Constraint Satisfaction Problems – Part 2

Deepak Kumar
October 2021

1

CSP Formulation (as a special case of search)

- State is defined by ***n*** variables

$$\{x_1, x_2, \dots, x_n\}$$

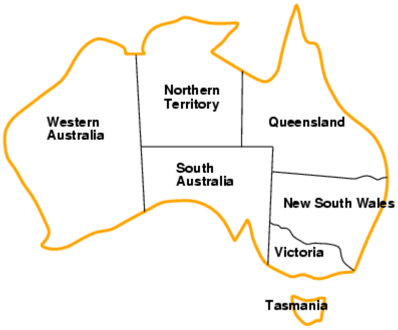
- Variables can take on values from a domain set (One for each variable)

$$\{D_1, D_2, \dots, D_n\}$$

- Goal test is a set of constraints specifying allowable combinations of values of variables (subsets)
- This allows general purpose algorithms without resorting to domain specific heuristics.

2

Example: Map-Coloring



- **Variables:** *WA, NT, Q, NSW, V, SA, T*
- **Domains:** $D_i = \{red, green, blue\}$
- **Constraints:** adjacent regions must have different colors

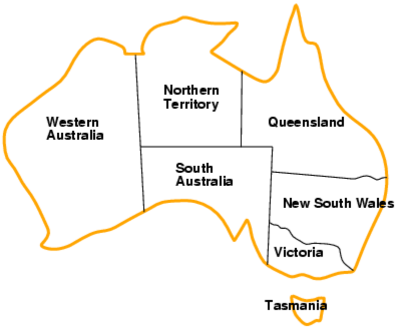
e.g., $WA \neq NT$

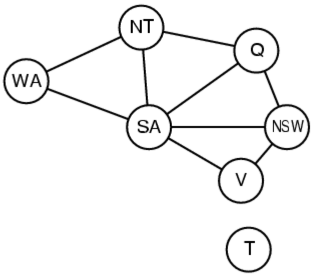
or
(WA, NT)
$\in \{(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)\}$

3

Constraint Graph Representation of CSP

- Binary CSP: each constraint relates two variables
- Constraint graph: nodes are variables, edges/arcs are constraints



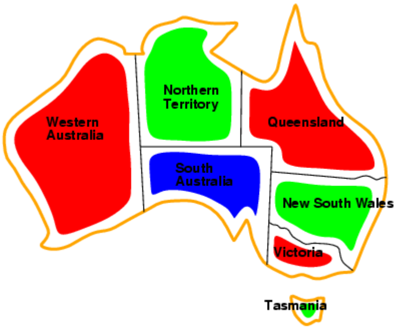


4

Example: Map-Coloring

- Solutions are complete and consistent assignments

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



5

Start with a basic search algorithm…

Initial State: Empty assignment { }

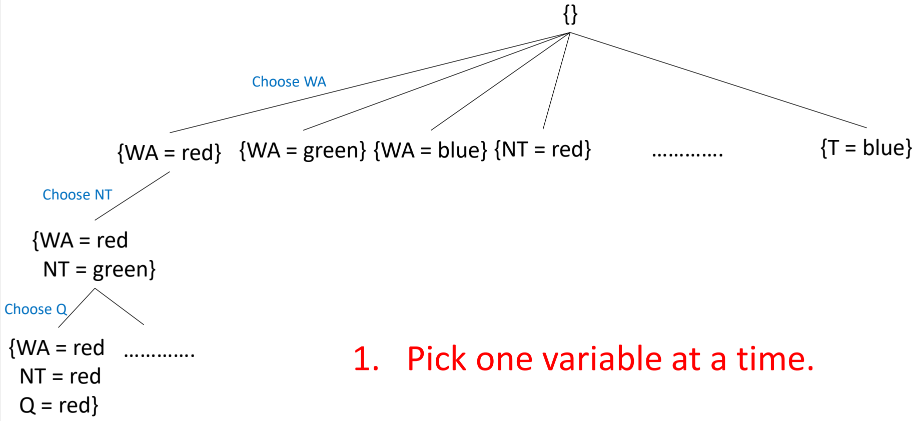
Successor Function: assign a value to an unassigned variable

Goal Test: current assignment complete & consistent?

