Constraint Programming II: Solving CPs using Propagation and Basic Search

Slides draw upon material from:
6.034 notes, by Tomas Lozano Perez;
AIMA, by Stuart Russell & Peter Norvig;
Constraint Processing, by Rina Dechter.

Brian C. Williams
Enrique Fernandez
16.410/413
October 28th, 2015
Assignments

• Remember:
  • Problem Set #6: Out today. Due next Wednesday
  • Project Part 1 (16.413): Due on Nov 6th

• Reading:
  • Today and Monday:
    [AIMA] Ch. 6.2-5; Constraint Satisfaction.

• To Learn More: Constraint Processing, by Rina Dechter.
  • Ch. 5: General Search Strategies: Look-Ahead.
  • Ch. 6: General Search Strategies: Look-Back.
  • Ch. 7: Stochastic Greedy Local Search.
## Midterm results

<table>
<thead>
<tr>
<th></th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>42</td>
<td>31</td>
<td>30</td>
<td>93</td>
</tr>
<tr>
<td>Min</td>
<td>12</td>
<td>4</td>
<td>0</td>
<td>35</td>
</tr>
<tr>
<td>Avg</td>
<td>31</td>
<td>20</td>
<td>18</td>
<td>69</td>
</tr>
<tr>
<td>Median</td>
<td>32</td>
<td>21</td>
<td>18</td>
<td>70</td>
</tr>
<tr>
<td>Std</td>
<td>6.45</td>
<td>7.73</td>
<td>6.36</td>
<td>14.98</td>
</tr>
</tbody>
</table>

| #     | 0  | 0  | 1  | 0    |

---

**Problem 1**
- mean: 30.8, median: 31.5, std: 6.5, max: 42, min: 12

**Problem 2**

**Problem 3**
- mean: 17.7, median: 18.0, std: 6.4, max: 30, min: 0

---

Fall 2015 16.410 Midterm Grade
- mean: 68.5, median: 69.5, std: 15.0, max: 35, min: 93
### Problem 1

#### 1-A : BFS Search

<table>
<thead>
<tr>
<th>Grade</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>

- mean: 4.6, median: 5.0, std: 0.9, max: 5, min: 1

#### 1-B : Search Issues

<table>
<thead>
<tr>
<th>Grade</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
</tr>
</tbody>
</table>

- mean: 12.1, median: 13.0, std: 2.6, max: 15, min: 7

#### 1-C : Complexity

<table>
<thead>
<tr>
<th>Grade</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

- mean: 4.8, median: 4.0, std: 3.7, max: 10, min: 0

#### 1-D : RRT

<table>
<thead>
<tr>
<th>Grade</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>1</td>
</tr>
</tbody>
</table>

- mean: 9.3, median: 11.0, std: 3.6, max: 12, min: 0
<table>
<thead>
<tr>
<th></th>
<th>2-A</th>
<th>2-B</th>
<th>2-C</th>
<th>P2 (Total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>8</td>
<td>8</td>
<td>15</td>
<td>31</td>
</tr>
<tr>
<td>Min</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Avg</td>
<td>6</td>
<td>5</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>Median</td>
<td>7</td>
<td>5</td>
<td>12</td>
<td>21</td>
</tr>
<tr>
<td>Std</td>
<td>2.66</td>
<td>2.43</td>
<td>5.14</td>
<td>7.73</td>
</tr>
<tr>
<td># 0s</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

2-A : Heuristics
- mean: 5.5, median: 6.5, std: 2.7, max: 8, min: 0

2-B : Search Properties
- mean: 4.8, median: 5.0, std: 2.4, max: 8, min: 0

2-C : FF heuristic
- mean: 9.8, median: 12.0, std: 5.1, max: 15, min: 0

Problem 2
<table>
<thead>
<tr>
<th></th>
<th>3-A</th>
<th>3-B</th>
<th>P3 (Total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>15</td>
<td>15</td>
<td>30</td>
</tr>
<tr>
<td>Min</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Avg</td>
<td>10</td>
<td>7</td>
<td>18</td>
</tr>
<tr>
<td>Median</td>
<td>10</td>
<td>8</td>
<td>18</td>
</tr>
<tr>
<td>Std</td>
<td>3.68</td>
<td>4.92</td>
<td>6.36</td>
</tr>
<tr>
<td># 0s</td>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

