COMPSCI 761:
ADVANCED TOPICS IN ARTIFICIAL INTELLIGENCE

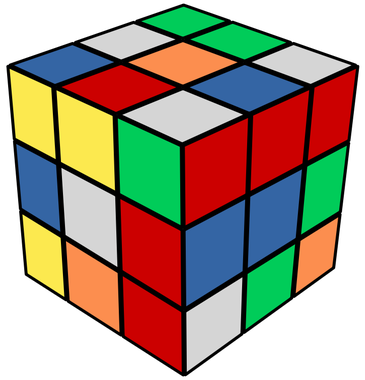
CONSTRAINT SATISFACTION PROBLEM I

Anna Trofimova, August 2022

TODAY

- CSP Definition recap
- CSP Backtracking
- CSP Inference

RECAP: SOME SEARCH PROBLEMS TYPES



Models with atomic representation have no internal structure; the state either does or does not match the goal state.

In Rubik's cube, the tiles alignment is either correct or not.

Models with factored representation have a set of variables, each of which has a value.

Every state on the map is a variable that has a colour value.

RECAP: CONSTRAINT SATISFACTION PROBLEMS (CSP)

- Constraint Satisfaction Problems are defined by a set of variables X_i, each with a domain D_i of possible values, and a set of constraints C that specify allowable combinations of values.

- The aim is to find an assignment of the variables X_i from the domains D_i in such a way that none of the constraints C are violated.
  - i.e. all of the constraints C are satisfied

EXAMPLE: MAP-COLOURING

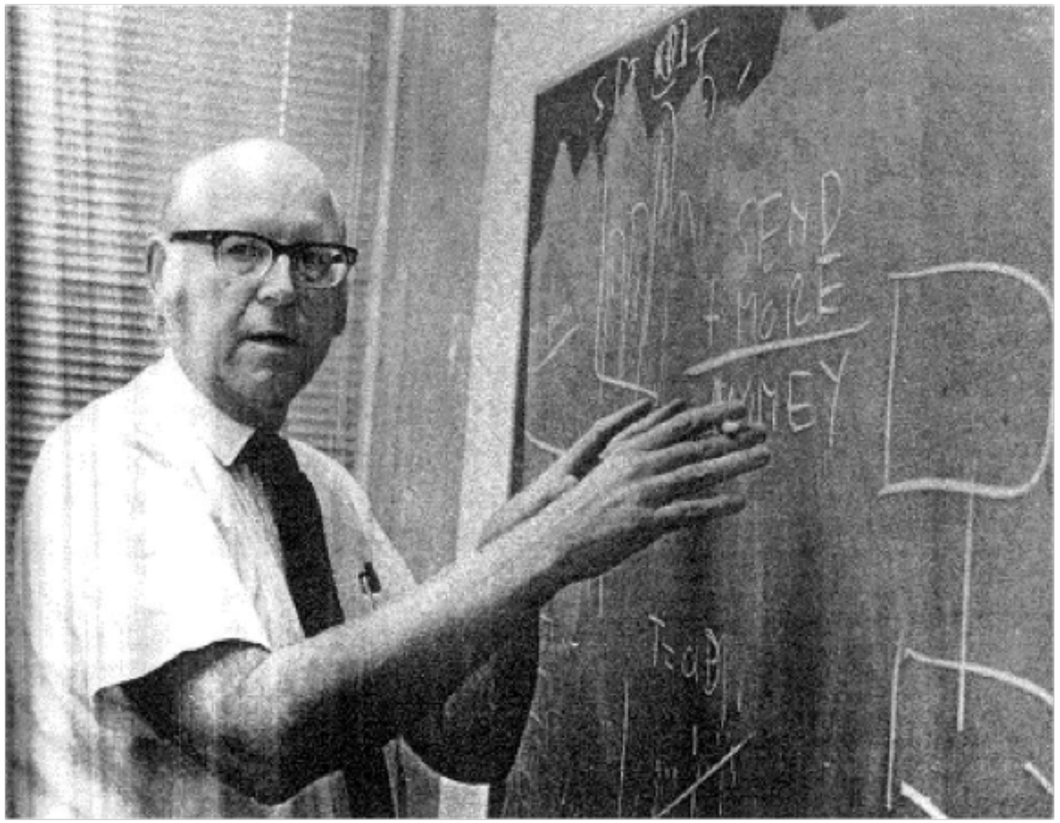
Variables: WA, NT, Q, NSW, V, SA, T

Domains: Di = {red, green, blue}

Constraints: adjacent regions must have different colours
e.g. WA≠ NT, etc.

or (WA,NT) in {(red, green), (red, blue),(green, red), (green, blue), (blue, red),(blue, green)}

RECAP: CRYPTARITHMETIC WITH ALLEN NEWELL



Book: Intended Rational Behavior

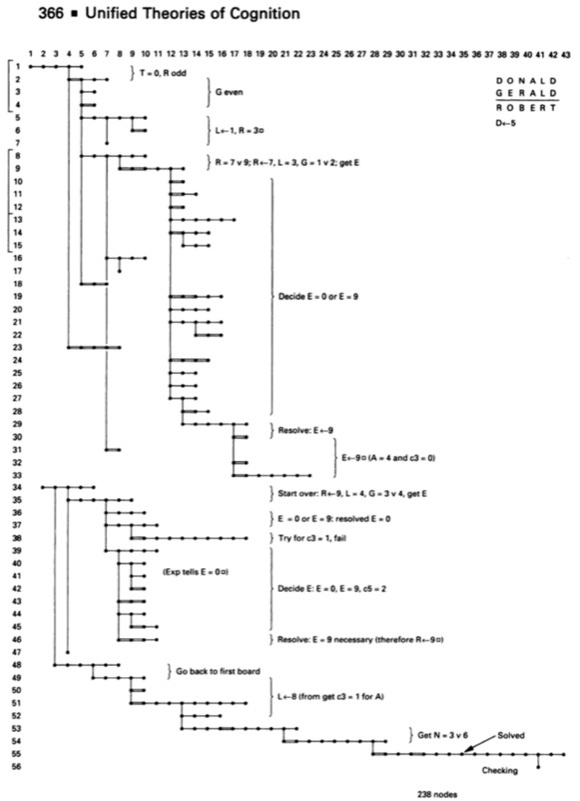


Figure 7-2. Problem-behavior graph of subject S3 on *DONALD + GERALD = ROBERT.*

CRYPTARITHMETIC WITH HIDDEN VARIABLES

- We can add “hidden” variables to simplify the constraints.

```
    T W O
+
    T W O
-----------
  F O U R
```

