CSP Warm-up

Assign Red, Green, or Blue
Neighbors must be different

1) What is your brain doing to solve these?
2) How would you solve these with search (BFS, DFS, etc.?)

Sudoku

<table>
<thead>
<tr>
<th>1</th>
<th></th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>
AI: Representation and Problem Solving
Constraint Satisfaction Problems (CSPs)

Instructors: Pat Virtue & Stephanie Rosenthal
Slide credits: Pat Virtue, http://ai.berkeley.edu
Announcements

• HW3 due Wednesday!
• P1 due Thursday, you can work in pairs!
• Watch your time management!
What is Search For?

• Planning: sequences of actions
  • The path to the goal is the important thing
  • Paths have various costs, depths
  • Heuristics give problem-specific guidance

• Identification: assignments to variables
  • The goal itself is important, not the path
  • All paths at the same depth (for some formulations)
  • CSPs are specialized for identification problems
Constraint Satisfaction Problems
Constraint Satisfaction Problems

• Standard search problems:
  • State is a “black box”: arbitrary data structure
  • Goal test can be any function over states
  • Successor function can also be anything
Constraint Satisfaction Problems

• Standard search problems:
  • State is a “black box”: arbitrary data structure
  • Goal test can be any function over states
  • Successor function can also be anything

• Constraint satisfaction problems (CSPs):
  • A special subset of search problems
  • State is defined by variables $X_i$ with values from a domain $D$ (sometimes $D$ depends on $i$)
  • Goal test is a set of constraints specifying allowable combinations of values for subsets of variables
Real-World CSPs

• Assignment problems: e.g., who teaches what class
• Timetabling problems: e.g., which class is offered when and where?
• Hardware configuration
• Transportation scheduling
• Factory scheduling
• Circuit layout
• Fault diagnosis
• ... lots more!

• Many real-world problems involve real-valued variables...
Shelf Organization

The shelves that store products that will be shipped to you (e.g., Amazon) are optimized so that items that ship together are stored on the same shelf.
Example: Map Coloring

• Variables: WA, NT, Q, NSW, V, SA, T

• Domains: \( D = \{\text{red, green, blue}\} \)

• Constraints: adjacent regions must have different colors
  
  Implicit: WA \( \neq \) NT
  
  Explicit: \((WA, NT) \in \{(\text{red, green}), (\text{red, blue}), \ldots\}\)

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

\[ \{WA=\text{red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}\} \]
Constraint Graphs
Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables

- Binary constraint graph: nodes are variables, arcs show constraints

- General-purpose CSP algorithms use the graph structure to speed up search. E.g., Tasmania is an independent subproblem!

[Demo: CSP applet (made available by aispace.org) -- n-queens]
Varieties of CSPs and Constraints
Example: N-Queens

- Formulation 1:
  - Variables: $X_{ij}$
  - Domains: $\{0, 1\}$
  - Constraints

\[
\forall i, j, k \ (X_{ij}, X_{ik}) \in \{(0, 0), (0, 1), (1, 0)\}
\forall i, j, k \ (X_{ij}, X_{kj}) \in \{(0, 0), (0, 1), (1, 0)\}
\forall i, j, k \ (X_{ij}, X_{i+k,j+k}) \in \{(0, 0), (0, 1), (1, 0)\}
\forall i, j, k \ (X_{ij}, X_{i+k,j-k}) \in \{(0, 0), (0, 1), (1, 0)\}
\sum_{i,j} X_{ij} = N
\]
Example: N-Queens

• Formulation 2:
  • Variables: $Q_k$
  • Domains: $\{1, 2, 3, \ldots N\}$
  • Constraints:

Implicit: $\forall i, j \text{ non-threatening}(Q_i, Q_j)$

Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}$

\[\ldots\]
Example: Cryptarithmetic

- Variables:
  \[ F \quad T \quad U \quad W \quad R \quad O \quad X_1 \quad X_2 \quad X_3 \]

- Domains:
  \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}

- Constraints:
  \[ \text{alldiff}(F, T, U, W, R, O) \]
  \[ O + O = R + 10 \cdot X_1 \]
  \[ \ldots \]
Example: Sudoku

