

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
 $\mathbf{D} = \text{dom}(V_1) \times \text{dom}(V_2) \times \dots \times \text{dom}(V_n)$. Test each assignment with the constraints.

- **Example:**

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

Generate-and-Test Algorithm

- Generate the assignment space
 $\mathbf{D} = \text{dom}(V_1) \times \text{dom}(V_2) \times \dots \times \text{dom}(V_n)$. Test each assignment with the constraints.

- **Example:**

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

- Can be implemented with n nested for-loops.

Generate-and-Test Algorithm

- Generate the assignment space
 $\mathbf{D} = \text{dom}(V_1) \times \text{dom}(V_2) \times \dots \times \text{dom}(V_n)$. Test each assignment with the constraints.

- **Example:**

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

- 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
 $\mathbf{D} = \text{dom}(V_1) \times \text{dom}(V_2) \times \dots \times \text{dom}(V_n)$. Test each assignment with the constraints.

- **Example:**

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

- 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 \mathbf{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 \mathbf{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 \mathbf{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 \mathbf{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.

A CSP can be solved by graph-searching:

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

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 $X_1 = v_1, \dots, X_k = v_k$.
Select a variable Y that isn't assigned in N .

For each value $y_i \in \text{dom}(Y)$

$X_1 = v_1, \dots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.

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 $X_1 = v_1, \dots, X_k = v_k$.
Select a variable Y that isn't assigned in N .

For each value $y_i \in \text{dom}(Y)$

$X_1 = v_1, \dots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.

- The start node is

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 $X_1 = v_1, \dots, X_k = v_k$.
Select a variable Y that isn't assigned in N .

For each value $y_i \in \text{dom}(Y)$

$X_1 = v_1, \dots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.

- The start node is the empty assignment.

A CSP can be solved by graph-searching:

- A node is an assignment values to some of the variables.
- Suppose node N is the assignment $X_1 = v_1, \dots, X_k = v_k$.
Select a variable Y that isn't assigned in N .

For each value $y_i \in \text{dom}(Y)$

$X_1 = v_1, \dots, X_k = v_k, Y = y_i$ 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

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 $X_1 = v_1, \dots, X_k = v_k$.
Select a variable Y that isn't assigned in N .

For each value $y_i \in \text{dom}(Y)$

$X_1 = v_1, \dots, X_k = v_k, Y = y_i$ 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.

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 $X_1 = v_1, \dots, X_k = v_k$.
Select a variable Y that isn't assigned in N .

For each value $y_i \in \text{dom}(Y)$

$X_1 = v_1, \dots, X_k = v_k, Y = y_i$ is a neighbour if it is consistent with the constraints that can be evaluated.

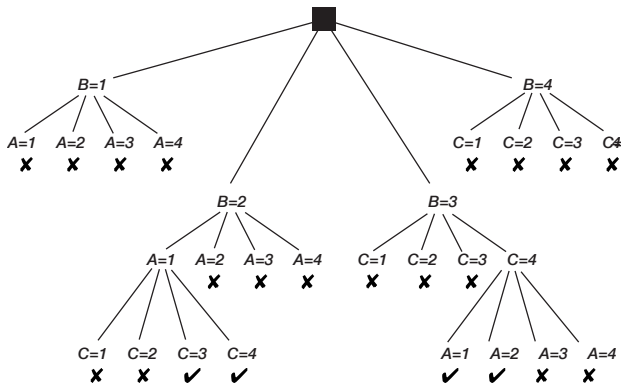
- The start node is the empty assignment.
- A goal node is a total assignment that satisfies the constraints.
- The search space depends on which variable is selected to be assigned for each node. There are no cycles or multiple paths to a node.

Simple Example 1

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

Simple Example 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:**

$$(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge \\ (C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge \\ (E < C) \wedge (E < D) \wedge (B \neq D).$$

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

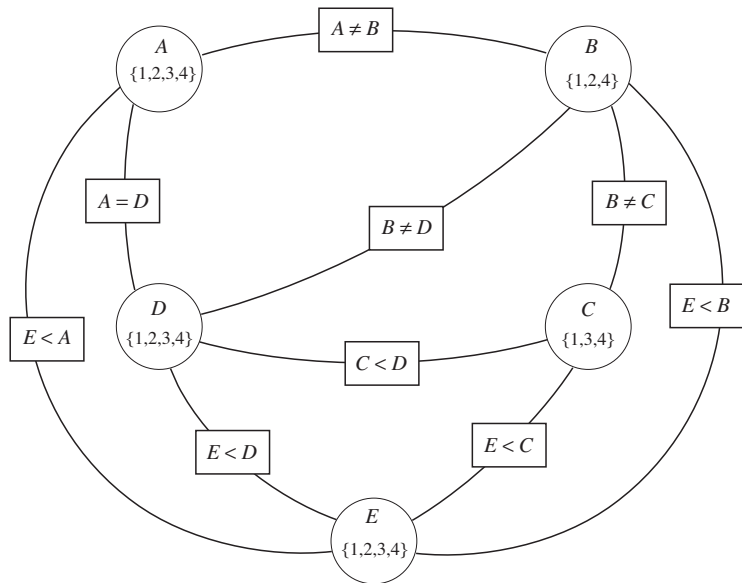
- There is a oval-shaped node for each variable.
- There is a rectangular node for each constraint.
- There is a domain of values associated with each variable node.
- There is an arc from variable X to each constraint that involves X .

An arc is written as $\langle X, r(X, \bar{Y}) \rangle$

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



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

Arc Consistency

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

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

$$D_X := \text{dom}(X)$$

$$\text{to_do} := \{ \langle X, c \rangle \mid c \in C \text{ and } X \in \text{scope}(c) \}$$

while to_do is not empty:

select and **remove** path $\langle X, c \rangle$ from to_do

suppose scope of c is $\{X, Y_1, \dots, Y_k\}$

$$ND_X := \{x \mid x \in D_X \text{ and}$$

$$\text{exists } y_1 \in D_{Y_1}, \dots, y_k \in D_{Y_k}$$

$$\text{s.th. } c(X = x, Y_1 = y_1, \dots, Y_k = y_k) = \text{true} \}$$

if $ND_X \neq D_X$:

$$\text{to_do} := \text{to_do} \cup \{ \langle Z, c' \rangle \mid X \in \text{scope}(c'),$$

$$c' \text{ is not } c, Z \in \text{scope}(c') \setminus \{X\} \}$$

$$D_X := ND_X$$

return $\{D_X \mid X \text{ is a variable}\}$

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 \implies
- Each domain has a single value \implies
- Some domains have more than one value \implies

Arc Consistency Algorithm

Three possible outcomes when all arcs are made arc consistent:

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

Arc Consistency Algorithm

Three possible outcomes when all arcs are made arc consistent:

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

Arc Consistency Algorithm

Three possible outcomes when all arcs are made arc consistent:

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

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.


```
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) = D1) or  
           Solve_one(CSP, domains with dom(X) = D2)
```

```
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
```

```
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
```

```
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) :  
  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  $\text{dom}(X) = D_1$ )  $\cup$   
           Solve_all(CSP, domains with  $\text{dom}(X) = D_2$ )
```

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: $make_AC(CSP)$

