CPSC 322: Introduction to Artificial Intelligence

CSPs: Arc Consistency and Domain Splitting

Textbook reference: [4.4, 4.5]

Instructor: Varada Kolhatkar
University of British Columbia

Credit: These slides are adapted from the slides of the previous offerings of the course. Thanks to all instructors for creating and improving the teaching material and making it available!

Announcements

- Assignment 2 has been released and is due on **21 Oct 11:59pm**.
- Midterm **practice questions** are available on Piazza.
- Midterm time and location
  Time: Friday, Oct 25th, from 6pm to 7pm
  Location: Woodward 2
  (Instructional Resources Centre-IRC) (WOOD) - 2
- My office hours: Fridays from 11am to noon in ICCS 185

Lecture outline

- Recap CSPs (~5 mins) 👈🏻
- Solving CSPs
  - Arc consistency (~25 mins)
  - Domain splitting (~25 mins)
- Class activity (~15 mins)
- Summary and wrap up

CSP: Motivation

"…By applying constraint programming, Optimatch successfully provides prioritization lists and near-optimal assignments that take into account all resources and positions in the pool, as well as the complex constraints defining a good match…By applying the Optimatch assignment scheme, ***up to 15 percent more matches can be found*** *compared to a semi-automated first-come-first-serve scheme. In addition, Optimatch results show* ***an increase in the average quality of the matches****."*

https://www.research.ibm.com/haifa/dept/vst/csp_wf.shtml

Recap: CSPs definition

A **constraint satisfaction problem** (CSP) consists of

- a set of **variables** V
- a **domain** $dom(V_i)$ for each variable $V_i \in V$
- a set of **constraints** C

<u>Simple example:</u>

$V = \{V_1\}$

$dom(V_1) = \{2,4,8,16\}$

$C = \{c_1, c_2\}$

$c_1 : V_1 \neq 8$

$c_2 : V_1 > 2$

Recap: Possible worlds and models

A possible world of a CSP is an assignment of values to **all** of its variables.

A **model/solution** of a CSP is an assignment of values to all of its variables that **satisfies** all of its constraints.

Simple example:

$V = \{V_1\}$

$dom(V_1) = \{2,4,8,16\}$

$C = \{c_1, c_2\}$

$c_1 : V_1 \neq 8$

$c_2 : V_1 > 2$

Possible worlds and models

$\{V_1 = 2\}$

$\{V_1 = 4\}$ (a model)

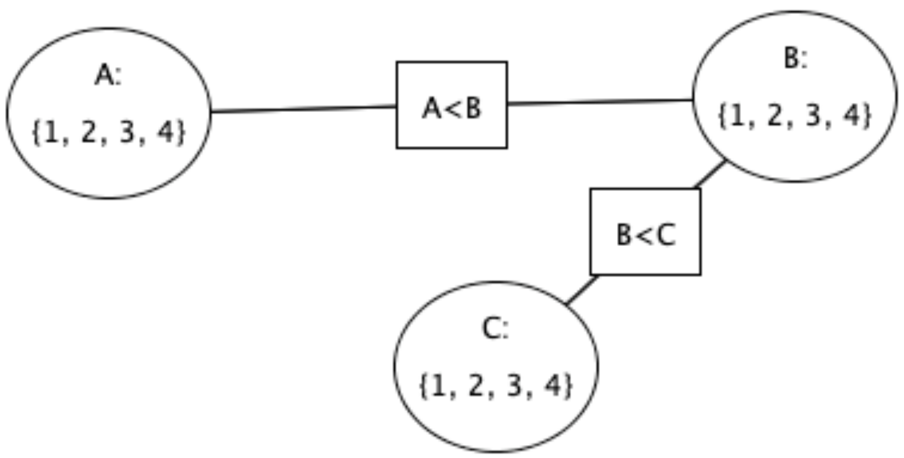
$\{V_1 = 8\}$

$\{V_1 = 16\}$ (a model)

Constraint networks

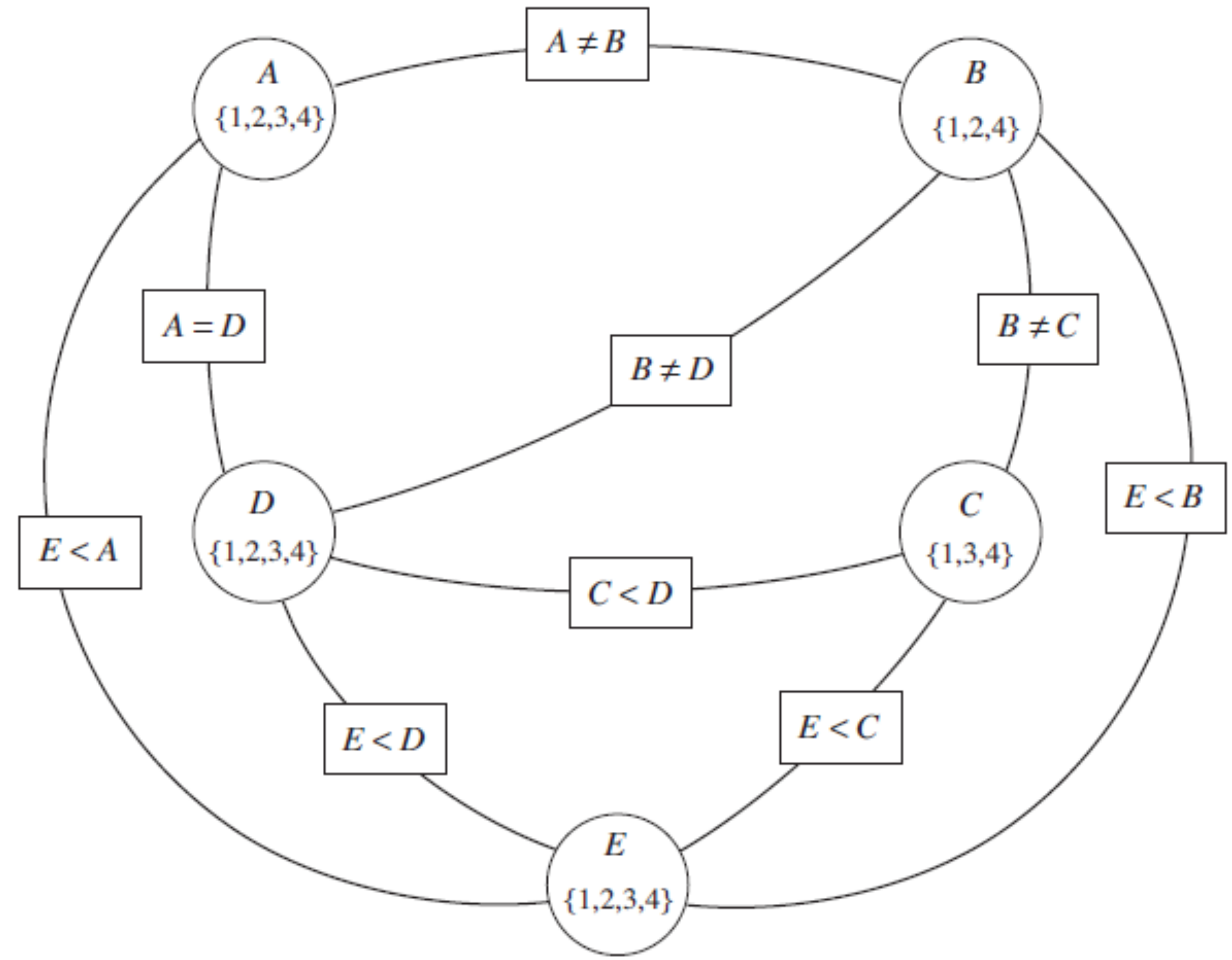
Definition: A **constraint network** is defined by a graph, with

