CSPB3202 Artificial Intelligence

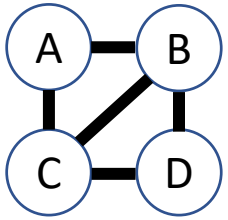
Search

University of Colorado **Boulder**

Note: Some slides were adapted from the materials from Prof. Pieter Abbeel and Prof. Dan Klein at UC Berkeley, and Russell & Norvig's AI book.

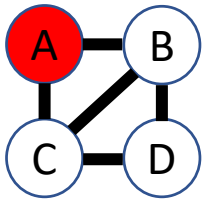
Constraint Satisfaction Problems

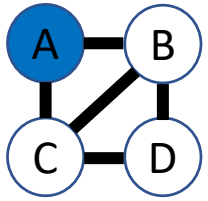
Solving CSP- Naïve Search: BFS

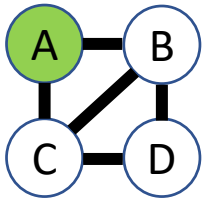


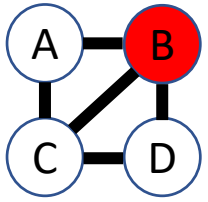
X = {A,B,C,D}

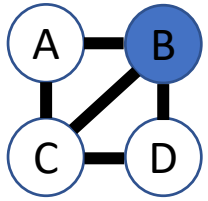
D = {red, blue, green}



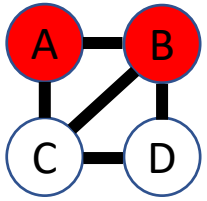


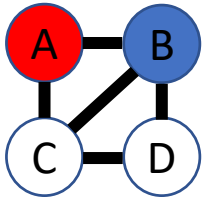


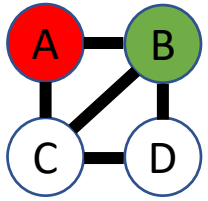




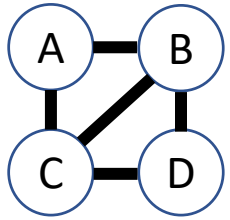
.....





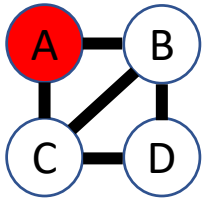


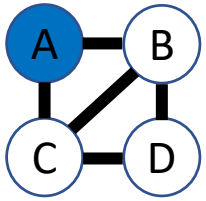
Solving CSP- Naïve Search: DFS

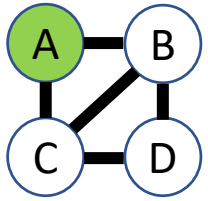


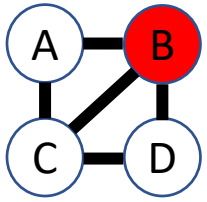
X = {A,B,C,D}

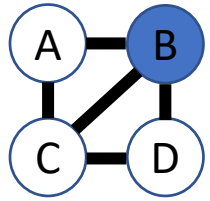
D = {red, blue, green}



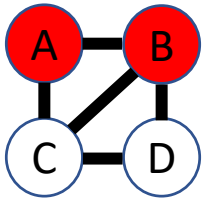


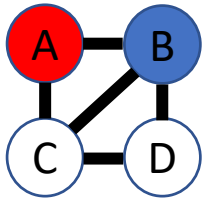


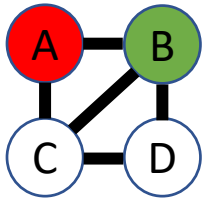


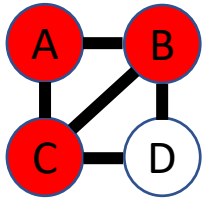


.....









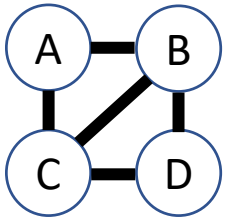
.....

