# Learning Objectives

At the end of the class you should be able to:

- show how constraint satisfaction problems can be solved with generate-and-test
- show how constraint satisfaction problems can be solved with search
- explain and trace arc-consistency of a constraint graph
- show how domain splitting can solve constraint problems

# Generate-and-Test Algorithm

- Generate the assignment space $D = dom(V_1) \times dom(V_2) \times \ldots \times dom(V_n)$. Test each assignment with the constraints.
- Example:

$$
\begin{aligned}
D &= dom(A) \times dom(B) \times dom(C) \times dom(D) \times dom(E) \\
&= \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3, \\
&= \{\langle 1,1,1,1,1 \rangle, \langle 1,1,1,1,2 \rangle, ..., \langle 4,4,4,4,4 \rangle\}.
\end{aligned}
$$

# Generate-and-Test Algorithm

- Generate the assignment space $\mathrm{D} = dom(V_1) \times dom(V_2) \times \ldots \times dom(V_n)$. Test each assignment with the constraints.
- Example:

$$
\begin{aligned}
\mathrm{D} &= dom(A) \times dom(B) \times dom(C) \times dom(D) \times dom(E) \\
&= \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3, \\
&= \{\langle 1,1,1,1,1\rangle, \langle 1,1,1,1,2\rangle, ..., \langle 4,4,4,4,4\rangle\}.
\end{aligned}
$$

- Can be implemented with $n$ nested for-loops.

# Generate-and-Test Algorithm

- Generate the assignment space $\mathrm{D} = dom(V_1) \times dom(V_2) \times \ldots \times dom(V_n)$. Test each assignment with the constraints.
- Example:

$$\begin{aligned} \mathrm{D} &= dom(A) \times dom(B) \times dom(C) \times dom(D) \times dom(E) \\ &= \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3, \\ &= \{\langle 1,1,1,1,1 \rangle, \langle 1,1,1,1,2 \rangle, \ldots, \langle 4,4,4,4,4 \rangle\}. \end{aligned}$$

- Can be implemented with $n$ nested for-loops.

```
for A in dom_A:
   for B in dom_B:
      ...
       if constraints are satisfied: return (A,B,...)
```

# Generate-and-Test Algorithm

- Generate the assignment space $\mathrm{D} = dom(V_1) \times dom(V_2) \times \ldots \times dom(V_n)$. Test each assignment with the constraints.
- Example:

$$
\begin{aligned}
\mathrm{D} &= dom(A) \times dom(B) \times dom(C) \times dom(D) \times dom(E) \\
&= \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3, \\
&= \{\langle 1,1,1,1,1 \rangle, \langle 1,1,1,1,2 \rangle, ..., \langle 4,4,4,4,4 \rangle\}.
\end{aligned}
$$

- Can be implemented with $n$ nested for-loops.

```
for A in dom_A:
   for B in dom_B:
      ...
         if constraints are satisfied: return (A,B,...)
```

- How many assignments need to be tested for $n$ variables each with domain size $d$?

# Backtracking Algorithms

- Systematically explore D by instantiating the variables one at a time

# Backtracking Algorithms

- Systematically explore D by instantiating the variables one at a time
- evaluate each constraint predicate as soon as all its variables are bound

# Backtracking Algorithms

- Systematically explore D by instantiating the variables one at a time
- evaluate each constraint predicate as soon as all its variables are bound
- any partial assignment that doesn't satisfy the constraint can be pruned.

# Backtracking Algorithms

- Systematically explore D by instantiating the variables one at a time
- evaluate each constraint predicate as soon as all its variables are bound
- any partial assignment that doesn't satisfy the constraint can be pruned.

Example Variables $A, B, C$, domains $\{1, 2, 3, 4\}$,
constraints $A < B, B < C$.

# Backtracking Algorithms

- Systematically explore D by instantiating the variables one at a time
- evaluate each constraint predicate as soon as all its variables are bound
- any partial assignment that doesn't satisfy the constraint can be pruned.

Example Variables $A, B, C$, domains $\{1, 2, 3, 4\}$,
constraints $A < B, B < C$.
Assignment $A = 1 \wedge B = 1$ is inconsistent with constraint $A < B$ regardless of the value of the other variables.