- one node for every variable (drawn as circles or ovals)
- one node for every constraint (drawn as rectangles)
- undirected edges running between variable nodes and constraint nodes whenever a given variable is involved in a given constraint.



Recap: Constraint network

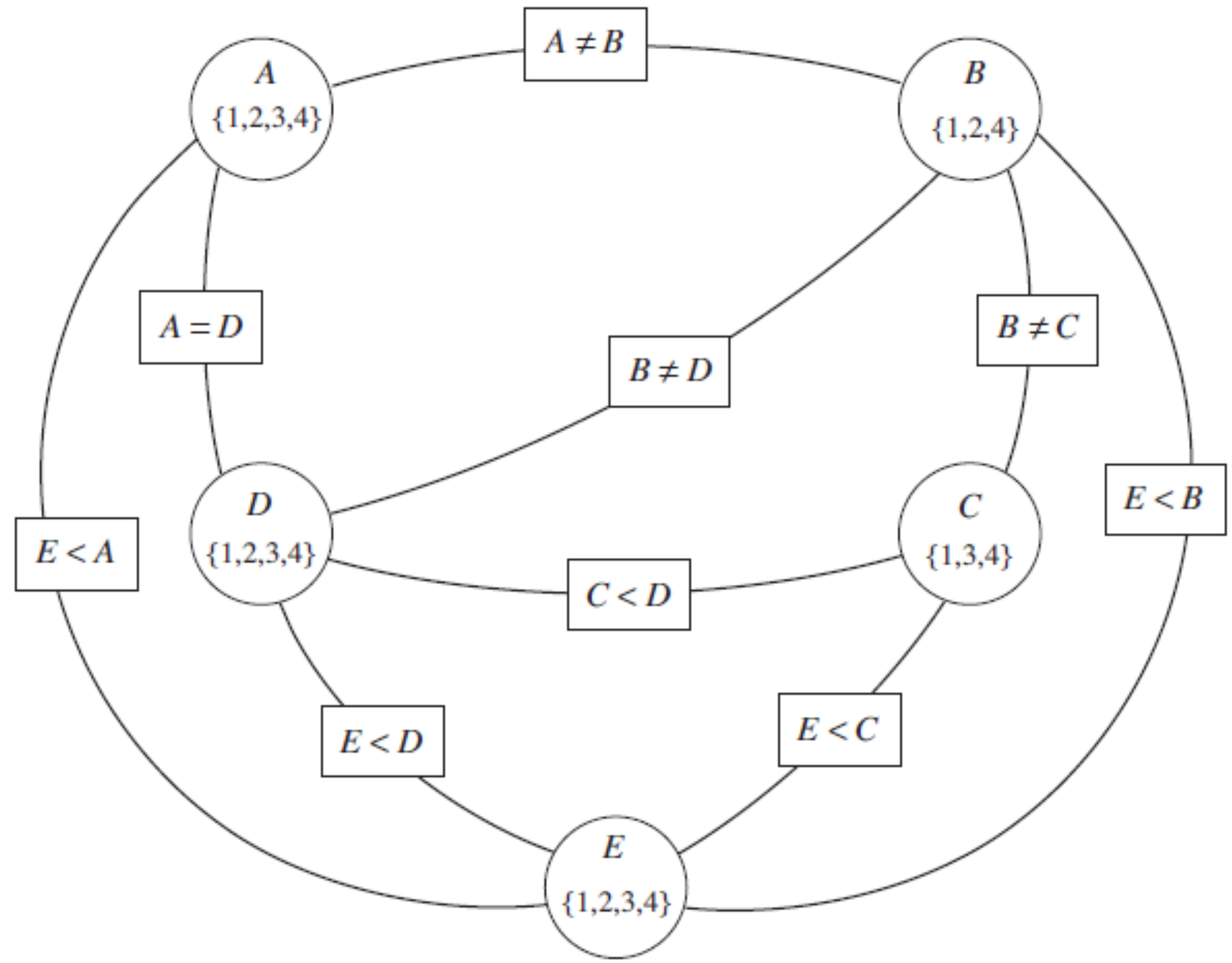
- How many variables?
- How many constraints?



Recap: Constraint network

- How many variables?
  5

- How many constraints?
  9



Solving CSPs

- Generate-and-Test (last lecture)
    - Time complexity $O(cd^k)$

$k \rightarrow$ **# variables**
$d \rightarrow$ **domain size**
$c \rightarrow$ **# constraints**

- Graph search (DFS with backtracking) (last lecture)
    - $O(d^k)$

Can we do better?

