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
  \[ D = \text{dom}(V_1) \times \text{dom}(V_2) \times \ldots \times \text{dom}(V_n). \]
  Test each assignment with the constraints.

- Example:

  \[ D = \text{dom}(A) \times \text{dom}(B) \times \text{dom}(C) \times \text{dom}(D) \times \text{dom}(E) \]

  \[ = \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \]

  \[ = \{\langle 1, 1, 1, 1, 1 \rangle, \langle 1, 1, 1, 1, 2 \rangle, \ldots, \langle 4, 4, 4, 4, 4 \rangle\}. \]
Generate-and-Test Algorithm

- Generate the assignment space
  \[ D = \text{dom}(V_1) \times \text{dom}(V_2) \times \ldots \times \text{dom}(V_n) \]. Test each assignment with the constraints.

- Example:

  \[
  D = \text{dom}(A) \times \text{dom}(B) \times \text{dom}(C) \times \text{dom}(D) \times \text{dom}(E)
  \]
  \[
  = \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\}
  \]
  \[
  = \{\langle 1, 1, 1, 1, 1 \rangle, \langle 1, 1, 1, 1, 2 \rangle, \ldots, \langle 4, 4, 4, 4, 4 \rangle\}.
  \]

- Can be implemented with \( n \) nested for-loops.
Generate-and-Test Algorithm

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

- Example:

  \[
  D = \text{dom}(A) \times \text{dom}(B) \times \text{dom}(C) \times \text{dom}(D) \times \text{dom}(E)
  = \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\}
  = \{\langle 1, 1, 1, 1, 1 \rangle, \langle 1, 1, 1, 1, 2 \rangle, \ldots, \langle 4, 4, 4, 4, 4 \rangle\}.
  \]

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

- Example:

  \[
  D = \text{dom}(A) \times \text{dom}(B) \times \text{dom}(C) \times \text{dom}(D) \times \text{dom}(E) \\
  = \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \times \{1, 2, 3, 4\} \\
  = \{\langle 1, 1, 1, 1, 1 \rangle, \langle 1, 1, 1, 1, 2 \rangle, \ldots, \langle 4, 4, 4, 4, 4 \rangle\}.
  \]