# CSP as Graph Searching

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.

# CSP as Graph Searching

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.
- Suppose node $N$ is the assignment $X_1 = v_1, \ldots, X_k = v_k$.
  Select a variable $Y$ that isn't assigned in $N$.
  For each value $y_i \in dom(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.

# CSP as Graph Searching

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.
- Suppose node $N$ is the assignment $X_1 = v_1, \ldots, X_k = v_k$.
  Select a variable $Y$ that isn't assigned in $N$.
  For each value $y_i \in dom(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.
- The start node is

# CSP as Graph Searching

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.
- Suppose node $N$ is the assignment $X_1 = v_1, \ldots, X_k = v_k$.
  Select a variable $Y$ that isn't assigned in $N$.
  For each value $y_i \in dom(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.
- The start node is the empty assignment.

# CSP as Graph Searching

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.
- Suppose node $N$ is the assignment $X_1 = v_1, \ldots, X_k = v_k$.
  Select a variable $Y$ that isn't assigned in $N$.
  For each value $y_i \in dom(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.
- The start node is the empty assignment.
- A goal node is a

# CSP as Graph Searching

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.
- Suppose node $N$ is the assignment $X_1 = v_1, \ldots, X_k = v_k$.
  Select a variable $Y$ that isn't assigned in $N$.
  For each value $y_i \in dom(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.
- The start node is the empty assignment.
- A goal node is a total assignment that satisfies the constraints.

# CSP as Graph Searching

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.
- Suppose node $N$ is the assignment $X_1 = v_1, \ldots, X_k = v_k$.
  Select a variable $Y$ that isn't assigned in $N$.
  For each value $y_i \in dom(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.
- The start node is the empty assignment.
- A goal node is a total assignment that satisfies the constraints.
- The search space depends on which variable is selected to be assigned for each node. There are no cycles or multiple paths to a node.

# Simple Example 1

- Variables: $A, B, C$
- Domains: $\{1, 2, 3, 4\}$
- Constraints $A < B$, $B < C$

# Simple Example 2

- Variables: $A, B, C, D$
- Domains: $\{1, 2, 3, 4\}$
- Constraints $A < B$, $B < C, C < D$

# Simple Example 3

- Variables: $A, B, C, D, E$
- Domains: $\{1, 2, 3, 4\}$
- Constraints $A < B$, $B < C, C < D, D < E$

# Example: scheduling activities

- Variables: $A$, $B$, $C$, $D$, $E$ that represent the starting times of various activities.
- Domains: $dom(A) = \{1, 2, 3, 4\}$, $dom(B) = \{1, 2, 3, 4\}$, $dom(C) = \{1, 2, 3, 4\}$, $dom(D) = \{1, 2, 3, 4\}$, $dom(E) = \{1, 2, 3, 4\}$
- Constraints:

$$
\begin{aligned}
(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge \\
(C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge \\
(E < C) \wedge (E < D) \wedge (B \neq D).
\end{aligned}
$$

# Consistency Algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the variable is ruled impossible by any of the constraints.
- Example: Is the scheduling example domain consistent?

# Consistency Algorithms

- Idea: prune the domains as much as possible before selecting values from them.
- A variable is domain consistent if no value of the domain of the variable is ruled impossible by any of the constraints.
- Example: Is the scheduling example domain consistent? $dom(B) = \{1, 2, 3, 4\}$ isn't domain consistent as $B = 3$ violates the constraint $B \neq 3$.

## Constraint Network

- There is a oval-shaped node for each variable.

# Constraint Network

- There is a oval-shaped node for each variable.
- There is a rectangular node for each constraint.

# Constraint Network

- There is a oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.