- **Today: Arc consistency with domain splitting**

Today: Learning outcomes

From this lecture, students are expected to be able to:

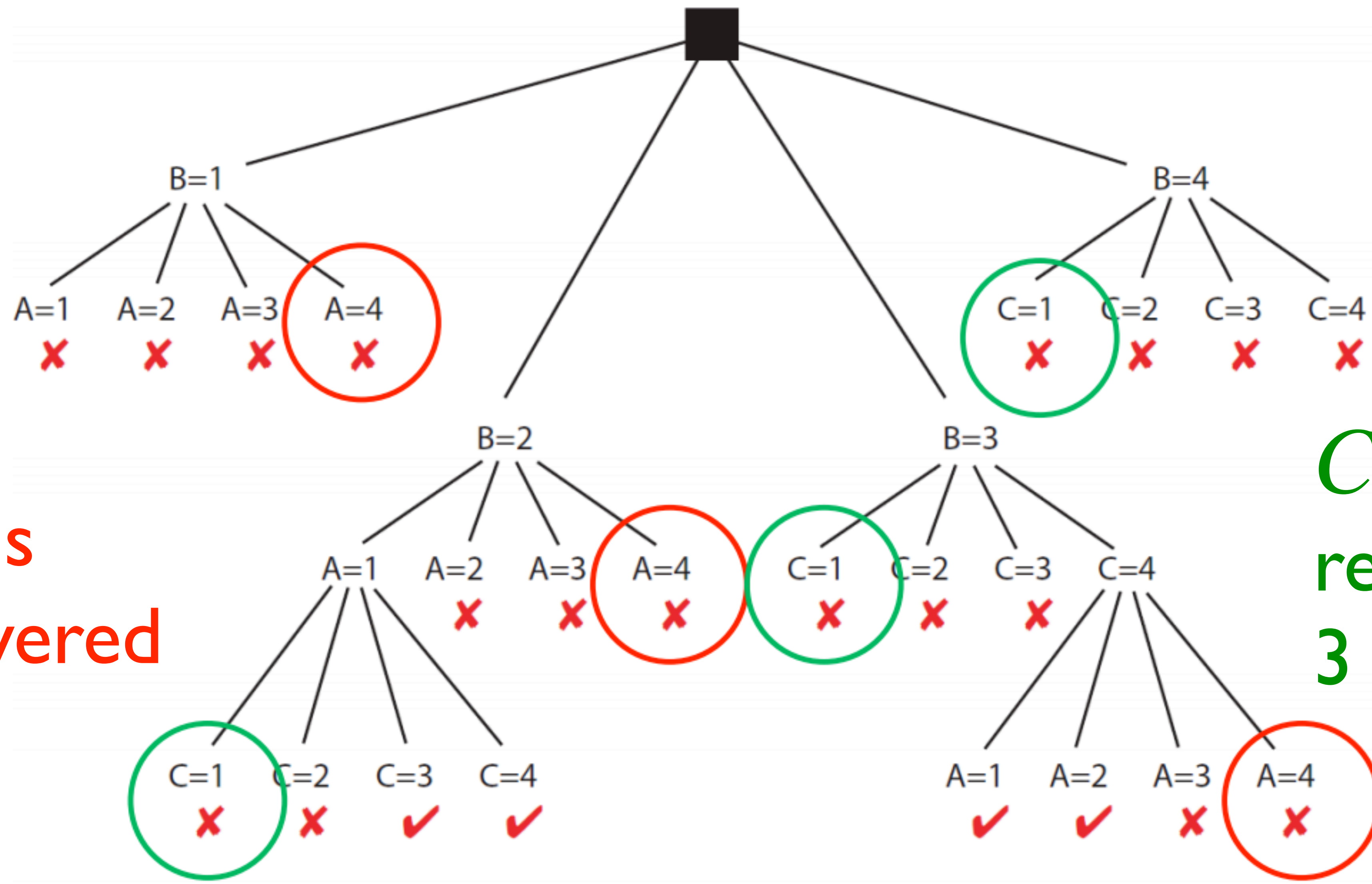
- Define/read/write/trace/debug the arc consistency algorithm. Compute its complexity and assess its possible outcomes.

- Define/read/write/trace/debug domain splitting and its integration with arc consistency.

Graph searching repeats work

<u>Toy problem</u>

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



$A \neq 4$ is rediscovered 4 times

$C \neq 1$ is rediscovered 3 times

Can we do better than search?

Key idea: Prune the domains as much as possible before searching for a solution.

Pruning for unary constraints

Key idea: Prune the domains as much as possible before searching for a solution. For **unary constraints**, prune the values in domains that do not satisfy the constraints.

Definition: A variable is **domain consistent** if no value of its domain is ruled impossible by any unary constraint.

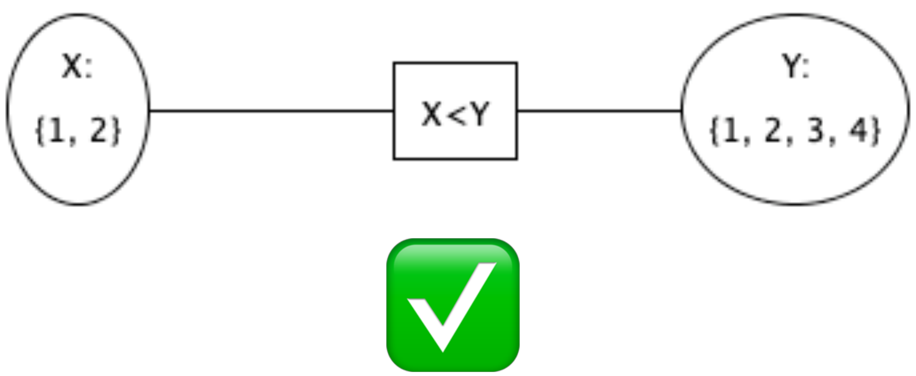
Example:
Suppose $dom(V_1) = \{1,2,3,4\}$ and a constraint is $V_1 \neq 4$, then V_1 is NOT domain consistent.

