Lecture 9
Arc Consistency
(4.5, 4.6)
Announcements

• Assignment 2 posted last Friday - Due Friday Feb. 10
  • Can already start doing the first 3 questions after today’s class

• Remember that I have posted in Connect:
  • Learning goals for the whole course
  • Short questions on material for the midterm

  ✓ Both midterm and final will have questions very similar, or even
    verbatim, from this set (about 40-60% of overall mark)

  ✓ You can answer these questions by studying the slides and textbook
    - Cover questions as we proceed through the material, come to see
      us if you have doubts on how to answer
    - BUT DO STUDY THE REVELANT MATERIAL FIRST!

  ✓ Do not wait until just before the midterm to review the questions, if you
    start now it will be much easier to cover all the relevant ones before
    the exam
Lecture Overview

Recap of Lecture 8
- Arc Consistency for CSP
- Domain Splitting
- Intro to SLS (time permitting)
Course Overview

Problem Type
- Static
  - Query
- Sequential
  - Planning

Environment
- Deterministic
  - Arc Consistency
  - Search
  - Vars + Constraints
  - Logics
  - STRIPS
  - Search
- Stochastic
  - Belief Nets
    - Variable Elimination
  - Decision Nets
    - Variable Elimination
  - Markov Processes
    - Value Iteration

Representation
- Reasoning Technique
  - We’ll start from CPS
We will look at Search for CSP

- Constraint Satisfaction Problems (CPS):
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function

- Query:
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function

- Planning:
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function
A constraint satisfaction problem (CSP) consists of:

- a set of variables \( V \)
- a domain \( \text{dom}(V) \) for each variable
- a set of constraints \( C \)

Constraints are restrictions on the values that one or more variables can take:

- Unary constraint: restriction involving a single variable
- k-ary constraint: restriction involving \( k \) different variables
  - We will mostly deal with binary constraints
- Constraints can be specified by
  1. listing all combinations of valid domain values for the variables participating in the constraint
  2. giving a function that returns true when given values for each variable which satisfy the constraint
Example: Map-Coloring

Variables $WA, NT, Q, NSW, V, SA, T$

Domains $D_i = \{\text{red, green, blue}\}$

Constraints: adjacent regions must have different colors

e.g., $WA \neq NT, NT \neq SA, NT \neq QU, \ldots,$

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>NT</th>
<th>SA</th>
<th>NT</th>
<th>QU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td>Green</td>
<td>Red</td>
<td>Green</td>
<td>Red</td>
<td>Green</td>
</tr>
<tr>
<td>Red</td>
<td>Bue</td>
<td>Red</td>
<td>Bue</td>
<td>Red</td>
<td>Bue</td>
</tr>
<tr>
<td>Green</td>
<td>Red</td>
<td>Green</td>
<td>Red</td>
<td>Green</td>
<td>Red</td>
</tr>
<tr>
<td>Green</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
</tr>
<tr>
<td>Blue</td>
<td>Red</td>
<td>Blue</td>
<td>Red</td>
<td>Blue</td>
<td>Red</td>
</tr>
<tr>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Green</td>
<td>Blue</td>
<td>Green</td>
</tr>
</tbody>
</table>

Or
A constraint satisfaction problem (CSP) consists of:

- a set of variables $V$
- a domain $\text{dom}(V)$ for each variable
- a set of constraints $\mathcal{C}$

A model of a CSP is an assignment of values to all of its variables (i.e., a possible world) that satisfies all of its constraints.

WA = red,
NT = green,
Q = red,
NSW = green,
V = red,
SA = blue,
T = green
Solving Constraint Satisfaction Problems

• Even the simplest problem of determining whether or not a model exists in a general CSP with finite domains is **NP-hard**
  • There is no known algorithm with worst case polynomial runtime.

