Announcements

- Assignment 3 is available
- Assignment 1 is now marked; see Canvas

Review: So far. . .

- An agent acts in an environment
- Agent has access to: abilities, goals/preferences, prior knowledge, observations, past experiences

Review: So far. . .

- An agent acts in an environment
- Agent has access to: abilities, goals/preferences, prior knowledge, observations, past experiences
- Search is used to find paths in graphs
- Search algorithms differ in how elements of frontier are selected
- Multiple-path pruning and loop pruning can reduce search
- Depth-bounded depth-first search (as used in iterative deepening and branch-and-bound) can save space

Review: So far. . .

- An agent acts in an environment
- Agent has access to: abilities, goals/preferences, prior knowledge, observations, past experiences
- Search is used to find paths in graphs
- Search algorithms differ in how elements of frontier are selected
- Multiple-path pruning and loop pruning can reduce search
- Depth-bounded depth-first search (as used in iterative deepening and branch-and-bound) can save space
- A constraint satisfaction problem involves a set a variables, a domain for each variable and a set of constraints.

Posing a Constraint Satisfaction Problem

A CSP is characterized by

- A set of variables $V_1, V_2, \ldots, V_n$.
- Each variable V_i has a domain $dom(V_i)$ the set of possible values for V_i. (We assume domains are finite.)
- A possible world or total assignment is an assignment of a value to each variable.

Posing a Constraint Satisfaction Problem

A CSP is characterized by

- A set of variables $V_1, V_2, \ldots, V_n$.
- Each variable V_i has a domain $dom(V_i)$
  the set of possible values for V_i.
  (We assume domains are finite.)
- A possible world or total assignment is an assignment of a value to each variable.
- A scope is a subset of the variables
- A hard constraint on a scope specifies which combination of values of the variables in the scope are legal.
  It is a function from the scope into $\{true, false\}$.
- A solution to CSP (a model) is possible world that satisfies all the constraints.

Clicker Question

Suppose there were 100 variables, each with domain size 17. How many possible worlds are there?

A 1700

B 117

C 17^{100}

D 100^{17}

E None of the above

Today: Learning Objectives

Today: Constraint Satisfaction Problems

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

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

- 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:

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

- 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?

Generate-and-test as Graph Searching

A CSP can be solved by graph-searching:

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

Generate-and-test as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors:
- Start node:
- Goal:

Generate-and-test as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors: 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.
- Start node:
- Goal:

Generate-and-test as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors: 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.
- Start node: the empty assignment
- Goal:

Generate-and-test as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors: 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.
- Start node: the empty assignment
- Goal: A goal node is a total assignment that satisfies all constraints.

Generate-and-test as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors: 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.
- Start node: the empty assignment
- Goal: A goal node is a total assignment that satisfies all constraints.
- There are no cycles or multiple paths to a node.
  The search space does not depend in the variable selected.
  All paths to a solution have same length.

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:

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

CSP as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors:
- Start node:
- Goal:

CSP as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors: 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.
- Start node:
- Goal:

CSP as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors: 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.
- Start node: the empty assignment
- Goal:

CSP as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors: 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.
- Start node: the empty assignment
- Goal: A goal node is a total assignment.

CSP as Graph Searching

A CSP can be solved by graph-searching:

- Nodes: A node is an assignment values to some of the variables.
- Neighbors: 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.
- Start node: the empty assignment
- Goal: A goal node is a total assignment.
- 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.
  Depth-first search is appropriate.

Simple Examples

Example 1:

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

Simple Examples

Example 1:

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

Example 2:

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

Simple Examples

Example 1:

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

Example 2:

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

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:

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

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?

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.

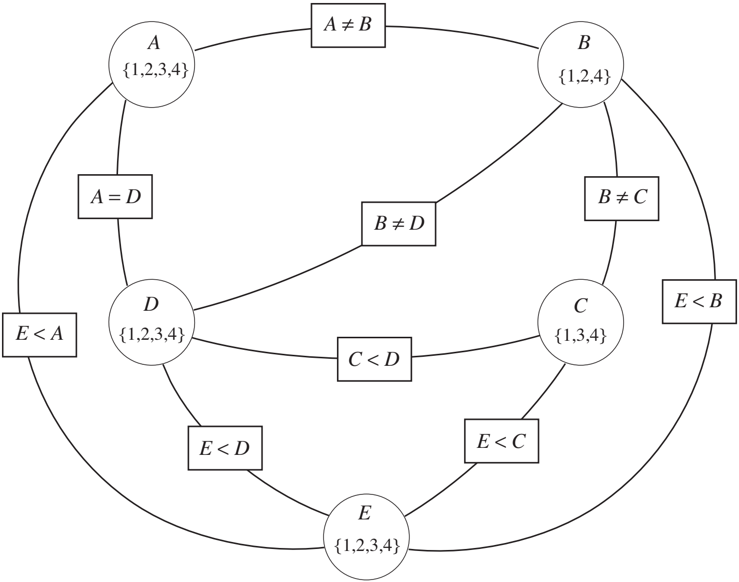
Constraint Network

- 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.