Arc consistency

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

Arc consistency

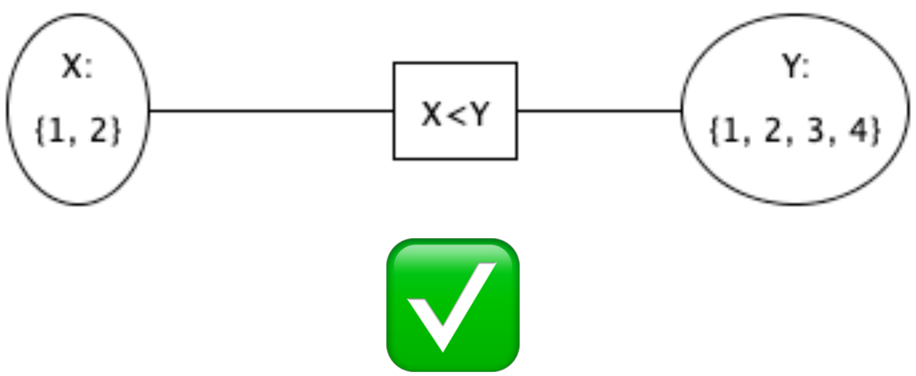
Definition: An arc $< X, r(X, Y) >$ is **arc consistent** if for each value $x \in dom(X)$, there is some value $y \in dom(Y)$, such that $r(x, y)$ is satisfied.



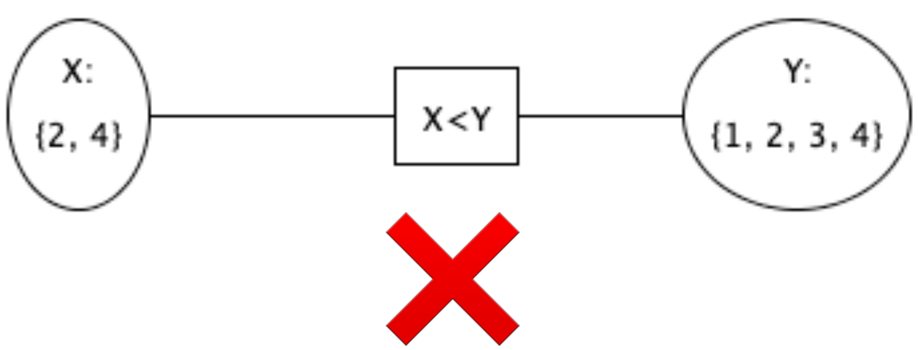
Arc $< X, r(X, Y) >$ is consistent.
$dom(X) = \{1,2\}, dom(Y) = \{1,2,3,4\}$
Arc consistent: Both $X = 1$ and $X = 2$ have OK values for Y.

Arc consistency

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



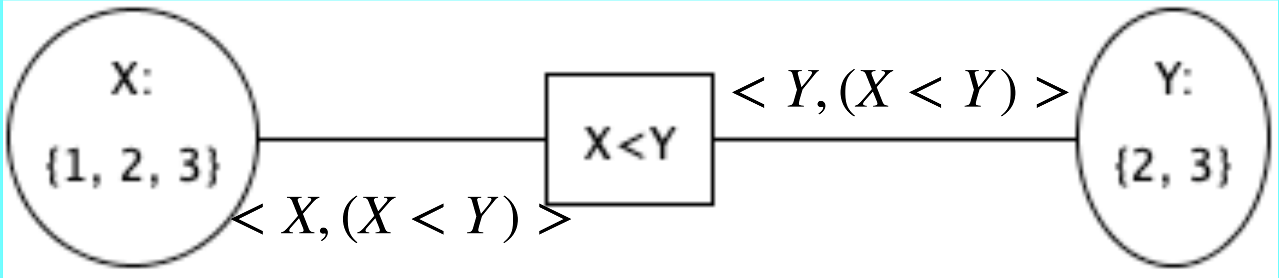
Arc $< X, r(X, Y) >$ is consistent.
$dom(X) = \{1,2\}, dom(Y) = \{1,2,3,4\}$
Arc consistent: Both $X = 1$ and $X = 2$ have OK values for Y.



Arc $< X, r(X, Y) >$ is not consistent.
$dom(X) = \{2,4\}, dom(Y) = \{1,2,3,4\}$
Not arc consistent because no value in $dom(Y)$ that satisfies $X < Y$ if $X = 4$.

Arc consistency

Consider the two arcs $< X, (X < Y) >$ and $< Y, (X < Y) >$ in the constraint network below.



A. Both arcs are consistent

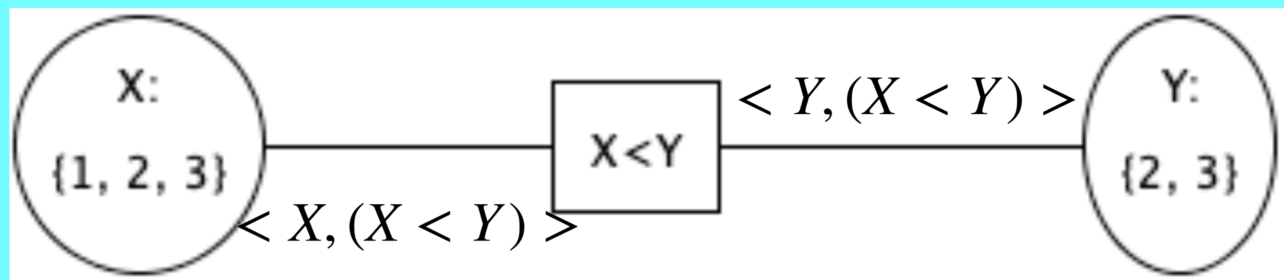
B. Left consistent and right inconsistent

C. Right consistent and left inconsistent

D. Both arcs are inconsistent

Arc consistency

Consider the two arcs $< X, (X < Y) >$ and $< Y, (X < Y) >$ in the constraint network below.



A. Both arcs are consistent

B. Left consistent and right inconsistent

C. Right consistent and left inconsistent ✅

D. Both arcs are inconsistent

Why arc consistency?

- Arc consistency is a desirable property because when an arc is non-consistent, the value that makes it inconsistent is never going to be part of the global solution.

- Waste of time to include that in backtracking.

Enforcing arc consistency

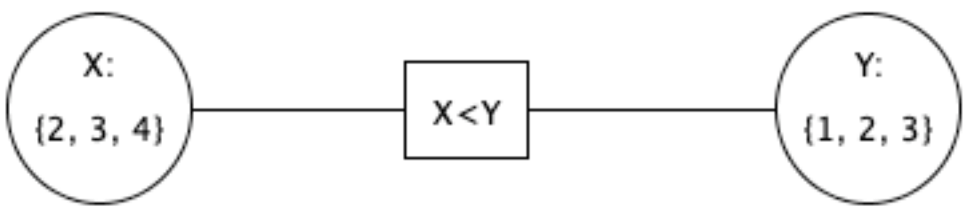
- If an $< X, r(X, Y) >$ is not arc consistent,
  - delete all values x in $dom(X)$ for which there is no corresponding value in $dom(Y)$ to make the arc $< X, r(X, Y) >$ consistent.
  - This removal can never rule out any models/solutions. Why?

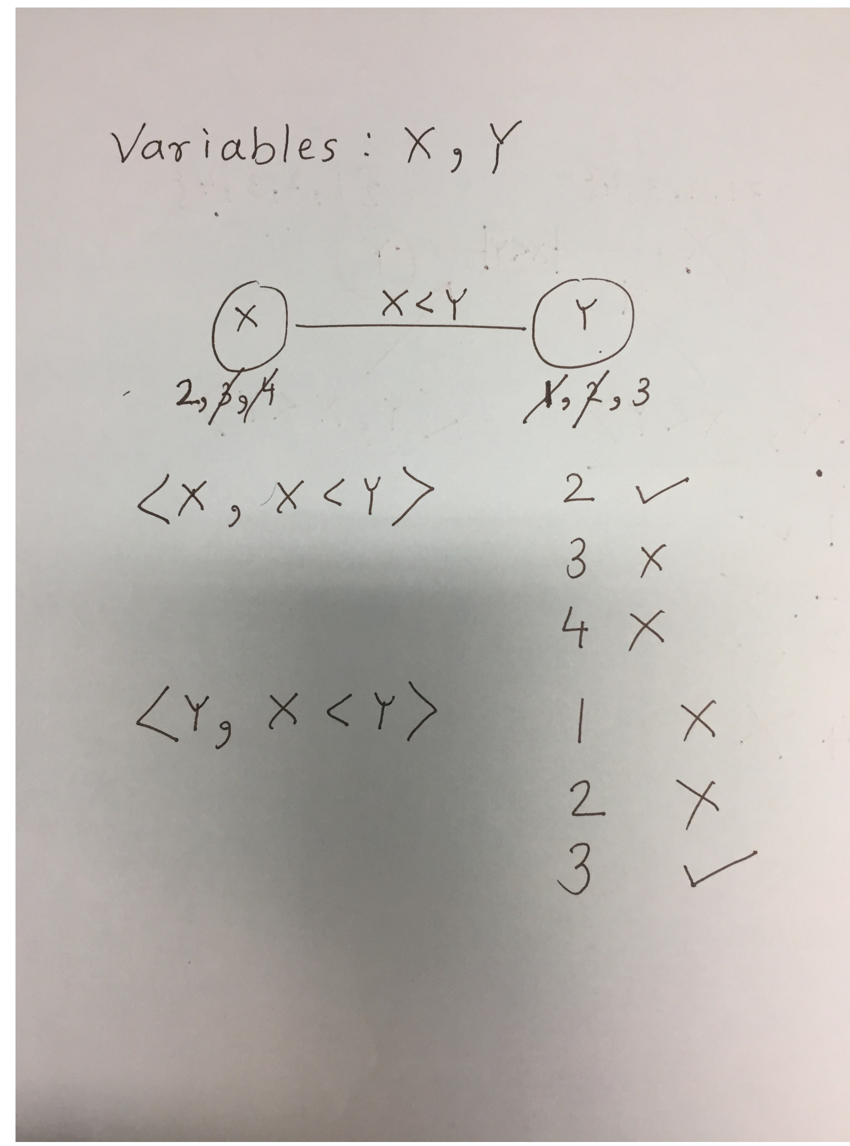
Example: enforcing arc consistency

<u>Toy problem</u>

Variables: X, Y; dom(X) = {2,3,4}; dom(Y) = {1,2,3};