• However, we can try to:
  • identify special cases for which algorithms are efficient
  • find efficient (polynomial) **consistency algorithms** that reduce the size of the search space
  • work on **approximation algorithms** that can find good solutions quickly, even though they may offer no theoretical guarantees
  • find algorithms that are fast on **typical** (not worst case) cases
Search-Based Approach

- **Constraint Satisfaction (Problems):**
  - **State:** assignments of values to a subset of the variables
  - **Successor function:** assign values to a “free” variable
  - **Goal test:** all variables assigned a value and all constraints satisfied?
  - **Solution:** possible world that satisfies the constraints
  - **Heuristic function:** none (all solutions at the same distance from start)

- **Planning:**
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function

- **Inference**
  - State
  - Successor function
  - Goal test
  - Solution
  - Heuristic function
Backtracking algorithms

• Explore search space via DFS but evaluate each constraint as soon as all its variables are bound.
• Any partial assignment that doesn’t satisfy the constraint can be pruned.

Check unary constraints on $V_1$
If not satisfied = PRUNE

Check constraints on $V_1$ and $V_2$ If not satisfied = PRUNE
Selecting variables in a smart way

- Backtracking relies on one or more heuristics to select which variables to consider next.
  - E.g, variable involved in the largest number of constraints:
    “If you are going to fail on this branch, fail early!”

- But we will look at an alternative approach that can do much better

  - Arc Consistency:
    - Key idea: prune the domains as much as possible before searching for a solution.
Def. A constraint network is defined by a graph, with
- one node for every variable (drawn as circle)
- one node for every constraint (drawn as rectangle)
- undirected edges running between variable nodes and constraint nodes whenever a given variable is involved in a given constraint.

- Three variables: A, B, C
- Two constraints: A<B, A>C
**Arc Consistency**

**Definition:**
An arc \(<x, r(x,y)>\) is **arc consistent** if for each value \(x\) in \(\text{dom}(X)\) there is some value \(y\) in \(\text{dom}(Y)\) such that \(r(x,y)\) is satisfied.

A network is arc consistent if all its arcs are arc consistent.

- **Not arc consistent:** No value in domain of B that satisfies A<B if A=3
- **Arc consistent:** Both B=2 and B=3 have ok values for A (e.g. A=1)
Definition:
An arc \(<x, r(x,y)>\) is arc consistent if for each value \(x\) in \(\text{dom}(X)\) there is some value \(y\) in \(\text{dom}(Y)\) such that \(r(x,y)\) is satisfied.

A network is arc consistent if all its arcs are arc consistent.
Definition:
An arc \( <x, r(x,y)> \) is arc consistent if for each value \( x \) in \( \text{dom}(X) \) there is some value \( y \) in \( \text{dom}(Y) \) such that \( r(x,y) \) is satisfied.
A network is arc consistent if all its arcs are arc consistent.

A. Both arcs are consistent
B. Left consistent, right inconsistent
C. Left inconsistent, right consistent
D. Both arcs are inconsistent

Left = \(<X, (X < Y/2)\>\)  \quad \text{Right} = \(<Y, (X < Y/2)\>\)
Arc Consistency

Definition:
An arc \( <x, r(x,y)> \) is arc consistent if for each value \( x \) in \( \text{dom}(X) \) there is some value \( y \) in \( \text{dom}(Y) \) such that \( r(x,y) \) is satisfied.

A network is arc consistent if all its arcs are arc consistent.

Arc consistent: Both \( X=2 \) and \( X=5 \) have ok values for \( Y \) (e.g. \( Y=17 \)).

Not arc consistent: No value in domain of \( X \) that satisfies \( X<Y/2 \) if

\[
\begin{align*}
X & : 2, 5, \\
Y & : 3, 5, 12 \\
\end{align*}
\]
Lecture Overview

- Recap of Lecture 8

Arc Consistency for CSP
- Domain Splitting
- Intro to SLS
Arc Consistency Algorithm

How can we enforce Arc Consistency?

• If an arc \(<X, r(X,Y)>\) is not arc consistent
  - Delete all values \(x\) in \(dom(X)\) for which there is no corresponding value in \(dom(Y)\)
  - This deletion makes the arc \(<X, r(X,Y)>\) arc consistent.
  - This removal can never rule out any models/solutions

WHY?

