Computability

Wang tiling

Square tiles with a color on each side. A set of such tiles is selected, and copies of the tiles are arranged side by side with matching colors, *without* rotating or reflecting them.

Question: whether it can tile the plane or not; whether this can be done in a periodic pattern.

There exist a finite set of Wang tiles that tiles the plane, but only aperiodically.

Halting Problem

```
def P(x):
    i = 0
    while i < len(x):
        if x[i] == “?”:
            return true
    return false
x = “test”
```

— never halts

Halting Problem: Given a program P and a string x, does the program halt on x?

Theorem (Turing 1940s)**:** The halting problem is not computable, i.e., there is no algorithm to solve it in general.

Proof: Assume by contradiction that there exists some algorithm HALT that solves the halting problem.

Consider this program:

```
def Z(P):
    if HALT(P, P):
        while true:   // if HALT says that P halts on input P, go into infinite loop
            i = 1
    else:             // if HALT says that P goes into infinite loop on input P
        return
```

Does Z halt on input Z?

If YES, then Z(Z) enters an infinite loop, in contradiction.

If NO, then Z(Z) returns after calling HALT, in contradiction.

So there is a contradiction to the existence of HALT. HALT cannot exist.

There are many uncomputable problems, but luckily, most problems we encounter in life are computable.

Nevertheless, most computational problems are only known to have exponential time algorithms.

The examples we saw are exceptions where polynomial time algorithms exist.

Other well known efficient algorithms are linear programming FastFourier transform.

How to deal with difficult problem?

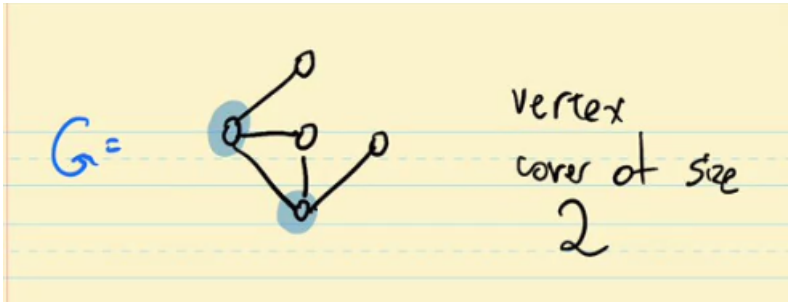
1. More hardware (parallel, GPU, TPU)

   Only takes you that far.
2. Assume more about problem or inputs we have
3. Approximate.

   Approximation algorithm.
4. Heuristic (deep learning).

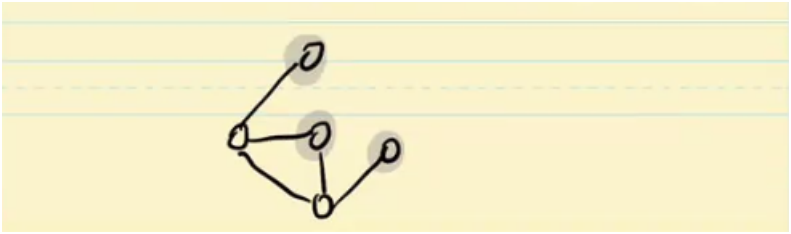
P is the class of problems having polynomial time algorithms (MST, ASAP, ...).

Def. A vertex cover is a set of vertices touching all edges.



VC problem: given G, find smallest vertex cover.

Def. An independent set is a set of vertices without any edge inside it (no edge has both ends inside).



IS problem: given G, find largest independent set.

Claim: VC $\leq$ IS

(if we solve IS, we also solve VC)

Proof: Given a graph G, we find the largest independent set S, and output $V - S$. This is the smallest vertex cover.

Remark, also IS $\leq$ VC.

Another example: 3SAT

$(X \vee Y \vee \overline{Z}) \wedge (X \vee \overline{Y} \vee W) \wedge (\overline{X} \vee \overline{Y} \vee \overline{W}) \wedge (Y \vee W \vee Z) \vee \overline{W} \vee \overline{Z})$

$n = 4$ variables, $m = 5$ clauses

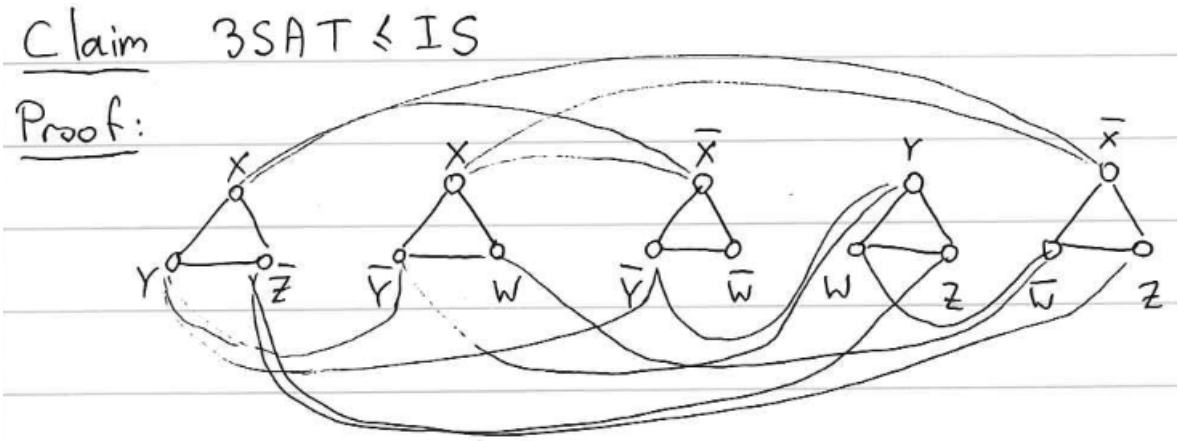
Answer: $X = T, Y = T, Z = F, W = F$; $X = T, Y = T, Z = T, W = F$; $X = T, Y = F, Z = T, W = T$; $X = T, Y = F, Z = T, W = F$

Best algorithm takes exponential time.

Claim: 3SAT $\leq$ IS

(if we solve IS, we also solve 3SAT)

Proof:



This graph has an independent set of size m if and only if the formula is satisfiable.

NP: Yes/No problems where we can efficiently verify a solution.

NP-complete: in NP and as hard as any problem in NP

Theorem [Cook-levin 71] 3SAT is NP-complete.

Corollary. VC & IS are also NP complete.

P: problem we can efficiently solve

P vs. NP question. It is widely believed that $P \neq NP$ but proving it is beyond reach.

Approximating VC

Even though VS is NP-complete (and likely requires exponential time), we can efficiently find an approximate solution that is guaranteed to be not much worse than the optimal.

Approximation algorithm:

1. couple the vertices until we can’t
2. output all coupled vertices

The result is a vertex cover since otherwise we could couple more vertices.

Theorem. The algorithm gives a 2-approximation.

Proof: The optimal solution must include at least one of each couple, whereas we took both.

Remark. Getting better approximation than 2 is a hard problem.