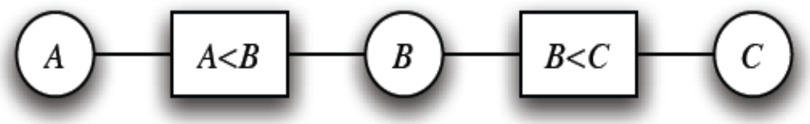
Constraints: X < Y

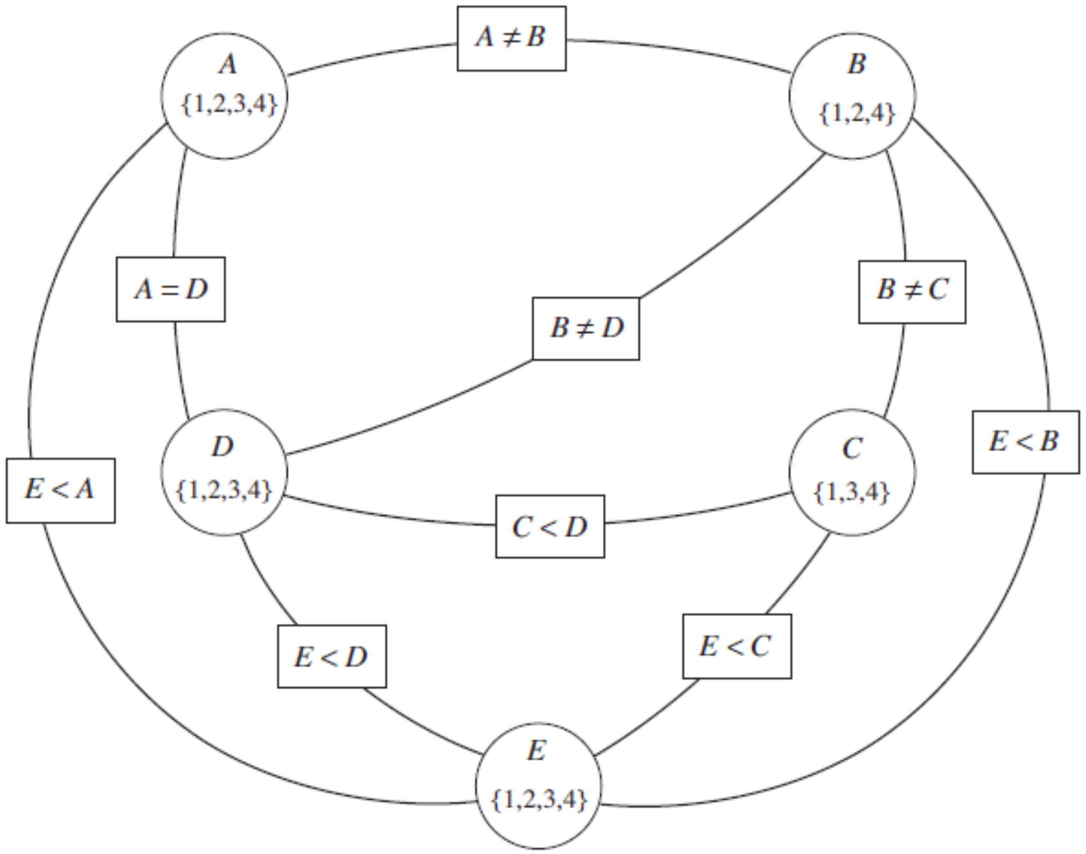




Enforcing arc consistency

How can we make a constraint network arc consistent?
Arc consistency algorithm





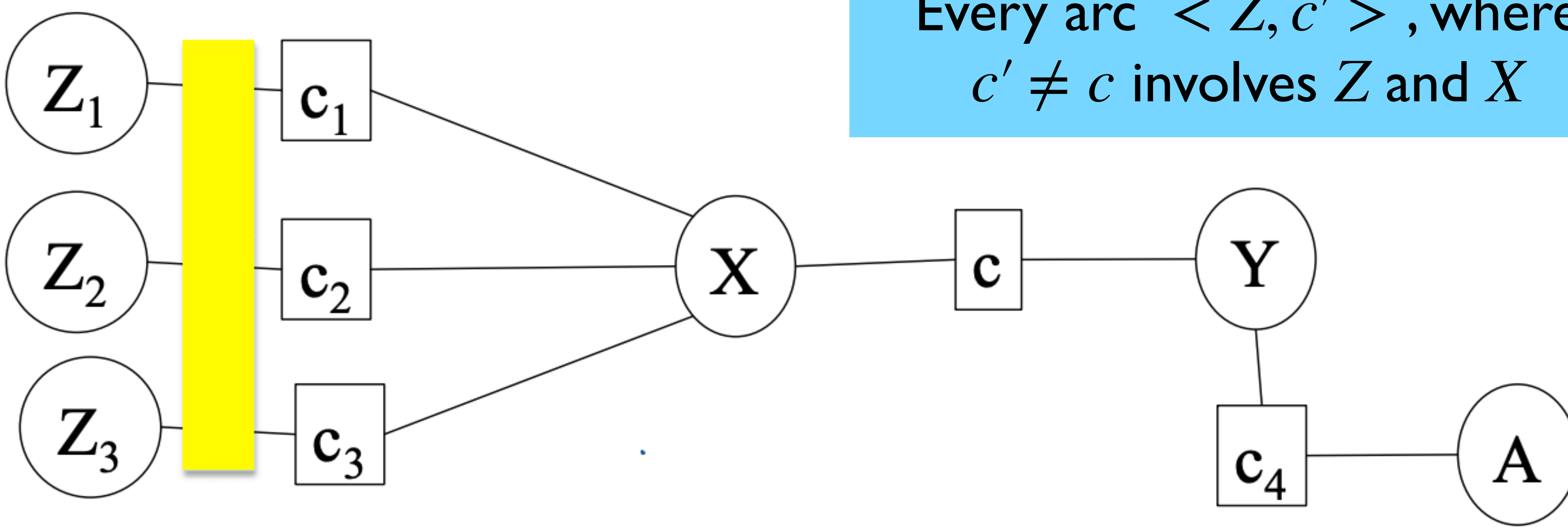
Arc consistency algorithm: high-level strategy

- Consider the arcs in turn, making each arc consistent.
- Reconsider arcs that could be made inconsistent again by this pruning of the domains
- Eventually reach a ‘fixed point’: all arcs consistent

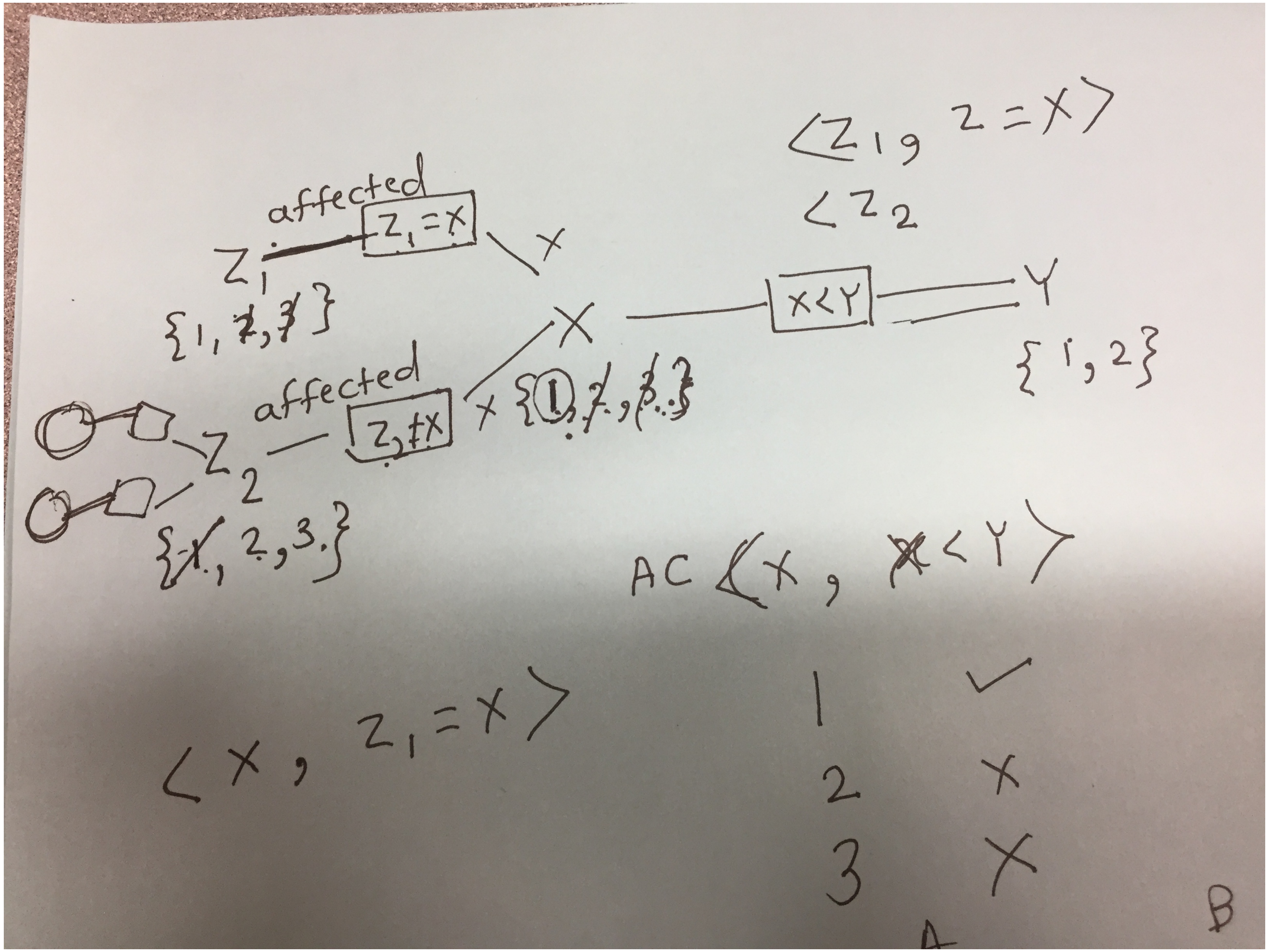
Which arcs need to be considered?