Download this example (SimpleCSP) from Schedule page

Save to a local file and then open file in the Aispace applet Consistency Based CSP Solver
Arc Consistency Algorithm

How can we enforce Arc Consistency?
• If an arc \(<X, r(X,Y)>\) is not arc consistent
  - Delete all values \(x\) in \(\text{dom}(X)\) for which there is no corresponding value in \(\text{dom}(Y)\)
  - This deletion makes the arc \(<X, r(X,Y)>\) arc consistent.
  - This removal can never rule out any models/solutions

Algorithm: general idea
• Go through all the arcs in the network
• Make each arc consistent by pruning the appropriate domain, when needed
• Reconsider consistent arcs that could be made inconsistent again by this pruning
• Eventually reach a ‘fixed point’: all arcs consistent
Arc Consistency

Arc consistent: For each value in \text{dom}(C), there is one in \text{dom}(A) that satisfies \(A > C\) (namely \(A = 3\))

Not arc consistent: No value in domain of B that satisfies \(A < B\) if \(A = 3\)

Try to build this simple network in AISpace
Arc Consistency

Arc consistent: For each value in \( \text{dom}(C) \), there is one in \( \text{dom}(A) \) that satisfies \( A > C \) (namely \( A = 3 \))

Not arc consistent: No value in domain of \( B \) that satisfies \( A < B \) if \( A = 3 \)

After pruning \( A = 3 \): Not arc consistent anymore: For \( C = 2 \), there is no value in \( \text{dom}(A) \) that satisfies \( A > C \): prune!
Which arcs need to be reconsidered?

- When Arc Consistency reduces the domain of a variable $X$ to make an arc $\langle X, c \rangle$ arc consistent, which arcs that were already consistent does it need to reconsider?
Which arcs need to be reconsidered?

- Consider the arcs in turn, making each arc consistent
- Reconsider arcs that could be made inconsistent again by this pruning

DO trace on ‘simple problem 1’ and on ‘scheduling problem 1’, trying to predict
- which arcs are not consistent and
- which arcs need to be reconsidered after each removal
Arc consistency algorithm
(for binary constraints)

Procedure GAC(V, dom, C)

Inputs

V: a set of variables
dom: a function such that dom(X) is the domain of variable X
C: set of constraints to be satisfied

Output

arc-consistent domains for each variable

Local

D_X is a set of values for each variable X
TDA is a set of arcs

1: for each variable X do
2: \( D_X \leftarrow \text{dom}(X) \)
3: TDA \( \leftarrow \{ \langle X, c \rangle | \ X \in V, c \in C \text{ and } X \in \text{scope}(c) \} \)
4: while (TDA \( \neq \{\} \)) do
5: \( \text{select } \langle X, c \rangle \in \text{TDA} \)
6: TDA \( \leftarrow \text{TDA} \setminus \{\langle X, c \rangle \} \)
7: \( \text{ND}_X \leftarrow \{x | x \in D_X \text{ and } \exists y \in D_Y \text{ s.t. (x, y) satisfies c}\} \)
8: if (\( \text{ND}_X \neq D_X \)) then
9: \( \text{TDA} \leftarrow \text{TDA} \cup \{ \langle Z, c' \rangle | X \in \text{scope}(c'), c' \neq c, Z \in \text{scope}(c') \setminus \{X\} \} \)
10: \( D_X \leftarrow \text{ND}_X \)
11: return \( \{D_X | X \text{ is a variable}\} \)
Arc Consistency Algorithm: Complexity

• Let’s determine Worst-case complexity of this procedure (compare with DFS)
  • let the max size of a variable domain be $d$
  • let the number of variables be $n$
  • Worst-case time complexity of Backtracking (DFS with pruning?)
Arc Consistency Algorithm: Complexity

- Let’s determine Worst-case complexity of this procedure (compare with DFS)
  - let the max size of a variable domain be $d$
  - let the number of variables be $n$
  - Worst-case time complexity of Backtracking (DFS with pruning?)