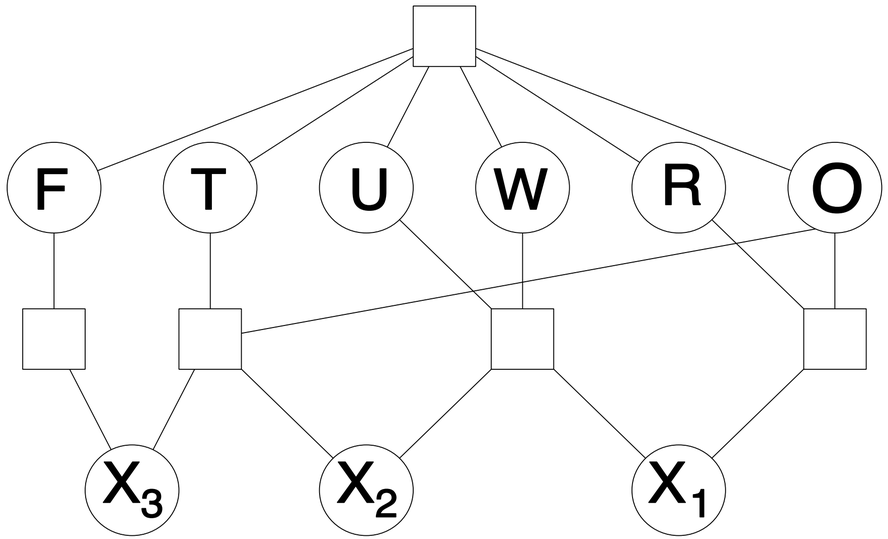
Variables: ?
Domains: ?

Constraints: ?

CRYPTARITHMETIC WITH HIDDEN VARIABLES

- We can add “hidden” variables to simplify the constraints.

```
    T W O
+
    T W O
-----------
  F O U R
```



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:
AllDifferent(F,T,U,W,R,O)
$O + O = R + 10 \cdot X_1$, etc.

REAL-WORLD CSPS

- Assignment problems (e.g. who teaches what class)
- Timetabling problems (e.g. which class is offered when and where?)
- Hardware configuration (e.g. minimise space for circuit layout)
- Transport scheduling (e.g. courier delivery, vehicle routing)
- Factory scheduling (optimise assignment of jobs to machines)
- Gate assignment (assign gates to aircraft to minimise transit)

Many real world CSPs are also optimisation problems

REAL-WORLD CSPS - FACTORY SCHEDULING

- A robot (agent) needs to schedule a set of activities for a manufacturing process, involving casting, milling, drilling, and bolting.
- Each activity has a set of possible times at which it may start.
- The robot has to satisfy various constraints arising from prerequisite requirements and resource use limitations.
- For each activity there is a variable that represents the time that it starts:
  - A – start of casting
  - B – start of bolding
  - D – start of drilling
  - C – start of casting

REAL-WORLD CSPS - FACTORY SCHEDULING

- A robot (agent) needs to schedule a set of activities for a manufacturing process, involving casting, milling, drilling, and bolting.
- Each activity has a set of possible times at which it may start.
- The robot has to satisfy various constraints arising from prerequisite requirements and resource use limitations.
- For each activity there is a variable that represents the time that it starts:
  - A – start of casting
  - B – start of bolding
  - D – start of drilling
  - C – start of casting
- Constraints: $D < B$, $C \neq D$, $B = C+3$

REAL-WORLD CSPS - FACTORY SCHEDULING

Consider a constraint on the possible dates for three activities.

$(A \leq B) \wedge (B < 3) \wedge (B < C) \wedge \neg (A = B \wedge C \leq 3)$

Variables: A, B, C - variables that represent the date of each activity
Domain of each variable is: {1, 2, 3, …}
A **constraint** with scope:
This formula says that A is on the same date or before B
and it cannot be that A and B are on the same date and C is on or before day 3.

REAL-WORLD CSPS - FACTORY SCHEDULING

This constraint could instead have its relation defined its extension, as a table specifying the legal assignments:

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

TYPES OF CONSTRAINTS

- Unary constraints involve a single variable
  - M ≠ 0
- Binary constraints involve pairs of variables
  - SA ≠ WA
- Higher-order constraints involve 3 or more variables
  - $Y = D + E$ or $Y = D + E - 10$
- Inequality constraints on Continuous variables
  - EndJob1 + 5 ≤ StartJob3
- Soft constraints (Preferences)
  - 11am lecture is better than 8am lecture!

PATH SEARCH VS CONSTRAINT SATISFACTION

Important difference between path search problems and CSPs

- Path Search Problems (e.g. Rubik's Cube)
  - Knowing the final state is easy
  - Difficult part is how to get there