6

Backtracking Search

Western Australia
Northern Territory
Queensland
South Australia
New South Wales
Victoria
Tasmania



d (7*3=21)

d (6*3=18)

d^n (6! * 3^7)

1. Pick one variable at a time.

2. Check constraints as you go.
(incremental goal testing)

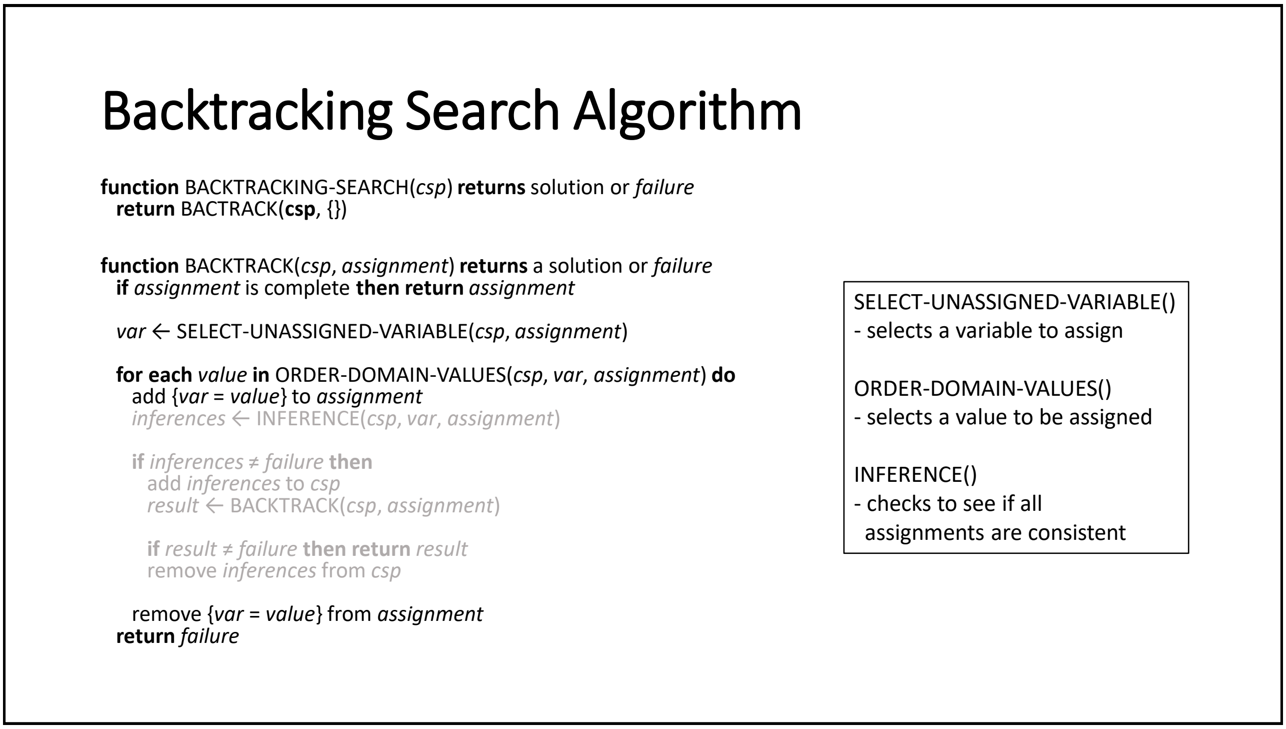
Backtracking Search Algorithm

```
function BACKTRACKING-SEARCH(csp) returns solution or failure
  return BACTRACK(csp, {})

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment

  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)

  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    add {var = value} to assignment
    inferences ← INFERENCE(csp, var, assignment)

    if inferences ≠ failure then
      add inferences to csp
      result ← BACKTRACK(csp, assignment)

      if result ≠ failure then return result
      remove inferences from csp

    remove {var = value} from assignment
  return failure
```