Check unary constraints on $V_1$
If not satisfied = PRUNE

Check constraints on $V_1$ and $V_2$ If not satisfied = PRUNE
Arc Consistency Algorithm: Complexity

- Let’s determine Worst-case complexity of this procedure (compare with DFS............)
  - let the max size of a variable domain be $d$
  - let the number of variables be $n$
  - The max number of binary constraints is $2^n$
Arc Consistency Algorithm: Complexity

• Let’s determine Worst-case complexity of this procedure (compare with DFS)
  • let the max size of a variable domain be \( d \)
  • let the number of variables be \( n \)
  • The max number of binary constraints is

• How many times, at worst, the same arc can be inserted in the ToDoArc list?
Arc Consistency Algorithm: Complexity

• Let’s determine Worst-case complexity of this procedure (compare with DFS)
  • let the max size of a variable domain be \( d \)
  • let the number of variables be \( n \)
  • The max number of binary constraints is \( ? \)
  • How many times, at worst, the same arc can be inserted in the ToDoArc list?

• How many steps are involved in checking the consistency of an arc?
Arc Consistency Algorithm: Complexity

- Let’s determine Worst-case complexity of this procedure (compare with DFS $O(d^n)$)
  - let the max size of a variable domain be $d$
  - let the number of variables be $n$
  - The max number of binary constraints is $\frac{n \times (n-1)}{2}$
  - How many times, at worst, the same arc can be inserted in the ToDoArc list? $O(d)$
  - How many steps are involved in checking the consistency of an arc? $O(d^2)$

- Overall complexity:
Arc Consistency Algorithm: Interpreting Outcomes

• Three possible outcomes (when all arcs are arc consistent):
  • Each domain has a single value,
    ✓ e.g. built-in AI Space example “Scheduling problem 1”
    ✓ We have: a (unique) solution.
  • At least one domain is empty,
    ✓ We have: No solution! All values are ruled out for this variable.
    ✓ e.g. try this graph (can easily generate it by modifying Simple Problem 2)
  • Some domains have more than one value,
Can we have an arc consistent network with non-empty domains that has no solution?
Arc Consistency Algorithm: Interpreting Outcomes

• Three possible outcomes (when all arcs are arc consistent):
  • Each domain has a single value,
    ✓ e.g. built-in AlSpace example “Scheduling problem 1”
    ✓ We have: a (unique) solution.
  • At least one domain is empty,
    ✓ We have: No solution! All values are ruled out for this variable.
    ✓ e.g. try this graph (can easily generate it by modifying Simple Problem 2)
  • Some domains have more than one value,
    ✓ There may be: one solution, multiple ones, or none
    ✓ Need to solve this new CSP (usually simpler) problem:
      - same constraints, domains have been reduced
Search vs. Domain Splitting

• Arc consistency ends: Some domains have more than one value → may or may not have a solution
  A. Apply Depth-First Search with Pruning or

  B. Split the problem in a number of disjoint cases CSP_i: for instance

  CSP with \( \text{dom}(X) = \{x_1, x_2, x_3, x_4\} \) becomes

  CSP_1 with \( \text{dom}(X) = \{x_1, x_2\} \) and
  CSP_2 with \( \text{dom}(X) = \{x_3, x_4\} \)

• Solution to CSP is the union of solutions to CSP_i
Example

Run “Scheduling Problem 2” in Alspace
- Try spitting on E (select 2 first, then 3 then 4)
Another Example

“Crossword 1” in Aispace,

- try splitting on D3 and then A3 (always “select half”)
Domain splitting

- For each subCSP, which arcs have to be on the ToDoArcs list when we get the subCSP by splitting the domain of X?
Searching by domain splitting

CSP, apply AC

If domains with multiple values

Split on one

CSP₁, apply AC

If domains with multiple values

Split on one

CSP₂, apply AC

If domains with multiple values…..Split on one
Another formulation of CSP as search

Arc consistency with domain splitting

- States: vector \((D(V_1), \ldots, D(V_n))\) of remaining domains, with \(D(V_i) \subseteq \text{dom}(V_i)\) for each \(V_i\)
- Start state: vector of original domains \((\text{dom}(V_1), \ldots, \text{dom}(V_n))\)
- Successor function:
  - reduce one of the domains + run arc consistency
- Goal state: vector of unary domains that satisfies all constraints
  - That is, only one value left for each variable
  - The assignment of each variable to its single value is a model
- Solution: that assignment
Arc consistency + domain splitting: example

