CSP Formulation
(as a special case of search)

• State is defined by n variables

\{x_1, x_2, \ldots, x_n\}

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

\{D_1, D_2, \ldots, 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.
Example: Map-Coloring

- **Variables:** \(WA, NT, Q, NSW, V, SA, T\)

- **Domains:** \(D_i = \{\text{red, green, blue}\}\)

- **Constraints:** adjacent regions must have different colors

  e.g., \(WA \neq NT\)

  or

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

Example: Map-Coloring

- **Solutions are** complete and consistent assignments

  \[
  \{WA = \text{red}, \\
  NT = \text{green}, \\
  Q = \text{red}, \\
  NSW = \text{green}, \\
  V = \text{red}, \\
  SA = \text{blue}, \\
  T = \text{green}\}
  \]
Constraint Graph Representation of CSP

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

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?
Backtracking Search

1. Pick one variable at a time.
2. Check constraints as you go.
   (incremental goal testing)

Backtracking Search Algorithm

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

function RECURSIVE-BACKTRACKING( assignment, csp ) returns a solution, or 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 according to 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 Search Algorithm

Can we detect inevitable failure?

Which variable to pick next?

Which value to assign next?

These are general purpose heuristics.

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

function $\text{BACKTRACKING-SEARCH}(csp)$ returns a solution, or failure
return $\text{RECURSIVE-BACKTRACKING}([], csp)$

function $\text{RECURSIVE-BACKTRACKING}(\text{assignment}, csp)$ returns a solution, or failure
  if $\text{ assignment}$ is complete then return $\text{ assignment}$
  var $\leftarrow$ $\text{SELECT-UNASSIGNED-VARIABLE}(\text{Variables}[csp], \text{ assignment}, csp)$
  for each value in $\text{ORDER-DOMAIN-VALUES}(\text{var}, \text{ assignment}, csp)$ do
    if value $\text{ is consistent with } \text{ assignment}$ according to $\text{Constraints}[csp]$ then
      add $\{ \text{ var } = \text{ value } \}$ to $\text{ assignment}$
      result $\leftarrow$ $\text{RECURSIVE-BACKTRACKING}(\text{assignment}, csp)$
      if result $\neq$ failure then return result
      remove $\{ \text{ var } = \text{ value } \}$ from $\text{ assignment}$
  return failure
Most constrained variable

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

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

Least constraining value

• Given a variable, choose the least constraining value:
  • the one that rules out the fewest values in the remaining variables
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 – Constraint propagation
  • Forward Checking
  • Arc Consistency

Forward checking

• Idea
  • Keep track of remaining legal values for unassigned variables
  • Terminate search when any variable has no legal values
Backtracking Search w/ Forward Checking

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

function RECURSIVE-BACKTRACKING(assignment, csp) returns a solution, or 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 according to 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
```

Inferences \(\leftarrow FC(csp, var, assignment)\)
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:

- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally

```
WA  NT  Q  NSW  V  SA  T
```
Constraint Graph Representation of CSP

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

Arc consistency

- Simplest form of propagation makes each arc consistent
- $X \rightarrow Y$ is consistent iff
  - for every value $x$ of $X$ there is some allowed $y$ w/o violating any constraints
Arc consistency

- Simplest form of propagation makes each arc **consistent**
- $X \rightarrow Y$ 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

Arc consistency

- Simplest form of propagation makes each arc **consistent**
- $X \rightarrow Y$ 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 \textit{consistent}
• \(X \rightarrow Y\) 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

• If \(X\) loses a value, neighbors of \(X\) need to be rechecked
Arc consistency

• Simplest form of propagation makes each arc consistent
• $X \rightarrow Y$ 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

Arc consistency algorithm AC-3

```
function AC-3(csp) returns the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: queue, a queue of arcs, initially all the arcs in csp
  while queue is not empty do
    $(X_i, X_j) \leftarrow$ Remove-First(queue)
    if RM-INCONSISTENT-VALUES($X_i$, $X_j$) then
      for each $X_k$ in Neighbors[$X_i$] do
        add ($X_k$, $X_i$) to queue

function RM-INCONSISTENT-VALUES($X_i$, $X_j$) returns true iff remove a value
  removed $\leftarrow$ false
  for each $x$ in Domain[$X_i$] do
    if no value $y$ in Domain[$X_j$] allows $(x,y)$ to satisfy constraint($X_i$, $X_j$)
      then delete $x$ from Domain[$X_i$], removed $\leftarrow$ true
  return removed
```

• Time complexity: $O(cd^3)$
  $c$ is the # of constraints, $d$ is the size of largest domain
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 (AC-3 algorithm)

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