Backtracking

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

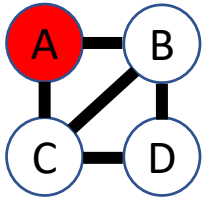
Backtracking

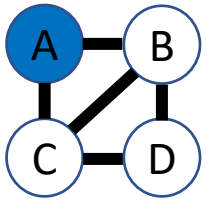
Idea: Check the constraints as we go

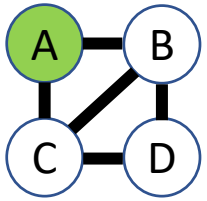


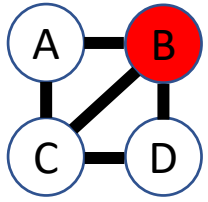
X = {A,B,C,D}

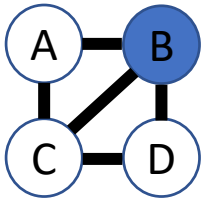
D = {red, blue, green}



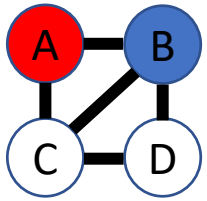


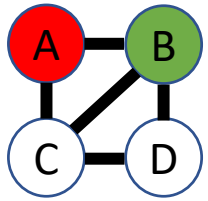


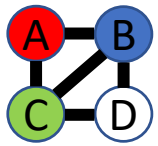


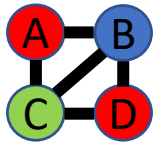


.....





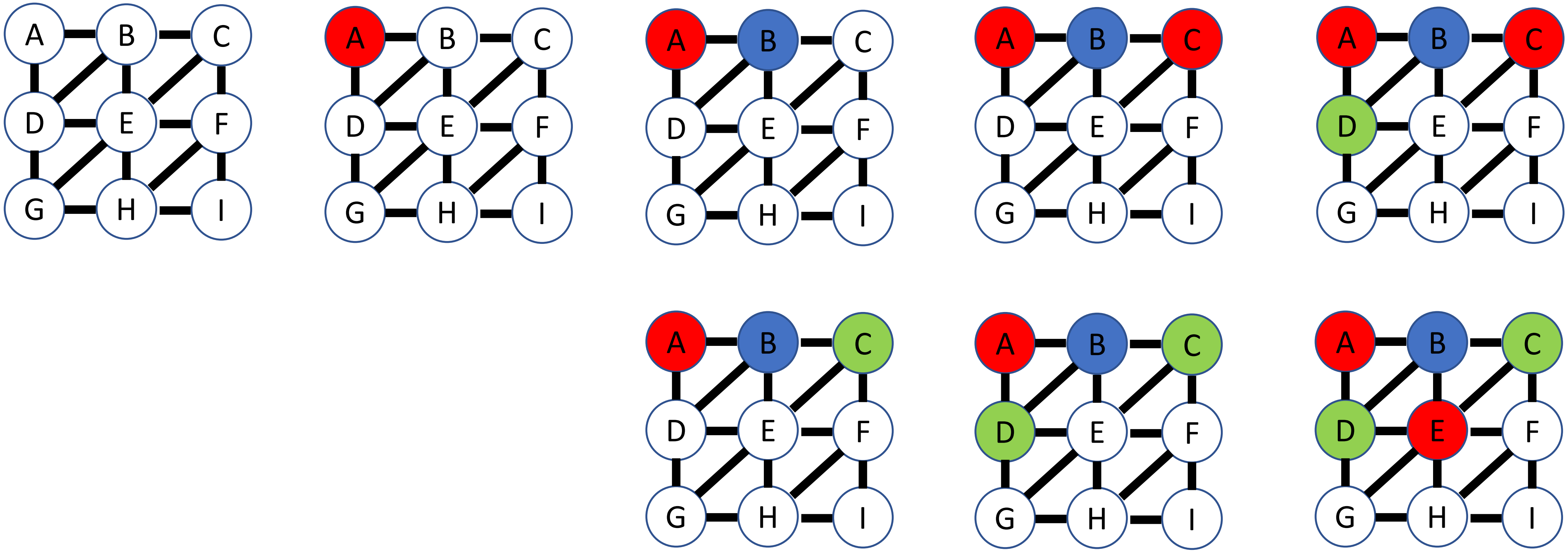




Backtracking

X = {A,B,C,D,...}

D = {red, blue, green}



Improving backtracking

- Filtering : Check if there is any violation in the future
- Ordering : Pick next variable wisely
- Structure : Can we take advantage of certain structure of the problem?