- **Variables:**
  - Each (open) square

- **Domains:**
  - \( \{1,2,\ldots,9\} \)

- **Constraints:**
  - 9-way alldiff for each column
  - 9-way alldiff for each row
  - 9-way alldiff for each region
  - (or can have a bunch of pairwise inequality constraints)
Varieties of CSPs

• Discrete Variables
  • Finite domains
    • Size $d$ means $O(d^n)$ complete assignments
    • E.g., Boolean CSPs, including Boolean satisfiability (NP-complete)
  • Infinite domains (integers, strings, etc.)
    • E.g., job scheduling, variables are start/end times for each job
    • Linear constraints solvable, nonlinear undecidable

• Continuous variables
  • E.g., start/end times for Hubble Telescope observations
  • Linear constraints solvable in polynomial time by LP methods
Varieties of Constraints

• Varieties of Constraints
  • Unary constraints involve a single variable (equivalent to reducing domains), e.g.:
    \[ SA \neq \text{green} \]
  • Binary constraints involve pairs of variables, e.g.:
    \[ SA \neq \text{WA} \]
  • Higher-order constraints involve 3 or more variables: e.g., cryptarithmetic column constraints

• Preferences (soft constraints):
  • E.g., red is better than green
  • Often representable by a cost for each variable assignment
  • Gives constrained optimization problems
  • (We’ll ignore these until we get to Bayes’ nets)
Solving CSPs
Standard Search Formulation

- Standard search formulation of CSPs

- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}
  - Successor function: assign a value to an unassigned variable
  - Goal test: the current assignment is complete and satisfies all constraints

- We’ll start with the straightforward, naïve approach, then improve it
Breadth First Search

... All possible first variables
Check: Is there a solution?
Breadth First Search
**Breadth First Search**

```
<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

For the first few rows, WA and NT are colored blue, Q is colored green, NSW and V are empty, and SA is empty.
Breadth First Search

WA  NT  Q  NSW  V  SA

...
**Depth First Search**

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Demo
What is wrong with general search?

• When do you fail?
Backtracking Search
Backtracking Search

• Backtracking search is the basic uninformed algorithm for solving CSPs

• Idea 1: One variable at a time
  • Variable assignments are commutative, so fix ordering
  • I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  • Only need to consider assignments to a single variable at each step

• Idea 2: Check constraints as you go
  • I.e. consider only values which do not conflict previous assignments
  • Might have to do some computation to check the constraints
  • “Incremental goal test”

• Depth-first search with these two improvements is called backtracking search (not the best name)

• Can solve n-queens for n ≈ 25
Backtracking Example
Backtracking Search

```plaintext
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 Search

General Search checks consistency on full assignment

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 Search

function BACKTRACKING-SEARCH(csp) returns solution/failure
    assignment ← {};
    result ← RECURSIVE-BACKTRACKING(assignment, csp); return result

function RECURSIVE-BACKTRACKING(assignment, csp) returns solution/failure
    if assignment is complete then return assignment
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[assignment] then
            add \{var = value\} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp); if result \neq failure then return result
            remove \{var = value\} from assignment
    return failure

return BACKTRACKING-SEARCH(csp)
Backtracking Search

function \textbf{Backtracking-Search}(csp) returns solution/failure
  return \textbf{Recursive-Backtracking}({}, csp)

function \textbf{Recursive-Backtracking}(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← \textbf{Select-Unassigned-Variable}({\textsc{Variables}}[csp], assignment, csp)
  for each value in \textbf{Order-Domain-Values}(var, assignment, csp) do
    if value is consistent with assignment given \textsc{Constraints}[csp] then
      add \{var = value\} to assignment
      result ← \textbf{Recursive-Backtracking}(assignment, csp)
      if result \neq \textit{failure} then return result
    remove \{var = value\} from assignment
  return failure

• Backtracking = DFS + variable-ordering + fail-on-violation
• What are the choice points?
Backtracking Search

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 = DFS + variable-ordering + fail-on-violation
- What are the choice points?
Demo Coloring – Backtracking
Improving Backtracking

• General-purpose ideas give huge gains in speed

• Filtering: Can we detect inevitable failure early?