- Can be implemented with \( n \) nested for-loops.

  ```python
  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

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

Assignment A = 1 \land B = 1 is inconsistent with constraint A < B regardless of the value of the other variables.
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.

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

Assignment A = 1 \land B = 1 is inconsistent with constraint A < B regardless of the value of the other variables.
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 $A = 1 \land B = 1$ 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 $X_1 = v_1, \ldots, X_k = v_k$.
  
  **Select** a variable $Y$ that isn’t assigned in $N$.
  
  For each value $y_i \in \text{dom}(Y)$
  
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ 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 \( X_1 = v_1, \ldots, X_k = v_k \).
  Select a variable \( Y \) that isn’t assigned in \( N \).
  For each value \( y_i \in \text{dom}(Y) \)
  \( X_1 = v_1, \ldots, X_k = v_k, Y = y_i \) is a neighbour if it is consistent with the constraints that can be evaluated.
- The start node is
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 \( y_i \in \text{dom}(Y) \)
  \( X_1 = v_1, \ldots, X_k = v_k, Y = y_i \) is a neighbour if it is consistent with the constraints that can be evaluated.
- The start node is the empty assignment.
A 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 $y_i \in \text{dom}(Y)$, $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ 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.
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 $y_i \in \text{dom}(Y)$
  $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ 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.
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 $y_i \in \text{dom}(Y)$
    - $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ 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:** $\text{dom}(A) = \{1, 2, 3, 4\}$, $\text{dom}(B) = \{1, 2, 3, 4\}$, $\text{dom}(C) = \{1, 2, 3, 4\}$, $\text{dom}(D) = \{1, 2, 3, 4\}$, $\text{dom}(E) = \{1, 2, 3, 4\}$

- **Constraints:**

\[
(B \neq 3) \land (C \neq 2) \land (A \neq B) \land (B \neq C) \land \\
(C < D) \land (A = D) \land (E < A) \land (E < B) \land \\
(E < C) \land (E < D) \land (B \neq D).
\]
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?
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? $\text{dom}(B) = \{1, 2, 3, 4\}$ isn’t domain consistent as $B = 3$ violates the constraint $B \neq 3$. 
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 an 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 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 an 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 $\langle X, r(X, Y) \rangle$

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

\[ \{1,2,3,4\} \{1,2,4\} \{1,2,3,4\} \{1,3,4\} \{1,2,3,4\} \]

\[ A \quad \neq \quad B \]
\[ B \quad \neq \quad D \]
\[ C \quad < \quad D \]
\[ A \quad = \quad D \]
\[ E \quad < \quad A \]
\[ B \quad \neq \quad C \]
\[ E \quad < \quad B \]
\[ E \quad < \quad D \quad E \quad < \quad C \]
Arc Consistency

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

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

©D.L. Poole and A.K. Mackworth 2010-2020
Arc Consistency

- An arc $\langle X, r(X, \overline{Y}) \rangle$ is arc consistent if, for each value $x \in \text{dom}(X)$, there is some value $\overline{y} \in \text{dom}(\overline{Y})$ 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 $\text{dom}(X)$ for which there is no corresponding value in $\text{dom}(\overline{Y})$ can be deleted from $\text{dom}(X)$ 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?
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 \( X \):

\[
D_X := \text{dom}(X)
\]

\[
TDA := \{ \langle X, c \rangle \mid c \in C \text{ and } X \in \text{scope}(c) \}
\]

while \( TDA \) is not empty:

- select and remove path \( \langle X, c \rangle \) from \( TDA \)
- suppose scope of \( c \) is \( \{ X, Y_1, \ldots, Y_k \} \)

\[
ND_X := \{ x \mid x \in D_X \text{ and } \exists y_1 \in D_{Y_1}, \ldots, y_k \in D_{Y_k} \text{ s.th. } c(X = x, Y_1 = y_1, \ldots, Y_k = y_k) = \text{true} \}
\]

if \( ND_X \neq D_X \):

\[
TDA := TDA \cup \{ \langle Z, c' \rangle \mid X \in \text{scope}(c'), c' \text{ is not } c, Z \in \text{scope}(c') \setminus \{X\} \}
\]

\[
D_X := ND_X
\]

return \( \{ D_X \mid X \text{ is a variable} \} \)
Three possible outcomes when all arcs are made arc consistent:

- One domain is empty $\Rightarrow$ no solution
- Each domain has a single value $\Rightarrow$ unique solution
- Some domains have more than one value $\Rightarrow$ there may or may not be a solution
Three possible outcomes when all arcs are made arc consistent:

- One domain is empty ➞
- Each domain has a single value ➞
- Some domains have more than one value ➞
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 $\implies$
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 \(\implies\)
Three possible outcomes when all arcs are made arc consistent:
- One domain is empty $\Rightarrow$ no solution
- Each domain has a single value $\Rightarrow$ unique solution
- Some domains have more than one value $\Rightarrow$ there may or may not be a solution
Consider binary constraints
- Each variable domain is of size $d$
- There are $e$ arcs.
- Checking an arc takes time $O(d^2)$

Each constraint needs to be checked at most $d$ times.

Thus the algorithm $GAC$ takes time $O(ed^3)$.

Solving a CSP is an NP-complete problem where:
- The number of variables
- Give a solution it can be checked in polynomial time
- But it can be made arc consistent in polynomial time.

Making the network arc consistent does not solve the problem. We need to search for a solution.
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
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
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?
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

\[\text{Solve\_one}(CSP, domains) :\]

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 \(\text{Solve\_one}(CSP, domains \text{ with } \text{dom}(X) = D_1)\) or 
    \(\text{Solve\_one}(CSP, domains \text{ with } \text{dom}(X) = D_2)\)
Finding set of all solutions with AC and domain splitting

$\text{Solve}_{all}(\text{CSP}, \text{domains}) :$

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

\[\text{Solve\_all}(CSP, \text{domains}) :\]
\[\quad \text{simplify } CSP \text{ with arc-consistency}\]
\[\quad \text{if } \text{one domain is empty:}\]
\[\quad \quad \text{return } \{\}\]
\[\quad \text{else if all domains have one element:}\]
\[\quad \quad \text{return}\]
\[\quad \text{else:}\]
\[\quad \quad \text{select variable } X \text{ with domain } D \text{ and } |D| > 1\]
\[\quad \quad \text{partition } D \text{ into } D_1 \text{ and } D_2\]
\[\quad \quad \text{return}\]
Finding set of all solutions with AC and domain splitting

\[ \text{Solve\_all}(CSP, \text{domains}) : \]
\[ \text{simplify CSP with arc-consistency} \]
\[ \text{if one domain is empty:} \]
\[ \quad \text{return } \{\} \]
\[ \text{else if all domains have one element:} \]
\[ \quad \text{return } \{\text{solution of that element for each variable}\} \]
\[ \text{else:} \]
\[ \quad \text{select variable } X \text{ with domain } D \text{ and } |D| > 1 \]
\[ \quad \text{partition } D \text{ into } D_1 \text{ and } D_2 \]
\[ \quad \text{return} \]
Finding set of all solutions with AC and domain splitting

\[ \text{Solve\_all}(CSP, \text{domains}) : \]
\[
\begin{align*}
\text{simulate \ CSP \ with \ arc-consistency} \\
\text{if \ one \ domain \ is \ empty:} \\
\quad \text{return} \ \{\} \\
\text{else \ if \ all \ domains \ have \ one \ element:} \\
\quad \text{return} \ \{\text{solution \ of \ that \ element \ for \ each \ variable}\} \\
\text{else:} \\
\quad \text{select \ variable} \ X \ \text{with \ domain} \ D \ \text{and} \ |D| > 1 \\
\quad \text{partition} \ D \ \text{into} \ D_1 \ \text{and} \ D_2 \\
\quad \text{return} \ \text{Solve\_all}(CSP, \text{domains \ with} \ dom(X) = D_1) \cup \text{Solve\_all}(CSP, \text{domains \ with} \ dom(X) = D_2)
\end{align*}
\]
Domain splitting leads to search space

- Nodes:
- Neighbors

Goal:
- Start node:
Domain splitting leads to search space
- Nodes: CSP with arc-consistent domains
- Neighbors

Goal:
Start node:
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
    - \(\text{make\_AC}(\text{CSP} \mid \text{dom}(X) = D_1)\)
    - \(\text{make\_AC}(\text{CSP} \mid \text{dom}(X) = D_2)\)
- **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
      - $\text{make-AC}(\text{CSP} \mid \text{dom}(X) = D_1)$
      - $\text{make-AC}(\text{CSP} \mid \text{dom}(X) = D_2)$
- Goal: all domains have size 1
- Start node:
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
    - $\text{make\_AC}(\text{CSP} \mid \text{dom}(X) = D_1)$
    - $\text{make\_AC}(\text{CSP} \mid \text{dom}(X) = D_2)$
- **Goal**: all domains have size 1
- **Start node**: $\text{make\_AC}(\text{CSP})$