**Problem 3**

- Mean: 17.7, Median: 18.0, Std: 6.4, Max: 30, Min: 0

**3-A: MDP**

- Mean: 10.4, Median: 10.0, Std: 3.7, Max: 15, Min: 0

**3-B: HMM**

- Mean: 7.2, Median: 7.5, Std: 4.9, Max: 15, Min: 0

16.410/16.413 Fall 2015
Constraint Problems are Everywhere

(a) Sudoku Puzzle

(b) The Solution
Constraint Satisfaction Problems (CSP)

Input: A Constraint Satisfaction Problem is a triple \(<V,D,C>\), where:
- \(V\) is a set of variables \(V_i\),
- \(D\) is a set of variable domains,
  - The domain of variable \(V_i\) is denoted \(D_i\),
- \(C = \{C_i\}\) is a set of constraints on assignments to \(V\),
  - Each constraint \(C_i = <S_i,R_i>\) specifies allowed variable assignments,
  - \(S_i\) the constraint’s scope, is a subset of variables \(V\),
  - \(R_i\) the constraint’s relation, is a set of assignments to \(S_i\).

Output: A full assignment to \(V\), from elements of \(V\)’s domain, such that all constraints in \(C\) are satisfied.

\[\begin{align*}
V & = \{V_1, V_2, V_3\} \\
D_1 & = \{R,G,B\} \\
C_{12} & = \{<R,G>, <G, R>, <B, R>, <B, G>\}
\end{align*}\]
Constraint Modeling (Programming) Languages

Features: Declarative specification of the problem that separates the formulation and the search strategy.

Example: Constraint Model of the Sudoku Puzzle in Number Jack (http://4c110.ucc.ie/numberjack/home).

```python
matrix = Matrix(N*N,N*N,1,N*N)
sudoku = Model( [AllDiff(row) for row in matrix.row],
                [AllDiff(col) for col in matrix.col],
                [AllDiff(matrix[x:x+N, y:y+N].flat)
                 for x in range(0,N*N,N)
                 for y in range(0,N*N,N)] )
```

16.410/16.413 Fall 2015
Constraint Problems are Everywhere

(a) Sudoku Puzzle
N-Queens

Place queens so that no queen can attack another.

Encoding

- Assume one queen per column.
- Determine what row each queen should be in.

Variables \( Q_1, Q_2, Q_3, Q_4 \).

Domains \( \{1, 2, 3, 4\} \).

Constraints \( Q_i <> Q_j \) "On different rows”.
\[ |Q_i - Q_j| <> |i-j| \] "Stay off the diagonals”.

Example \( C_{1,2} = \{(1,3) (1,4) (2,4) (3,1) (4,1) (4,2)\} \).
Outline

• Arc-consistency and constraint propagation.
• Analysis of constraint propagation.
• Solving CSPs using search.
Good News / Bad News

**Good News**
- Very general & interesting family of problems.
- Problem formulation used extensively in autonomy and decision making applications.

**Bad News**
Includes NP-Hard (intractable ?) problems.
Algorithmic Design Paradigm

Solving CSPs involves a combination of:

1. **Inference**
   - Solves partially by eliminating values that can’t be part of any solution (constraint propagation).
   - Makes implicit constraints explicit.

2. **Search**
   - Tries alternative assignments against constraints.
N-Queens

Inference

Eliminate values that can’t be part of any solution

Search

Try alternative assignments against constraints
Arc Consistency

Idea: Eliminate values of a variable domain that can *never satisfy* a specified constraint (an arc).