• Ordering:
  • Which variable should be assigned next?
  • In what order should its values be tried?

• Structure: Can we exploit the problem structure?
Filtering
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options.
- Forward checking: Cross off values that violate a constraint when added to the existing assignment.
Filtering: Keep track of domains for unassigned variables and cross off bad options

Forward checking: Cross off values that violate a constraint when added to the existing assignment
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment
Filtering: Forward Checking

- Filtering: Keep track of domains for unassigned variables and cross off bad options
- Forward checking: Cross off values that violate a constraint when added to the existing assignment

![Diagram of Australia with state abbreviations]

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Colors for WA]</td>
<td>![Colors for NT]</td>
<td>![Colors for Q]</td>
<td>![Colors for NSW]</td>
<td>![Colors for V]</td>
<td>![Colors for SA]</td>
</tr>
</tbody>
</table>

FAIL – variable with no possible values
Demo Coloring – Backtracking with Forward Checking
Filtering: Constraint Propagation

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

<table>
<thead>
<tr>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Filtering: 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! Why didn’t we detect this yet?

<table>
<thead>
<tr>
<th></th>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

![Map of Australia with states colored in red and blue]
Filtering: 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! Why didn’t we detect this yet?
- Constraint propagation: reason from constraint to constraint
Consistency of A Single Arc

• An arc $X \rightarrow Y$ is consistent iff for every $x$ in the tail there is some $y$ in the head which could be assigned without violating a constraint.

• Remove values in the domain of $X$ if there isn’t a corresponding legal $Y$.

• Forward checking: Enforcing consistency of arcs pointing to each new assignment.
Consistency of A Single Arc

• An arc $X \rightarrow Y$ is consistent iff for every $x$ in the tail there is some $y$ in the head which could be assigned without violating a constraint.

• Remove values in the domain of $X$ if there isn’t a corresponding legal $Y$.

• Forward checking: Enforcing consistency of arcs pointing to each new assignment.
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure **all** arcs are consistent:

  ```
  Remember: Delete from the tail!
  ```
Enforcing Arc Consistency in a CSP

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
    \((X_i, X_j)\) ← 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 \(X_i \leftarrow X_j\)
    then delete \(x\) from DOMAIN[\(X_i\)]; removed ← true
return removed
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Queue:
NT->WA
WA->SA
NT->SA
Q->SA
NSW->SA
V->SA

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Queue:
WA->SA
NT->SA
Q->SA
NSW->SA
V->SA
WA->NT
SA->NT
Q->NT

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Queue:
WA->SA
NT->SA
Q->SA
NSW->SA
V->SA
WA->NT
SA->NT
Q->NT

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Queue:
NT->SA  
Q->SA  
NSW->SA  
V->SA  
WA->NT  
SA->NT  
Q->NT

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Queue:
Q->SA
NSW->SA
V->SA
WA->NT
SA->NT
Q->NT

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Queue:
NSW->SA
V->SA
WA->NT
SA->NT
Q->NT

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Remember: Delete from the tail!
POLL: What gets added to the Queue?

• A simple form of propagation makes sure all arcs are consistent:

Queue:
A: NSW->Q, SA->Q, NT->Q
B: Q->NSW, Q->SA, Q->NT
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:

Queue:
- NT->Q
- SA->Q
- NSW->Q

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Queue:
NSW->Q
WA->NT
SA->NT
Q->NT
WA->SA
NT->SA
Q->SA
NSW->SA
V->SA

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

Queue:
WA->NT
SA->NT
Q->NT
WA->SA
NT->SA
Q->SA
NSW->SA
V->SA
V->NSW
Q->NSW
SA->NSW

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:

Queue:
- WA->NT
- SA->NT
- Q->NT
- WA->SA
- NT->SA
- Q->SA
- NSW->SA
- V->SA
- V->NSW
- Q->NSW
- SA->NSW

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:

Queue:
- SA->NT
- Q->NT
- WA->SA
- NT->SA
- Q->SA
- NSW->SA
- V->SA
- V->NSW
- Q->NSW
- SA->NSW

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

- A simple form of propagation makes sure all arcs are consistent:

Queue:
- SA->NT
- Q->NT
- WA->SA
- NT->SA
- Q->SA
- NSW->SA
- V->SA
- V->NSW
- Q->NSW
- SA->NSW

Remember: Delete from the tail!
Arc Consistency of an Entire CSP

• A simple form of propagation makes sure all arcs are consistent:

• Backtrack on the assignment of Q
• Arc consistency detects failure earlier than forward checking
• Can be run as a preprocessor or after each assignment
• What’s the downside of enforcing arc consistency?

Queue:
SA→NT
Q→NT
WA→SA
NT→SA
Q→SA
NSW→SA
V→SA
V→NSW
Q→NSW
SA→NSW

Remember: Delete from the tail!
Enforcing Arc Consistency in a CSP

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) ← 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

• Runtime: O(n^2d^3), can be reduced to O(n^2d^2)
• ... but detecting all possible future problems is NP-hard – why?
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 runs inside a backtracking search!

[Demo: coloring -- arc consistency]
[Demo: coloring -- forward checking]
Demo Coloring – Backtracking with Forward Checking – Complex Graph
Demo Coloring – Backtracking with Arc Consistency – Complex Graph
Ordering
Demo: Coloring -- Backtracking + Forward Checking (+ MRV)
Ordering: Minimum Remaining Values

• Variable Ordering: Minimum remaining values (MRV):
  • Choose the variable with the fewest legal left values in its domain

• Why min rather than max?
• Also called “most constrained variable”
• “Fail-fast” ordering
Ordering: Least Constraining Value

• Value Ordering: Least Constraining Value
  • Given a choice of variable, choose the least constraining value
  • I.e., the one that rules out the fewest values in the remaining variables
  • Note that it may take some computation to determine this! (E.g., rerunning filtering)

• Why least rather than most?

• Combining these ordering ideas makes 1000 queens feasible
Demo: Coloring -- Backtracking + Arc Consistency + Ordering
Structure
Problem Structure

• Extreme case: independent subproblems
  • Example: Tasmania and mainland do not interact

• Independent subproblems are identifiable as connected components of constraint graph

• Suppose a graph of $n$ variables can be broken into subproblems of only $c$ variables:
  • Worst-case solution cost is $O((n/c)(d^c))$, linear in $n$
  • E.g., $n = 80$, $d = 2$, $c = 20$
  • $2^{80} = 4$ billion years at 10 million nodes/sec
  • $(4)(2^{20}) = 0.4$ seconds at 10 million nodes/sec
Tree-Structured CSPs

- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$

- This property also applies to probabilistic reasoning (later): an example of the relation between syntactic restrictions and the complexity of reasoning
Tree-Structured CSPs

- Algorithm for tree-structured CSPs:
  - Order: Choose a root variable, order variables so that parents precede children

  - Remove backward: For $i = n : 2$, apply $\text{RemoveInconsistent}(\text{Parent}(X_i), X_i)$
  - Assign forward: For $i = 1 : n$, assign $X_i$ consistently with $\text{Parent}(X_i)$

- Runtime: $O(n d^2)$ (why?)
Tree-Structured CSPs

• Claim 1: After backward pass, all root-to-leaf arcs are consistent
  • Proof: Each $X \rightarrow Y$ was made consistent at one point and $Y$’s domain could not have been reduced thereafter (because $Y$’s children were processed before $Y$)

• Claim 2: If root-to-leaf arcs are consistent, forward assignment will not backtrack
  • Proof: Induction on position

• Why doesn’t this algorithm work with cycles in the constraint graph?
  • Note: we’ll see this basic idea again with Bayes’ nets
Summary: CSPs

- CSPs are a special kind of search problem:
  - States are partial assignments
  - Goal test defined by constraints

- Basic solution: backtracking search

- Speed-ups:
  - Ordering
  - Filtering
  - Structure