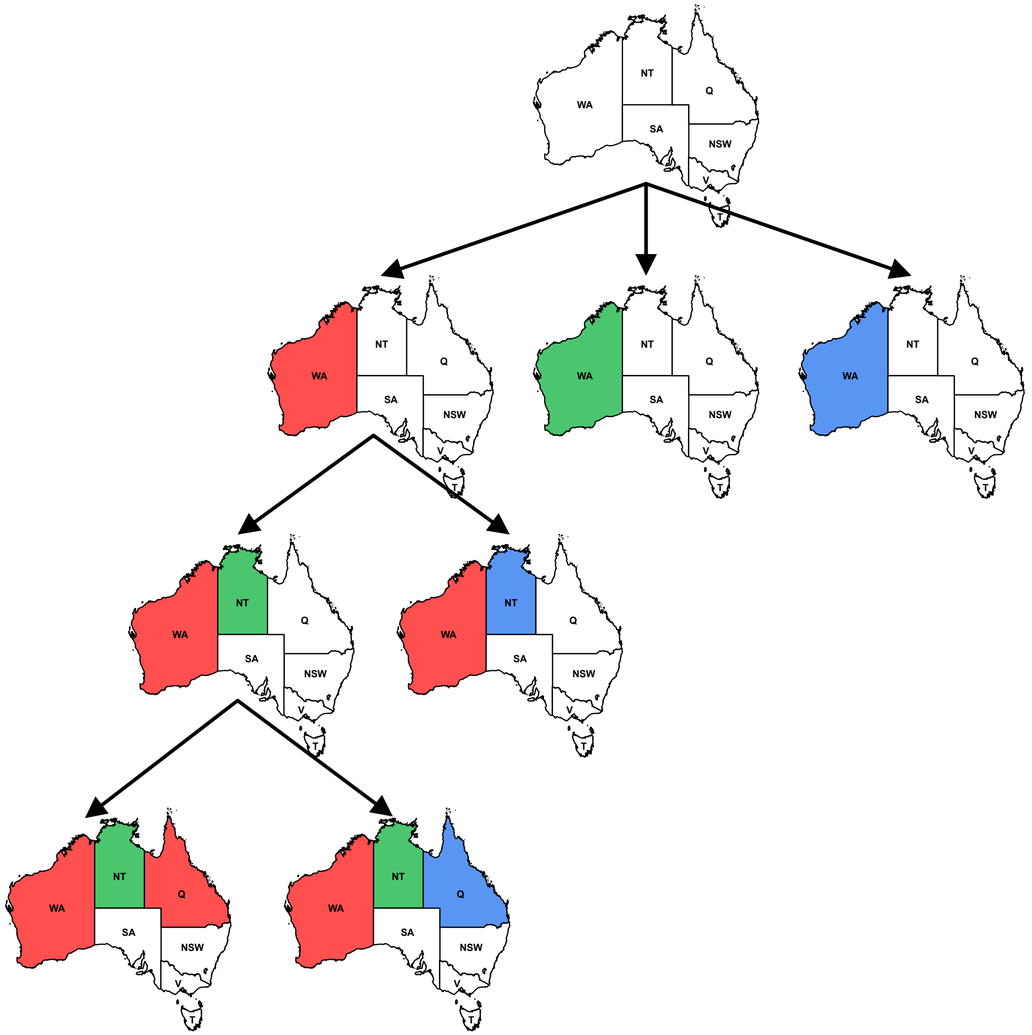
- Constraint Satisfaction Problems (e.g. n-Queens)
  - Difficult part is knowing the final state
  - How to get there is easy

BACKTRACKING SEARCH

CSPs can be solved by assigning values to variables one by one, in different combinations. Whenever a constraint is violated, we go back to the most recently assigned variable and assign it a new value.
This can be achieved by a Depth First Search on a special kind of state space, where states are defined by the values assigned so far:

- Initial state: the empty assignment.
- Successor function: assign a value to an unassigned variable that does not conflict with previously assigned values of other variables. (If no legal values remain, the successor function fails.)
- Goal test: all variables have been assigned a value, and no constraints have been violated.

BACKTRACKING EXAMPLE



BACKTRACKING SEARCH SPACE PROPERTIES

The search space for this Depth First Search has certain very specific properties:

- If there are n variables, every solution will occur at exactly depth n.
- Variable assignments are commutative

  [WA = red then NT = green] same as [NT = green then WA = red]

IMPROVEMENTS TO BACKTRACKING SEARCH

General-purpose methods can give huge gains in speed:

1. Which variable should be assigned next?

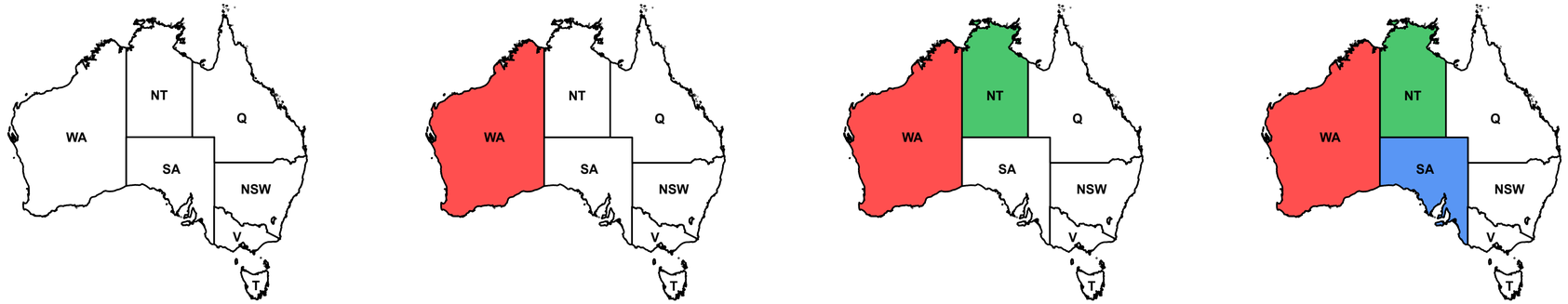
2. In what order should its values be tried?

3. Can we detect inevitable failure early?

Improving backtracking efficiency

MINIMUM REMAINING VALUES

- Minimum Remaining Values (MRV)
  - choose the variable with the fewest legal values



NT or SA should go first?

