  - A*, SMA*

- Local Search
  - Hill Climbing
Today

- Constraint Satisfaction Problems
  - Examples
  - Definitions

- Making smart choices going forward
  - Minimum Remaining Values heuristic
  - Forward checking
  - Constraint Propagation

- Making smart choices going backward (when we get stuck)
  - Backjumping

- Local Search Strategies

States as Black Boxes

- Search methods so far impose minimal requirements on states:
  - Generate successors
  - Evaluate domain-specific heuristics
  - Apply goal test

- From point of view of search algorithm, states are **black boxes** — no relevant internal structure
Exploiting Structure in States

- Constraint Satisfaction Problems (CSPs)
  - Standard, structured, and simple representation
  - Enabling use of general-purpose algorithms
  - Achieving performance improvements without domain-specific heuristics

Variables, Domains, Constraints

- Variables
  - \{entrée, dessert\}

- Domains
  - The set of values a variable can take
  - \entrée \in \{ steak, fish, lasagne \}
  - \dessert \in \{ pie, jello, ice cream \}

- Constraints
  - A relationship between two or more variables
  - \text{calories(entrée)} + \text{calories(dessert)} < 1000
  - \text{calium(entrée)} + \text{calium(dessert)} > 100
Example: 8 Queens

Find an arrangement of queens such that no queen attacks another

8 Queens as CSP

- **Variables**, $X_1, \ldots, X_n$
  - One for each queen ($n=8$)
  - Assume one queen per column

- **Variable domains**
  - Row location of queen in column $i$, $X_i \in \{1, \ldots, 8\}$

- **Constraints**, $C_1, \ldots, C_m$
  - No queens can attack each other
    - $X_i \neq X_j$, $i \neq j$
    - $X_i \neq X_j + k$, $|i - j| = k$
Example: Map Labeling

Map Labeling as CSP

- **Variables**
  - City label locations

- **Domains**
  - {NW, NE, SW, SE}

- **Constraints**
  - Labels of nearby cities do not overlap

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AA</td>
<td>Detroit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NW</td>
<td>NE, SE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NE</td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SW</td>
<td>NW, NE, SE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SE</td>
<td>NW, NE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legal assignments
Example: 3SAT

\[(P_1 \lor \neg P_2 \lor \neg P_3) \land (P_1 \lor \neg P_2 \lor \neg P_4) \land (P_1 \lor \neg P_3 \lor \neg P_4) \land \]
\[(-P_1 \lor P_2 \lor \neg P_3) \land (P_2 \lor \neg P_3 \lor \neg P_4) \land (-P_1 \lor P_2 \lor \neg P_4) \land \]
\[(-P_1 \lor \neg P_2 \lor P_3) \land (-P_1 \lor P_3 \lor \neg P_4) \land (-P_2 \lor P_3 \lor \neg P_4) \land \]
\[(-P_1 \lor \neg P_2 \lor P_4) \land (-P_1 \lor \neg P_3 \lor P_4) \land (-P_2 \lor \neg P_3 \lor P_4) \land \]
\[(-P_1 \lor \neg P_2 \lor \neg P_3) \]

Variables?
Domains?
Constraints?

Other examples?

- What other CSPs can you think of?
Example: Crossword Puzzle

Example: Sudoku
Cryptarithmetic

- Variables
  - letters
- Domains
  - \{0,\ldots,9\}
- Constraints
  - Columns have to add up right, including carries
  - Letters stand for distinct digits
  - S, M are non-zero

CSPs as Search (“F” grade)

- For uninformed search formulation we need: 
  \{state_0, successors(n), is-goal(n), path-cost(n)\}
  
  - state: Assignments for each variable.
  - state_0: No variables yet assigned
  - successors(n): All possible variable assignments for all unassigned variables
  - is-goal(n): All variables assigned satisfying all constraints?
  - path-cost(n): any constant value

- Which search algorithm?
- Run time?
CSPs as Search ("D")

- For uninformed search formulation we need:
  
  \{ state_0, successors(n), is-goal(n), path-cost(n) \}

  - state: Assignments for each variable.
  - state_0: No variables yet assigned
  - successors(n): All consistent possible variable assignments for all unassigned variables
  - is-goal(n): All variables assigned?
  - path-cost(n): any constant value

- Run time?

Cryptarithmetic Search Tree ("D")

- Branching factor
  - \(10^n\) at level 1
  - \(9(n-1)\) at level 2
  - \(8(n-2)\) at level 3, ...

