4 Constraint satisfaction search

Applications of propositional logic: automated reasoning about simple facts

4.1 Question answering

How to answer $A \models \gamma$?

Could implement with resolution:

$$\begin{aligned} A \models \gamma \quad & \text{iff} \quad A \cup \{\neg\gamma\} \text{ unsatisfiable} \\ & \text{iff} \quad A \cup \{\neg\gamma\} \vdash \bot \text{ using resolution} \end{aligned}$$

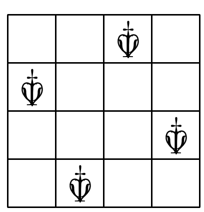
Could also implement with evaluations:

Search for a satisfying assignment for $A \cup \{\neg\gamma\}$
If none found, assert $A \models \gamma$

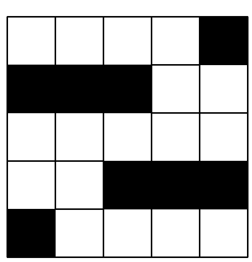
4.2 Can represent constraint satisfaction problems

Examples

1. Pigeonhole principle
2. N-queens problem



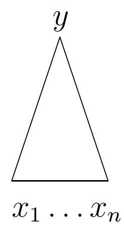
3. Designing crossword puzzles



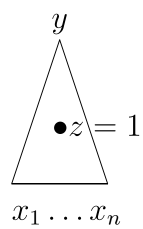
Given dictionary: aardvark
abdomen
⋮
zebra
zygote

4. Circuit testing

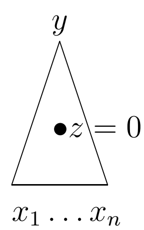
Boolean circuit $\equiv$ propositional formula



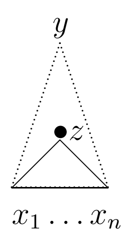
$F_{\mathbf{x}\to y}(\mathbf{x})$ original circuit



$F_{\mathbf{x}\to y|z=1}(\mathbf{x})$ z stuck at 1



$F_{\mathbf{x}\to y|z=0}(\mathbf{x})$ z stuck at 0



$F_{\mathbf{x}\to z}(\mathbf{x})$ internal circuit to z

Can use satisfiability search to design test inputs to identify faults:

– To test if z is stuck at 0

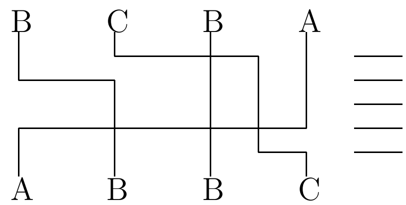
Need $\mathbf{x}$ such that $F_{\mathbf{x}\to z}(\mathbf{x}) = 1$ and $F_{\mathbf{x}\to y|z=1}(\mathbf{x}) \neq F_{\mathbf{x}\to y|z=0}(\mathbf{x})$
so find assignment $\mathbf{x}$ that satisfies prop'n $F_{\mathbf{x}\to z} \wedge (F_{\mathbf{x}\to y|z=1} \oplus F_{\mathbf{x}\to y|z=0})$

– To test if z is stuck at 1

Need $\mathbf{x}$ such that $F_{\mathbf{x}\to z}(\mathbf{x}) = 0$ and $F_{\mathbf{x}\to y|z=1}(\mathbf{x}) \neq F_{\mathbf{x}\to y|z=0}(\mathbf{x})$
so find $\mathbf{x}$ that satisfies proposition $\neg F_{\mathbf{x}\to z} \wedge (F_{\mathbf{x}\to y|z=1} \oplus F_{\mathbf{x}\to y|z=0})$

5. Channel routing (VLSI design)

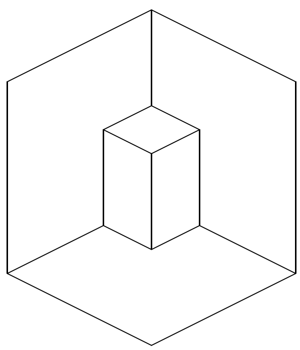
   Connect labelled pins



   Can only go along fixed set of channels

   Must make perpendicular crossings

6. Polyhedral scene interpretation (R&N2 Sect 24.4)



   label junction types (e.g. "innies" vs. "outies")

4.3 Implementing propositional reasoning

Search
– search space of resolution derivations
– search space of truth value assignments to primitive propositions

4.4 Constraint satisfaction search

Searching a finite product space

$$\begin{array}{llll} \text{variables} & p_1 & \ldots & p_n \\ \text{values} & v_{11} & \ldots & v_{n1} \\ & \vdots & & \vdots \\ & v_{1k_1} & \ldots & v_{nk_n} \end{array}$$

Given a set of constraints $\alpha_1, \ldots, \alpha_k$
Looking for assignment $p_1 = v_1 \ldots p_n = v_n$ that satisfies all of the constraints

Systematic strategy 1: Enumerate assignments

Dumb
Exponential time 2^n

But *complete*
– if satisfying assignment exists, guaranteed to find it
– if none exists, then proves non-existence

Constraint satisfaction is NP-complete
Can we be clever about exponential time algorithms?