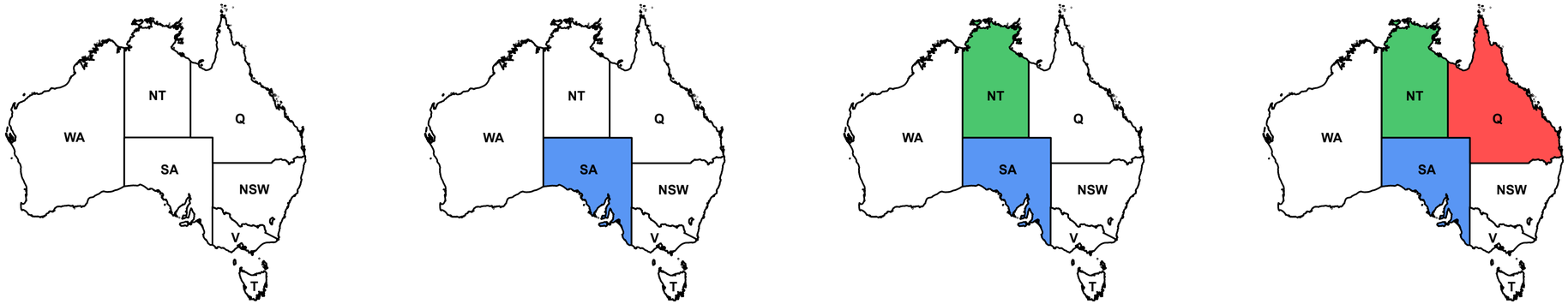
DEGREE HEURISTIC

- Tie-breaker among MRV variables
- Degree heuristic:
  - choose the variable with the most constraints on remaining variables

Which state is the most constrained?

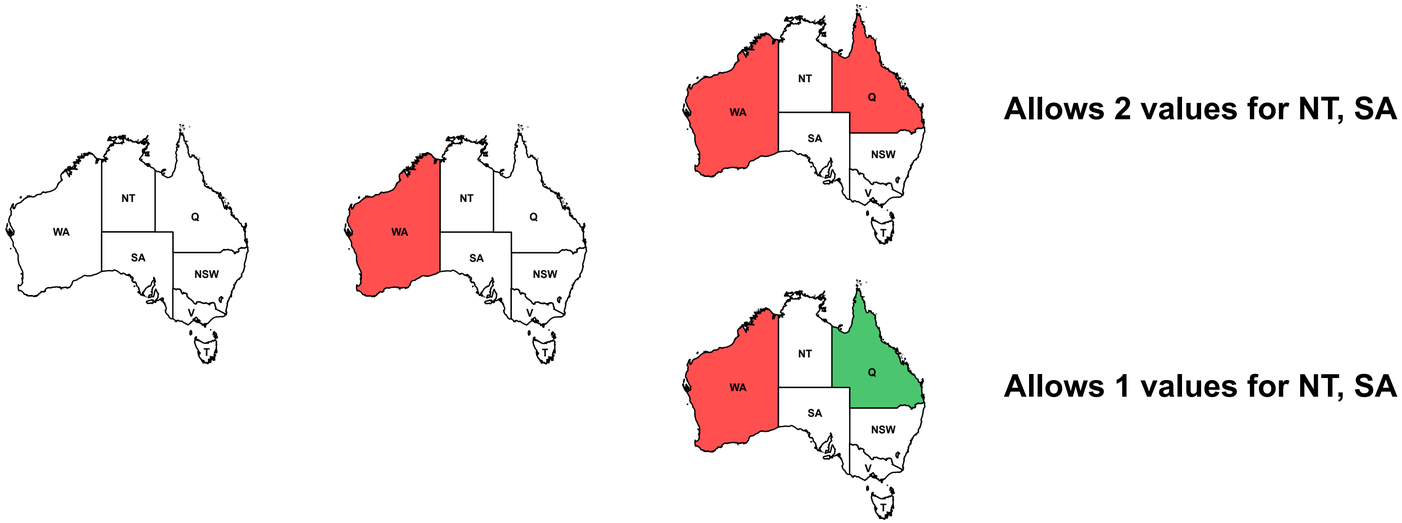
DEGREE HEURISTIC

- Tie-breaker among MRV variables
- Degree heuristic:
  - choose the variable with the most constraints on remaining variables



LEAST CONSTRAINING VALUE

- Given a variable, choose the least constraining value:
  the one that rules out the fewest values in the remaining variables

