Reducing Queries to SAT

Consider the sentence $\Delta = (A \vee B \vee \neg C) \wedge (\neg A \vee C) \wedge (A \wedge C \wedge \neg D)$
We treat each clause as a constraint, and solve as a CSP
  $\rightarrow$ if $w = \{A = T, B = F, C = T, D = F\}$, then $w \models \Delta$

Thus we can see why SAT is the prototypical NP-Complete problem.
We will now discuss various methods to solve SAT problems

Sat Solvers

There are two ways to solve SAT problems that are considered standard:

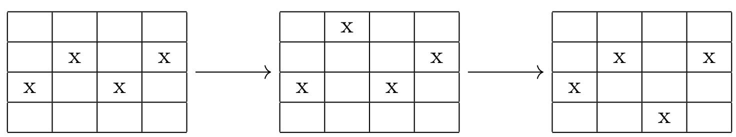
1. Backtrack Search (DFS + Failure Detection)
2. Local Search (Very Fast, Incomplete Search)

In the context of SAT, backtrack search is called DPLL (initials of the 4 creators).
There are a few tools commonly used to make DPLL faster.

1. uses a degree heuristic to determine value ordering
2. can learn from the past via conflict clause learning
3. can perform component analysis to break problems down
4. utilizes random restarts
5. utilizes unit resolution (resolution with a single term clause — linear & incomplete)

Local Search is much simpler and faster, but incomplete.

1. Guess a truth assignment
2. check whether the rule holds.
   YES $\rightarrow$ DONE NO $\rightarrow$ try again



N=5 N=2 N=0

It turns out that the N queens problem is simple for local search;
  This is because the solutions are densely distributed.

Solutions are densely distributed when a problem is **under-constrained**.
Solutions are sparsely distributed when a problem is **over-constrained**.
The hardest problems tend to be at the threshold of the two.

1. Hill Climbing

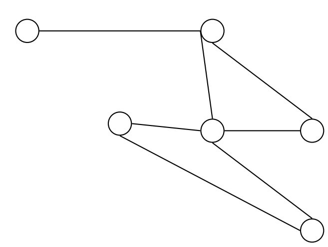
The difficulty of these algorithms comes in the way we determine what truth assignment to use.
There are two standard approaches to this: We can imagine the complete assignments in a graph.

Hill-climbing is choosing the next node by heuristic.
We have two major issues:

- local extrema
  solution: utilize random restarts
- side-moves solution: consecutive side-move limits

Our algorithm therefore takes the form of an embedded loop.
There is no guarantee this visits every world; we have two choices to deal with this:

1. allow the algorithm to complete — complete but can loop infinitely
2. terminate after a time limit — incomplete



neighborhood structure

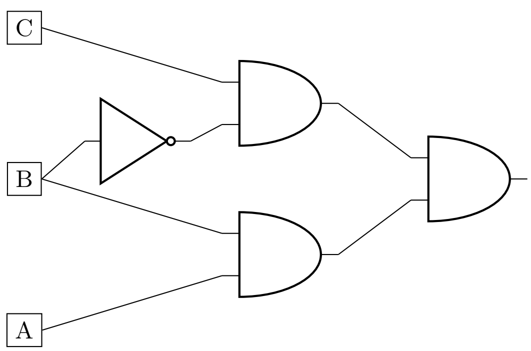
2. Stochastic methods

Stochastic ≡ probabilistic
A common approach is **simulated annealing**
While not at goal:
  pick a neighbor randomly
  if it is better, go there
  else, there is a % chance based on how much worse it is and the depth of the neighbor
End

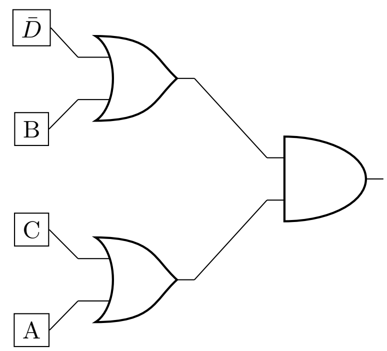
By exploring less when deeper, we can avoid local extrema and have a complete algorithm (provided we allow the algorithm to run for infinite time).

Compiling Sentences To Tractable Circuits

Circuits are more compact than formulas, since circuits can reuse components!



DNF $(A \wedge \neg B) \vee (B \wedge C)$



CNF $(A \vee C) \wedge (B \vee \neg D)$

It therefore makes sense to use them to express complicated logic.

Tractable NNF Circuits

NNF circuits are not inherently tractable, but they are given certain restrictions.

If the sub-circuits of all AND gates in a circuit share no variables, we say it is **decomposable**
This property allows us to solve components of the circuit separately.
  an AND gate is satisfiable iff all its sub-circuits are
  an OR gate is satisfiable iff one of its sub-circuits are
If the circuit is decomposable, The circuit is tractable for SAT.

If the inputs to an OR gate are mutually exclusive, we say it is **deterministic**.
If a circuit is decomposable and deterministic, it is a d-DNNF circuit.
  d-DNNF circuits are tractable for #SAT ("sharp-SAT").
    #SAT counts the number of satisfying assignments in linear time.

If all sub-circuits of an OR gate share all variables, the gate is said to be **smooth**.
  These can be easier to work with, but are logically equivalent to d-DNNFs.

Complexity:

- SAT is NP complete ≡ it is not thought to be solvable in polynomial time
- #SAT is #P complete ≡it involves counting the number of acceptable NP solutions
- Maj-SAT is PP complete ≡ a probabilistic algorithm is run a set # of times for a polynomial time

While these methods are increasingly comple, they all can repeat information over time or space.
We address a solution to this issue in the next lecture.