SELECT-UNASSIGNED-VARIABLE()
- selects a variable to assign

ORDER-DOMAIN-VALUES()
- selects a value to be assigned

INFERENCE()
- checks to see if all
  assignments are consistent



Backtracking Search Algorithm

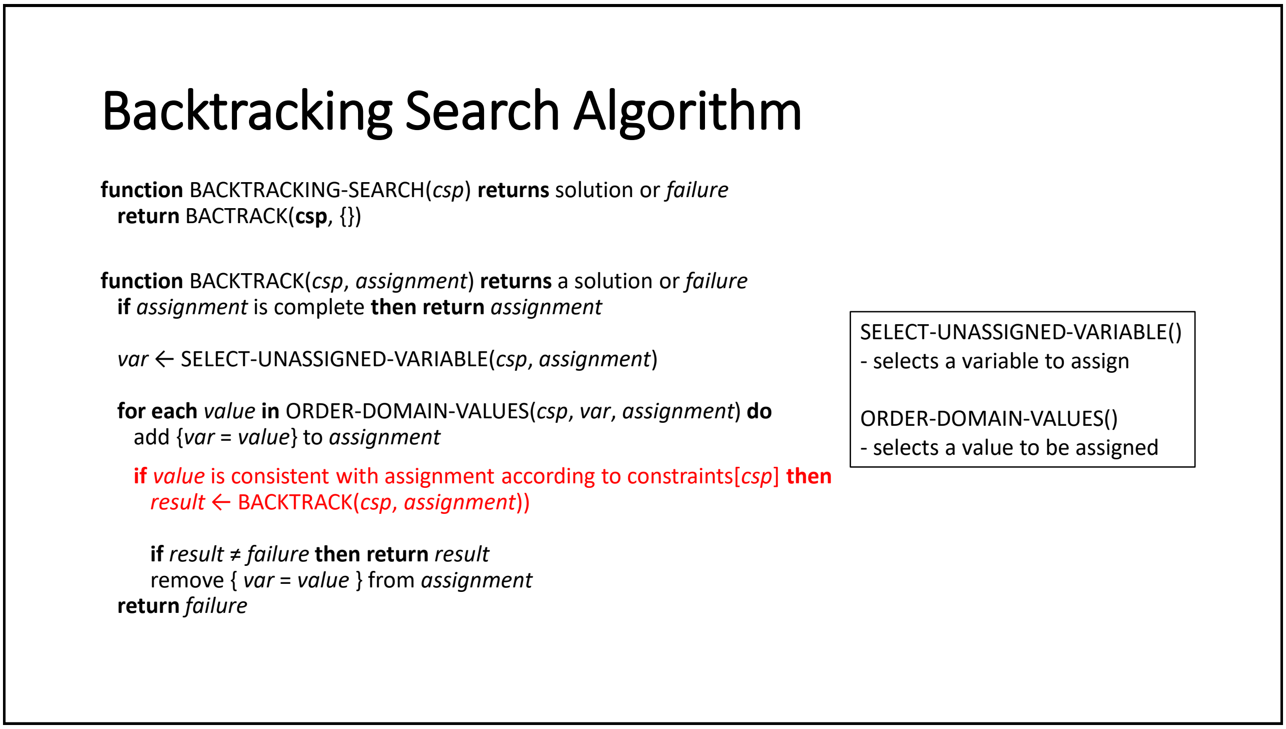
```
function BACKTRACKING-SEARCH(csp) returns solution or failure
  return BACTRACK(csp, {})

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment

  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)

  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    add {var = value} to assignment
    inferences ← INFERENCE(csp, var, assignment)

    if inferences ≠ failure then
      add inferences to csp
      result ← BACKTRACK(csp, assignment)

      if result ≠ failure then return result
      remove inferences from csp

    remove {var = value} from assignment
  return failure
```

SELECT-UNASSIGNED-VARIABLE()
- selects a variable to assign

ORDER-DOMAIN-VALUES()
- selects a value to be assigned

INFERENCE()
- checks to see if all
  assignments are consistent

9



Backtracking Search Algorithm

```
function BACKTRACKING-SEARCH(csp) returns solution or failure
  return BACTRACK(csp, {})

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment

  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)

  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    add {var = value} to assignment
    if value is consistent with assignment according to constraints[csp] then
      result ← BACKTRACK(csp, assignment))

      if result ≠ failure then return result
      remove { var = value } from assignment
  return failure
```

SELECT-UNASSIGNED-VARIABLE()
- selects a variable to assign

ORDER-DOMAIN-VALUES()
- selects a value to be assigned

10

Backtracking Search Algorithm

```
function BACKTRACKING-SEARCH(csp) returns solution or failure
  return BACTRACK(csp, {})

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment

  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)

  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    add {var = value} to assignment

    if value is consistent with assignment according to constraints[csp] then
      result ← BACKTRACK(csp, assignment))

      if result ≠ failure then return result
      remove { var = value } from assignment
  return failure
```

Which variable to pick next?

