Constraint Satisfaction Problems

Mostly adapted from chapter 4 of AI 2E by David Poole and Alan Macworth

Constraint Satisfaction Problem

A CSP is characterised by:

- A set of **variables** $V_1, V_2, \ldots, V_n$
- A set of associated **domains**, one for each variable. A domain D_{Vi} is the set of values that V_i can assume.
- A set of **constraints** on various subsets of the variables which specify legal combinations of values for these variables.

A **solution** to the CSP is an *n*-tuple of values for the variables that satisfies all the constraints.

Example CSP: scheduling activities

- Variables: A, B, C, D, E that represent the starting times of various activities.
- Domains: $\mathbf{D}_A = \{1, 2, 3, 4\}$, $\mathbf{D}_B = \{1, 2, 3, 4\}$, $\mathbf{D}_C = \{1, 2, 3, 4\}$, $\mathbf{D}_D = \{1, 2, 3, 4\}$, $\mathbf{D}_E = \{1, 2, 3, 4\}$
- Constraints:

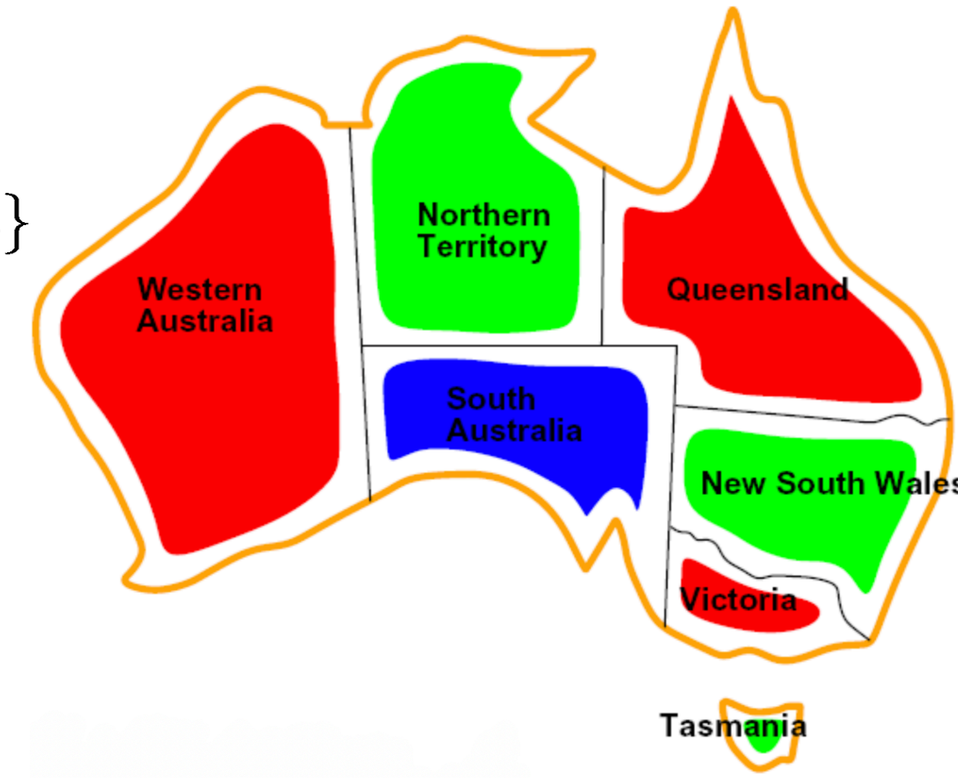
$$(B \neq 3) \wedge (C \neq 2) \wedge (A \neq B) \wedge (B \neq C) \wedge$$
$$(C < D) \wedge (A = D) \wedge (E < A) \wedge (E < B) \wedge$$
$$(E < C) \wedge (E < D) \wedge (B \neq D).$$

Example CSP: Australian map colouring

- Variables: *WA, NT, Q, NSW, V, SA, T*
- Domains: the same for all variables: *{red, green, blue}*
- Constraints: adjacent regions must have different colours. (e.g. *WA≠NT, ...*)

Solutions are assignments satisfying all constraints, e.g.:

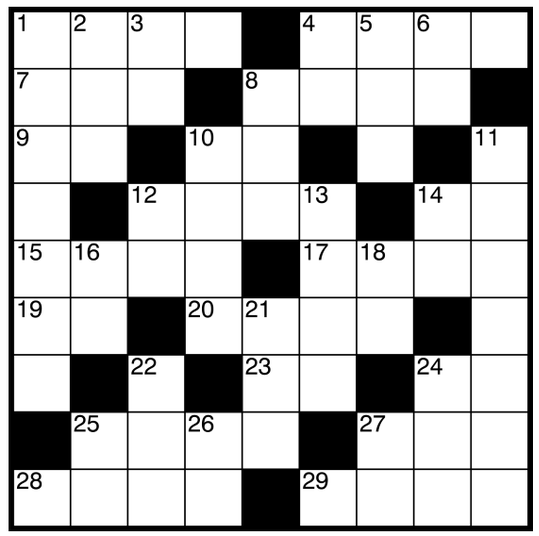
$$\{WA = red, NT = green, Q = red,$$
$$NSW = green, V = red, SA = blue, T = green\}$$

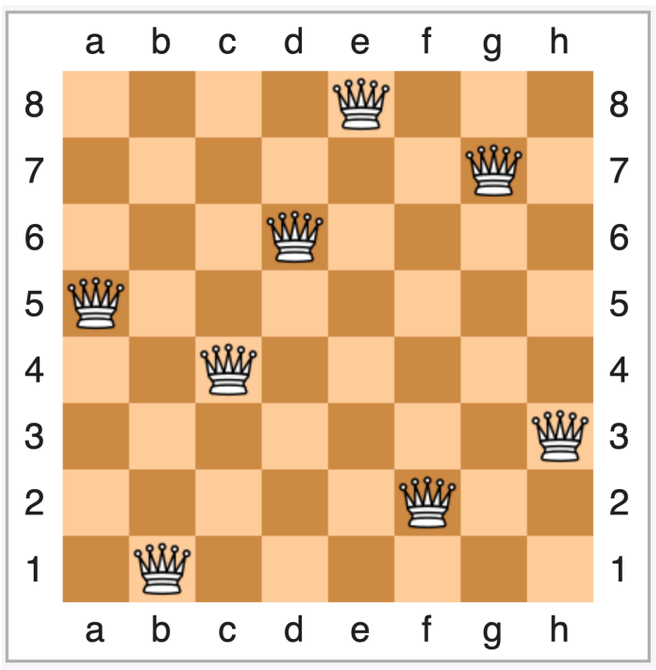


More CSP examples: logic puzzles

- Sudoko
- Crosswords
- Eight queens puzzle

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9





Practical CSP examples

- Scheduling, Timetabling, Rostering

- Car sequencing (assembly lines)

- Positioning agents with constraints such as:
  - minimum distance between each pair
  - maximum distance of the closest agent
  - maximum distance from a command centre

Generate-and-Test Algorithm

- Generate the assignment space $\mathbf{D} = \mathbf{D}_{V_1} \times \mathbf{D}_{V_2} \times \ldots \times \mathbf{D}_{V_n}$. Test each assignment with the constraints.
- Example:

$$\begin{aligned} \mathbf{D} &= \mathbf{D}_A \times \mathbf{D}_B \times \mathbf{D}_C \times \mathbf{D}_D \times \mathbf{D}_E \\ &= \{1,2,3,4\} \times \{1,2,3,4\} \times \{1,2,3,4\} \\ &\quad \times \{1,2,3,4\} \times \{1,2,3,4\} \\ &= \{\langle 1,1,1,1,1\rangle, \langle 1,1,1,1,2\rangle, ..., \langle 4,4,4,4,4\rangle\}. \end{aligned}$$