- # of leaves = \(n!10\)

- But only \(10^n\) complete assignments!
Another Key Observation

- The consistency of an assignment depends only on the values assigned to the variables.
- The order in which the variables were assigned is irrelevant! (Commutivity)

CSPs as Search (“C”)

- For uninformed search formulation we need: 
  \{state_0, successors(n), is-goal(n), path-cost(n) \}

  - state: Assignments for each variable.
  - state_0: No variables yet assigned
  - successors(n): All consistent possible variable assignments for a single unassigned variables
  - is-goal(n): All variables assigned?
  - path-cost(n): any constant value

- Run time?
Cryptarithmetic Search Tree ("C")

**SEND + MORE**

**MONEY**

<table>
<thead>
<tr>
<th>S = 0</th>
<th>S = 1</th>
<th>S = 2</th>
<th>S = 3</th>
<th>S = 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>E = 1</td>
<td>E = 2</td>
<td>E = 8</td>
<td>E = 9</td>
<td></td>
</tr>
</tbody>
</table>

Branching factor:
- 10 at level 1
- 9 at level 2
- 8 at level 3, ...

Picking which variable to expand

- **Plain backtracking** picks next variable arbitrarily
  - Inefficient!

- **MRV (minimum remaining values)** heuristic
  - Choose variable with fewest legal values remaining
  - aka *most constrained variable*
  - If any variable has no legal values, MRV will choose that and detect failure immediately

- **Degree** heuristic
  - choose variable with largest number of constraints on unassigned variables
How many values remain?

- How do we compute which values remain for a variable?
  - Determining this exactly requires solving the problem!
  - How can we efficiently reduce the size of the domain?

How many values remain?

- Forward Checking
- Arc Consistency
- k-Consistency
- MAC Consistency

*Don’t worry, these are all basically the same idea, applied to varying extremes!*
Forward Checking (FC)

- Whenever a variable $X$ is assigned
  - Examine each unassigned variable $Y$ connected to $X$ by a constraint
  - Delete from $Y$'s domain any value inconsistent with the value chosen for $X$
  - If assignment becomes impossible (anywhere), backtrack.

Forward Checking Example

Assign: WA=R
Assign: Q = G
Assign: Q = G
Arc Consistency

- Basic idea:
  - Whenever we reduce the domain for a node, reprocess the edges it’s connected to.

- Reprocessing: for an edge between (A,B)
  - remove any values from Dom(A) for which there is no value in Dom(B) that satisfies the edge.
  - and vice-versa
  - (If domains got smaller, we must reprocess more edges!)

- Arc Consistency is very powerful
  - Can solve many problems by itself!

Arc Consistency: Run-Time

- Processing a single arc:
  - O(d^2): (for each value in A, check each value in B)

- Each arc processed at most _____ times
  - 2d - 1: Arcs only reprocessed when Dom(A) or Dom(B) gets smaller... at worst one value at a time. (But we know to stop when Dom{}=0).

- At most ____ arcs/edges (fully connected)
  - n(n-1)/2 (fully connected)

- Total runtime: O(n^2d^3)
  - Remember: CSP includes 3SAT, which is NP-complete.
  - How can Arc Consistency be polynomial time?
**k-Consistency**

- **k-consistent:**
  - For any consistent assignment of \( k - 1 \) variables, exists consistent value of any \( k \)th

- **Strongly k-consistent:**
  - \( j \)-consistent for all \( j \leq k \)

- **Special cases**
  - \( k = 1 \): node consistency (maintained by FC)
  - \( k = 2 \): arc consistency
  - \( k = n \): Problem is (almost) solved.

- Can choose to enforce higher-order consistency after each assignment.
  - Albeit at greater computational costs.

**Your turn: Special Constraints**

- **Goal:** Want to be able to enforce constraints at the highest possible level in the search tree in order to maximize pruning.

- Assume all variables have an integer domain \( \{1,9\} \) and that you know the current set of permissible values for each variable.

- Reformulate these constraints so that they can be applied as early as possible in the search tree:
  - (Note: there may be different constraints that you can apply at different levels!)

- Assume domain of all variables is initially \( \{1-9\} \)

  1. All-Different\( (x_1, x_2, x_3, x_4) \)
  2. All-Different\( (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9) \)
  3. All-Same\( (x_1, x_2, x_3) \);
  4. \( \text{Sum}(x_1, x_2, x_3, x_4) = 30 \)
  5. \( \text{Sum}(x_1, x_2, x_3, x_4) \) is Odd
  6. \( \text{Product}(x_1, x_2, x_3, x_4) = 18 \)
  7. \( \text{IsPrime}(100*x_1 + 10*x_2 + x_3) \)
Picking values

- We’ve talked a lot about which variable to substitute...

- Does it matter which order we try the values in the domain?
  - Yes! If we guess well, we’ll find a solution faster.

Picking Values

- Least constrained value
  - Which value rules out the fewest values nearby?
  - Pursue most promising directions first

- Other heuristics
  - Most probable *a priori*
  - Cryptograms: for a given ciphertext word, try common plaintext words first.
MRV vs. LCV?