Which value to assign next?

These are general purpose heuristics.

11

Improving Backtracking Search – Ordering Variables & Values

- Which variable to pick next?
  MRV- Most constrained variable (one with fewest remaining values)

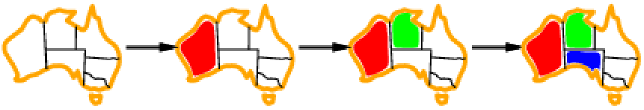
- Which value to assign next?
  Least constraining value first

- Also, we can use the *degree heuristic*
  Pick the variable with the highest degree in the constraint graph

12

Most constrained variable

- Most constrained variable:
  choose the variable with the fewest legal values

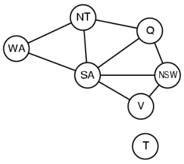


- a.k.a. minimum remaining values (MRV) heuristic

Degree Heuristic

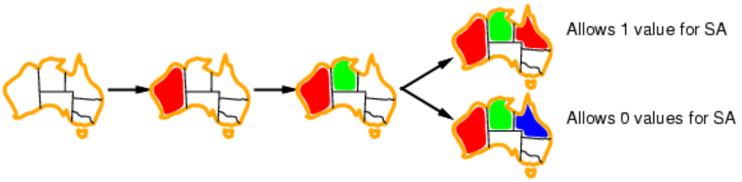
- Pick the variable with the highest degree in the constraint graph

  Useful in picking the very first variable
  (when no variables have been assigned)



Least constraining value

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



Backtracking Search Algorithm

Which variable to pick first?
Degree Heuristic

Which variable to pick next?
MRV Heuristic

Which value to assign next?
LCV Heuristic

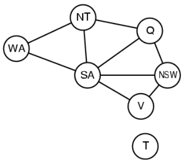
These are general purpose heuristics.

```
function BACKTRACKING-SEARCH(csp) returns solution or failure
  return BACTRACK(csp, {})

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment

  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)

  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    add {var = value} to assignment
    if value is consistent with assignment according to constraints[csp] then
      add { var = value } to assignment
      result ← BACKTRACK(csp, assignment))

      if result ≠ failure then return result
      remove { var = value } from assignment
  return failure
```

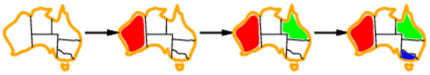
Improving Backtracking Search

- **Ordering**
  - Which variable to pick next?
    MRV- Most constrained variable (one with fewest remaining values)
  - Which value to assign next?
    Least constraining value first
- **Filtering/Inference** [Interleaving search & inference]
  - Forward Checking
  - Arc Consistency

Forward Checking (Filtering/Inference)

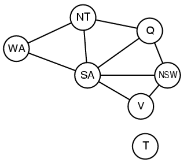


- Idea
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

Forward Checking (Filtering/Inference)



- Can also help if combined with MRV heuristic
  After WA=red, we have constrained NT & SA to (green, blue)
  All others have three colors possible.
  Pick one of NT or SA to color next, instead of Q.