Clicker Question

$dom(X) = \{1, 3, 5\}$
$dom(Y) = \{2, 3, 4\}$
Making the arc $\langle X, X < Y \rangle$ arc consistent:

- A does nothing because it is already arc consistent
- B results in just 5 being removed from the domain of X
- C results in 3 and 5 being removed from the domain of X
- D results in 3 and 5 being removed from the domain of X, and 2 and 3 removed from the domain of Y
- E results in 3 being removed from both domains

Clicker Question

$dom(X) = \{1, 3, 5\}$
$dom(Y) = \{2, 3, 4\}$
Making the arc $\langle Y, X < Y \rangle$ arc consistent:

A does nothing because it is already arc consistent

B results in just 2 being removed from the domain of Y

C results in 2 and 3 being removed from the domain of Y

D results in the domain of Y becoming empty

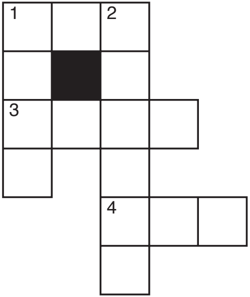
E results in 3 being removed from each domain

Clicker Question

$dom(X) = \{1, 3, 5\}$
$dom(Y) = \{2, 3, 4\}$
Making the arc $\langle X, X \neq Y \rangle$ arc consistent:

- A does nothing because it is already arc consistent
- B results in just 3 being removed from the domain of X
- C results in 1 and 5 being removed from the domain of X
- D results in 1, 3 and 5 being removed from the domain of X
- E results in both domains becoming empty

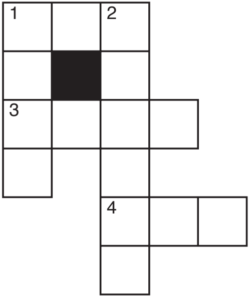
Clicker Question



$dom(\text{1-across}) = \{ant, big, bus, car, has\}$
$dom(\text{2-down}) = \{\text{ginger, search, symbol, yogurt}\}$
For the constraint C: 3rd letter of 1-across = 1st letter of 2-down.
After making $\langle \text{1-across}, C\rangle$ arc consistent, the domain of 1-across is

- A $\{ant, big, bus, car, has\}$
- B $\{big, bus, car, has\}$
- C $\{big, bus, has\}$
- D $\{ant, big, car\}$
- E $\{ant, car\}$

Clicker Question



$dom(\text{1-across}) = \{ant, big, bus, car, has\}$
$dom(\text{2-down}) = \{\text{ginger, search, symbol, yogurt}\}$
For the constraint C: 3rd letter of 1-across = 1st letter of 2-down.
After making $\langle \text{2-down}, C \rangle$ arc consistent, the domain of 2-down is

- A {ginger, search, symbol, yogurt}
- B {ginger, yogurt}
- C {ginger, search, symbol}
- D {yogurt}
- E {}

Clicker Question

$dom(X) = \{1, 3, 5\}$
$dom(Y) = \{2, 3, 4\}$
Making the arc $\langle X, X = Y \rangle$ arc consistent:

A does nothing because it is already arc consistent

B results in just 3 being removed from the domain of X

C results in 1 and 5 being removed from the domain of X

D results in 1, 3 and 5 being removed from the domain of X

E results in both domains becoming empty

Clicker Question

$dom(X) = \{1, 3, 5\}$
$dom(Y) = \{2, 3, 4\}$
Making the arc $\langle X, X = Y + 1 \rangle$ arc consistent:

A does nothing because it is already arc consistent

B results in 3 being removed from the domain of X

C results in 1 being removed from the domain of X

D results in 3 and 5 being removed from the domain of X

E results in domain of X becoming empty

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:

$\quad D_X := dom(X)$

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

while to_do is not empty:

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

$\quad$ **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 $ND_X \neq D_X$:

$\qquad to\_do := to\_do \cup \{\langle Z, c'\rangle \mid X \in scope(c'),$

$\qquad\qquad c'$ 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 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.
  - It is often best to split a domain in half.

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.
  - It is often best to split a domain in half.
  - Do we need to restart from scratch?

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: $make\_AC(CSP)$

Clicker Question

Which of the following is false:

A If there is a solution, arc consistency will not make any domains empty

B Arc consistency always halts for finite CSPs

C Arc consistency involves checking constraints multiple times

D Arc consistency always results in singleton domains if there is only one solution.

Clicker Question

What is not true of arc consistency with domain splitting:

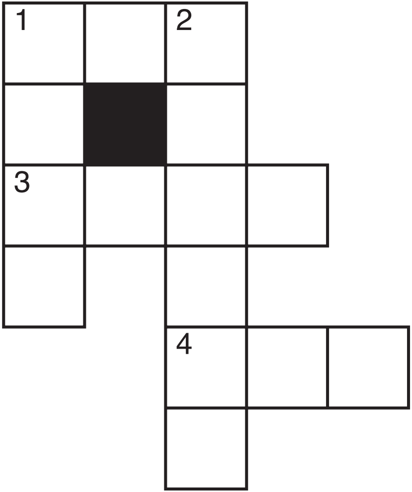
A Arc consistency needs domain splitting to solve problems in general

B Together they can solve CSPs in polynomial time

C They typically result in a smaller search space than using search without arc consistency

D They always terminate with a solution if there is one for finite CSPs

Example: Crossword Puzzle



Words:

ant, big, bus, car, has
book, buys, hold,
lane, year
beast, ginger, search,
symbol, syntax