Definition: arc \(<x_i, x_j>\) is arc consistent if \(<x_i, x_j>\) and \(<x_j, x_i>\) are directed arc consistent.
Arc Consistency
Definition: arc $\langle x_i, x_j \rangle$ is directed arc consistent if

- for every $a_i$ in $D_i$,
  - there exists some $a_j$ in $D_j$ such that
    - assignment $\langle a_i, a_j \rangle$ satisfies constraint $C_{ij}$,
  - $\forall a_i \in D_i, \exists a_j \in D_j$ such that $\langle a_i, a_j \rangle \in C_{ij}$
- $\forall$ denotes “for all,” $\exists$ denotes “there exists” and $\in$ denotes “in.”
Revise: A directed arc consistency procedure

Definition: arc $<x_i, x_j>$ is directed arc consistent if
$\forall a_i \in D_i, \exists a_j \in D_j$ such that $<a_i, a_j> \in C_{ij}$.

Revise $(x_i, x_j)$

Input: Variables $x_i$ and $x_j$ with domains $D_i$ and $D_j$ and constraint relation $R_{ij}$.

Output: pruned $D_i$, such that $x_i$ is directed arc-consistent relative to $x_j$.

1. for each $a_i \in D_i$
2. if there is no $a_j \in D_j$ such that $<a_i, a_j> \in R_{ij}$,
3. then delete $a_i$ from $D_i$.
4. endif
5. endfor

Constraint Processing,
by R. Dechter
pgs 54-56.
Directed Arc Consistency

Revise($x_1$, $x_2$):

Definition: arc $<x_i, x_j>$ is arc consistent if $<x_i, x_j>$ and $<x_j, x_i>$ are directed arc consistent.

Definition: Problem is arc consistent if all pairs of variables are arc consistent.
Full Arc Consistency over All Constraints via Constraint Propagation

Definition: arc $<x_i, x_j>$ is directed arc consistent if
$$\forall a_i \in D_i, \exists a_j \in D_j \text{ such that } <a_i, a_j> \in C_{ij}.$$ 

Constraint Propagation:

To achieve (directed) arc consistency over CSP:

1. For every arc $C_{ij}$ in CSP, with tail domain $D_i$, call Revise.

2. Repeat until quiescence:
   - If an element was deleted from $D_i$, then repeat Step 1. (AC-1)
Full Arc-Consistency via AC-1

AC-1(CSP)

Input: A constraint satisfaction problem CSP = <X, D, C>.
Output: CSP', the largest arc-consistent subset of CSP.

1. repeat
2. for every c_{ij} ∈ C,
3. Revise(x_i, x_j)  
4. Revise(x_j, x_i)  
5. endfor
6. until no domain is changed.
Full Arc Consistency via Constraint Propagation

Definition: arc $<x_i, x_j>$ is directed arc consistent if

$$\forall a_i \in D_i, \exists a_j \in D_j \text{ such that } <a_i, a_j> \in C_{ij}.$$ 

Constraint Propagation:

To achieve (directed) arc consistency over CSP:

1. For every arc $C_{ij}$ in CSP, with tail domain $D_i$, call Revise.
2. Repeat until quiescence:
   
   If an element was deleted from $D_i$, then
   
   repeat Step 1, \hspace{1cm} (AC-1)
   
   OR call Revise on each arc with head $D_i$ \hspace{1cm} (AC-3)
   
   (use FIFO Q, and remove duplicates).
Full Arc-Consistency via AC-3 (Waltz CP)

AC-3(CSP)

Input: A constraint satisfaction problem CSP = \( \langle X, D, C \rangle \).

Output: CSP’, the largest arc-consistent subset of CSP.