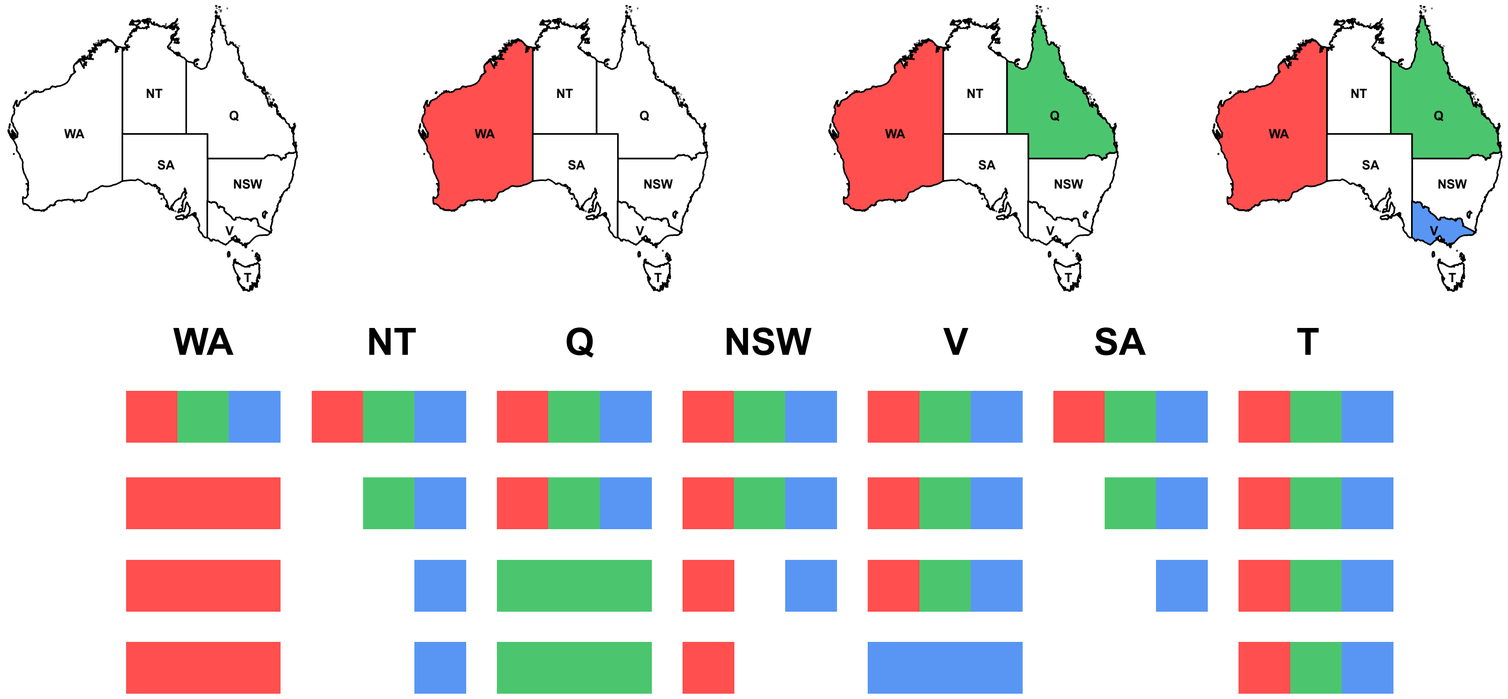


- (More generally, 3 allowed values would be better than 2, etc.)
  - Combining these heuristics makes multi-variable CSP with big domains feasible

FORWARD CHECKING

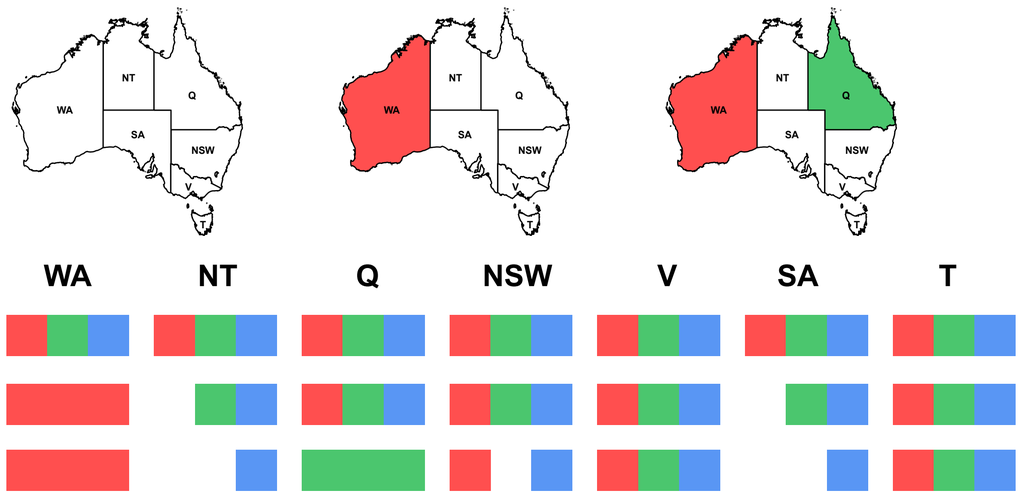
Idea:

- Keep track of remaining legal values for unassigned variables
- Terminate search when any variable has no legal values
  - prune off that part of the search tree, and backtrack



CONSTRAINT PROPAGATION

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:



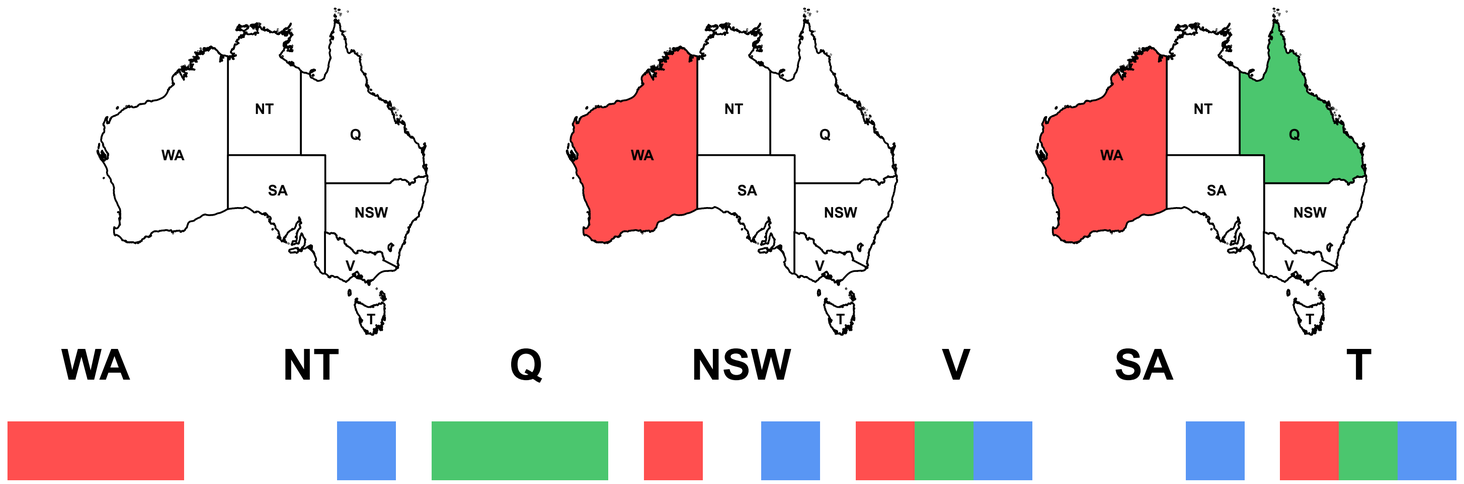
NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

ARC CONSISTENCY

Simplest form of propagation makes each arc consistent

X → *Y* is consistent if *for* ***every*** *value x of X there is* ***some*** *allowed y*