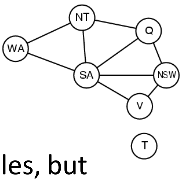
WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦
🟥	🟦	🟩	🟥	🟦		🟥🟩🟦

Backtracking w/ Forward Checking

```
function BACKTRACKING-SEARCH(csp) returns solution or failure
  return BACTRACK(csp, {})

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment

  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)

  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    add {var = value} to assignment

    if value is consistent with assignment according to constraints[csp] then

      add { var = value } to assignment

      result ← BACKTRACK(csp, assignment))

      if result ≠ failure then return result
      remove { var = value } from assignment
  return failure
```

```
inferences ← $FC(csp, var, assignment)$
if inferences ≠ failure
   add inferences to current assignment
```

Constraint propagation



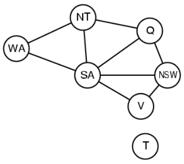
- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

WA	NT	Q	NSW	V	SA	T
🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦
🟥	🟩🟦	🟥🟩🟦	🟥🟩🟦	🟥🟩🟦	🟩🟦	🟥🟩🟦
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

- NT and SA cannot both be blue! [*Arc Inconsistency*]
- Constraint propagation repeatedly enforces constraints locally

21

Arc Consistency



- Simplest form of propagation makes each arc consistent
- *X* →*Y* is consistent iff

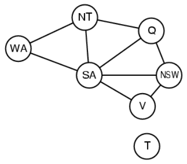
  for every value *x* of *X* there is some allowed *y* w/o violating any constraints

1. Check V and NSW – OK

WA	NT	Q	NSW	V	SA	T
🟥	🟦	🟩	🟥🟦	🟥🟩🟦	🟦	🟥🟩🟦

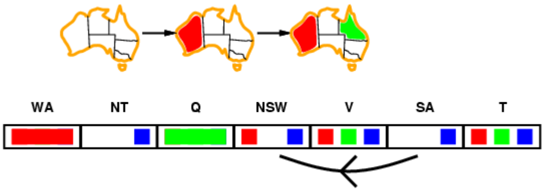
22

Arc Consistency



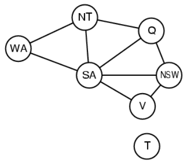
- Simplest form of propagation makes each arc consistent
- *X* →*Y* is consistent iff

  for every value *x* of *X* there is some allowed *y* w/o violating any constraints



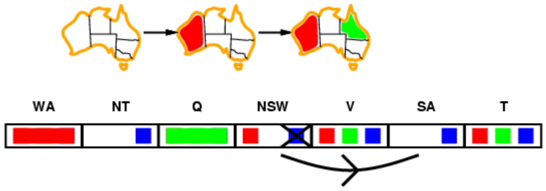
1. Check V and NSW – OK
2. Check SA and NSW – OK

Arc Consistency



- Simplest form of propagation makes each arc consistent
- *X* →*Y* is consistent iff

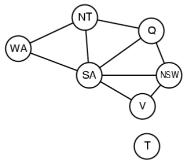
  for every value *x* of *X* there is some allowed *y* w/o violating any constraints



1. Check V and NSW – OK
2. Check SA and NSW – OK
3. Check NSW and SA

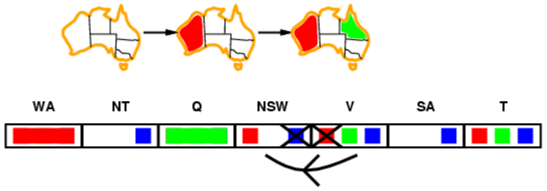
   R is OK, B is not

Arc Consistency



- Simplest form of propagation makes each arc consistent
- *X* →*Y* is consistent iff

  for every value *x* of *X* there is some allowed *y* w/o violating any constraints



- If *X* loses a value, neighbors of *X* need to be rechecked

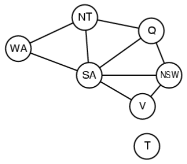
1. Check V and NSW – OK
2. Check SA and NSW – OK
3. Check NSW and SA

   R is OK, B is not
4. Check V and NSW

   R is not OK, delete

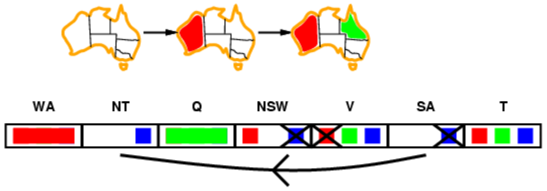
25

Arc Consistency



- Simplest form of propagation makes each arc consistent
- *X* →*Y* is consistent iff

  for every value *x* of *X* there is some allowed *y* w/o violating any constraints



- If *X* loses a value, neighbors of *X* need to be rechecked
- Arc consistency detects failure earlier than forward checking
- Can be run as a preprocessor or after each assignment