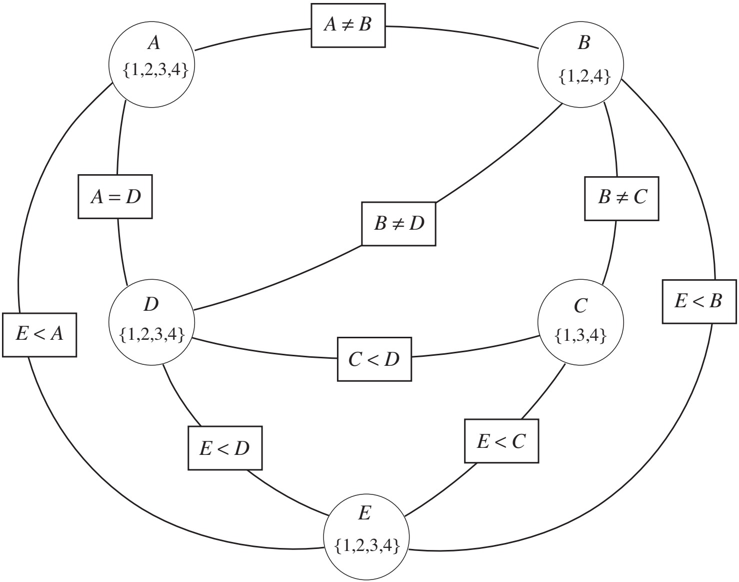
# Constraint Network

- There is a oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.
- There is an arc from variable $X$ to each constraint that involves $X$.

# Constraint Network

- There is a oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.
- There is an arc from variable $X$ to each constraint that involves $X$.
  An arc is written as $\langle X, r(X, \overline{Y})\rangle$
  E.g., $\langle X, X < Y\rangle$, $\langle Y, X < Y\rangle$
  $\langle X, X + Y = Z\rangle$, $\langle Y, X + Y = Z\rangle$, $\langle Z, X + Y = Z\rangle$

# Example Constraint Network



# Arc Consistency

- An arc $\langle X, r(X, \overline{Y}) \rangle$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\overline{y} \in dom(\overline{Y})$ such that $r(x, \overline{y})$ is satisfied.

# Arc Consistency

- An arc $\langle X, r(X, \overline{Y}) \rangle$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\overline{y} \in dom(\overline{Y})$ such that $r(x, \overline{y})$ is satisfied.
- A network is arc consistent if all its arcs are arc consistent.
- What if arc $\langle X, r(X, \overline{Y}) \rangle$ is *not* arc consistent?

# Arc Consistency

- An arc $\langle X, r(X, \overline{Y})\rangle$ is arc consistent if, for each value $x \in dom(X)$, there is some value $\overline{y} \in dom(\overline{Y})$ such that $r(x, \overline{y})$ is satisfied.
- A network is arc consistent if all its arcs are arc consistent.
- What if arc $\langle X, r(X, \overline{Y})\rangle$ is *not* arc consistent?
  All values of $X$ in $dom(X)$ for which there is no corresponding value in $dom(\overline{Y})$ can be deleted from $dom(X)$ to make the arc $\langle X, r(X, \overline{Y})\rangle$ consistent.

# Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?

# Arc Consistency Algorithm

- The arcs can be considered in turn making each arc consistent.
- When an arc has been made arc consistent, does it ever need to be checked again?
  An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the $Y$'s is reduced.

# Generalized Arc Consistency

**for each** variable $X$:
$\quad D_X := dom(X)$
$TDA := \{\langle X, c\rangle \mid c \in C \text{ and } X \in scope(c)\}$
**while** $TDA$ is not empty:
$\quad$ **select** and **remove** path $\langle X, c\rangle$ from $TDA$
$\quad$ **suppose** scope of $c$ is $\{X, Y_1, \ldots, Y_k\}$
$\quad ND_X := \{x \mid x \in D_X$ and
$\qquad\qquad$ exists $y_1 \in D_{Y_1}, \ldots, y_k \in D_{Y_k}$
$\qquad\qquad$ s.th. $c(X = x, Y_1 = y_1, \ldots, Y_k = y_k) = true\ \}$
$\quad$ if $ND_X \neq D_X$:
$\qquad TDA := TDA \cup \{\langle Z, c'\rangle \mid X \in scope(c'),$
$\qquad\qquad c'$ is not $c$, $Z \in scope(c') \setminus \{X\}\}$
$\qquad D_X := ND_X$
**return** $\{D_X \mid X$ is a variable$\}$