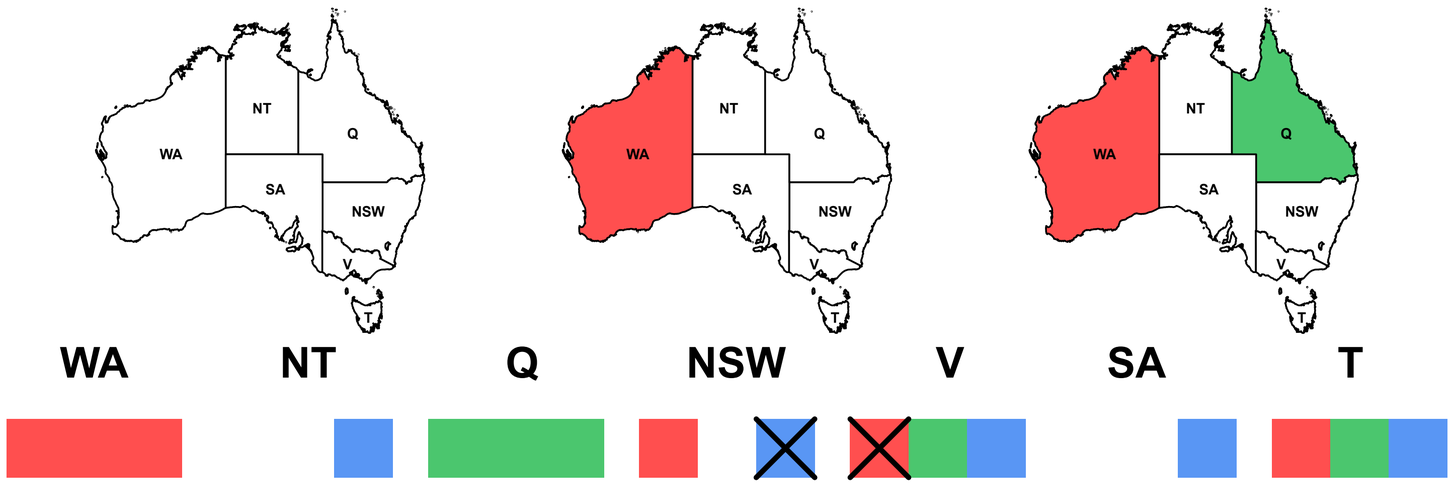


If *X* looses a value, neighbours of *X* need to be rechecked.

ARC CONSISTENCY

Simplest form of propagation makes each arc consistent

X → *Y* is consistent if *for* ***every*** *value x of X there is* ***some*** *allowed y*



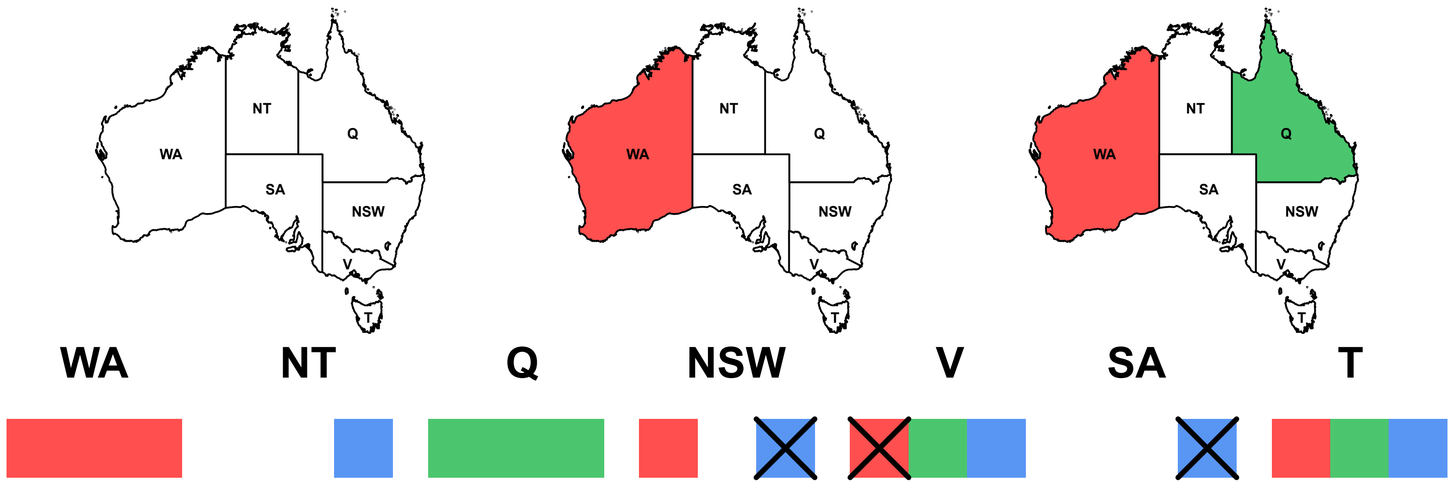
Arc consistency detects failure earlier than forward checking

Can be run as a preprocessor after each assignment

ARC CONSISTENCY

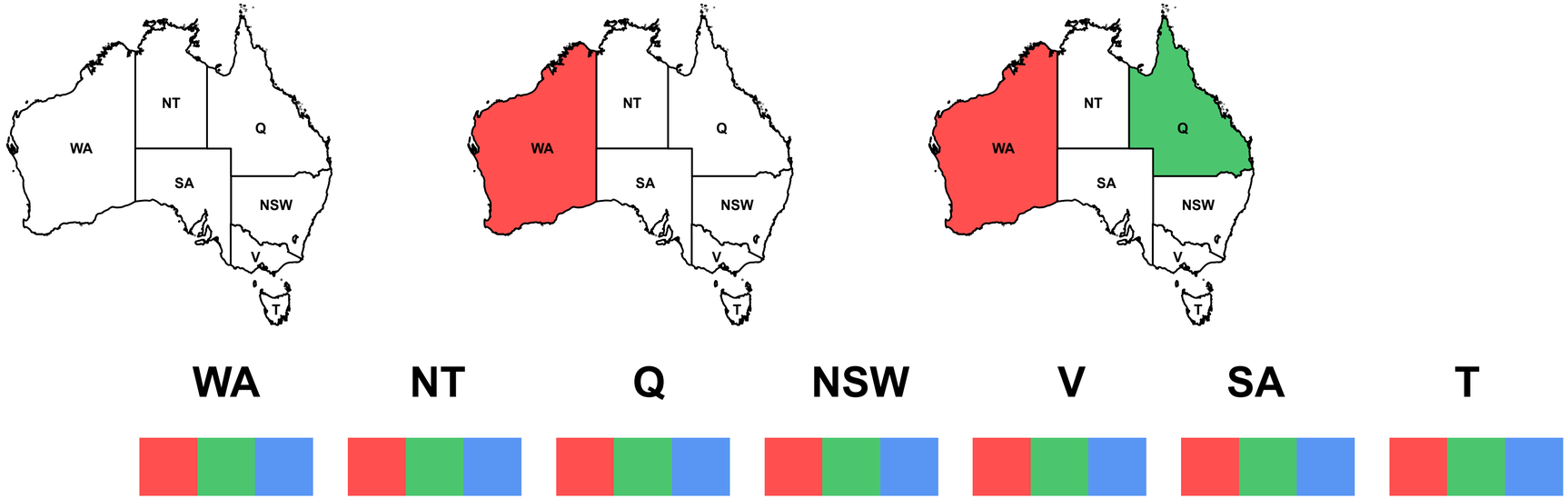
Simplest form of propagation makes each arc consistent