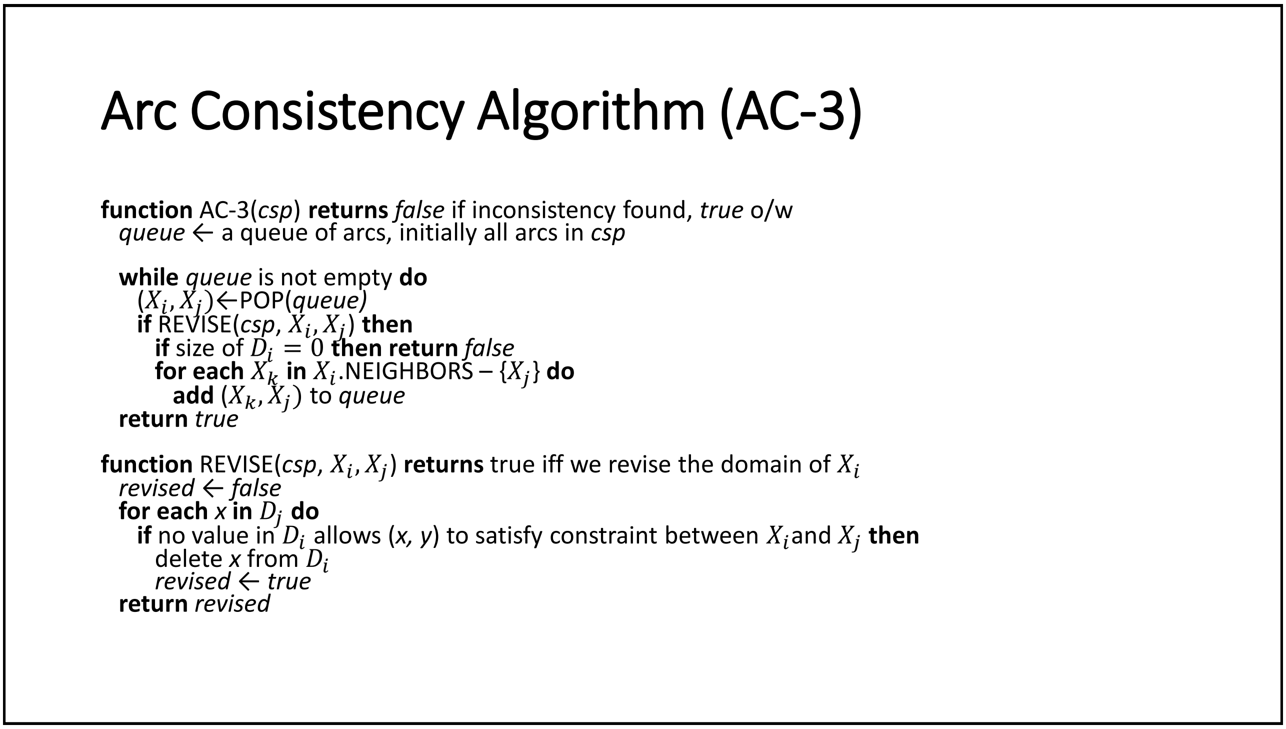
1. Check V and NSW – OK
2. Check SA and NSW – OK
3. Check NSW and SA

