ANU COMP3620/6320
Constraint Satisfaction Problems

Chapter 5: “AI: A Modern Approach”

Lecturer: Scott Sanner

Slides Adapted from Instructor’s Material

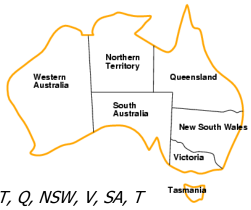
Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Local search for CSPs
- Connections to prop. and first-order logic
- Summary
- Extensions

Constraint satisfaction problems (CSPs)

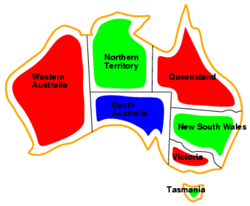
- Standard search problem:
  - state is a "black box“ – any data structure that supports successor function, heuristic function, and goal test
- CSP:
  - state is defined by variables X_i with values from domain D_i
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables
- Simple example of a formal representation language
- Allows useful general constraint search algorithms with more power than standard search algorithms

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

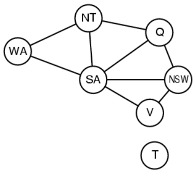
Example: Map-Coloring



- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

Constraint graph

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



Varieties of CSPs

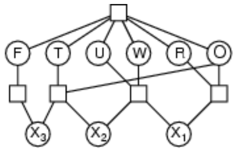
- Discrete variables
  - finite domains:
    - *n* variables, domain size *d* → $O(d^n)$ complete assignments
    - e.g., Boolean CSPs, incl.~Boolean satisfiability (NP-complete)
  - infinite domains:
    - integers, strings, etc.
    - e.g., job scheduling, variables are start/end days for each job
    - need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
- Continuous variables
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

Varieties of constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., SA ≠ WA
- Higher-order constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints

Example: Cryptarithmetic

```
   T W O
 + T W O
 -------
 F O U R
```



- Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
- Constraints: *Alldiff (F,T,U,W,R,O)*
  - $O + O = R + 10 \cdot X_1$
  - $X_1 + W + W = U + 10 \cdot X_2$
  - $X_2 + T + T = O + 10 \cdot X_3$
  - $X_3 = F,\ T \neq 0,\ F \neq 0$

Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling
- Many real-world problems involve real-valued variables

Standard search formulation (incremental)

Let's start with the straightforward approach, then fix it

States are defined by the values assigned so far

- Initial state: the empty assignment { }
- Successor function: assign a value to an unassigned variable that does not conflict with current assignment
  → fail if no legal assignments
- 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
   → use depth-first search
3. Path is irrelevant, so can also use complete-state formulation
4. $b = (n - \ell)d$ at depth ℓ, hence $n! \cdot d^n$ leaves

Backtracking search

- Variable assignments are commutative, i.e.,
  - [WA = red then NT = green]
  - same as [NT = green then WA = red]
- Only consider assignments to a one variable at each node
  → b = d and there are d^n leaves
- DFS for CSPs is called backtracking search
  - basic uninformed algorithm for CSPs
- Can solve *n*-queens for $n \approx 25$

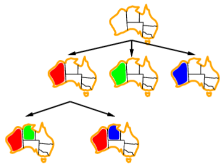
Backtracking search

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

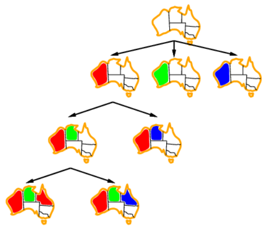
Backtracking example

Backtracking example

Backtracking example



Backtracking example



Improving backtracking efficiency

- General constraint search methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?

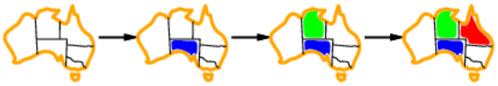
Most constrained variable

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

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

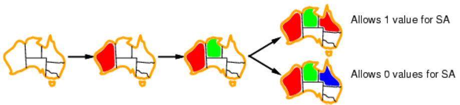
Most constraining variable

- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables



Least constraining value

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



- 1000 queens feasible with these heuristics

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

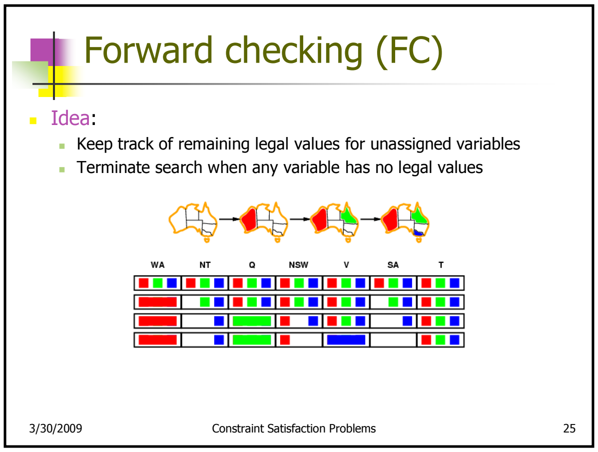
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

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

Forward checking

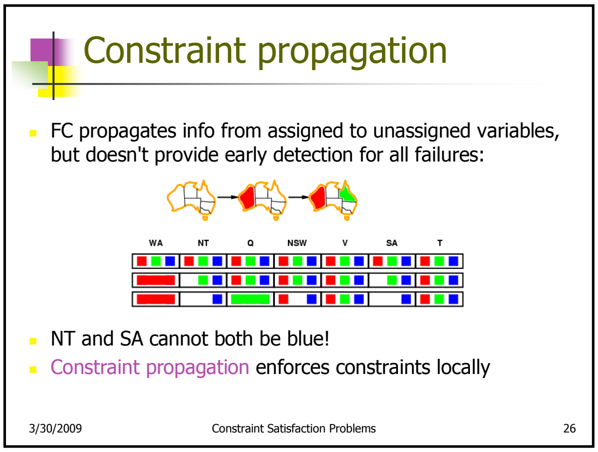
- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

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



Forward checking (FC)

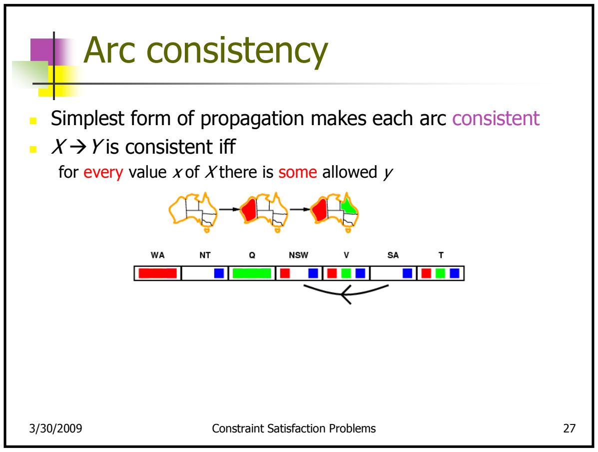
- *Idea*:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values



Constraint propagation

- FC propagates info from assigned to unassigned variables, but doesn't provide early detection for all failures:

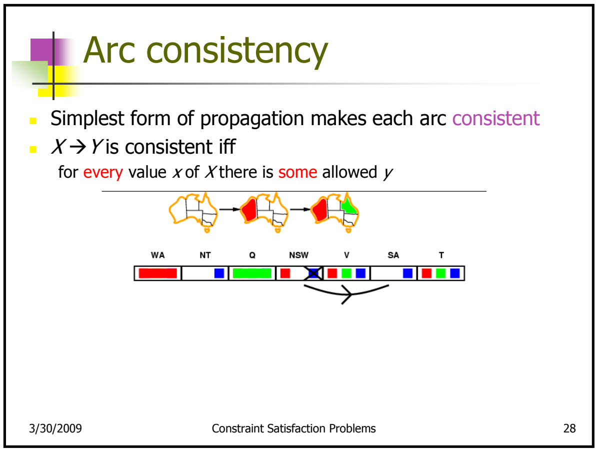
- NT and SA cannot both be blue!
- *Constraint propagation* enforces constraints locally



Arc consistency

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

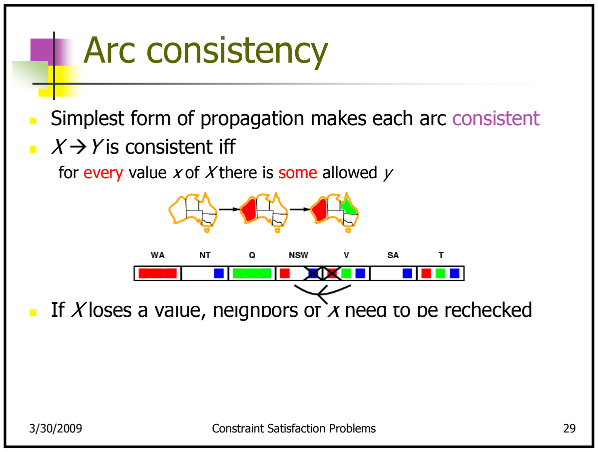
  for *every* value x of X there is *some* allowed y



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

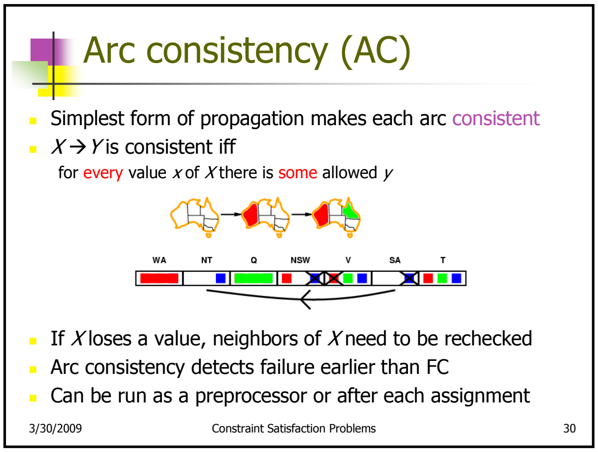


Arc consistency

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

- 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 FC
- 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 ← 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 ← true
  return removed
```

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

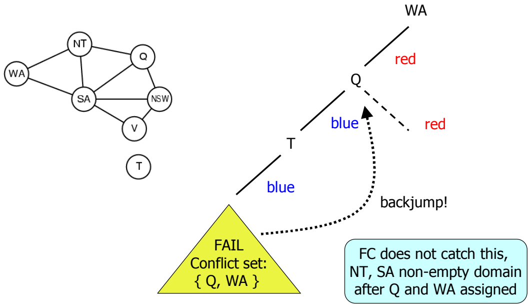
Backjumping

- Backtracking may waste search
  - reason for conflicts could be far back in tree
  - then local backtracking doomed to fail
- Conflict set of variable v
  - previously assigned variables in constraints that conflict with all of v's assignments
  - can *backjump* to most recent variable in conflict set
- But forward checking subsumes backjumping
  - FC does early check if assignments force domain(v) = ∅

Conflict-directed Backjumping

- Backjumping
  - Only analyzes conflicts at leaves of tree
- Conflict-directed backjumping
  - Internal nodes union conflict sets from leaves minus any unassigned variables
- Can be combined with forward checking
  - FC-CBJ – highly effective CSP technique

CBJ Example



FC does not catch this, NT, SA non-empty domain after Q and WA assigned

Partial Assignment vs. Local Search

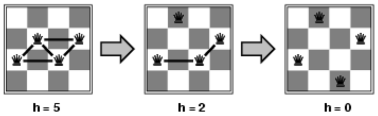
- Previously we built a tree of partial assignments
  - Branch fills in more assignments until solution found
  - Or no solution on that branch, then backtrack (or jump)
- But we can also start with a random *full assignment* that may have conflicts
  - Then we perturb the assignment to reduce conflicts (on average)
  - Same idea as simulated annealing
  - Known as *(stochastic) local search*

Local search for CSPs

- Hill-climbing, simulated annealing typically work with "complete" states, i.e., all variables assigned
- To apply to CSPs:
  - allow states with unsatisfied constraints
  - operators reassign variable values
- Variable selection: randomly select any conflicted variable
- Value selection by min-conflicts heuristic:
  - choose value that violates the fewest constraints
  - i.e., hill-climb with *h(n)* = total number of violated constraints

Example: 4-Queens

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



- Random initial state: solves *n*-queens in ~constant time for arbitrary *n* with high probability (e.g., *n* = 10,000,000)

Summary

- CSPs are a special kind of problem:
  - states defined by values of a fixed set of variables
  - constraints represented as sets of legal assignments to variables
  - usually focus on finding *satisfying solution*
- Backtracking (backjumping) = DFS with one var. 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 – AC) does additional work to detect inconsistencies between unassigned variables
- Can perform local search
  - iterative min-conflicts is usually effective in practice

CSPs vs. Propositional Logic

- Binary variable CSP = propositional logic
  - Prop. logic constraints = (CNF) formulae
  - More compact than set-based CSP constraints

CSPs vs. First-order Logic

- General CSPs equivalent to restricted FOL
  - Need typed variables & domain closure axioms
    - $\forall\, v_1\ NT(v_1) \Rightarrow (v_1 = red \lor v_1 = blue \lor v_1 = yellow)$
  - Constraints = disjuncts of variable (in)equalities
    - $\forall\, v_1, v_2\ NT(v_1) \land SA(v_2) \Rightarrow (v_1 = red \Rightarrow v_2 \neq red)$
  - Satisfiability = ∃ query (and binding extraction)
    - $\exists\, v_1, v_2\ NT(v_1) \land SA(v_2)$

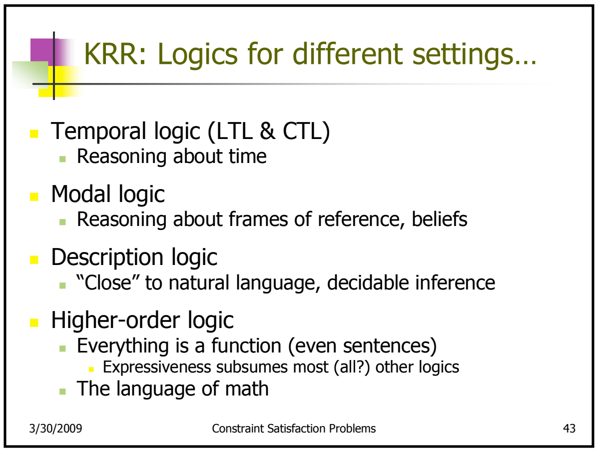
But CSP solvers faster than general FOL inference

Summary for Last 2 Weeks

- An agent receives observations (tell KB)
- Makes "safe" or "high-value" decisions (ask KB)
- Making inferences from a knowledge base (i.e., constraints) crucial to AI
- Various types of knowledge representation…

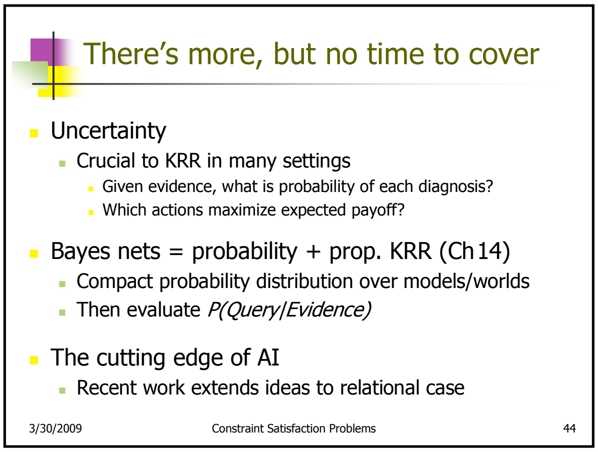
Summary for Last 2 Weeks

- Propositional logic ← Used extensively in hardware verification
  - State consists of boolean variables
  - (Complex) logical formulae as constraints
- First-order logic (FOL) ← More theoretical & complex, but finding applications
  - "Lifted" relational logic
  - Compact constraints over ∞ worlds
- Constraint sat. problems (CSPs) ← Extensive use in industry, iLog top software provider
  - State consists of k-ary, integer (or continuous variables)
  - Expressive midpoint b/w prop. logic and FOL
  - Advanced search techniques building on prop. logic DPLL



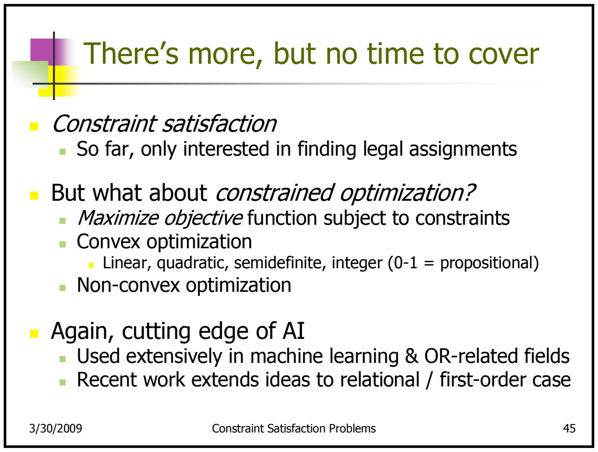
KRR: Logics for different settings…

- Temporal logic (LTL & CTL)
  - Reasoning about time
- Modal logic
  - Reasoning about frames of reference, beliefs
- Description logic
  - "Close" to natural language, decidable inference
- Higher-order logic
  - Everything is a function (even sentences)
    - Expressiveness subsumes most (all?) other logics
  - The language of math



There's more, but no time to cover

- Uncertainty
  - Crucial to KRR in many settings
    - Given evidence, what is probability of each diagnosis?
    - Which actions maximize expected payoff?
- Bayes nets = probability + prop. KRR (Ch 14)
  - Compact probability distribution over models/worlds
  - Then evaluate *P(Query|Evidence)*
- The cutting edge of AI
  - Recent work extends ideas to relational case



There's more, but no time to cover

- *Constraint satisfaction*
  - So far, only interested in finding legal assignments
- But what about *constrained optimization?*
  - *Maximize objective* function subject to constraints
  - Convex optimization
    - Linear, quadratic, semidefinite, integer (0-1 = propositional)
  - Non-convex optimization
- Again, cutting edge of AI
  - Used extensively in machine learning & OR-related fields
  - Recent work extends ideas to relational / first-order case