X → *Y* is consistent if *for* ***every*** *value x of X there is* ***some*** *allowed y*



For some problems, it can speed up the search enormously.
For others, it may slow the search due to computational overheads.

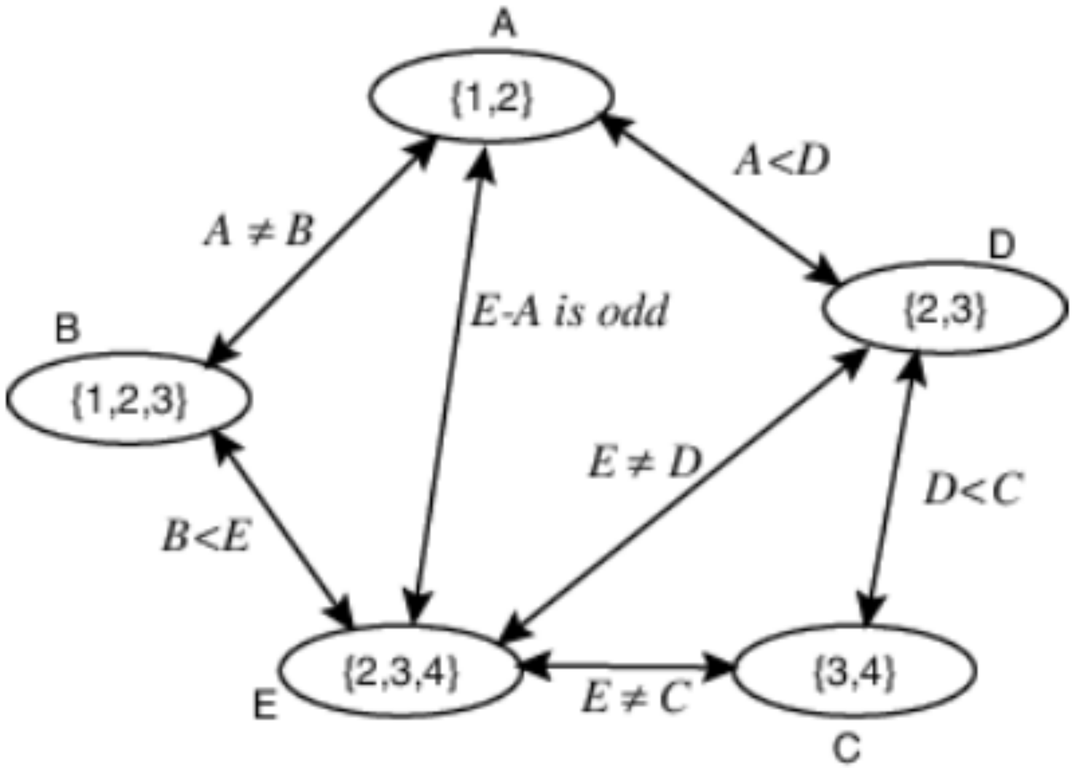
ARC CONSISTENCY EXAMPLE



VARIABLE ELIMINATION

- If there is only one variable, return the intersection of its (unary) constraints
- Otherwise
  - Select a variable X
  - Join the constraints in which X appears, forming constraint R1
  - Project R1 onto its variables other than X, forming R2
  - Replace all of the constraints in which X appears by R2
  - Recursively solve the simplified problem, forming R3
  - Return R1 joined with R3