- Minimum Remaining Values
  - Pick variable with fewest values left in its domain

- Least Constrained Value:
  - Pick value with most possible children

- Why do we maximize one and minimize the other?
  - To solve the problem, we must eventually assign every variable, so pick the one with the smallest branching factor (MRV).
  - Once we’ve picked a variable, we must ultimately rule out all possibilities, so look for most promising values first.
  - (Picking an unpromising avenue is just going to waste time)

BT Refinement: Perspectives

Look Back
- Reasoning about what to do after failure
- Backjumping
  - Backtrack to some decision before most recent

Look Ahead
- Reasoning about what to do after successful assignment
- Examples
  - Ordering heuristics
  - Constraint propagation: FC, MAC, ... MkC
Basic Back Goal

- Our goal is to jump back up as far as possible
  - Safe jump: don’t miss a solution
  - Know that the sub-tree we skip is unsolvable

Basic Back Jumping

- **Conflict Set(\(X_i\))**: For each value in Dom\(\{X_i\}\), what is the earliest variable that is inconsistent with it?
  - Suppose Dom\(\{X_i\}\) = \{\(v_1, v_2\}\)
  - Suppose \(X_i=v_1\) is incompatible with the current values of \(X_3\) and \(X_5\). We’d have to go all the way back up to \(X_3\) to make \(X_i=v_1\) possible.
  - Suppose \(X_i=v_2\) is incompatible with the current values of \(X_4\), \(X_5\), and \(X_6\). We’d have to go all the way back up to \(X_4\) to make \(X_i=v_2\) possible.
  - Conflict Set\(\{X_i\}\) = \{\(X_3, X_4\}\)

- We backjump to \(X_4\): a different value of \(X_4\) might allow an assignment of \(X_i (X_i = v_2)\).

*Forward Checking?*
6-Queens Example

from (Kondrak & van Beek, 1997)
6-Queens Example

Q₆ conflict set = \{1,2,3,5\}, Jump back to 5…

6-Queens Example

Q₅: nothing left to try. back up.
### 6-Queens Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q₄: Try row 6.

### 6-Queens Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Q₅: Try row 4.
### 6-Queens Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Q</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1</td>
<td>Q</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>1</td>
<td>Q</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Q</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
<td>Q</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Q₆ conflict set = \{1,2,3,4\}

### 6-Queens Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Q</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1</td>
<td>Q</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>1</td>
<td>Q</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Q</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
<td>Q</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Backjump to Q₄
Conflict-Directed Backjumping (CBJ)

- Maintain conflict set containing past variables that failed consistency checks with current instantiation
- Backtrack (or backjump) to most recent variable $Z$ in conflict set
- On backjump, transfer conflict set (not counting $Z$)

6-Queens Example

```
  1   1   Q  3  2
  2   Q  1 1 1 1 1
  3   1   Q 2 3 3
  4   1   3
  5   Q  2 1 2 2
  6   2   1 3
```

from (Kondrak & van Beek, 1997)
### 6-Queens Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>Q</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>Q</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Q</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Q</td>
<td></td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1</td>
<td>3</td>
<td>Q</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>Q</td>
<td>2</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>2</td>
<td></td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

1 2 3 4 5 6

Q₆ conflict set = \{1,2,3,5\}, Jump back to 5…
6-Queens Example

Combine Q6 conflicts with local conflicts
\{1,2,3\} = \{1,2,3,5\} U \{1,2,3\} - 5

6-Queens Example

Backjump to Q3!
Conflict-based BackJump Wrap-Up

- By transferring conflict set, we preserve key information across backtracks, pruning a much larger part of the search space.

Disconnected Sub-graphs
Sub problems

- Finding independent sub-problems is rare, but wonderful
  - Original problem: $O(d^n)$
  - Split in two equal-sized sub-problems: $O(d^{n/2})$

Your turn

- Let ‘abcde’ be a 5 digit number (a!=0) such that:
  - $a = 3b$
  - $b = 3^c$
  - $a = d+1$
  - $d = 2e$

- Is this an easy or hard problem?
Trees

- Trees are great too!

- Starting from the leaves:
  - Apply arc consistency to the parent, removing values from parent domain.
  - Now, the leaves can always find a value consistent with their parent.

- Start from the root:
  - Pick any value for the node consistent with its parent.

- Runtime?
  - $nd^2 + nd$

Trees

- Let ‘abcde’ be a 5 digit number (a!=0) such that:
  - $a = 3b$
  - $b = 3^c$
  - $a = d+1$
  - $d = 2e$

There are two solutions! 31021 and 93184
Transforming Problems into Trees

It’s a tree if we could remove SA!

Tree-ification

- Pick nodes $S$ that turn the problem into a tree

- For all possible assignments to $S$
  - Solve the induced tree
Local Search

- Local search is applicable to CSP too!

- Advantages
  - Can be very fast
  - Replanning
    - Produces solutions similar to earlier solutions

Local Search for CSPs

- Search in space of complete assignments

- Min-conflicts heuristic
  - Choose variable to reassign
  - Pick value minimizing number of conflicts with neighbors in constraint graph

- For $n$-queens, search time empirically independent of $n$
  - Solutions are fairly densely distributed around the state space: any initial guess never has far to go!
Next Time

- Logical Agents
  - Construct a world containing knowledge
  - Rules of inference
  - Ask agent to deduce things