NP: Smart Enumeration
Backtracking (Decision problems)

- Remember NP: guess-and-test approach

Backtracking:

- Systematically enumerate configurations + test
- Additionally: try detect dead-ends early plus other speed-ups
3Sat: Backtracking: naive

solve(openVars, assignedVars, clauses)
   IF openVars == empty THEN
      IF check(clauses, assignedVars)
         return assignedVars
   ELSE var <- chooseVar(openVars)
      solve(openVars - var,
         assignedVars + (var <- true),
         clauses)
      solve(openVars - var,
         assignedVars + (var <- false),
         clauses)
3Sat: improved Backtracking

solve(openVars, assignedVars, clauses)
  IF clauses == true THEN return assignedVars
  IF clauses == false THEN return null
  var <- chooseVar(openVars)
  solve(openVars - var,
        assignedVars + (var <- true),
        simplify(clauses, (var <- true)))
  solve(openVars - var,
        assignedVars + (var <- false),
        simplify(clauses, (var <- false)))
Branch-and-Bound

Backtracking analog for optimisation:
- Systematic enumeration plus
- Cutoff if current config will be worse than current best solution
- “anytime” algorithm, approximate if stopped early
Branch-and-Bound for TSP

solveTSP(openCities, currentPath)

IF length(currentPath) > currentBest THEN
    return null

IF openCities == null
    AND length(currentPath) > currentBest THEN
        currentBest <- length(currentPath)

FOR city in openCities DO
    solveTSP(openCities-city, currentPath+city)
Complete vs. Imcomplete

- Backtracking / branch&Bound are complete, i.e. if you are patient enough, they will terminate either with:
  - an (optimal) solution
  - no solution (and then you know there is no solution)
- Incomplete / heuristic methods might give you some (sub-optimal?) solution quickly, but if there is no solution, they may loop infinitely.
- Also, if they terminate early (max #iterations), and return no solution, then there is no absolute guarantee that there is really no solution.
Construct vs. modify/repair

- **TSP construct**: e.g. “nearest-neighbor” always connect the next closest node
- **TSP modify/repair**: start with a random tour and improve it, e.g. with k-opt moves:
  - Lin Kernighan Heuristics (LKH)
  - Lots of methods: random-restart hill-climbing, simulated annealing, genetic algorithms, tabu search, cross-entropy method, ant-colony optimisation, particle-swarm optimisation, ...
- **3Sat**: random var-assignment, flip vars that minimize number of “conflicts” (WalkSAT)
GSAT procedure for 3SAT

for I = 1 to MAX-TRIES
    T = random truth assignment
    for j = 1 to MAX-FLIPS
        if T satisfies CNF then return T
        Flip any var in T that results in greatest decrease in # of unsatisfied clauses
        (maybe 0 or even negative, I.e. worse)
    return “no solution found”
WalkSAT modification of GSAT

replace “Flip ..” with

With prob P: pick some variable in some unsatisfied clause and flip its truth-value
With prob 1-P: follow GSAT procedure, I.e. make best possible local move

WalkSAT can be a lot faster than GSAT
Why?
Phase transition for Random 3SAT

below Threshold: under-constrained, lots solution
at Threshold: hard, very small number of solutions
above Threshold: over-constrained, no solutions
Phase transition

- below threshold: great for GSAT/WalkSAT
- at threshold: hard, GSAT/WalkSAT may need large number of restarts
- above threshold: GSAT/WalkSAT are incomplete, I.e. cannot prove “no solution”, but complete methods based on backtracking (e.g. Davis-Putnam algorithm) might terminate proving “no solution”