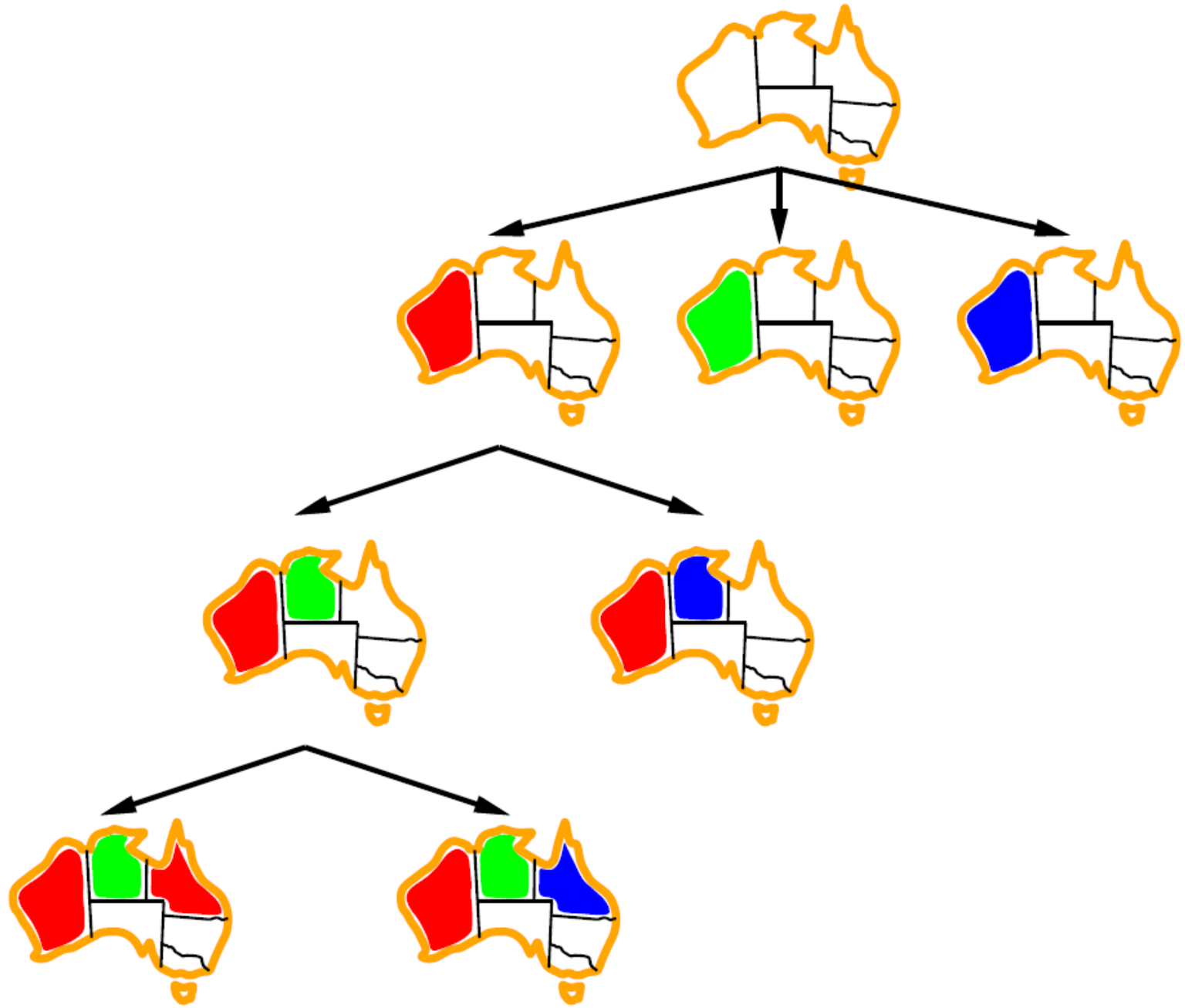
- Generate-and-test is always exponential in the number of variables.

Backtracking algorithm

- 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 Assignment $A = 1 \wedge B = 1$ is inconsistent with constraint $A \neq B$ regardless of the value of the other variables.