# Arc Consistency Algorithm

Three possible outcomes when all arcs are made arc consistent:

# Arc Consistency Algorithm

Three possible outcomes when all arcs are made arc consistent:

- One domain is empty $\Longrightarrow$
- Each domain has a single value $\Longrightarrow$
- Some domains have more than one value $\Longrightarrow$

# Arc Consistency Algorithm

Three possible outcomes when all arcs are made arc consistent:

- One domain is empty $\Longrightarrow$ no solution
- Each domain has a single value $\Longrightarrow$
- Some domains have more than one value $\Longrightarrow$

# Arc Consistency Algorithm

Three possible outcomes when all arcs are made arc consistent:

- One domain is empty $\Longrightarrow$ no solution
- Each domain has a single value $\Longrightarrow$ unique solution
- Some domains have more than one value $\Longrightarrow$

# Arc Consistency Algorithm

Three possible outcomes when all arcs are made arc consistent:

- One domain is empty $\Longrightarrow$ no solution
- Each domain has a single value $\Longrightarrow$ unique solution
- Some domains have more than one value $\Longrightarrow$ there may or may not be a solution

# Complexity of Arc Consistency

- Consider binary constraints
- Each variable domain is of size $d$
- There are $e$ arcs.
- Checking an arc takes time

# Complexity of Arc Consistency

- Consider binary constraints
- Each variable domain is of size $d$
- There are $e$ arcs.
- Checking an arc takes time $O(d^2)$
  $\langle X, c(X,Y)\rangle$ for each value for $X$, check each value for $Y$
- Each constraint needs to be checked at most

# Complexity of Arc Consistency

- Consider binary constraints
- Each variable domain is of size $d$
- There are $e$ arcs.
- Checking an arc takes time $O(d^2)$
  $\langle X, c(X, Y)\rangle$ for each value for $X$, check each value for $Y$
- Each constraint needs to be checked at most $d$ times.
  $\langle X, c(X, Y)\rangle$ rechecked when a value for $Y$ is removed.
- Thus the algorithm $GAC$ takes time

# Complexity of Arc Consistency

- Consider binary constraints
- Each variable domain is of size $d$
- There are $e$ arcs.
- Checking an arc takes time $O(d^2)$
  $\langle X, c(X, Y)\rangle$ for each value for $X$, check each value for $Y$
- Each constraint needs to be checked at most $d$ times.
  $\langle X, c(X, Y)\rangle$ rechecked when a value for $Y$ is removed.
- Thus the algorithm *GAC* takes time $O(ed^3)$.

# Complexity of Arc Consistency

- Consider binary constraints
- Each variable domain is of size $d$
- There are $e$ arcs.
- Checking an arc takes time $O(d^2)$
  $\langle X, c(X, Y)\rangle$ for each value for $X$, check each value for $Y$
- Each constraint needs to be checked at most $d$ times.
  $\langle X, c(X, Y)\rangle$ rechecked when a value for $Y$ is removed.
- Thus the algorithm $GAC$ takes time $O(ed^3)$.

Solving a CSP is an NP-complete problem where $n$ the number of variables

- Give a solution it can be checked in polynomial time
- But it can be made arc consistent in polynomial time. How?

# Complexity of Arc Consistency

- Consider binary constraints
- Each variable domain is of size $d$
- There are $e$ arcs.
- Checking an arc takes time $O(d^2)$
  $\langle X, c(X, Y) \rangle$ for each value for $X$, check each value for $Y$
- Each constraint needs to be checked at most $d$ times.
  $\langle X, c(X, Y) \rangle$ rechecked when a value for $Y$ is removed.
- Thus the algorithm $GAC$ takes time $O(ed^3)$.

Solving a CSP is an NP-complete problem where $n$ the number of variables

- Give a solution it can be checked in polynomial time
- But it can be made arc consistent in polynomial time. How?
  Making the network arc consistent does not solve the problem. We need to search for a solution.

# Finding solutions with AC and domain splitting

To solve a CSP:

- Simplify with arc-consistency
- If a domain is empty, return no solution
- If all domains have size 1, return solution found
- Else split a domain, and recursively solve each half.