1. \textbf{for} every \( c_{ij} \in C \),
2. \( \text{queue} \leftarrow \text{queue} \cup \{<x_i, x_j>, <x_j, x_i>\} \)
3. \textbf{endfor}
4. \textbf{while} \( \text{queue} \neq {} \)
5. select and delete arc \( <x_i, x_j> \) from \( \text{queue} \)
6. Revise\((x_i, x_j)\)
7. \textbf{if} Revise\((x_i, x_j)\) caused a change in \( D_i \)
8. \textbf{then} \( \text{queue} \leftarrow \text{queue} \cup \{<x_k, x_i> \mid k \neq i, k \neq j\} \)
9. \textbf{endif}
10. \textbf{endwhile}

Constraint Processing, by R. Dechter
pgs 58-59.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

Each undirected arc denotes two directed arcs.
Graph Coloring

Initial Domains

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

Arcs to examine

\[ V_1 - V_2, V_1 - V_3, V_2 - V_3 \]

• Introduce queue of arcs to be examined.
• Start by adding all arcs to the queue.
Graph Coloring
Initial Domains

<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>
</tbody>
</table>

Arcs to examine
\[ V_1 - V_2, V_1 - V_3, V_2 - V_3 \]

- \( V_i - V_j \) denotes two arcs, between \( V_i \) and \( V_j \).
- \( V_i > V_j \) denotes an arc from \( V_i \) to \( V_j \).
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 &gt; V_2$</td>
<td></td>
</tr>
</tbody>
</table>

• Delete disallowed tail values.

Arcs to examine

$V_2 > V_1$, $V_1 - V_3$, $V_2 - V_3$

• $V_i - V_j$ denotes two arcs, between $V_i$ and $V_j$.
• $V_i > V_j$ denotes an arc from $V_i$ to $V_j$. 
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 &gt; V_2$</td>
<td>none</td>
</tr>
</tbody>
</table>

Arcs to examine

$V_2 > V_1, V_1 - V_3, V_2 - V_3$

- Delete disallowed tail values.
- $V_i - V_j$ denotes two arcs, between $V_i$ and $V_j$.
- $V_i > V_j$ denotes an arc from $V_i$ to $V_j$. 
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains

Arc examined | Value deleted
-------------|----------------
$V_1 > V_2$   | none
$V_2 > V_1$   |

Arcs to examine

$V_1 - V_3, V_2 - V_3$

- Delete disallowed tail values.
- $V_i - V_j$ denotes two arcs, between $V_i$ and $V_j$.
- $V_i > V_j$ denotes an arc from $V_i$ to $V_j$. 
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 &gt; V_2$</td>
<td>none</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td>none</td>
</tr>
</tbody>
</table>

- Delete disallowed tail values.
- $V_i - V_j$ denotes two arcs, between $V_i$ and $V_j$.
- $V_i > V_j$ denotes an arc from $V_i$ to $V_j$. 
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
</tbody>
</table>

Arcs to examine

$V_1 - V_3$, $V_2 - V_3$

- Delete disallowed tail values.
- $V_i - V_j$ denotes two arcs, between $V_i$ and $V_j$.
- $V_i > V_j$ denotes an arc from $V_i$ to $V_j$. 
**Constraint Propagation Example AC-3**

**Graph Coloring**

Initial Domains

<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>$V_1 &gt; V_3$</td>
<td></td>
</tr>
</tbody>
</table>

**Arcs to examine**

- $V_3 > V_1$, $V_2 - V_3$

- Delete disallowed tail values.

- $V_i - V_j$ denotes two arcs, between $V_i$ and $V_j$.

- $V_i > V_j$ denotes an arc from $V_i$ to $V_j$. 

• Delete disallowed tail values.
**Constraint Propagation Example AC-3**

**Graph Coloring**

**Initial Domains**

<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>$V_1 &gt; V_3$</td>
<td>$V_1(G)$</td>
</tr>
</tbody>
</table>

**Arcs to examine**

$V_3 > V_1, V_2 - V_3$

**IF** An element of a variable’s domain is removed,
**THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>$V_1 &gt; V_3$</td>
<td>$V_1(G)$</td>
</tr>
</tbody>
</table>

Arcs to examine

$V_3 > V_1$, $V_2 - V_3$, $V_2 > V_1$, $V_3 > V_1$

IF An element of a variable’s domain is removed,
THEN add all arcs to that variable to the examination queue.
**Constraint Propagation Example AC-3**

Graph Coloring

Initial Domains

<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>$V_1 &gt; V_3$</td>
<td>$V_1(G)$</td>
</tr>
<tr>
<td>$V_3 &gt; V_1$</td>
<td></td>
</tr>
</tbody>
</table>

- Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed,
**THEN** add all arcs to that variable to the examination queue.
Graph Coloring

Initial Domains

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 \rightarrow V_2$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 &gt; V_3$</td>
<td>$V_1(G)$</td>
</tr>
<tr>
<td>$V_3 &gt; V_1$</td>
<td>none</td>
</tr>
</tbody>
</table>

- Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed,
**THEN** add all arcs to that variable to the examination queue.
Graph Coloring

Initial Domains

<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>

• Delete unmentioned tail values.

**IF** An element of a variable’s domain is **removed,**
**THEN** add all arcs to that variable to the examination queue.
**Constraint Propagation Example AC-3**

**Graph Coloring**

Initial Domains

<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>
<tr>
<td>$V_2 &gt; V_3$</td>
<td></td>
</tr>
</tbody>
</table>

- Delete unmentioned tail values.

**Arcs to examine**

- $V_3 > V_2$, $V_2 > V_1$

---

**IF** An element of a variable’s domain is removed,

**THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>V₁ - V₂</td>
<td>none</td>
</tr>
<tr>
<td>V₁ - V₃</td>
<td>V₁(G)</td>
</tr>
<tr>
<td>V₂ &gt; V₃</td>
<td>V₂(G)</td>
</tr>
</tbody>
</table>

Arcs to examine
V₃ > V₃, V₂ > V₁

• Delete unmentioned tail values.

IF An element of a variable’s domain is removed, THEN add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 &gt; V_3$</td>
<td>$V_2(G)$</td>
</tr>
</tbody>
</table>

• Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed, **THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 &gt; V_3$</td>
<td>$V_2(G)$</td>
</tr>
</tbody>
</table>

Arcs to examine

- $V_3 > V_2$, $V_2 > V_1$, $V_1 > V_2$

• Delete unmentioned tail values.

IF An element of a variable’s domain is removed,
THEN add all arcs to that variable to the examination queue.
**Graph Coloring**

**Initial Domains**

<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>
<tr>
<td>$V_2 &gt; V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_3 &gt; V_2$</td>
<td></td>
</tr>
</tbody>
</table>

- Delete unmentioned tail values.

**Arcs to examine**

- $V_2 > V_1, V_1 > V_2$

**Constraint Propagation Example AC-3**

**Different-color constraint**

**IF** An element of a variable’s domain is **removed,**

**THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 &gt; V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_3 &gt; V_2$</td>
<td>none</td>
</tr>
</tbody>
</table>

• Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed,  
**THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

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

• Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed,
**THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td></td>
</tr>
</tbody>
</table>

• Delete unmentioned tail values.

IF An element of a variable’s domain is removed,
THEN add all arcs to that variable to the examination queue.
**Constraint Propagation Example AC-3**

**Graph Coloring**

**Initial Domains**

<table>
<thead>
<tr>
<th>Arc examined</th>
<th>Value deleted</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1 \rightarrow V_2$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 \rightarrow V_3$</td>
<td>$V_1(G)$</td>
</tr>
<tr>
<td>$V_2 \rightarrow V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td>none</td>
</tr>
</tbody>
</table>

- Delete unmentioned tail values.

**Arcs to examine**

- $V_1 > V_2$

**Example**

IF An element of a variable’s domain is removed,

THEN add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring

Initial Domains

<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>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 &gt; V_2$</td>
<td></td>
</tr>
</tbody>
</table>

Arcs to examine

- Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed, **THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 &gt; V_2$</td>
<td>$V_1(R)$</td>
</tr>
</tbody>
</table>

• Delete unmentioned tail values.

