What is search for?
- single agent
- determenistic actions
- fully observed state
- discrete state space

- planning: sequence of actions
  - the path to the goal is the important thing
  - paths have different costs
  - Heuristics give problem specific guidance

- identification: assignments to variables
  - the goal is important, not the path
  - all paths have same depth (for some formulations)
  - CSP are a specialized kind

CSP

standard search problem:

- state is a "black box": arbitrary data structure
  powerfull but limiting abstraction
- goal test and successor functions could be anything

CSP's: we have a structure

- a state is made of variables X_i with values from a domain D (sometimes D depends on i)
- Goal test is a set of constraints that tell us legal combinations of values for subsets of variables

Constraint graphs

an arc for each constraint, doesn't tell which constraint it is but only that there is one

binary CSP: all constraint relates at most two variables
general purpose CSP algorithms use the graph structure to speed up search

Varietes of CSP

- discrete variables
  - finite domains

    size d means $O(d^n)$ complete assigments

    eg. boolean CSP

  - infinite domains (integers, strings etc.)

    eg. job scheduling, variables start/end times for each job

    linear constraints solvable, non lineor undecidable
  - continuos variables

    eg. start/end times for telescope observation

    linear constraints solvable in polynomial tyme by LP method

Varietes of constraints

- unary costraints: involve a single variable (= reducing domains)

  eg. SA ≠ green
- binary costraints

  eg SA ≠ WA
- Higher-order costraints

Preferences (soft costraints)

often representable by a cost for each variable assigment

gives constrained optimization problems (ignove until Bayes nets)

Solving CSPs

standard formulation

states defined by values assigned so far (partial assigments)

- initial state: empty assigment {}
- successor function: assigns value to variable
- goal state: the current assigment is complete and sadisfies all costraints

  ↳ BFS is bad bcz it has to explore the uhole tree.
  we'll use DFS based approaches

Naive search problem: tries all assignments

Backtracking search

basic uninformed alg. for solving CSPs

we apply to DFS the following improvements

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
   - each step assignment to a single variable
2) check costraints as you go
   i.e. consider only values that do not conflict with previous assignments

Improving backtracking

- Ordering : what variable should we assign next?
  in what order should its values be tried?
- Structure : exploit the problem structure
- filter : can we detect inevitable failure early?

Filtering: Keep track of domains for unassigned variables and cross off bad options

Forward checking

don't pursue values that violate a costraint when added to the existing assignment

doesn't provide early detection for all failures

Only propagates info from assigned to unassigned variables, can't see conflicts between unassigned variables.

Arc Consistency

An arc $x \rightarrow y$ is consistent if and only if $\forall x$ in the tail $\exists y$ in the head which could be assigned without any violation of costraints.

To make an arc consistent we can remove elements from the tail

- if x loses a value neighbours of x must be checked again
- Arc consistency detects failure earlier than forward checking
- can be run as a preprocessor or after each assignment

  downside: really runtime heavy (like A*)

An algorithm for this that only works for binary CSP is AC-3, runtime $O(n^2 d^3)$

limitations:

- can have one solution left
- multiple solutions left
- no solutions left

Ordering

- variable ordering: minimum remaining values (MRV)

  choose variable with less legal values in its domain

  min bcz it will fail fast

- value ordering: Least constraining value

  given a var. choose the lcv

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

  it may take some computation to determine

K-consistency

- 2-consistency: for each pair of nodes, any consistent assignment to one can be extended to the other (the one we saw before)
- K-consistency: for each K-nodes any consistent assignment to K-1 can be extended to the k-th node

strong k consistency: also K-1, K-2 ... consistent

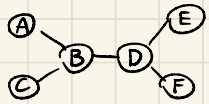
n-consistency means we can solve without backtracking
↑ n nodes in graph

Structure

extreme case: independent sub-problem → identifiable as connected components of constraint graph

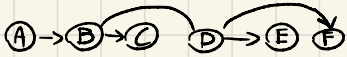
tree structured CSP

if the constraint graph has no loops the CSP can be solved in $O(nd^2)$ (instead of the typical $O(d^n)$)



algorithm:

- order: choose root variable, order parent→children



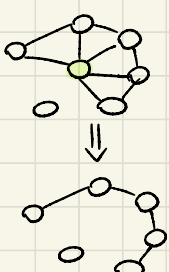
- remove backward: for $i = n : 2$ apply removeInconsistent(parent(X_i), X_i)
  we only have one arch pointing to a node bcz it's a tree

claim: after backward pass, all root-to-leaf arcs are consistent

claim: if root-to-leaf arcs are consistent, forward assignment will not backtrack

Nearly tree-structured CSPs

making the structure a tree by deleting nodes



- conditioning: instantiate a variable, prune it's neighbor's domains (assign)
- cutset conditioning: instantiate in all possible ways a set of variables such that the remaining graph is a tree
  resolve the tree and you found a solution if united with the assignments

cutset of size c runtime $O(d^c \cdot (n-c)d^2)$ very fast for very small c

tree decomposition

idea: make a tree of mega-variables

each mega variable is part of the original CSP

subproblems overlap to ensure consistent solutions

Iterative algorithms for CSPs

typically work with "complete" states, i.e. all variables assigned

N.B. not necessarily legal

to apply to CSPs:

- take an assignment with unsatisfied constraints
- operators (instead of successor) reassign variable values
- no fringe

algorithm:

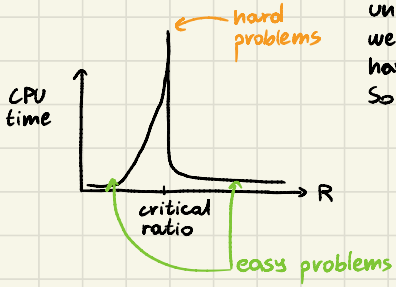
- variable selection: random conflicted variable
- value selection: min-conflicts heuristic

  choose a value that violates the fewest constraints

performance: really fast really often

but the performance depends on the ratio:

$$R = \frac{|\text{constraints}|}{|\text{variables}|}$$

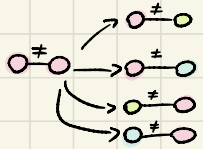


unfortunate because we want to solve hard problems.
So min conflicts is often ineffective in practice

Local search

- tree search keep unexploved alternatives on the fringe (ensures completeness)
- local search : improve a single option until you can't make better (no fringe)

new successor function: local changes



generally much faster and more memory efficient (but incomplete and sub-optimal)

Hill climbing

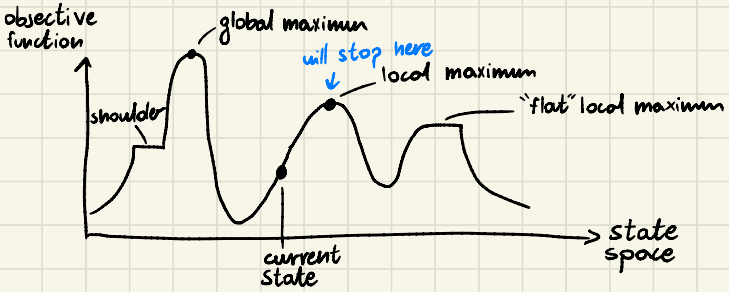
idea: start wherever

repeat: move to the best neighboring state

if no neighbors better than current, quit

not complete, not optimal

what's good? Really easy to apply



Simulated annealing

idea : Escape local maxima by allowing downhill moves

but makes them rarer as you go

it has a guaranteed optimality