   R is OK, B is not
4. Check V and NSW

   R is not OK, delete
5. Check SA and NT

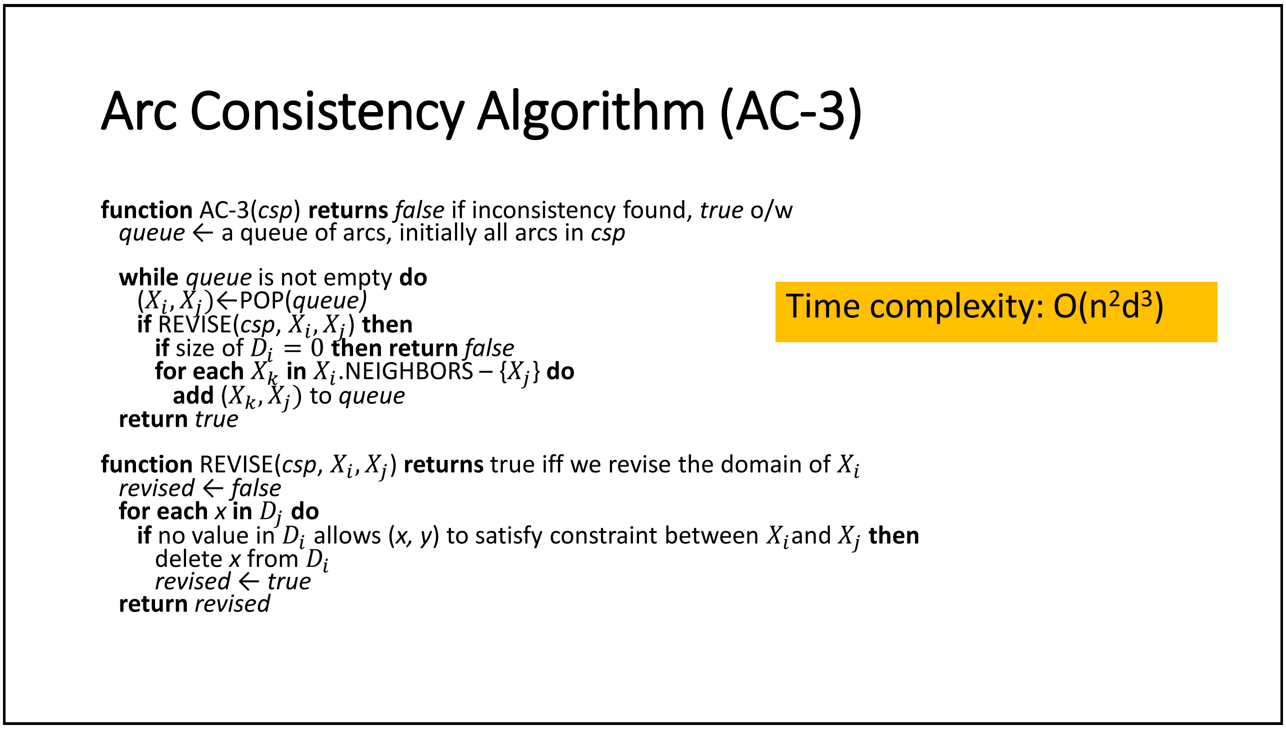
   Failure!

26



Arc Consistency Algorithm (AC-3)

```
function AC-3(csp) returns false if inconsistency found, true o/w
  queue ← a queue of arcs, initially all arcs in csp

  while queue is not empty do
    $(X_i, X_j)$←POP(queue)
    if REVISE(csp, $X_i, X_j$) then
      if size of $D_i = 0$ then return false
      for each $X_k$ in $X_i$.NEIGHBORS – $\{X_j\}$ do
        add $(X_k, X_j)$ to queue
  return true

function REVISE(csp, $X_i, X_j$) returns true iff we revise the domain of $X_i$
  revised ← false
  for each x in $D_j$ do
    if no value in $D_i$ allows (x, y) to satisfy constraint between $X_i$ and $X_j$ then
      delete x from $D_i$
      revised ← true
  return revised
```



Arc Consistency Algorithm (AC-3)

```
function AC-3(csp) returns false if inconsistency found, true o/w
  queue ← a queue of arcs, initially all arcs in csp

  while queue is not empty do
    $(X_i, X_j)$←POP(queue)
    if REVISE(csp, $X_i, X_j$) then
      if size of $D_i = 0$ then return false
      for each $X_k$ in $X_i$.NEIGHBORS – $\{X_j\}$ do
        add $(X_k, X_j)$ to queue
  return true

function REVISE(csp, $X_i, X_j$) returns true iff we revise the domain of $X_i$
  revised ← false
  for each x in $D_j$ do
    if no value in $D_i$ allows (x, y) to satisfy constraint between $X_i$ and $X_j$ then
      delete x from $D_i$
      revised ← true
  return revised
```

Time complexity: $O(n^2d^3)$

Improving Backtracking Search

- **Ordering**
  - Which variable to pick next?
    Most constrained variable (one with fewest remaining values)
  - Which value to assign next?
    Least constraining value first
- **Filtering**
  - Forward Checking
  - Arc Consistency

