**Solving CSPs**

Solving CSPs involves some combination of:

1. Constraint propagation, to eliminate values that could not be part of any solution
2. Search, to explore valid assignments

**Constraint Propagation (aka Arc Consistency)**

Arc consistency eliminates values from domain of variable that can never be part of a consistent solution.

\[ V_i \rightarrow V_j \]

Directed arc \((V_i, V_j)\) is arc consistent if \(\forall x \in D_i \exists y \in D_j\) such that \((x,y)\) is allowed by the constraint on the arc
Constraint Propagation (aka Arc Consistency)

Arc consistency eliminates values from domain of variable that can never be part of a consistent solution.

\[ V_i \rightarrow V_j \]

Directed arc \((V_i, V_j)\) is arc consistent if

\[ \forall x \in D_i \exists y \in D_j \text{ such that } (x,y) \text{ is allowed by constraint} \]

We can achieve consistency on arc by deleting values from \(D_i\) (domain of variable at tail of constraint arc) that fail this condition.

Assume domains are size at most \(d\) and there are \(e\) binary constraints.

A simple algorithm for arc consistency is \(O(ed^3)\) – note that just verifying arc consistency takes \(O(d^2)\) for each arc.
Constraint Propagation Example

Graph Coloring
Initial Domains are indicated

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each undirected constraint arc is really two directed constraint arcs, the effects shown above are from examining BOTH arcs.
Constraint Propagation Example

Graph Coloring
Initial Domains are indicated

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 - V_2$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 - V_3$</td>
<td>$V_1(G)$</td>
</tr>
</tbody>
</table>

Constraint Propagation Example

Graph Coloring
Initial Domains are indicated

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 - V_2$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 - V_3$</td>
<td>$V_1(G)$</td>
</tr>
</tbody>
</table>
Constraint Propagation Example

Graph Coloring
Initial Domains are indicated

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 - V_2$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
</tbody>
</table>

Constraint Propagation Example

Graph Coloring
Initial Domains are indicated

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 - V_2$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_1 - V_2$</td>
<td>$V_1(R)$</td>
</tr>
</tbody>
</table>
Constraint Propagation Example

Graph Coloring
Initial Domains are indicated

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 - V_2$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 - V_3$</td>
<td>$V_3(G)$</td>
</tr>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_1 - V_2$</td>
<td>$V_1(R)$</td>
</tr>
<tr>
<td>$V_1 - V_3$</td>
<td>none</td>
</tr>
<tr>
<td>$V_2 - V_3$</td>
<td>none</td>
</tr>
</tbody>
</table>
But, arc consistency is not enough in general

Graph Coloring

arc consistent but no solutions

arc consistent but 2 solutions B,R,G ; B,G,R.
But, arc consistency is not enough in general

Graph Coloring

- arc consistent but **no** solutions
- arc consistent but 2 solutions \( \text{B, R, G} \); \( \text{B, G, R} \).
- arc consistent but 1 solution
  - Assume B, R not allowed

Need to do search to find solutions (if any)
Searching for solutions – backtracking (BT)

When we have too many values in domain (and/or constraints are weak) arc consistency doesn’t do much, so we need to search. Simplest approach is pure backtracking (depth-first search).

V₁ assignments

V₂ assignments

V₃ assignments

Inconsistent with V₁ = R

Backup at inconsistent assignment
Searching for solutions – backtracking (BT)

When we have too many values in domain (and/or constraints are weak) arc consistency doesn’t do much, so we need to search. Simplest approach is pure backtracking (depth-first search).

- **V₁ assignments**
  - Inconsistent with V₁ = R

- **V₂ assignments**
  - Backup at inconsistent assignment

- **V₃ assignments**
  - Backup at inconsistent assignment

Inconsistent with V₂ = G
Searching for solutions – backtracking (BT)

When we have too many values in domain (and/or constraints are weak) arc consistency doesn’t do much, so we need to search. Simplest approach is pure backtracking (depth-first search).

Combine Backtracking & Constraint Propagation

A node in BT tree is partial assignment in which domain of variables has been set (tentatively) to singleton set.

Use constraint propagation (arc-consistency) to propagate effect of this tentative assignment i.e. eliminate values inconsistent with current values.
Combine Backtracking & Constraint Propagation

A node in BT tree is **partial** assignment in which domain of variables has been set (tentatively) to singleton set.

Use constraint propagation (arc-consistency) to propagate effect of this tentative assignment i.e. eliminate values inconsistent with current values.

**Question:** How much propagation to do?

**Answer:** Not much, just local propagation from domains with unique assignments, which is called forward checking (FC). This conclusion is not necessarily obvious, but it generally holds in practice.
Backtracking with Forward Checking (BT-FC)

When examining assignment $V_i = d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

```
V_1 assignments
V_2 assignments
V_3 assignments
```

```
R
V_1

R, G
V_2

R, G
V_3
```
**Backtracking with Forward Checking (BT-FC)**

When examining assignment $V_i = d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

- $V_1$ assignments
- $V_2$ assignments
- $V_3$ assignments

We have a conflict whenever a domain becomes empty.

When backing up, need to restore domain values, since deletions were done to reach consistency with tentative assignments considered during search.
Backtracking with Forward Checking (BT-FC)

When examining assignment $V_i = d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

V1 assignments

V2 assignments

V3 assignments

Backtracking with Forward Checking (BT-FC)

When examining assignment $V_i = d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

V1 assignments

V2 assignments

V3 assignments
Backtracking with Forward Checking (BT-FC)

When examining assignment $V_i = d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

![Diagram of constraint graph with nodes labeled V1, V2, and V3, and edges connecting them with values R, G, and B.](image)

- $V_1$ assignments
- $V_2$ assignments
- $V_3$ assignments
Backtracking with Forward Checking (BT-FC)

When examining assignment $V_i = d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

When examining assignment $V_i = d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.
When examining assignment $V_i = d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.
### Backtracking with Forward Checking (BT-FC)

When examining assignment $V_i = d_k$, remove any values inconsistent with that assignment from neighboring domains in constraint graph.

- **$V_1$ assignments**
- **$V_2$ assignments**
- **$V_3$ assignments**

No need to check previous assignments. Generally preferable to pure BT.

### BT-FC with dynamic ordering

Traditional backtracking uses fixed ordering of variables & values, e.g., random order or place variables with many constraints first.

You can usually do better by choosing an order dynamically as the search proceeds.

- **Most constrained variable**
  
  when doing forward-checking, pick variable with fewest legal values to assign next (minimizes branching factor)

- **Least constraining value**
  
  choose value that rules out the smallest number of values in variables connected to the chosen variable by constraints.

E.g., this combination improves feasible n-queens performance from about $n = 30$ with just FC to about $n = 1000$ with FC & ordering.
Which country should we color next

What color should we pick for it?

E most-constrained variable (smallest domain)

RED least-constraining value (eliminates fewest values from neighboring domains)