Artificial Intelligence

Constraint Satisfaction Problems

Outline

- Constraint Satisfaction Problems (CSP)
- Backtracking search for CSPs
- Local search for CSPs

Constraint Satisfaction Problems (CSP)

- CSP vs Standard Search Problem
  - 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
    - Allows useful general-purpose algorithms with more power than standard search algorithms
- More abstract representation format/language
- Allows use of general-purpose algorithms with more power than standard search algorithms

Constraint Satisfaction Problem Definition

- CSP is formally defined by a set of variables, X_1, X_2, …, X_n, and a set of constraints, C_1, C_2, …, C_m.
- Each variable X_i has a non-empty domain D_i of possible values.
- Each constraint Ci involves some subset of the variables and specifies the allowable combinations of values for that subset.
- A state of the problem is defined by an assignment of values to some or all of the variables: $\{X_i = v_i, X_j = v_j, \ldots\}$.
- An assignment that does not violate any constraints is called a consistent or legal assignment.
- A complete assignment is one in which every variable is mentioned.
- A solution to a CSP is a complete and consistent assignment.
- Some CSPs also require a solution that maximizes an objective function.

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
- Notice that many real-world problems involve real-valued variables

Varieties of CSP

- 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., StartJob1 + 5 ≤ StartJob3
- Continuous variables
  - e.g., start/end times for Hubble Space Telescope observations
  - linear constraints solvable in polynomial time by linear programming

Constraint Graph

- Varieties of constraints
  - Unary constraints involve a single variable
  - Binary constraints involve pairs of variables
  - Higher-order constraints involve 3 or more variables
- Binary CSP: each constraint relates at most two variables
- Constraint graph: nodes are variables, arcs are constraints

Standard search formulation (incremental)

- Idea: 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

Properties of CSP

- Problem definition is the same for all CSPs.
- Every solution appears at depth n with n variables —> use depth-first search.
- Path is irrelevant.
- Variable assignments are commutative
- Only need to consider assignments to a single variable at each node —> b = d and there are (dn) leaves

Backtracking Search

- Depth-first search for CSPs with single-variable assignments is called backtracking search

- Backtracking search is the basic uninformed algorithm for CSPs

```
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
```

Improving Backtracking Efficiency

- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
    —> Most constrained variable, i.e., minimum remaining values (MRV) heuristic
    - Tie-breaker among most constrained variables
      —> Most constraining variable
  - In what order should its values be tried?
    —> Least constraining value
  - Can we detect inevitable failure early?

Detecting Failure Early

- Forward Checking
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values
- Constraint propagation
  - Simplest method: repeatedly enforces constraints locally
- 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
  - 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*

Local Search for CSP

- 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

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
- 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
- Iterative min-conflicts is usually effective in practice