IF An element of a variable’s domain is removed,
THEN add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td>none</td>
</tr>
<tr>
<td>$V_1 &gt; V_2$</td>
<td>$V_1(R)$</td>
</tr>
</tbody>
</table>

Arcs to examine

- $V_2 > V_1$, $V_3 > V_1$

• Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed,

**THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 - V_1$</td>
<td>$V_1(R)$</td>
</tr>
</tbody>
</table>

Arcs to examine

$V_2 > V_1, V_3 > V_1$

- Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed, **THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 - V_1$</td>
<td>$V_1(R)$</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td></td>
</tr>
</tbody>
</table>

- Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed,
**THEN** add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 - V_1$</td>
<td>$V_1(R)$</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td>none</td>
</tr>
</tbody>
</table>

Arcs to examine

$V_3 > V_1$

• Delete unmentioned tail values.

IF An element of a variable’s domain is removed,

THEN add all arcs to that variable to the examination queue.
Constraint Propagation Example AC-3

Graph Coloring
Initial Domains

<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>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 - V_1$</td>
<td>$V_1(R)$</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td>none</td>
</tr>
<tr>
<td>$V_3 &gt; V_1$</td>
<td>none</td>
</tr>
</tbody>
</table>

Arcs to examine

- Delete unmentioned tail values.

**IF** An element of a variable’s domain is removed,
**THEN** add all arcs to that variable to the examination queue.
**Constraint Propagation Example AC-3**

**Graph Coloring**

Initial Domains

<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>
<tr>
<td>$V_2 - V_3$</td>
<td>$V_2(G)$</td>
</tr>
<tr>
<td>$V_2 - V_1$</td>
<td>$V_1(R)$</td>
</tr>
<tr>
<td>$V_2 &gt; V_1$</td>
<td>none</td>
</tr>
<tr>
<td>$V_3 &gt; V_1$</td>
<td>none</td>
</tr>
</tbody>
</table>

**Arcs to examine**

IF examination queue is empty

THEN arc (pairwise) consistent.
Outline

• Arc-consistency and constraint propagation.
• Analysis of constraint propagation.
• Solving CSPs using search.
What is the Complexity of AC-1?

AC-1(CSP)

Input: A network of constraints CSP = <X, D, C>.

Output: CSP’, the largest arc-consistent subset of CSP.

1. repeat
2. for every $c_{ij} \in C$,
3. Revise($x_i, x_j$)
4. Revise($x_j, x_i$)
5. endfor
6. until no domain is changed.

Assume:

• There are $n$ variables.
• Domains are of size at most $k$.
• There are $e$ binary constraints.
What is the Complexity of AC-1?

Assume:

• There are $n$ variables.
• Domains are of size at most $k$.
• There are $e$ binary constraints.

Which is the correct complexity?

1. $O(k^2)$,
2. $O(enk^2)$,
3. $O(enk^3)$,
4. $O(nek)$. 
Revise: A directed arc consistency procedure

Revise \((x_i, x_j)\)

**Input:** Variables \(x_i\) and \(x_j\) with domains \(D_i\) and \(D_j\) and constraint relation \(R_{ij}\).

**Output:** pruned \(D_i\), such that \(x_i\) is directed arc-consistent relative to \(x_j\).

1. for each \(a_i \in D_i\)
2. if there is no \(a_j \in D_j\) such that \(<a_i, a_j> \in R_{ij}\)
3. then delete \(a_i\) from \(D_i\).
4. endif
5. endfor

Complexity of Revise?

\[= O(k^2).\]

where \(k = \max_i |D_i|\)
Full Arc-Consistency via AC-1

AC-1(CSP)

Input: A network of constraints CSP = <X, D, C>.
Output: CSP’, the largest arc-consistent subset of CSP.

1. repeat
2. for every c_{ij} ∈ C,
3. Revise(x_i, x_j)
4. Revise(x_j, x_i)
5. endfor
6. until no domain is changed.

Complexity of AC-1?

= O(nk*e*revise),
= O(enk^3),

where k = \max_i |D_i|,
    n = |X|, e = |C|.
What is the Complexity of Constraint Propagation using AC-3?

Assume:

• There are \( n \) variables.
• Domains are of size at most \( k \).
• There are \( e \) binary constraints.

Which is the correct complexity?

1. \( O(k^2) \),
2. \( O(ek^2) \),
3. \( O(ek^3) \),
4. \( O(ek) \).
**Full Arc-Consistency via AC-3**

AC-3(CSP)

**Input:** A network of constraints CSP = \(<X, D, C>\).

**Output:** CSP’, the largest arc-consistent subset of CSP.

1. for every \(c_{ij} \in C\), \(O(e)\) +
2. queue ← queue \(\cup \\{<x_i,x_j>, <x_i,x_j>\}\)
3. endfor
4. while queue \(\neq \{\}\)
5. select and delete arc \(<x_i, x_j>\) from queue
6. Revise\((x_i, x_j)\)
7. if Revise\((x_i, x_j)\) caused a change in \(D_i\).
8. then queue ← queue \(\cup \\{<x_k,x_i> | k \neq i, k \neq j\}\)
9. endif
10. endwhile

**Complexity of AC-3?**

\[= O(e+ek^2) = O(ek^3),\]  
where \(k = \max_i |D_i|, n = |X|, e = |C|\).
Is arc consistency sound and complete?

An arc consistent solution selects a value for every variable from its arc consistent domain.

Soundness: All solutions to the CSP are arc consistent solutions?

• Yes,
• No.

Completeness: All arc-consistent solutions are solutions to the CSP?

• Yes,
• No.

R, G
R, G
R, G
Incomplete: Arc consistency doesn’t rule out all infeasible solutions

Graph Coloring

Arc consistent, but no solutions.

Arc consistent, but 2 solutions, not 8.

<table>
<thead>
<tr>
<th>B, R, G</th>
</tr>
</thead>
<tbody>
<tr>
<td>B, G, R</td>
</tr>
</tbody>
</table>
To Solve CSPs We Combine

1. Arc consistency (via constraint propagation):
   • Eliminates values that are shown locally to not be a part of any solution.

2. Search:
   • Explores consequences of committing to particular assignments.

Methods that Incorporate Search:
• Standard Search,
• Back Track Search (BT),
• BT with Forward Checking (FC),
• Dynamic Variable Ordering (DV),
• Iterative Repair (IR),
• Conflict-directed Back Jumping (CBJ).
Solving CSPs using Generic Search

- **State**
  - Partial assignment to variables, made thus far.

- **Initial State**
  - No assignment.

- **Operator**
  - Creates new assignment \( (X_i = v_{ij}) \).
    - Select any unassigned variable \( X_i \).
    - Select any one of its domain values \( v_{ij} \).
  - Child extends parent assignments with new.

- **Goal Test**
  - All variables are assigned.
  - All constraints are satisfied.

- **Branching factor?**
  \( \rightarrow \text{Sum of domain size of all variables} \quad O(|v|^*|d|). \)

- **Performance?**
  \( \rightarrow \text{Exponential in the branching factor} \quad O(|v|^*|d|^{|v|}). \)
Search Performance on N Queens

- Standard Search,
- A handful of queens.
- Backtracking.
Solving CSPs with Standard Search

Standard Search:
- Children select any value for any variable \([O(|v|*|d|)]\).
- Test complete assignments for consistency against CSP.

Observations:
1. The order in which variables are assigned does not change the solution.
   - Many paths denote the same solution,
     - \(|v|!\).
   ➔ Expand only one path (i.e., use one variable ordering).

1. We can identify and prune a dead end before we assign all variables.
   - Extensions to inconsistent partial assignments are always inconsistent.
   ➔ Check consistency after each assignment.
Next: Back Track Search (BT)

1. Expand assignments of one variable at each step.
2. Pursue depth first.
3. Check consistency after each expansion, and backup.

Preselect order of variables to assign.

Assign designated variable.