Systematic strategy 2: Backtrack search

Search partial assignments

$p_1 = v_1 \quad p_2 = v_2 \quad p_3 =? \quad \ldots \quad p_n =?$
$\longrightarrow$

if any constraint becomes violated, backup and try alternative value
if all constraints satisfied, halt immediately

procedure <u>backtrack</u> $(p_j \ldots p_n)$
  for each value v_j of p_j
    $p_j := v_j$
    if no constraint violated
      <u>backtrack</u>$(p_{j+1} \ldots p_n)$
      if succeeds, **return** satisfying assignment
  if all fail, **return** fail

E.g. is $\{\neg a \vee b,\ \neg b \vee c,\ \neg c,\ \neg a\}$ satisfiable?

Enumerate	a	b	c		Backtrack	a	b	c	
	1	1	1	$\times$		1	-	-	$\times$
	1	1	0	$\times$		0	1	1	$\times$
	1	0	1	$\times$		0	1	0	$\times$
	1	0	0	$\times$		0	0	1	$\times$
	0	1	1	$\times$		0	0	0	$\surd$
	0	1	0	$\times$					
	0	0	1	$\times$					
	0	0	0	$\surd$					

Speedup 1: Constraint propagation (forward checking)

Every time a variable is assigned, eliminate values from forward variables if possible

E.g. $\{\neg a \vee b,\ \neg b \vee c,\ \neg c,\ \neg a\}$

a	b	c		
1	1	1	(b, c forced)	$\times$
0	1	1	(c forced)	$\times$
0	0	1	$\times$	
0	0	0	$\surd$	

Speedup 2: Take free moves

If a constraint can be satisfied by a variable assignment, without making other constraints tighter, do so.

Other speedups

- Variable/value ordering heuristics
- More elaborate constraint propagation
- Lemmas: remember resolvents from each backtrack
- Conflict directed backjumping: backtrack as high as possible in search tree given conflict and reasons for forced moves

4.5 Unsystematic search

Unsystematic strategy 1: Random search

Global random: sample independent random assignments

Local random: start with a random assignment, then follow random walk by flipping single variable value

Actually works well if there is a large proportion of satisfying assignments!
But never halts if a solution does not exist

Unsystematic strategy 2: Greedy local search

Use heuristic = # violated constraints

```
while solution not found and not bored
    take random assignment
    while solution not found and walk length bound not exceeded
        evaluate neighboring assignments under heuristic
        step to best neighbor
```

Dumb?

Although unsystematic search runs forever if a solution doesn't exist, it can be astonishingly fast if a solution exists.

E.g. Hard random 3-SAT problems

4.3 times as many constraints as variables
Each constraint involves 3 variables

# vars	backtrack +tricks	heuristic greedy
50	1.5s	0.5s
100	3m	10s
150	10h	25s
200		2m
250		3m
300		13m
350		20m

Speedup 1: Simulated annealing

Take bad moves randomly:

> If neighbor better than current assignment under h, move to neighbor, else move to neighbor with probability $e^{(h(\text{curr})-h(\text{neigh}))/\text{temp}}$

temp is a parameter that controls randomness vs. greediness

Speedup 2: Minimax optimization

Putweights on constraints

repeat

Primal search: update assignment to minimize weighted violation, until stuck
Dual step: update weights to increase weighted violation, until unstuck

until solution found, or bored

# vars	backtrack +tricks	heuristic greedy	minimax
50	1.5s	0.5s	0.001s
100	3m	10s	0.01s
150	10h	25s	0.1s
200		2m	0.25s
250		3m	0.4s
300		13m	1s
350		20m	2.5s

4.6 Readings

Russell and Norvig 2nd Ed., Chapter 5.
Dean, Allen, Aloimonos, Section 4.4.

Mitchell, Selman and Levesque, Hard and easy distributions of SAT problems, *Proceedings AAAI-92*.

Selman, Levesque and Mitchell, A new method for solving hard satisfiability problems, *Proceedings AAAI-92*.

Schuurmans, Southey and Holte, The exponentiated subgradient algorithm for heuristic Boolean programming, *Proceedings IJCAI-01*.