3 variables: A, B, C
Domains: all \{1,2,3,4\}
A=B, B=C, A \neq C

\begin{align*}
(\{1,2,3,4\}, \{1,2,3,4\}, \{1,2,3,4\}) & \xrightarrow{AC \ (arc\ consistency)} (\{1,2,3,4\}, \{1,2,3,4\}, \{1,2,3,4\}) \\
(\{1,3\}, \{1,2,3,4\}, \{1,2,3,4\}) & \xrightarrow{AC} (\{1,3\}, \{1,3\}, \{1,3\}) \\
(\{2,4\}, \{1,2,3,4\}, \{1,2,3,4\}) & \xrightarrow{AC} (\{2,4\}, \{2,4\}, \{2,4\})
\end{align*}

\begin{align*}
A & \in \{1,3\} \\
B & \in \{1\} \\
& \xrightarrow{AC} (\{1,3\}, \{1\}, \{1,3\}) \\
& \xrightarrow{AC} (\{\}, \{\}, \{\}) \quad \text{No solution}
\end{align*}

\begin{align*}
A & \in \{2,4\} \\
B & \in \{3\} \\
& \xrightarrow{AC} (\{1,3\}, \{3\}, \{1,3\}) \\
& \xrightarrow{AC} (\{\}, \{\}, \{\}) \quad \text{No solution}
\end{align*}

\begin{align*}
A & \in \{2,4\} \\
B & \in \{2\} \\
& \xrightarrow{AC} (\{2,4\}, \{2\}, \{2,4\}) \\
& \xrightarrow{AC} (\{\}, \{\}, \{\}) \quad \text{No solution}
\end{align*}

\begin{align*}
A & \in \{2,4\} \\
B & \in \{4\} \\
& \xrightarrow{AC} (\{2,4\}, \{4\}, \{2,4\}) \\
& \xrightarrow{AC} (\{\}, \{\}, \{\}) \quad \text{No solution}
\end{align*}
Arc consistency + domain splitting: another example

3 variables: A, B, C
Domains: all \{1,2,3,4\}
A=B, B=C, A=C

A \in \{1,3\}
B \in \{1\}, \{3\}

Solution
\{1\}, \{1\}, \{1\}

Solution
\{3\}, \{3\}, \{3\}

AC (arc consistency)

Solution
\{2\}, \{2\}, \{2\}

Solution
\{4\}, \{4\}, \{4\}
Searching by domain splitting

CSP, apply AC
If domains with multiple values
Split on one

CSP₁, apply AC
If domains with multiple values
Split on one

CSP₂, apply AC
If domains with multiple values
Split on one

How many CSPs do we need to keep around at a time?
Assume solution at depth m and b children at each split
Systematically solving CSPs: Summary

• Build Constraint Network

• **Apply Arc Consistency**
  - One domain is empty →
  - Each domain has a single value →
  - Some domains have more than one value →

• **Apply Depth-First Search with Pruning** OR

• **Split the problem** in a number of disjoint cases
  - Apply Arc Consistency to each case, and repeat
Limitation of Systematic Approaches

- Many CSPs (scheduling, DNA computing, etc.) are simply too big for systematic approaches
- If you have $10^5$ vars with $\text{dom}(\text{var}_i) = 10^4$

<table>
<thead>
<tr>
<th>Systematic Search</th>
<th>Constraint Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branching factor $b =$</td>
<td>Size =</td>
</tr>
<tr>
<td>Solution depth $d =$</td>
<td></td>
</tr>
<tr>
<td>Complexity =</td>
<td>Complexity of AC =</td>
</tr>
</tbody>
</table>
Learning Goals for CSP

• Define possible worlds in term of variables and their domains
• Compute number of possible worlds on real examples
• Specify constraints to represent real world problems differentiating between:
  • Unary and k-ary constraints
  • List vs. function format
• Verify whether a possible world satisfies a set of constraints (i.e., whether it is a model, a solution)
• Implement the Generate-and-Test Algorithm. Explain its disadvantages.
• Solve a CSP by search (specify neighbors, states, start state, goal state). Compare strategies for CSP search. Implement pruning for DFS search in a CSP.
  • Define/read/write/trace/debug the arc consistency algorithm. Compute its complexity and assess its possible outcomes
• Define/read/write/trace/debug domain splitting and its integration with arc consistency