# Finding one solutions with AC and domain splitting

$Solve\_one(CSP, domains)$ :
    simplify $CSP$ with arc-consistency
    **if** one domain is empty:
        **return** False
    **else if** all domains have one element:
        **return** solution of that element for each variable
    **else**:
        **select** variable $X$ with domain $D$ and $|D| > 1$
        **partition** $D$ into $D_1$ and $D_2$
        **return** $Solve\_one(CSP, domains$ with $dom(X) = D_1)$ or
            $Solve\_one(CSP, domains$ with $dom(X) = D_2)$

# Finding set of all solutions with AC and domain splitting

$Solve\_all(CSP, domains)$ :

   simplify $CSP$ with arc-consistency

   **if** one domain is empty:

      **return**

   **else if** all domains have one element:

      **return**

   **else**:

      **select** variable $X$ with domain $D$ and $|D| > 1$

      **partition** $D$ into $D_1$ and $D_2$

      **return**

# Finding set of all solutions with AC and domain splitting

*Solve_all*$(CSP, domains)$ :

simplify *CSP* with arc-consistency

**if** one domain is empty:

**return** $\{\}$

**else if** all domains have one element:

**return**

**else**:

**select** variable $X$ with domain $D$ and $|D| > 1$

**partition** $D$ into $D_1$ and $D_2$

**return**

# Finding set of all solutions with AC and domain splitting

$Solve\_all(CSP, domains)$ :
  simplify $CSP$ with arc-consistency
  **if** one domain is empty:
    **return** $\{\}$
  **else if** all domains have one element:
    **return** $\{$solution of that element for each variable$\}$
  **else**:
    **select** variable $X$ with domain $D$ and $|D| > 1$
    **partition** $D$ into $D_1$ and $D_2$
    **return**

# Finding set of all solutions with AC and domain splitting

$Solve\_all(CSP, domains)$ :
    simplify $CSP$ with arc-consistency
    **if** one domain is empty:
        **return** $\{\}$
    **else if** all domains have one element:
        **return** $\{$solution of that element for each variable$\}$
    **else**:
        **select** variable $X$ with domain $D$ and $|D| > 1$
        **partition** $D$ into $D_1$ and $D_2$
        **return** $Solve\_all(CSP, domains$ with $dom(X) = D_1) \cup$
            $Solve\_all(CSP, domains$ with $dom(X) = D_2)$

# AC and domain splitting as search

Domain splitting leads to search space

- Nodes:
- Neighbors
- Goal:
- Start node:

# AC and domain splitting as search

Domain splitting leads to search space

- Nodes: CSP with arc-consistent domains
- Neighbors

- Goal:
- Start node:

# AC and domain splitting as search

Domain splitting leads to search space

- Nodes: CSP with arc-consistent domains
- Neighbors of $CSP$:
  if all domains are non-empty:
    select variable $X$ with domain $D$ and $|D| > 1$
    partition $D$ into $D_1$ and $D_2$
    neighbors are
    - $make\_AC(CSP \mid dom(X) = D_1)$
    - $make\_AC(CSP \mid dom(X) = D_2)$
- Goal:
- Start node:

# AC and domain splitting as search

Domain splitting leads to search space

- Nodes: CSP with arc-consistent domains
- Neighbors of $CSP$:
  if all domains are non-empty:
    select variable $X$ with domain $D$ and $|D| > 1$
    partition $D$ into $D_1$ and $D_2$
    neighbors are
    - $make\_AC(CSP \mid dom(X) = D_1)$
    - $make\_AC(CSP \mid dom(X) = D_2)$
- Goal: all domains have size 1
- Start node:

# AC and domain splitting as search

Domain splitting leads to search space

- Nodes: CSP with arc-consistent domains
- Neighbors of $CSP$:
  if all domains are non-empty:
    select variable $X$ with domain $D$ and $|D| > 1$
    partition $D$ into $D_1$ and $D_2$
    neighbors are
    - $make\_AC(CSP \mid dom(X) = D_1)$
    - $make\_AC(CSP \mid dom(X) = D_2)$
- Goal: all domains have size 1
- Start node: $make\_AC(CSP)$