When we reduce the domain of a variable X to make an arc $< X, c >$ arc consistent, which arcs do we need to consider?

Every arc $< Z, c' >$, where $c' \neq c$ involves Z and X



Which arcs need to be considered?



Arc consistency pseudo code

```
TDA → all arcs in constraint network

while (TDA is not empty):
    select arc a from TDA
    if a is not consistent then
        make a consistent
        add arcs to TDA that may now be
inconsistent
```

Arc consistency algorithm (for binary constrinats)

```
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
        DX is a set of values for each variable X
        TDA is a set of arcs

    for each variable X do
        DX ←dom(X)
    TDA ←{⟨X,c⟩| X ∈ V, c ∈ C  and X ∈ scope(c)}

    while (TDA  {})
        select ⟨X,c⟩ ∈TDA
        TDA ←TDA  \ {⟨X,c⟩}
        NDX ←{x| x ∈ DX and  y ∈ DY s.t. (x, y) satisfies c}
        if (NDX ≠  DX) then
            TDA ← TDA  ∪ { ⟨Z,c'⟩ | X ∈ scope(c'), c'  c, Z ∈ scope(c') \ {X} }
            DX ← NDX

     return {DX| X  is a variable}
```

TDA: ToDoArc, blue arcs in AIspace

Scope of constraint c is the set of variables involved in that constraint

X's domain changed $\Rightarrow$ arcs (Z, c') for variables Z sharing a constraint c' with X could become inconsistent .

ND_x: values x for X for which there a value y supporting x

Arc consistency algorithm: Interpreting outcomes

Three possible outcomes (when all arcs are arc consistent):

Arc consistency algorithm: Interpreting outcomes

Three possible outcomes (when all arcs are arc consistent):

Each domain has a single value.
We have a (unique) solution.
E.g., Variables = {A, B, C} with domains {1,2,3}; constraints: A < B, B <C

Arc consistency algorithm: Interpreting outcomes

Three possible outcomes (when all arcs are arc consistent):

Each domain has a single value.
We have a (unique) solution.
E.g., Variables = {A, B, C} with domains {1,2,3}; constraints: $A < B$, $B < C$

At least one domain is empty.
No solution! All values are ruled out for this variable.
E.g., Variables = {A, B, C}; dom{A} = {1}, dom(B) = {1,2}, dom(C) = {1,2};
constraints: $A = B$, $B = C$, $A \neq C$

Arc consistency algorithm: Interpreting outcomes

Three possible outcomes (when all arcs are arc consistent):

Each domain has a single value.
We have a (unique) solution.
E.g., Variables = {A, B, C} with domains {1,2,3}; constraints: $A < B$, $B < C$

At least one domain is empty.
No solution! All values are ruled out for this variable.
E.g., Variables = {A, B, C}; dom{A} = {1}, dom(B) = {1,2}, dom(C) = {1,2};
constraints: $A = B$, $B = C$, $A \neq C$

Some domains have more than one value
There may be a solution, multiple ones, or none. Need to solve this new CSP (usually simpler) problem: same constraints, domains have been reduced
E.g., built-in example "simple problem 2"

Arc consistency algorithm: Complexity

Let max size of a variable domain be d.
Let the number of variables be n.
Let the number of constraints be c
Let all constraints be **binary**.

How often will we prune the domain of variable V?

A. $O(n)$

B. $O(d)$

C. $O(n \times d)$

D. $O(c)$

Arc consistency algorithm: Complexity

Let max size of a variable domain be d.
Let the number of variables be n.
Let the number of constraints be c
Let all constraints be **binary**.

How often will we prune the domain of variable V?

A. $O(n)$

B. $O(d)$ ✅

C. $O(n \times d)$

D. $O(c)$

Arc consistency algorithm: Complexity

Let max size of a variable domain be d.
Let the number of variables be n.
Let the number of constraints be c
Let all constraints be **binary**.

How often will we prune the domain of variable V?

A. $O(n)$

B. $O(d)$ ✅

d possible values for each variable.

C. $O(n \times d)$

D. $O(c)$

Arc consistency algorithm: Complexity

Let max size of a variable domain be d.
Let the number of variables be n.
Let the number of constraints be c
Let all constraints be **binary**.

How many arcs will be put on the ToDoArc (TDA) list when pruning domain of variable V?
O(degree of variable V)
Across all variables, sum of degrees of all variables =
$2 \times$ number of constraints $= 2 \times c$
Together: we'll put $O(dc)$ arcs on the ToDoArc list

Arc consistency algorithm: Complexity

Let max size of a variable domain be d.
Let the number of variables be n.
Let the number of constraints be c
Let all constraints be **binary**.

Checking consistency for each arc.
Have to check constraints for each value in the head of the arc to the each value in the tail of the arc.

$O(d^2)$

Arc consistency algorithm: Complexity

Let max size of a variable domain be d.
Let the number of variables be n.
Let the number of constraints be c
Let all constraints be **binary**.

Overall complexity: arcs put on the TDA list × constraints consistency checking for each arc = $O(cd^3)$

Compare to $O(d^n)$ of DFS! Arc consistency is MUCH faster.

Arc consistency algorithm: Complexity

Let max size of a variable domain be d.
Let the number of variables be n.
Let the number of constraints be c
Let all constraints be **binary**.

Scales **linearly** with the number of constraints and **cubically** with the size of the domain

Overall complexity: arcs put on the TDA list $\times$ constraints consistency checking for each arc = $O(cd^3)$

Compare to $O(d^n)$ of DFS! Arc consistency is MUCH faster.

Arc consistency algorithm: Complexity

Let max size of a variable domain be d.
Let the number of variables be n.
Let the number of constraints be c
Let all constraints be **binary**.

Scales **linearly** with the number of constraints and **cubically** with the size of the domain

Overall complexity: arcs put on the TDA list $\times$ constraints consistency checking for each arc = $O(cd^3)$

Compare to $O(d^n)$ of DFS! Arc consistency is MUCH faster.

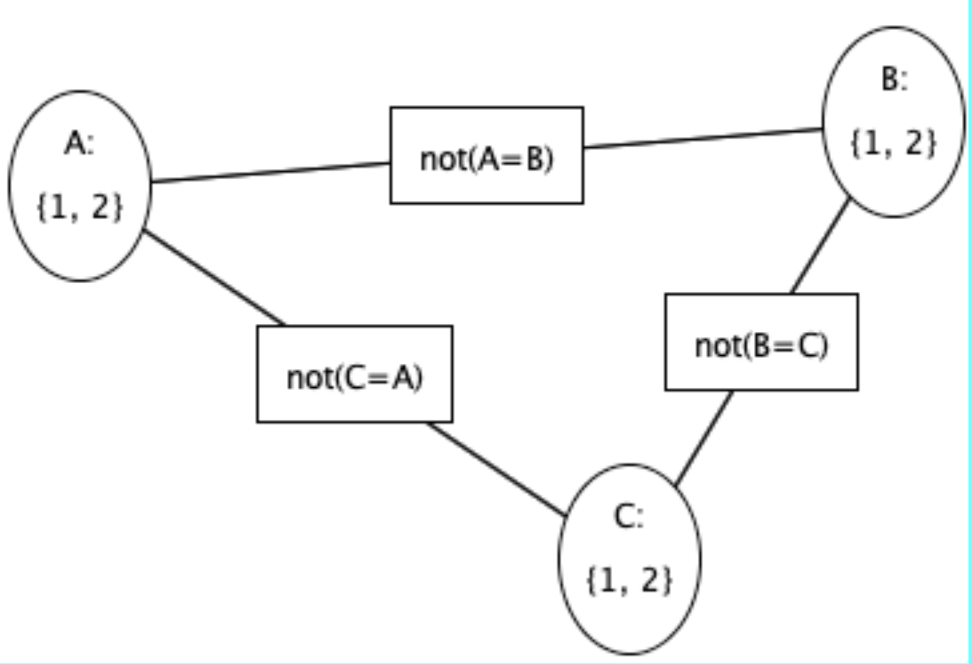
Scales **exponentially** with the number of variables

Domain splitting

Can we have an arc consistent network with non-empty domains that has no solution?

A. Yes

B. No

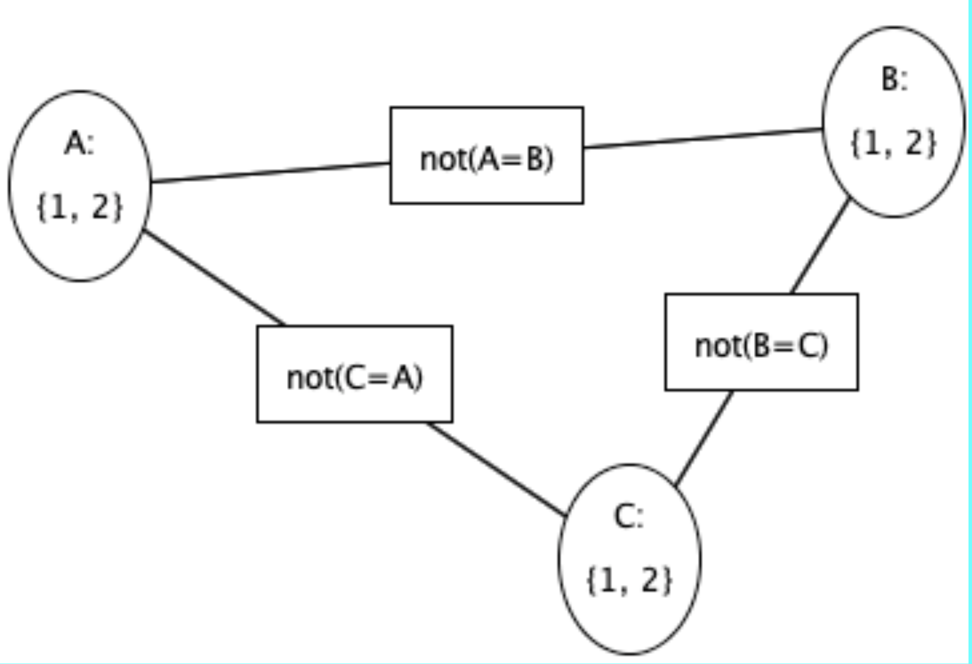


Domain splitting

Can we have an arc consistent network with non-empty domains that has no solution?

A. Yes ✅

B. No



Domain splitting (case analysis)

- Arc consistency ends: Some domains have more than one value → may or may not have a solution
  - Apply Depth-First Search with Pruning or
  - Split the problem in a number of disjoint cases
  - Solution to CSP is the **union** of solutions to these disjoint cases

Example:
CSP with $dom(X) = \{x_1, x_2, x_3, x_4\}$ becomes
CSP_1 with $dom(X) = \{x_1, x_2\}$
CSP_2 with $dom(X) = \{x_3, x_4\}$
Solution is the union of solutions of CSP_1 and CSP_2

Domain splitting in action

Trace it on "simple problem 2"

Coming up

Readings for next class

- 4.5 Domain Splitting
- 4.7 Local Search