Constraint Satisfaction Problem

Radek Mařík

FEE CTU, K13132

May 9, 2017

[1] Based on the slides created by Stuart Russell

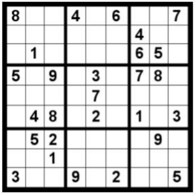
Outline

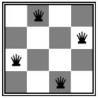
1. Methodology Overview
   - Basic Definitions
2. Algorithms
   - Generate and Test
   - Backtracking search
   - Variable and value ordering heuristics
   - Forward checking
   - Node consistency, arc consistency, path consistency
   - Local search for CSPs: min-conflict heuristic

Objectives [RN10]

Constraint Satisfaction Problem (CSP)

1. Define **possible worlds** in term of **variables** and their **domains**
2. Specify **constraints** to represent real world problems
3. Verify whether a possible world satisfies a set of constraints





What is a CSP? [RN10]

A CSP is defined by

1. A finite set $\mathcal{V}$ of variables V_i, $i = 1, \ldots, n$
2. A nonempty domain $D_i = \text{dom}(V_i)$ of possible values for each variable $V_i \in \mathcal{V}$
3. A finite set of constraints $C_1, C_2, \ldots, C_m$
   - Each constraint C_i limits the values that variables can take
   - for subsets of the variables

Example 1

- $\mathcal{V} = \{V_1\}$
  - $D_1 = \text{dom}(V_1) = \{1,2,3,4\}$
- $\mathcal{C} = \{C_1, C_2\}$
  - C_1: $V_1 \neq 2$
  - C_2: $V_1 > 1$

- $\mathcal{V} = \{V_1, V_2\}$
  - $D_1 = \{1,2,3\}$, $D_2 = \{1,2\}$
- $\mathcal{C} = \{C_1, C_2, C_3\}$
  - C_1: $V_1 \neq 2$
  - C_2: $V_1 + V_2 < 5$
  - C_3: $V_1 > V_2$

Possible Worlds [RN10]

CSP model

- A model of a CSP is an assignment of values to all of its variables that satisfies all of its constraints.
  - i.e. a model is a possible world that satisfies all constraints

Example 2

- $\mathcal{V} = \{V_1\}$
  - $D_1 = \text{dom}(V_1) = \{1, 2, 3, 4\}$
- $\mathcal{C} = \{C_1, C_2\}$
  - C_1: $V_1 \neq 2$
  - C_2: $V_1 > 1$
- All models for this CSP:
  - $\{V_1 = 3\}$
  - $\{V_1 = 4\}$

Possible Worlds [RN10]

CSP model

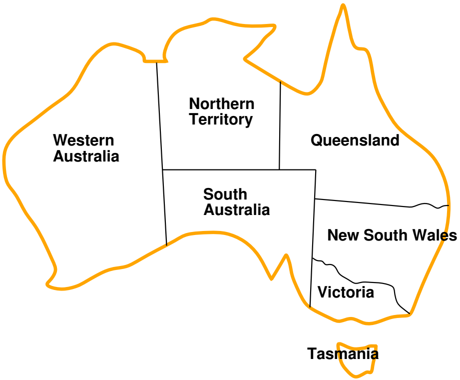
- A model of a CSP is an assignment of values to all of its variables that satisfies all of its constraints.
  - i.e. a model is a possible world that satisfies all constraints
  - a CSP solution

Example 3

- $\mathcal{V} = \{V_1, V_2\}$
  - $D_1 = \{1, 2, 3\}$, $D_2 = \{1, 2\}$
- $\mathcal{C} = \{C_1, C_2, C_3\}$
  - C_1: $V_1 \neq 2$
  - C_2: $V_1 + V_2 < 5$
  - C_3: $V_1 > V_2$

- Possible worlds for this CSP:
  - $\{V_1 = 1, V_2 = 1\}$
  - $\{V_1 = 1, V_2 = 2\}$
  - $\{V_1 = 2, V_2 = 1\}$ (a model)
  - $\{V_1 = 2, V_2 = 2\}$
  - $\{V_1 = 3, V_2 = 1\}$ (a model)
  - $\{V_1 = 3, V_2 = 2\}$

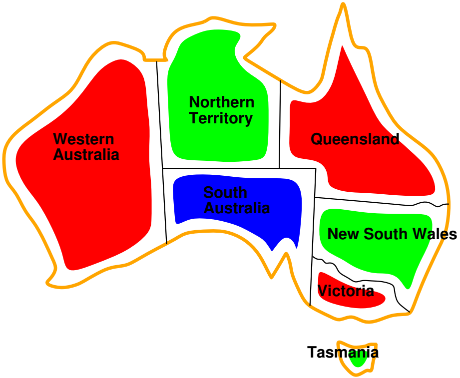
Example: Map Coloring Problem



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

- e.g. $WA \neq NT$ (if the language allows this), or
- $(WA, NT) \in \{(red, green), (red, blue), (green, red), \ldots\}$

Example: Map Coloring Model



Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green,$
$V = red, SA = blue, T = green\}$

Constraints

Constraints are restrictions on the values that one or more variables can take:

- Unary constraint: a restriction involving a single variable
  - e.g.: $V2 \neq 2$
- k-ary constraint: a restriction involving k different variables
  - e.g. binary ($k = 2$): $V_1 + V_2 < 5$
  - e.g. 3-ary: $V_1 + V_2 + V_4 < 5$
  - We will mostly deal with binary constraints ($k = 2$).
- Constraints can be specified by
  1. listing all combinations of valid domain values for the variables participating in the constraint
     - e.g. for constraint $V_1 > V_2$ and dom$(V1) = \{1,2,3\}$ and dom$(V2) = \{1,2\}$: $\{v(V_1, V_2)_i\} = \{(2,1),(3,1),(3,2)\}$
  2. giving a function (predicate) that returns true if given values for each variable which satisfy the constraint else false: $V_1 > V_2$

Scope of a Constraint

- Each constraint C_i is a pair $<$ scope, relation $>$
- **Relation** . . . a list of allowed combinations of variable values.

Scope

The scope of a constraint is the set of variables that are involved in the constraint

Example 4

- $V_2 \neq 2$ has scope $\{V_2\}$
- $V_1 > V_2$ has scope $\{V_1, V_2\}$
- $V_1 + V_2 + V_4 < 5$ has scope $\{V_1, V_2, V_4\}$
- How many variables are in the scope of a k-ary constraint?
  - k variables

Finite Constraint Satisfaction Problem

FCSP

A finite constraint satisfaction problem (FCSP) is a CSP with a finite set of variables and a finite domain for each variable.

- We will only study finite CSPs here but many of the techniques carry over to countably infinite and continuous domains.
  We use CSP here to refer to FCSP.
  - The scope of each constraint is automatically finite since it is a subset of the finite set of variables.

Solution Variants

We may want to solve the following problems with a CSP:

- determine whether or not a model exists
- find a model
- find all of the models
- count the number of models
- find the best model, given some measure of model quality
  - this is now an optimization problem
- determine whether some property of the variables holds in all models

What is a solution?

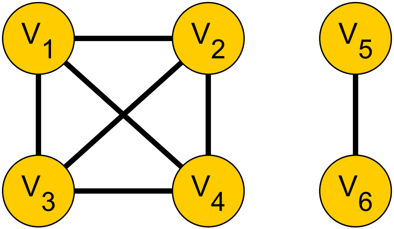
- A **state** is an assignment of values to some or all variables.
  - An assignment is **complete** when every variable has a value.
  - An assignment is **partial** when some variables have no values.
- **Consistent assignment**
  - the assignment does not violate the constraints
- A **solution** to a CSP is a complete and consistent assignment.
- Some CSPs require a solution that maximizes an *objective function*.
- **Preferences** (soft contraints): can be represented using costs, and lead to **constrained optimization problems**.

Solving Constraint Satisfaction Problems

- Even the simplest problem of determining whether or not a model exists in a general CSP with finite domains is NP-hard
  - There is no known algorithm with worst case polynomial runtime.
  - We can't hope to find an algorithm that is polynomial for all CSPs.
- However, we can try to:
  - find efficient (polynomial) consistency algorithms that reduce the size of the search space
  - identify special cases for which algorithms are efficient
  - work on approximation algorithms that can find good solutions quickly, even though they may offer no theoretical guarantees
  - find algorithms that are fast on typical (not worst case) cases

Constraint Graph

- Binary CSP: each constraint relates at most two variables
- Constraint graph:
  - nodes are variables
  - arcs are binary constraints
- The structure of the graph can be exploited to provide problem solutions:
  - Graph can be used to simplify search
  - e.g. a decomposition into subproblems



Real-world CSPs

- Assignment problems (e.g. who teaches what class)
- Timetabling problems
  - e.g. which class is offered when and where?
- Hardware configuration
- Hardware verification (e.g. IBM)
- Transportation scheduling
- Factory scheduling
- Floor planning
- Puzzle solving (e.g. crosswords, sudoku)
- Software verification (small to medium programs)

Example: 4-Queens as a CSP

Assume one queen in each column. Which row does each one go in?

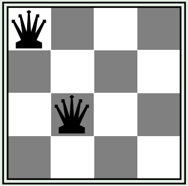
4 Queens Problem

Variables Q_1, Q_2, Q_3, Q_4

Domains $D_i = \{1, 2, 3, 4\}$

Constraints

- $Q_i \neq Q_j$ (cannot be in same row)
- $|Q_i - Q_j| \neq |i - j|$ (or same diagonal)



$Q_1 = 1 \quad Q_2 = 3$

Translate each constraint into a set of allowable values for its variables

E.g., values for (Q_1, Q_2) are $(1,3)$ $(1,4)$ $(2,4)$ $(3,1)$ $(4,1)$ $(4,2)$

Generate and Test (GT) Algorithms

- Systematically check all possible worlds
  - Possible worlds: cross product of domains

$$\mathrm{dom}(V_1) \times \mathrm{dom}(V_2) \times \cdots \times \mathrm{dom}(V_n)$$

- Generate and Test:
  1. Generate possible worlds one at a time
  2. Test constraints for each one.

Example 5

3 variables A, B, C
for $a \in$ dom(A)
 for $b \in$ dom(B)
  for $c \in$ dom(C)
   if $\{A = a,\ B = b,\ C = c\}$ satisfies all constraints
    return $\{A = a,\ B = b,\ C = c\}$
fail

Standard search formulation (incremental)

- Let's start with the straightforward, dumb approach, then fix it
- States are defined by the values assigned so far
  1. **Initial state**: the empty assignment, $\emptyset$
  2. **Successor function**: assign a value to an unassigned variable that does not conflict with current assignment.
     $\Rightarrow$ fail if no legal assignments (not fixable!)
  3. **Goal test**: the current assignment is complete

1. This is the same for all CSPs! 😈
2. Every solution appears at depth n with n variables
   $\Rightarrow$ use depth-first search
3. Path is irrelevant, so can also use complete-state formulation
4. $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!!!! ☹

Backtracking search

Variable assignments are commutative, 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 node

$\Rightarrow$ $b = d$ and there are d^n leaves

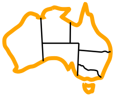
Depth-first search for CSPs with single-variable assignments is called backtracking search
Backtracking search is the basic uninformed algorithm for CSPs
Can solve n-queens for $n \approx 25$

Backtracking search

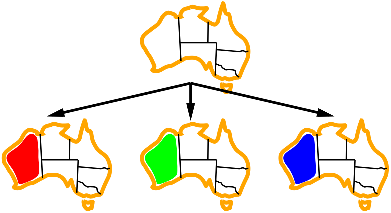
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

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
```

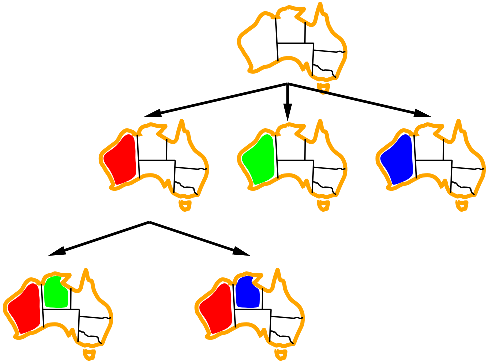
Backtracking example



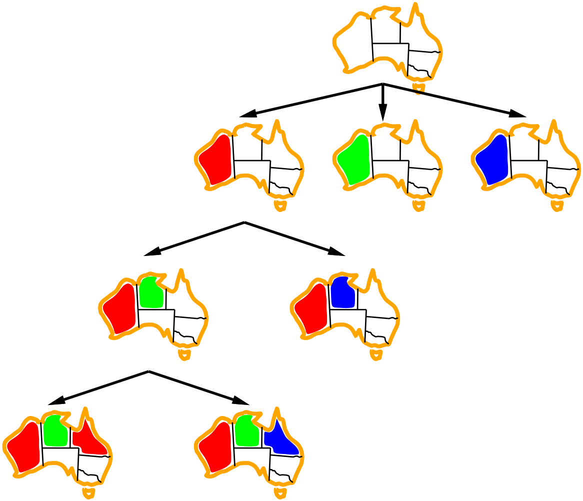
Backtracking example



Backtracking example



Backtracking example



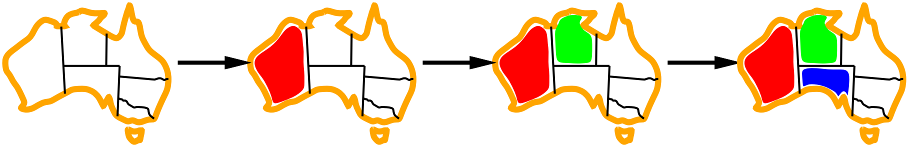
Improving backtracking efficiency

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?
4. Can we take advantage of problem structure?

Minimum remaining values

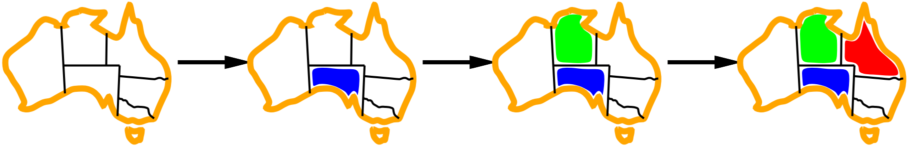
Minimum remaining values (MRV):
choose the variable with the fewest legal values



Degree heuristic

Tie-breaker among MRV variables
Degree heuristic:

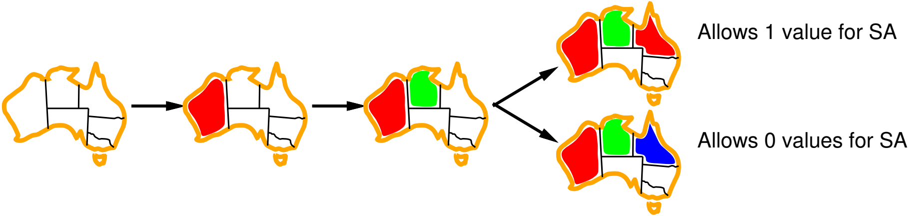
choose the variable with the most constraints on remaining variables



Least constraining value

Given a variable, choose the least constraining value:

the one that rules out the fewest values in the remaining variables

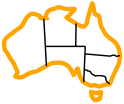


Combining these heuristics makes 1000 queens feasible

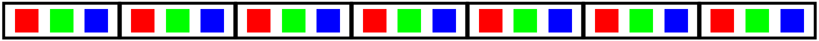
Forward checking

Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values



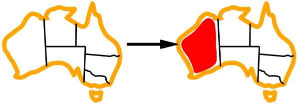
WA **NT** **Q** **NSW** **V** **SA** **T**

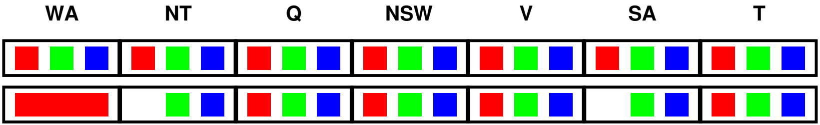


Forward checking

Idea: Keep track of remaining legal values for unassigned variables

Terminate search when any variable has no legal values

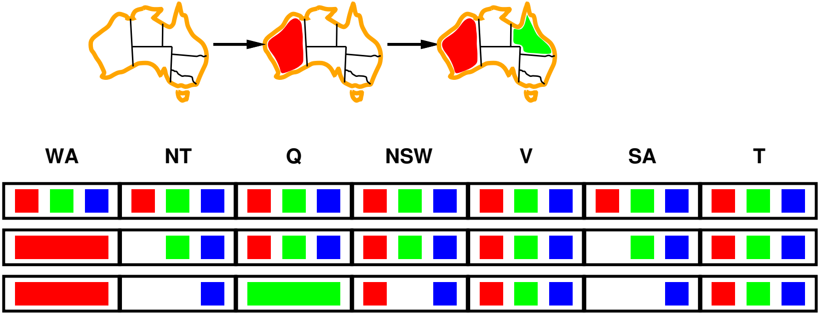




Forward checking

Idea: Keep track of remaining legal values for unassigned variables

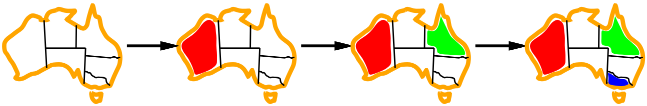
Terminate search when any variable has no legal values

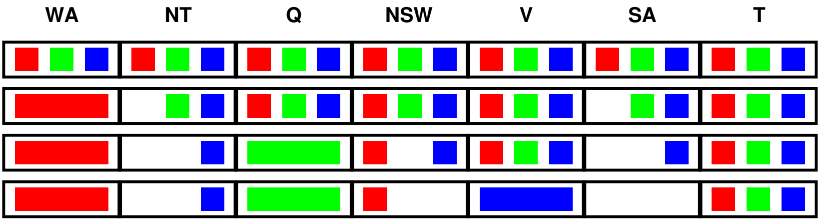


Forward checking

Idea: Keep track of remaining legal values for unassigned variables

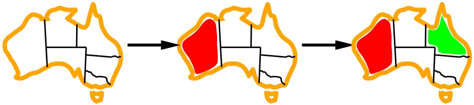
Terminate search when any variable has no legal values





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!

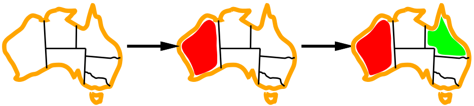
Constraint propagation repeatedly enforces constraints locally

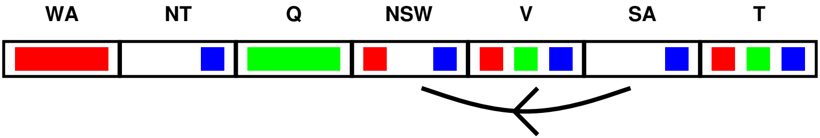
Arc consistency

Simplest form of propagation makes each arc consistent

$X \rightarrow Y$ is consistent iff

for **every** value x of X there is **some** allowed y

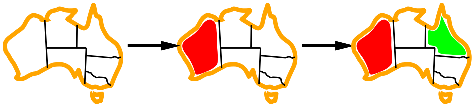


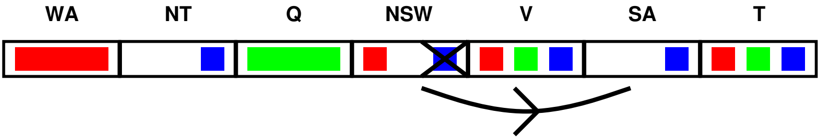


Arc consistency

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

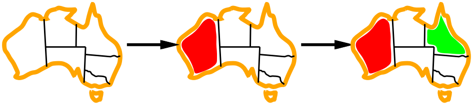


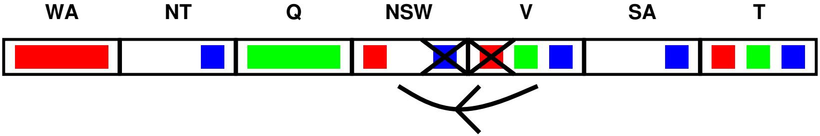


Arc consistency

Simplest form of propagation makes each arc consistent
$X \to Y$ is consistent iff

for **every** value x of X there is **some** allowed y





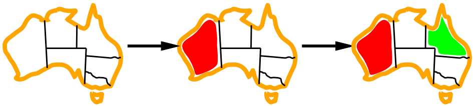
If X loses a value, neighbors of X need to be rechecked

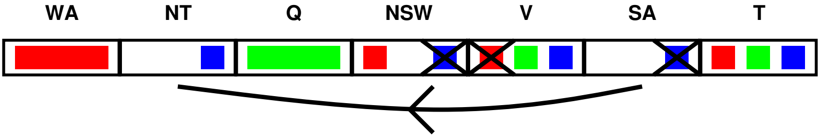
Arc consistency

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





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

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*

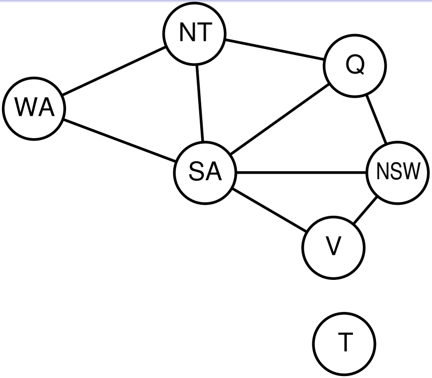
while *queue* is not empty **do**
  $(X_i,\ X_j) \leftarrow$ REMOVE-FIRST(*queue*)
  if REMOVE-INCONSISTENT-VALUES($X_i,\ X_j$) **then**
    for each X_k **in** NEIGHBORS[X_i] **do**
      add ($X_k,\ X_i$) to *queue*

---

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

$O(n^2d^3)$, can be reduced to $O(n^2d^2)$ (but detecting **all** is NP-hard)

Problem structure



- Tasmania and mainland are independent subproblems
- Identifiable as connected components of constraint graph

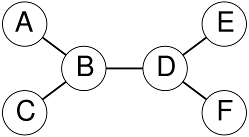
Problem structure contd.

- Suppose each independent subproblem has c variables out of n total
- n/c subproblems, each of which takes at most d^c work to solve
- Worst-case solution cost is $n/c \cdot d^c$, **linear** in n

$n = 80$, $d = 2$, $c = 20$

1. 2^{80} = 4 billion years at 10 million nodes/sec
2. $4 \cdot 2^{20}$ = 0.4 seconds at 10 million nodes/sec

Tree-structured CSPs



- Any try with n nodes has $n-1$ arcs.
- The graph made directed arc-consistent in $O(n)$ steps.
- Each step must compare up to $O(d)$ possible domain values for 2 variables.

Theorem 6

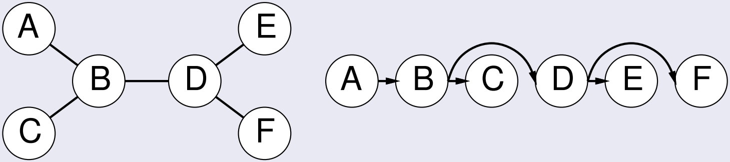
Theorem: If the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$ time

- Compare to general CSPs, where worst-case time is $O(d^n)$
- This property also applies to logical and probabilistic reasoning:
  - an important example of the relation between syntactic restrictions and the complexity of reasoning.

Algorithm for tree-structured CSPs

Algorithm

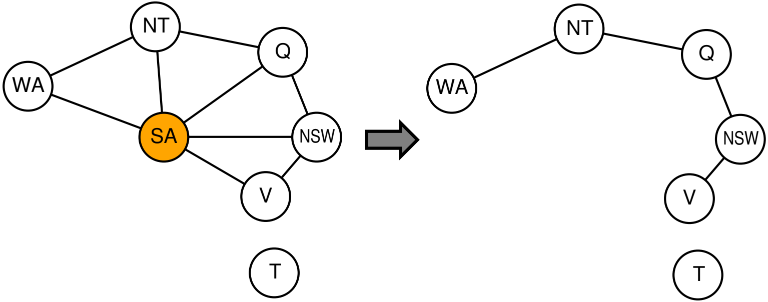
1. **Topological sort:**
   Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering



2. For j from n down to 2, apply
   REMOVEINCONSISTENT$(Parent(X_j), X_j)$
3. For j from 1 to n, assign X_j consistently with $Parent(X_j)$

Nearly tree-structured CSPs

Conditioning: instantiate a variable, prune its neighbors' domains



- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size $c \Rightarrow$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small c

Iterative algorithms for CSPs

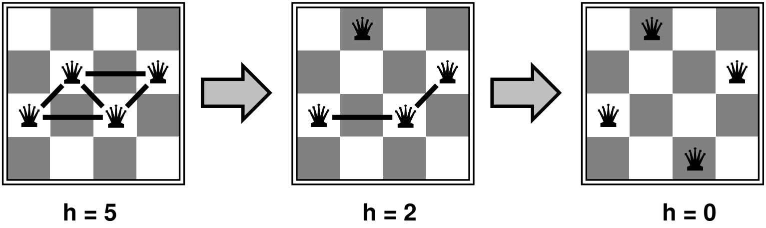
- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  1. allow states with unsatisfied constraints
  2. operators **reassign** variable values

Variable selection

- randomly select any conflicted variable
- min-conflicts heuristic:
  - choose value that violates the fewest constraints
  - i.e., hillclimb with $h(n)$ = total number of violated constraints

Example: 4-Queens

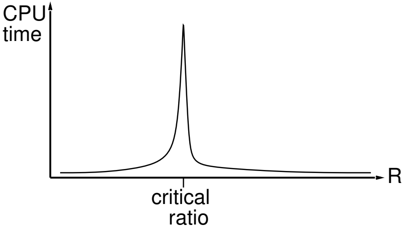
States: 4 queens in 4 columns ($4^4 = 256$ states)
Operators: move queen in column
Goal test: no attacks
Evaluation: $h(n)$ = number of attacks



Performance of min-conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability
  - e.g. queens n = 10,000,000 in ≈ 50 steps
- The same appears to be true for any randomly-generated CSP **except** in a narrow range of the ratio.

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



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.
- **Basic solution**:
  - Backtracking = depth-first search with one variable assigned per node
- **Speed-ups**:
  - 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
  - The CSP representation allows analysis of problem structure
    - Tree-structured CSPs can be solved in linear time
- **Iterative min-conflicts** is usually effective in practice

References I

Stuart J. Russell and Peter Norvig.
Artificial Intelligence, A Modern Approach.
Pre, third edition, 2010.