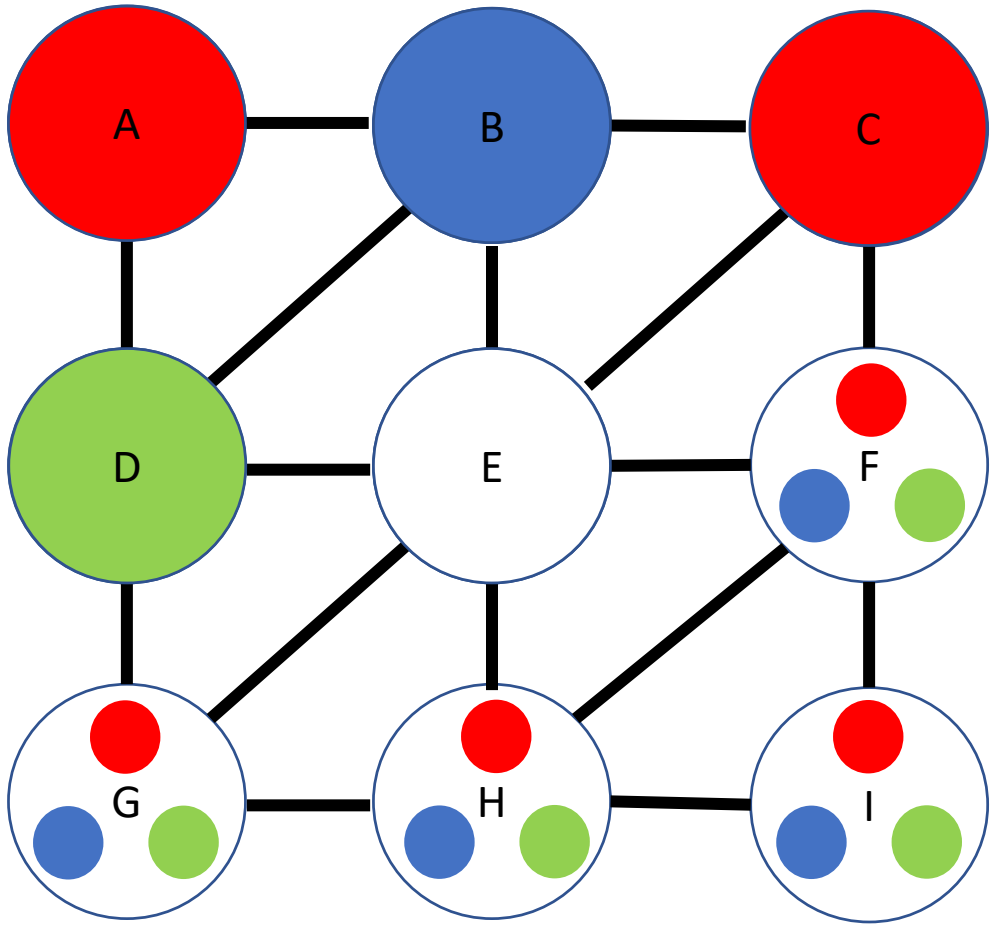
Filtering

Idea : Check if there is any violation in the future (Domain reduction)

- Forward Checking: Inferencing domain reduction on neighboring variables
- Arc Consistency: Propagate the inference/constraints to the entire graph

Forward checking

Idea: cross off non-valid options in the {D} for X_i



X = {A,B,C,D,…}
D = {red, blue, green}

Forward checking can still have a problem!

Arc Consistency- AC3

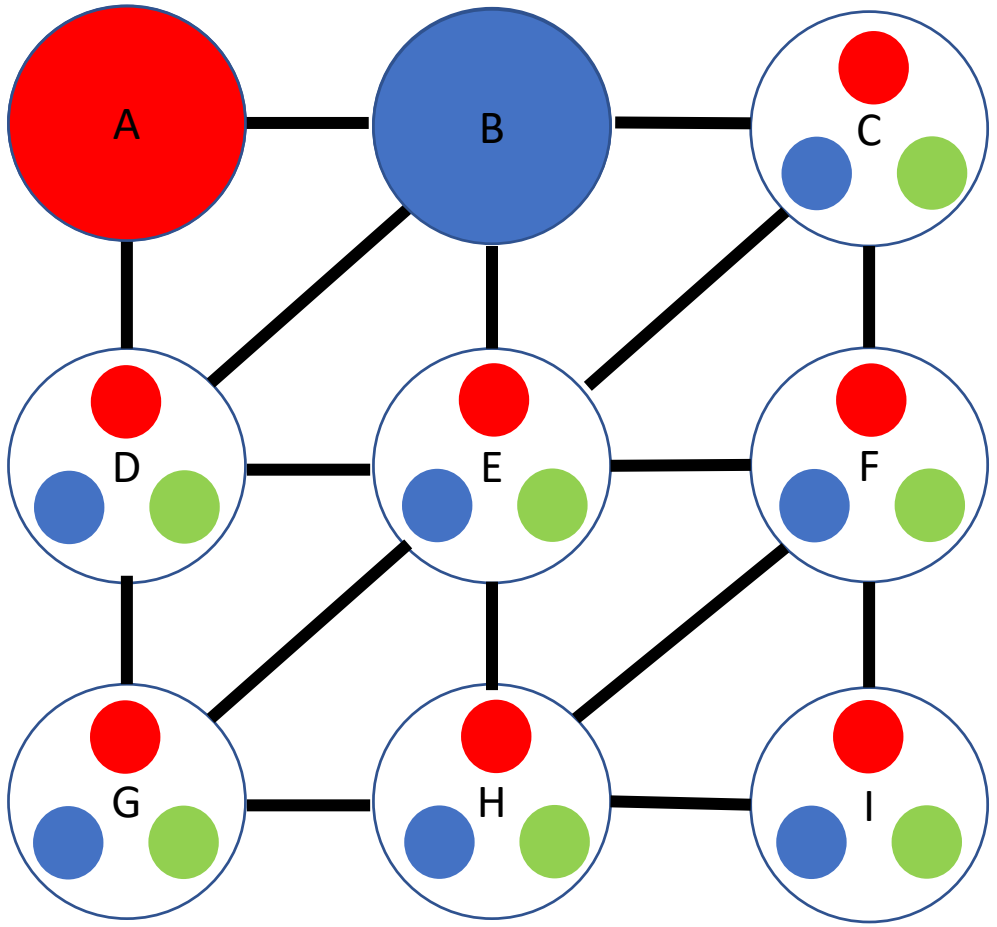
```
function AC-3( csp) returns the CSP, possibly with reduced domains
    inputs: csp, a binary CSP with variables {X_1, X_2, ..., X_n}
    local variables: queue, a queue of arcs, initially all the arcs in csp

    while queue is not empty do
        (X_i, X_j) ← REMOVE-FIRST(queue)
        if REMOVE-INCONSISTENT-VALUES(X_i, X_j) then
            for each X_k in NEIGHBORS[X_i] do
                add (X_k, X_i) to queue
```

```
function REMOVE-INCONSISTENT-VALUES( X_i, X_j) returns true iff succeeds
    removed ← false
    for each x in DOMAIN[X_i] do
        if no value y in DOMAIN[X_j] allows (x,y) to satisfy the constraint X_i ↔ X_j
            then delete x from DOMAIN[X_i]; removed ← true
    return removed
```

Arc Consistency- AC3

Idea: cross off non-valid options in the {D} for all *X*

X = {A,B,C,D,…}
D = {red, blue, green}



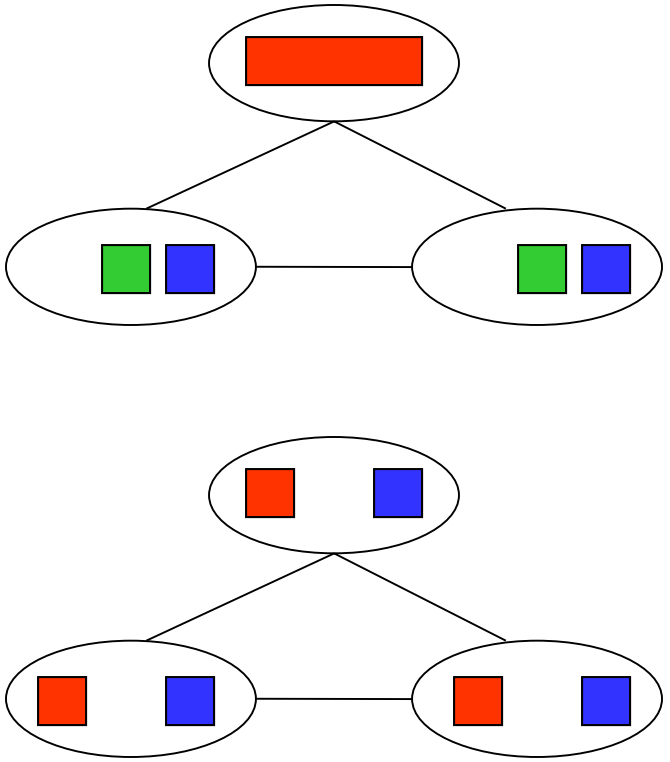
RULE 1: If a domain is removed/assigned, check all neighbors and neighbors or neighbors recursively!

RULE 2: Arc is consistent if there is at least a valid pair of assignments.

Limitations of Arc Consistency

- After enforcing arc consistency:
  - Can have one solution left
  - Can have multiple solutions left
  - Can have no solutions left (and not know it)

- Arc consistency still may need backtracking



What went wrong here?