Backtracking example



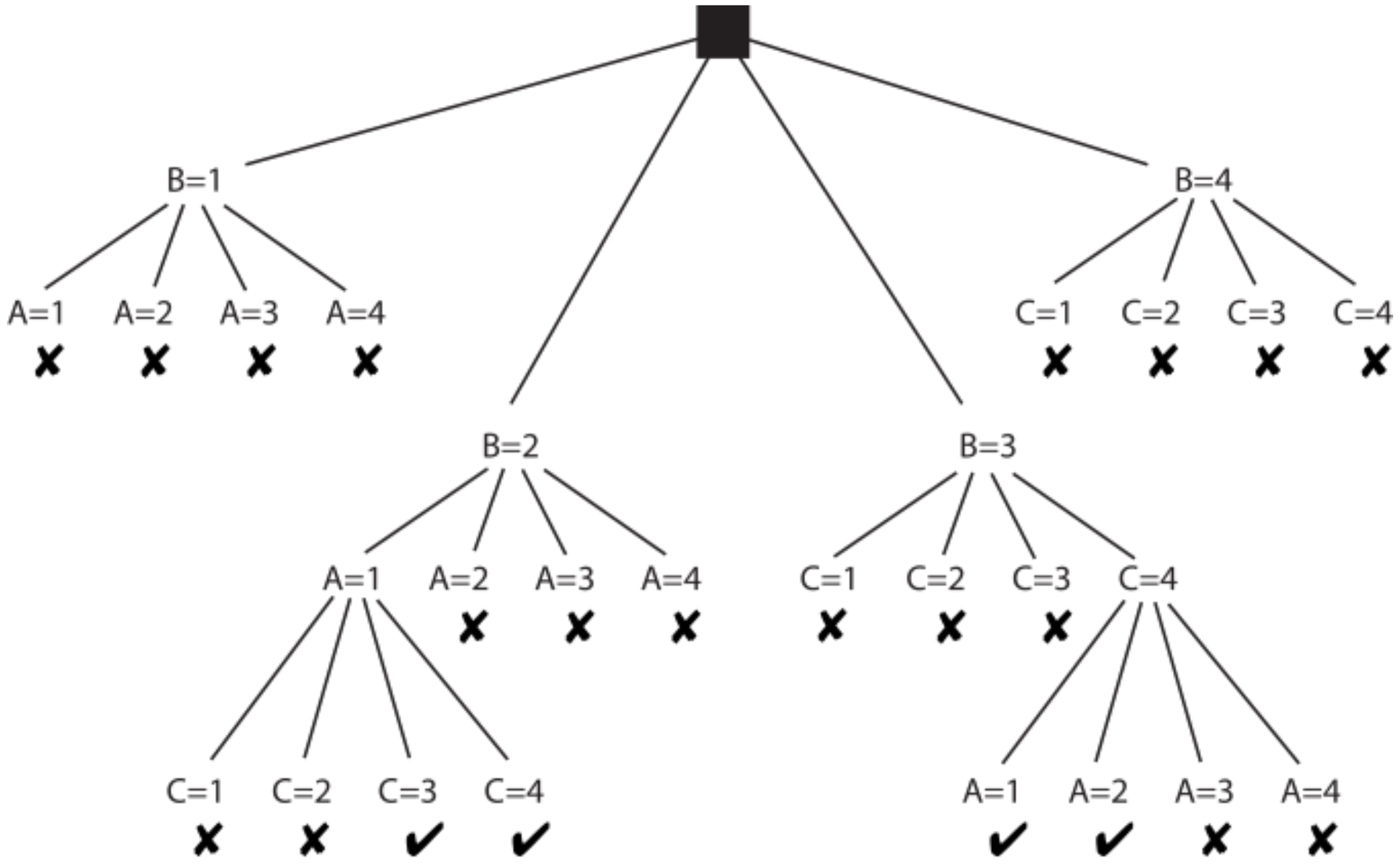
CSP as graph search

A CSP can be represented as a graph-searching algorithm:

- 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 $y_i \in dom(Y)$ there is a neighbour $X_1 = v_1, \ldots, X_k = v_k, Y = y_i$ if this assignment is consistent with the constraints on these variables.
- The start node is the empty assignment.
- A goal node is a total assignment that satisfies the constraints.

Graph search example

- Consider a CSP with variables *A*, *B*, and *C* each with domain *{1, 2, 3, 4}*. The constraints are *A* < *B* and *B* < *C*. The search tree may look like this:



CSP algorithms

- CSP is NP-hard. Why?

- While there is no escape from this, some instances of CSP can be solved more efficiently by exploiting certain properties.