VARIABLE ELIMINATION

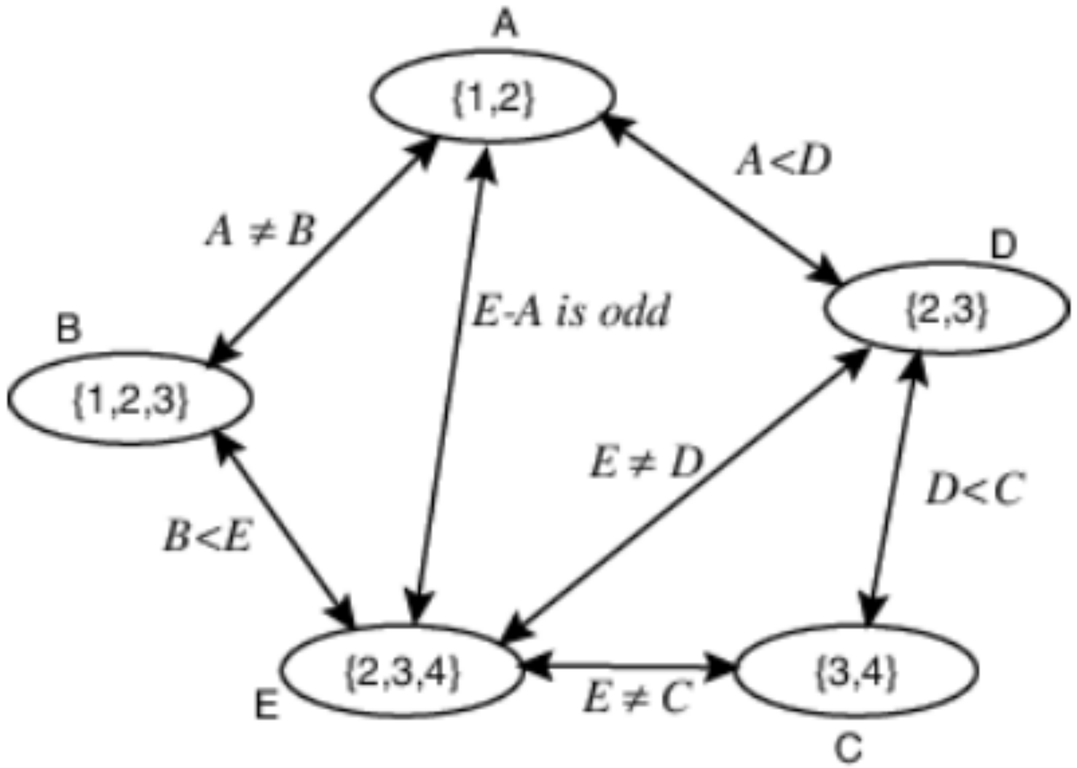


VARIABLE ELIMINATION

Constraints:
A ≠ B, E ≠ C, E ≠ D, A < D, B < E,
D < C, E-A is odd

Variables: A, B, C, D, E

Domains: A = {1,2}, B = {1,2,3},
C = {3,4}, D = {2,3}, E ={2,3,4}



VARIABLE ELIMINATION EXAMPLE

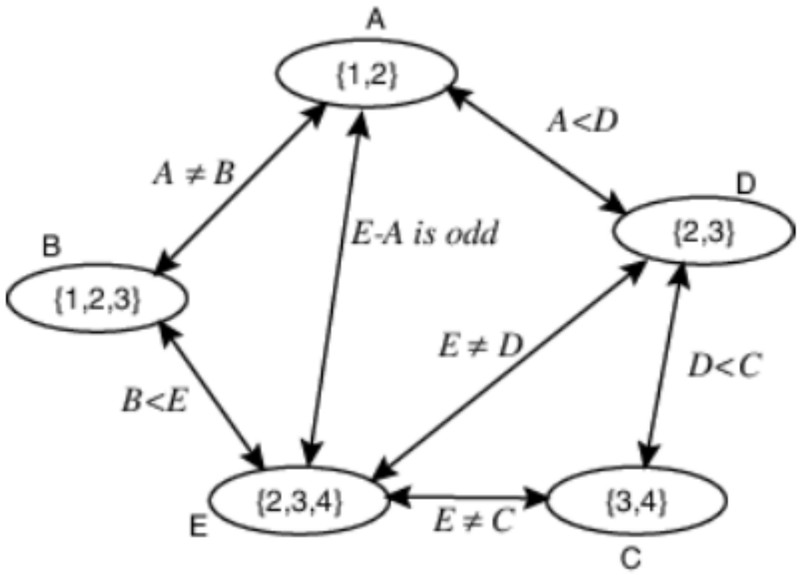
$r_1 : C \neq E$	C	E
	3	2
	3	4
	4	2
	4	3

$r_2 : C > D$	C
	3
	4
	4

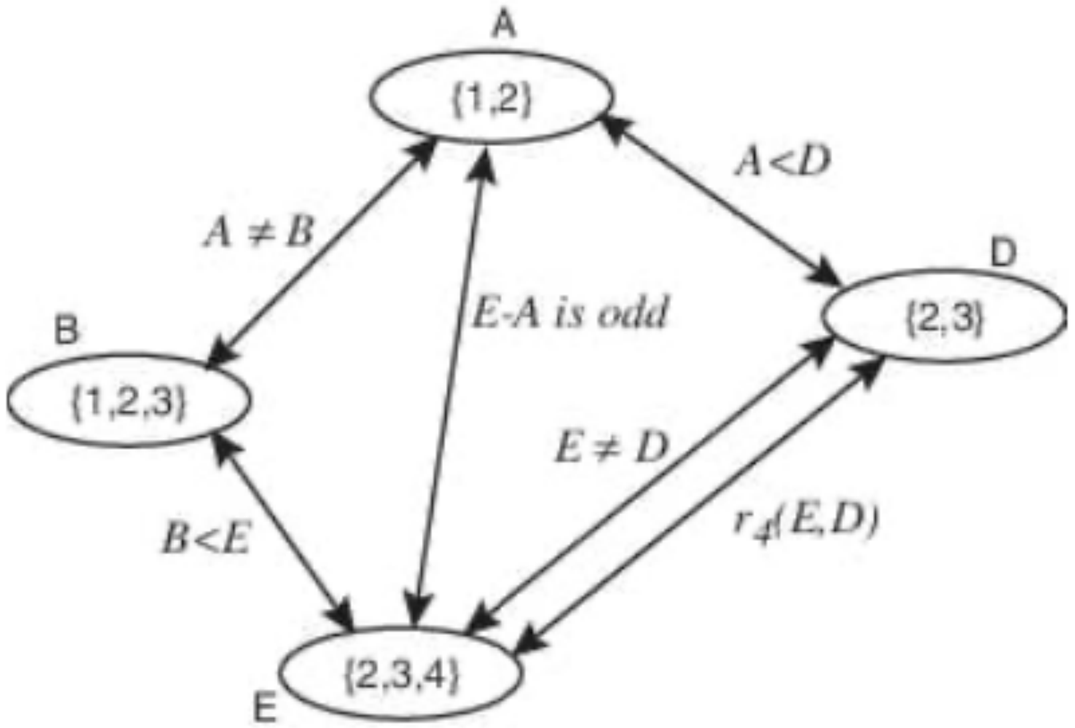
$r_3 : r_1 \bowtie r2$	C	D	E
	3	2	2
	3	2	4
	4	2	2
	4	2	3
	4	3	2
	4	3	3

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

↪ new constraint



VARIABLE ELIMINATION EXAMPLE



SUMMARY

- CSPs are a special kind of problem:
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly