constraint satisfaction problems (CSP)

- states: variables $X_i \in D_i$ and some constraints limiting the allowed combinations of values
- an **assignment** is:
    - **consistent** if there are no constraint violations
    - **complete** if there are no unassigned variables
- a **solution** is a *complete, consistent assignment*

restriction graph

- variables are nodes, constraints are edges
- used by algorithms to speed up searching

variables

- discrete
    - finite domains
        - future assumption: each variable's domain has d unique values
    - infinite domains
        - ex: real numbers, character strings
- continuous
    - linear constraints

constraints

- unary: $X_1 \neq 2$
- binary: $X_1 \neq X_2$
    - binary CSP problems: unary/binary constraints
- superior order (≥ 3)
- preferences (soft restrictions)
    - usually represented as costs
    - optimization problems

search algorithms

- initial state: no variables assigned
- successor function: assigning an unassigned variable
- objective testing: no variables unassigned
- solutions: depth n in the tree
- ramification factor: nd, $(n-1)d$, ...
    - $n!d^n$ leaves, d^n possible assignments

backtracking

- basically DFS
- assignments are commutative ($X_1 = 1, X_2 = 2$ is identical to $X_2 = 1, X_1 = 2$)
- one assignment per tree node

- the basic uninformed algorithm

improving backtracking

1. which variable must be assigned?
2. what order do the values need to be checked in?
3. can we detect failure early?
4. can we take advantage of the problem's structure?

minimum-remaining-values

- assign to the variable with the least allowed values (the most constrained)
- "fail-first" heuristic

least-constraining-value

- choose the value least constrained (which excludes the least values from adjacent variables)

forward checking

- idea: update the domain of unassigned variables
- when we select a value for a variable, eliminate it from connected variable's domains

constraint propagation

- forward checking propagates info to unassigned variables, but does not ensure early failure detection
- need to propagate between unassigned variables
- **arc consistency** - the simplest form of propagation, makes each arc consistent
  - $X \rightarrow Y$ consistent $\iff \forall x \in X, \exists $ allowed $y \in Y$
  - if X loses a value, X's neighbors need to be checked
  - fails faster than forward checking
  - can be executed as a preprocessing step or after every assignment
- using constraint propagation implies an increase in execution time
  - 3-consistency (path-consistency), k-consistency
    - *k-consistency* - any consistent assignment of $k-1$ variables can be extended to a k variable instantiation
  - if a CSP problem with n variables is n-consistent, then backtracking is no longer necessary
- a compromise is needed between propagation and searching
  - if propagation takes longer than searching, it's useless
- directional consistency

conflict-directed backjumping (CBJ)

- bkt: return to the previous variable to assign a new value
  - go back *one* level
- when we reach a point of conflict, we can try to identify the cause
  - go back to the source of the problem
- idea: keep a *conflict set* for each variable (updated with variable assignments)

- consider current variable X_i - its conflict set is the set of previously-assigned variables connected to X_i (due to constraints)
- if we can't find a valid assignment for X_i, go back to X_k, the deepest in X_i's conflict set
- update X_k's conflict set: $$CS(X_k) = CS(X_k) \cup CS(X_i) - {X_k} $$

problem structure and decomposition

- recognize **independent subproblems**
  - connected components of the constraint graph
  - solve them separately

tree-structured CSPs

- theorem: if the constraint graph is a tree, it can be solved in $O(nd^2)$ time
  - way better than the normal $O(d^n)$
- a CSP is *directed arc-consistent* for an ordering of the variables $X_1, X_2, ..., X_n \iff X_i$ is arc consistent with $X_j, \forall j > i$
  - choose a root variable
  - order the variables root-to-leaves s.t. the parent of a node precedes it
  - for $j = \overline{n,2}$, apply `MakeArcConsistent(Xj.parent, Xj)`
  - for $j = \overline{1,n}$, assign X_j

almost tree-structured CSPs

- choose a subset S of variables s.t. the constraint graph becomes a free after S is erased (*cutset*)
- for each assignment of the variables in S, erase inconsistent values from the domains of other variables

local search

- hill-climbing and simulated annealing work with complete assignments
- to apply them to CSPs:
  - allow states with unsatisfied constraints
  - have operators for variable re-assignment
- variable selection: a random, conflicting variable
- selecting value: using the *min-conflicts* heuristic
  - choose the value that creates the least conflicts
  - $h(n)$ - the number of violated constraints
- algorithm summary: start from an inconsistent, complete assignment, and gradually fix it to have less and less conflicts

conclusions

- CSPs - states (assignments), constraints
- BKT: DFS with one assignment per node
- heuristics for ordering the variables and selecting their values
- forward-checking prevents assignments which guarantee future failure; arc-consistency further constrains values

- representation using the constraint graph permits problem structure analysis; tree-like CSPs can be solved in linear time
- local-search methods are usually efficient in practice