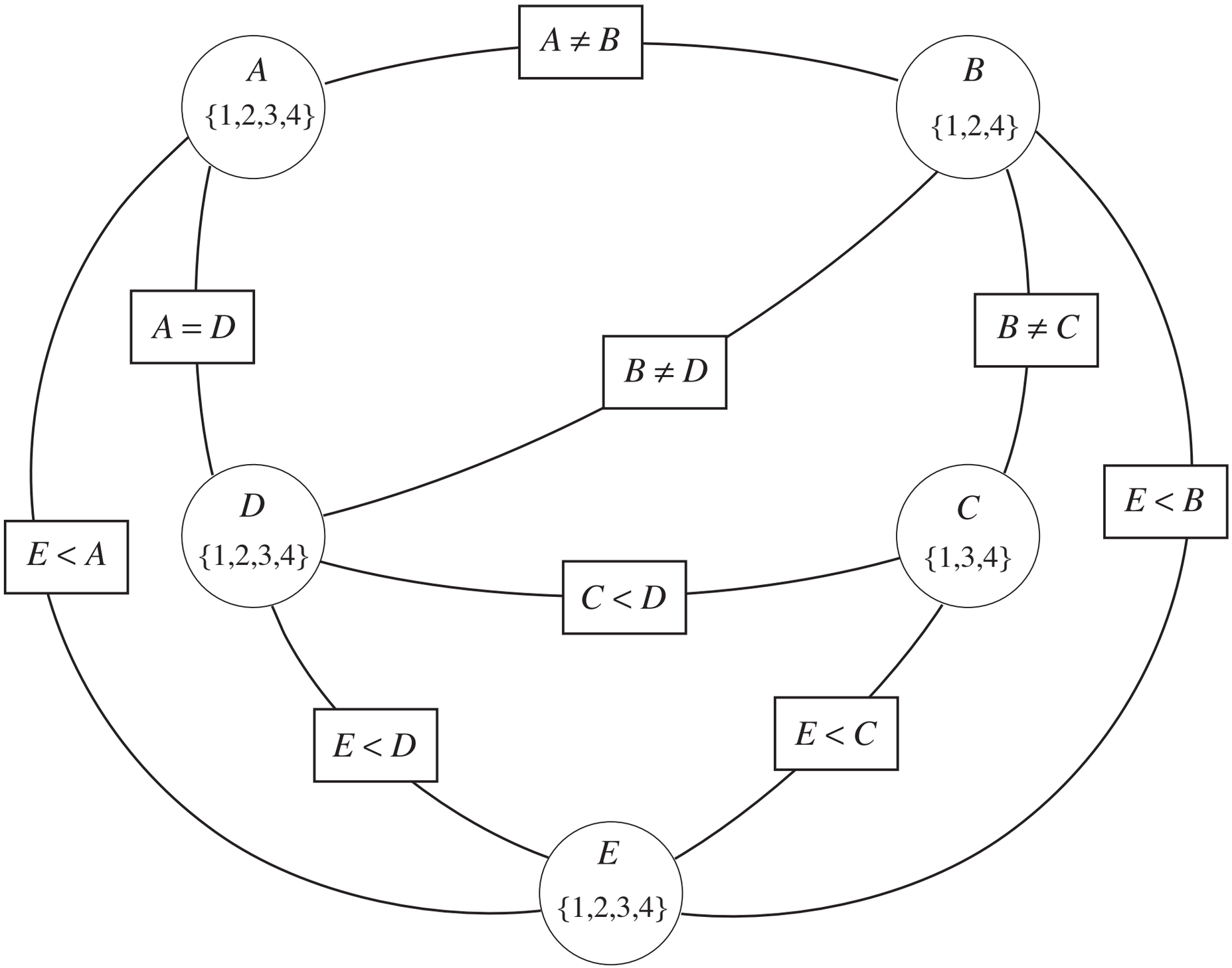
- We will look at three algorithms:
  - Arc consistency
  - Domain splitting
  - Variable elimination

Constraint networks

An instance of CSP can be represented as a 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*.

Example Constraint Network



Example CSP: Cryptarithmetic

- Variables:

  $F \; T \; U \; W \; R \; O \; X_1 \; X_2 \; X_3$

- Domains:

  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

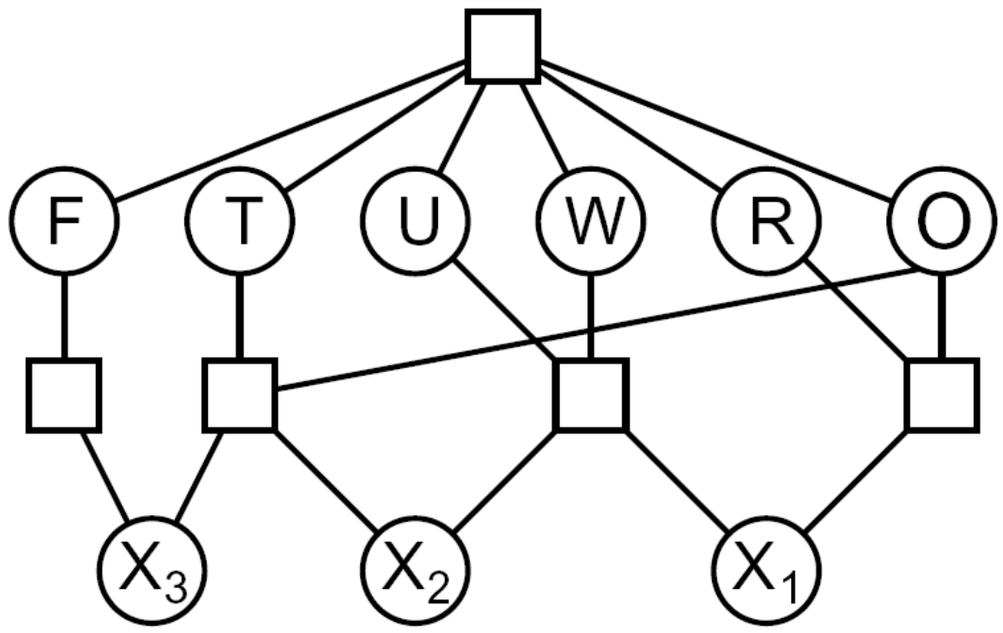
- Constraints:

  alldiff(F, T, U, W, R, O)

  $O + O = R + 10 \cdot X_1$

  $\cdots$

$$\begin{array}{cccc} & T & W & O \\ + & T & W & O \\ \hline F & O & U & R \end{array}$$



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.
- If an 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 $dom(\overline{Y})$ may be deleted from $dom(X)$ to make the arc $\langle X, r(X, \overline{Y}) \rangle$ consistent.

Arc consistency example

Consider an instance of CPS with variables *A, B, C* where the domains of all the variables are *{1, 2, 3, 4}* and the only constraint is *A* + *B* = *C*. Make the arc <*A, A*+*B*=*C*> arc-consistent (one iteration only).

Arc consistency algorithm

- The arcs can be considered in turn making each arc consistent.
- An arc $\langle X, r(X, \overline{Y}) \rangle$ needs to be revisited if the domain of one of the Y's is reduced.
- Three possible outcomes (when all arcs are 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

General Arc Consistency Algorithm

1: **procedure** $GAC(\langle Vs, dom, Cs\rangle)$
2: ▷ Returns arc consistent domains for CSP $\langle Vs, dom, Cs\rangle$
3: **return** $GAC2(\langle Vs, dom, Cs\rangle, \{\langle X, c\rangle \mid c \in Cs \text{ and } X \in scope(c)\})$
4: **procedure** $GAC2(\langle Vs, dom, Cs\rangle, to\_do)$
5: **while** $to\_do \neq \{\}$ **do**
6: **select** and **remove** $\langle X, c\rangle$ from to_do
7: let $\{Y_1, \ldots, Y_k\} = scope(c) \setminus \{X\}$
8: $ND := \{x \mid x \in dom[X] \text{ and exists } y_1 \in dom[Y_1] \ldots y_k \in dom[Y_k] \text{ such that } c(X{=}x, Y_1{=}y_1, \ldots, Y_k{=}y_k)\}$
9: **if** $ND \neq dom[X]$ **then**
10: $to\_do := to\_do \cup \{\langle Z, c'\rangle \mid \{X, Z\} \subseteq scope(c'), c' \neq c, Z \neq X\}$
11: $dom[X] := ND$
12: **return** dom

Domain splitting

- The idea is to split a problem into a number of disjoint cases and solve each case separately.
- The set of all solutions to the initial problem is the union of the solutions to each case.
- Example: If X is a variable with the domain {0, 1}, then all the solutions either have X=0 or X=1. To find out all the solutions, set X=0, find all the solutions with this assignment, and then set X=1 and find all the solutions with this assignment.
- Domain splitting algorithm:
  - Input: a CSP instance
  - Select a variable X that has more than one value in its domain;
  - Split the domain into two disjoint non-empty sets;
  - Create two new CPS instances that are the copy of the original except for the variable X where each new instance has one of the new split domains.
- The domain splitting algorithm is not an interesting algorithm on its own but can be combined with the Arc Consistency algorithm to find solutions.

Finding solutions when Arc Consistency Algorithm finishes

- If some domains have more than one element $\Longrightarrow$ search
- Split a domain, then recursively solve each half.
- We only need to revisit arcs affected by the split.
- It is often best to split a domain in half.

CSP Solver using Arc Consistency and Domain Splitting algorithms

1: **procedure** $CSP\_Solver(\langle Vs, dom, Cs\rangle)$
2: ▷ Returns a solution to CSP $\langle Vs, dom, Cs\rangle$ or *false* if there is no solution
3: **return** $Solve2(\langle Vs, dom, Cs\rangle, \{\langle X, c\rangle \mid c \in Cs \text{ and } X \in scope(c)\})$
4: **procedure** $Solve2(\langle Vs, dom, Cs\rangle, to\_do)$
5: $dom_0 := GAC2(\langle Vs, dom, Cs\rangle, to\_do)$
6: **if** there is a variable X such that $dom_0[X] = \{\}$ **then**
7: **return** *false*
8: **else if** for every variable X, $|dom_0[X]| = 1$ **then**
9: **return** solution with each variable X having the value in $dom_0[X]$
10: **else**
11: select variable X such that $|dom_0[X]| > 1$
12: partition $dom_0[X]$ into D_1 and D_2
13: $dom_1 :=$ a copy of dom_0 but with $dom_1[X] = D_1$
14: $dom_2 :=$ a copy of dom_0 but with $dom_2[X] = D_2$
15: $to\_do := \{\langle Z, c'\rangle \mid \{X, Z\} \subseteq scope(c'), Z \neq X\}$
16: **return** $Solve2(\langle Vs, dom_1, Cs\rangle, to\_do)$ **or** $Solve2(\langle Vs, dom_2, Cs\rangle, to\_do)$

Variable Elimination

Idea: eliminate the variables one-by-one passing their constraints to their neighbours.

- Consider a CSP that contains the variables *A*, *B*, and *C*, each with domain {1, 2, 3, 4}.
- Suppose the constraints that involve *B* are $A < B$ and $B < C$.
- There may be many other variables, but if *B* does not have any constraints in common with these variables, eliminating *B* will not impose any new constraints on these other variables.
- To remove *B*, first join on the relations that involve *B.*
- To get the relation on *A* and *C* induced by *B*, project this join onto *A* and *C*.

A	B
1	2
1	3
1	4
2	3
2	4
3	4

$\bowtie$

B	C
1	2
1	3
1	4
2	3
2	4
3	4

$=$

A	B	C
1	2	3
1	2	4
1	3	4
2	3	4

A	C
1	3
1	4
2	4

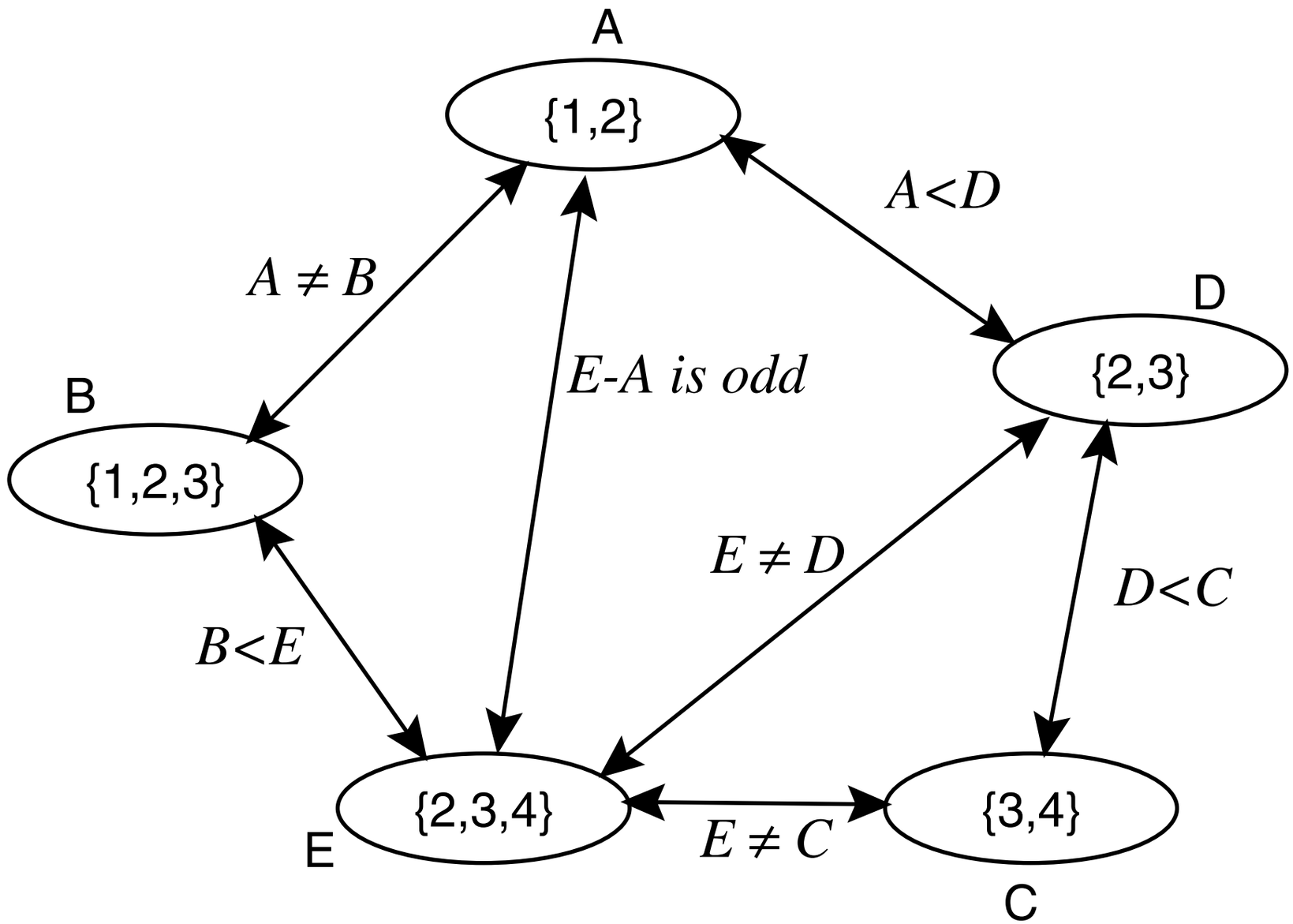
Variable Elimination Algorithm (the outline)

- If there is only one variable, return the intersection of the (unary) constraints that contain it
- Select a variable X
- Join the constraints in which X appears, forming constraint R_1
- Project R_1 onto its variables other than X, forming R_2
- Replace all of the constraints in which X_i appears by R_2
- Recursively solve the simplified problem, forming R_3
- Return R_1 joined with R_3

Variable Elimination Algorithm

1: **procedure** $VE\_CSP(Vs, Cs)$
2: **Inputs**
3: Vs: a set of variables
4: Cs: a set of constraints on Vs
5: **Output**
6: a relation containing all of the consistent variable assignments
7: **if** Vs contains just one element **then**
8: return the join of all the relations in Cs
9: **else**
10: **select** variable Xs to eliminate
11: $Vs' := Vs \setminus \{X\}$
12: $Cs_X := \{T \in Cs : T \text{ involves } X\}$
13: **let** R be the join of all of the constraints in Cs_X
14: **let** R' be R projected onto the variables other than X
15: $S := VE\_CSP(Vs', (Cs \setminus Cs_X) \cup \{R'\})$
16: **return** $R \bowtie S$

Variable Elimination Example



Variable Elimination Example (cont.)

$r_1 : C \neq E$	C	E
	3	2
	3	4
	4	2
	4	3

$r_2 : C > D$	C	D
	3	2
	4	2
	4	3

$r_3 : r_1 \bowtie r2$	C	D	E
	3	2	2
	3	2	4
	4	2	2
	4	2	3
	4	3	2
	4	3	3

$r_4 : \pi_{\{D,E\}} r_3$	D	E
	2	2
	2	3
	2	4
	3	2
	3	3

$\hookrightarrow$ new constraint

Variable Elimination Example (cont.)