29

Backtracking Search Algorithm

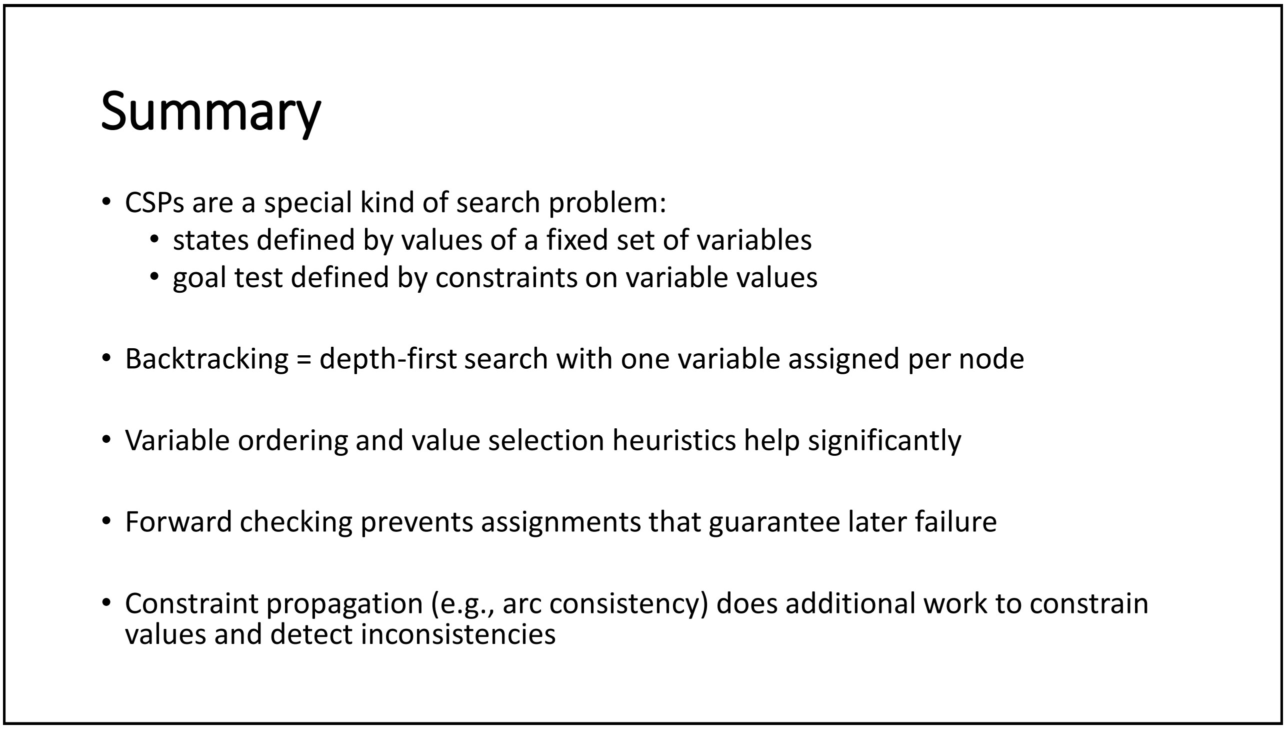
```
function BACKTRACKING-SEARCH(csp) returns solution or failure
  return BACTRACK(csp, {})

function BACKTRACK(csp, assignment) returns a solution or failure
  if assignment is complete then return assignment

  var ← SELECT-UNASSIGNED-VARIABLE(csp, assignment)

  for each value in ORDER-DOMAIN-VALUES(csp, var, assignment) do
    add {var = value} to assignment
    inferences ← INFERENCE(csp, var, assignment)

    if inferences ≠ failure then
      add inferences to csp
      result ← BACKTRACK(csp, assignment)

      if result ≠ failure then return result
      remove inferences from csp

    remove {var = value} from assignment
  return failure
```

SELECT-UNASSIGNED-VARIABLE()
- selects a variable to assign

ORDER-DOMAIN-VALUES()
- selects a value to be assigned

INFERENCE()
- checks to see if all assignments are consistent

30



Summary

- CSPs are a special kind of search 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
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies