Welcome!

CIS 1890: Solving Hard Problems in Practice

Teaching Staff

Rohan Menezes
Instructor

Charley Cunningham
Instructor

Sahitya Senapathy
Teaching Assistant

Eric Chen
Teaching Assistant

Goals for the course

- **Encounter** vital but provably hard problems
- **Discover** how industry experts tackle problems in practice
- **Experiment** with industrial tools using modern techniques
- **Apply** these tools to your own hard problems

Goals for the course

- **Encounter** vital but provably hard problems
- **Discover** how industry experts tackle problems in practice
- **Experiment** with industrial tools using modern techniques
- **Apply** these tools to your own hard problems

Goals for the course

- **Encounter** vital but provably hard problems
- **Discover** how industry experts tackle problems in practice
- **Experiment** with industrial tools using modern techniques
- **Apply** these tools to your own hard problems

Goals for the course

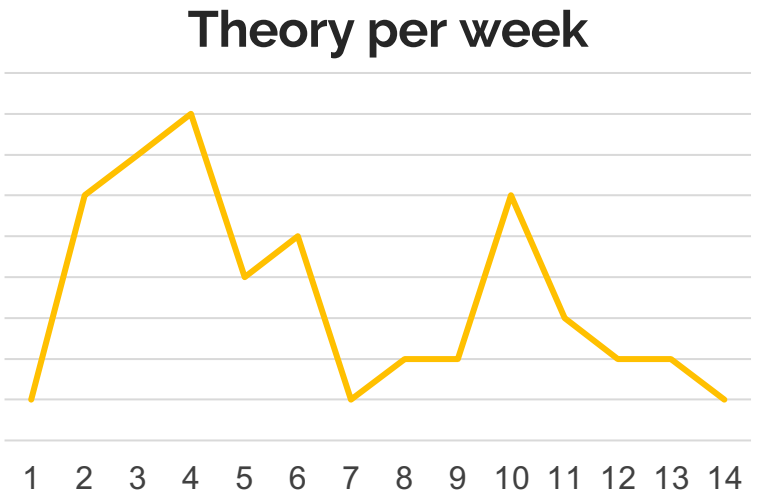
- **Encounter** vital but provably hard problems
- **Discover** how industry experts tackle problems in practice
- **Experiment** with industrial tools using modern techniques
- **Apply** these tools to your own hard problems

Course Logistics

- **Homework:** 4.5 programming assignments
  - HW0 (finger exercises) due next Tuesday before class
  - 1 late submission (48h), can't be used on project
- **Final Project:** solve your own hard problem, in pairs, tons of flexibility
- **Exams:** no
- **Office Hours:** schedule on cis.upenn.edu/~cis1890
- Detailed slides and code posted on website too
- Please make sure you're signed up for Ed and Gradescope

Theory? Practice?

- Interspersed theory & practice
  - “Practice”: applying techniques
  - “Theory”: how techniques work
  - No proofs
- First 4 weeks will have relatively more theory
- Remaining 10 weeks will be more practical



Grading

- Homework: 60%
- Final Project: 30%
- Attendance: 10%

- Final grades:
  don't worry too much about it.

19x Logistics

- This “recitation” is the main lecture for CIS 1890
  - Tuesday 5:15 – 6:45 in Towne 319
- The “lecture” slot is for all 19x courses
  - Covers general SWE stuff: command line, git, etc
  - Only 3 lectures (the first 3 weeks)
  - Can go in person or watch recording on cis.upenn.edu/~cis19x/

Covid Logistics

- If you have a reasonable suspicion that you have Covid, **don't come**
  - Email me **before class** and we'll work something out

Academic Integrity

- Work on assignments individually (except final project)
  - Discussion encouraged, but work should be yours
- OK: high-level discussions
  - “Can you help me understand the DPLL algorithm?”
- OK: low-level discussions
  - “How do I time my program in OR-Tools?”
- Be careful: mid-level discussions
  - Not OK: “How exactly do I write this constraint?”

Lecture 1: Hard Problems

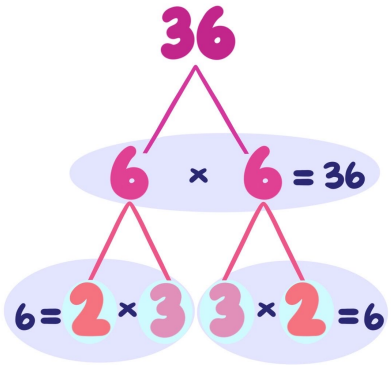
Rohan Menezes rohanmenezes@alumni.upenn.edu

What makes a problem

hard?

CIS 262 in 5 minutes

- **Decision problem**: some question that can be answered YES/NO for any input



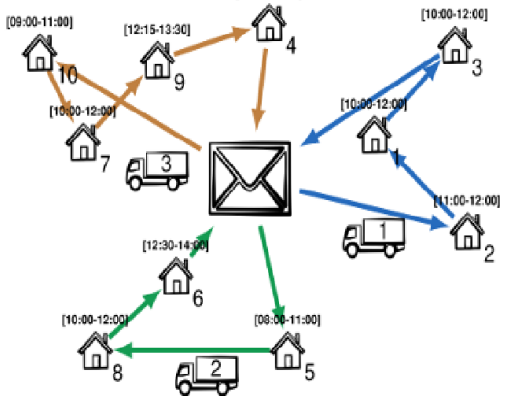
- **Optimization problem**: try to find the "best" out of many feasible solutions

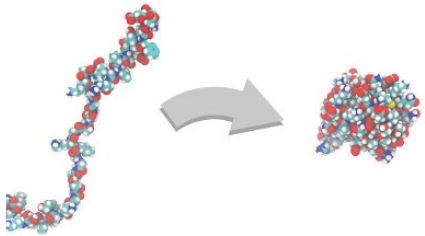
CIS 262 in 5 minutes

- Easy problem: we can solve it quickly for *any* input
  - Quickly: as input size grows, solving time grows at most polynomially
- Difficult problem: can't solve it quickly for every input
  - Solving time might grow exponentially in general

CIS 262 in 5 minutes

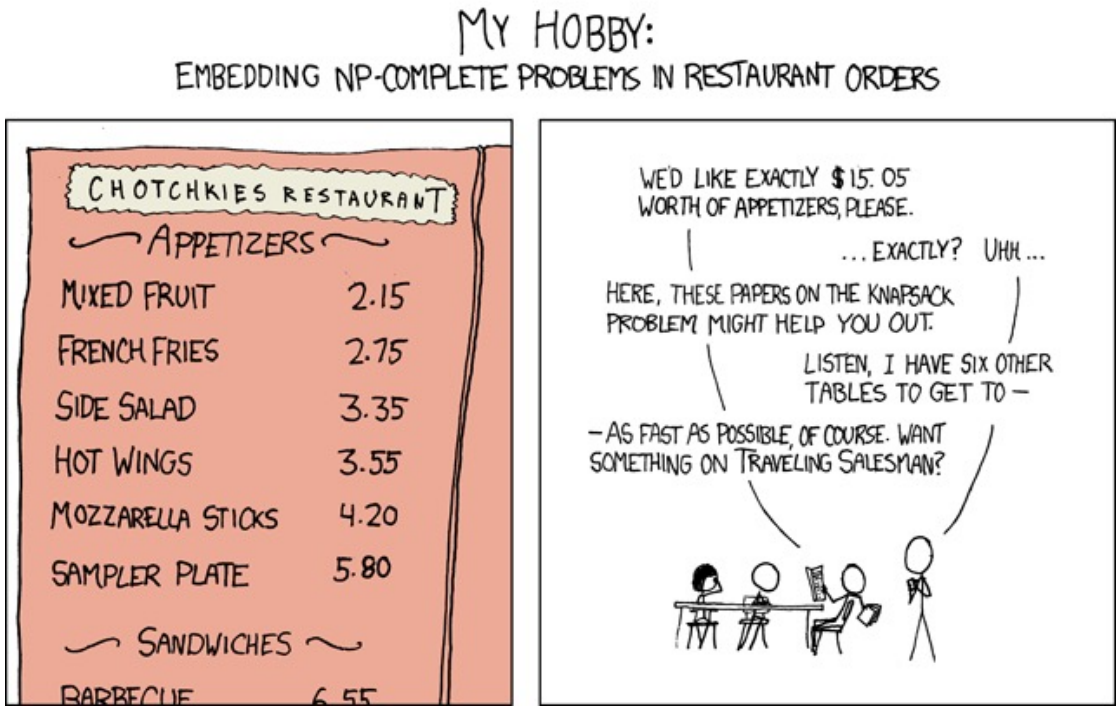
- NP-complete: tons of critical decision problems that turn out to be equivalent





protein folding

CIS 262 in 5 minutes

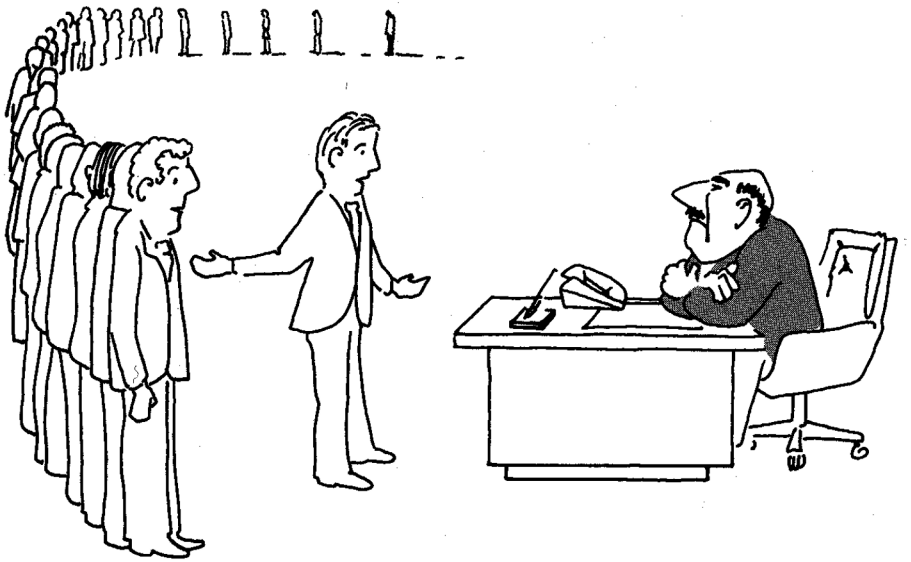


CIS 262 in 5 minutes

- **Probably difficult**: nobody has able to figure out how to solve these problems quickly in 50+ years

CMI

Our final definition of hard



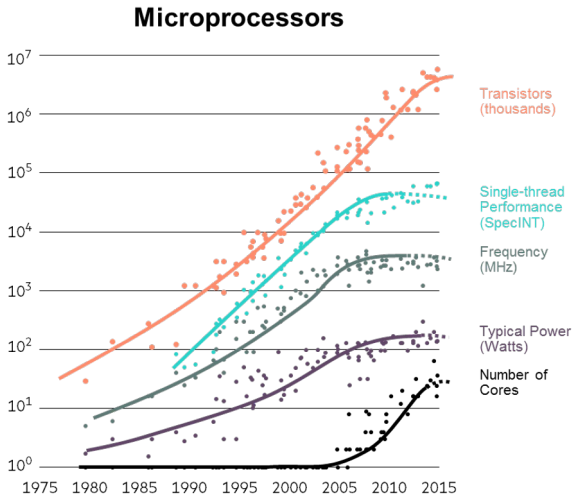
"I can't find an efficient algorithm, but neither can all these famous people."

We'll look at NP-complete problems (both decision and optimization varieties) in this course.

- Decision problems often ask "does there exist some solution?"
- In practice, we don't just want to determine if a solution exists; want to **find** a solution as well.

Does exponential runtime matter?

- **Moore, 1965:** number of transistors per chip **doubles** every two years
- Why bother with solvers? Just wait for faster computers
- **Issue 1:** if problems take $O(2^n)$ time, then even if computer speed doubles, we can only increase n by 1
- **Issue 2:** 55 years later, Moore's law is slowing down



How to solve it, then?

A hopeless challenge?

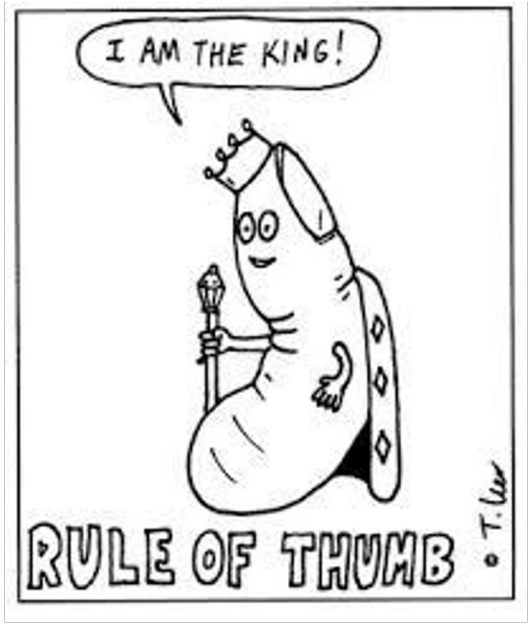
No! Worst case is pessimistic – remember QuickSort

Dealing with hardness

Things we won't focus on:

- Special case algorithms
  - Special cases of NP-complete problems might be in P
- Approximation algorithms
  - For optimization problems, find an "almost optimal" solution
- Monte Carlo algorithms
  - Randomized algorithms with small chance of incorrectness
- Las Vegas algorithms
  - Randomized algorithms with small chance of running slowly

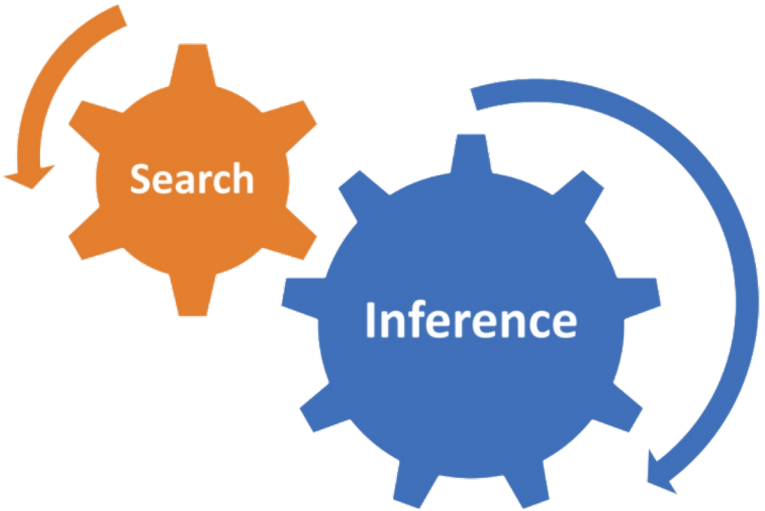
Heuristic Algorithms



Declarative Programming (Modeling)

```
SELECT orders.customer_name
FROM orders
WHERE orders.order_id > 1
AND orders.type == "shipped"
ORDER BY orders.order_id;
```

Search & Inference



The Universal Solver

- 1956-74: early efforts towards **general automated reasoning**
  - **1956:** Samuel's checkers program demonstrated on TV
  - **1959:** Simon, Shaw & Newell's *General Problem Solver*
  - **1964:** Bobrow's natural-language word problem solver
- 1971: introduction of NP-completeness
  - **General idea:** can solve one problem extremely well, and reduce all other problems to that problem

Classic hard problem: SAT

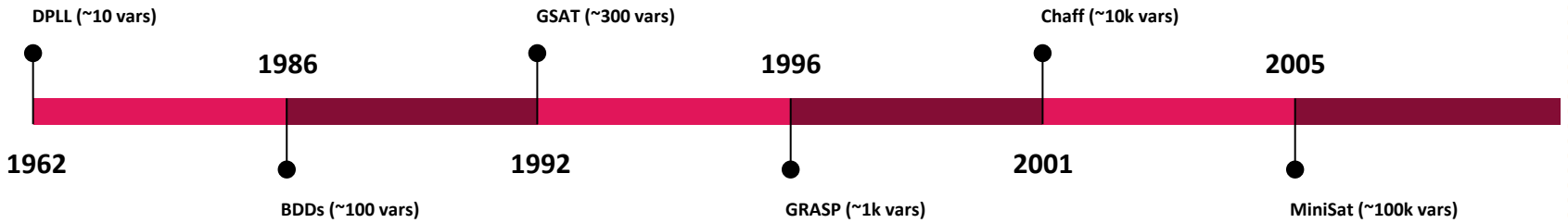
- **Satisfiability Problem:** Given a formula φ of boolean variables, does there exist a truth assignment that makes the entire formula evaluate to True?
  - Many problems can be encoded as SAT instances
  - Assignment: a choice of truth values for each variable
- **Ex:** $(x \vee y) \Rightarrow y$ is satisfiable with $\{x = T; y = T\}$
- **Ex:** $(x \wedge \overline{x})$ is unsatisfiable
- **Cook's Theorem (1971): SAT is NP-complete.**
  - First NP-complete problem!

Modern SAT solvers

- **SAT solvers:** black-boxes to quickly solve huge instances of SAT
- 1962: Davis, Putnam, and Loveland formulate precursor to most modern SAT solvers
- **GRASP** (UMich 1996) and **Chaff** (Princeton 2001): first practical, efficient SAT solvers

- The improvement in the performance of SAT solvers over the past 20 years is *revolutionary*!
  - Better marketing: Deep Solving

Timeline of SAT solvers



- Today: can solve instances with **millions** of variables
  - 1m vars: search space of assignments is $2^{1000000} \approx 9.9 \times 10^{301029}$
  - Age of universe $\approx 4.3 \times 10^{26}$ nanoseconds
- This chart refers to typical SAT instances found in industry applications

SAT terminology

- Assume only logical symbols are AND, OR, NOT
- **Literal:** a boolean variable (x) or its negation ($\overline{x}$)
  - (x) is called a **positive** literal, and ($\overline{x}$) is a **negative** literal
  - "a variable as it appears in a formula"
- **Clause:** a disjunction/OR of literals
  - e.g. $(\overline{x} \vee y \vee z)$

- Note: we would say that $(\overline{x} \vee y) \wedge (x \vee y)$ has 2 variables and 4 literals

Conjunctive Normal Form

- A boolean formula is in **conjunctive normal form (CNF)** if it is a conjunction/AND of clauses (i.e., an AND of ORs)
  - "a CNF" means "a formula in CNF"
- Ex: which of the following are in CNF?
  - $(\overline{x} \vee y \vee z) \wedge (x \Rightarrow w)$
  - $(\overline{x} \wedge y \wedge z) \vee (\overline{y} \wedge z)$
  - $(\overline{x} \vee y \vee z) \wedge (\overline{y} \vee z)$
  - $\overline{x} \vee y \vee z$
  - $x \wedge \overline{x}$

CNF-SAT: a loss of generality?

- It's convenient for SAT solvers to accept formulas in CNF, but what if we need to solve any other non-CNF boolean formula?
- **Every boolean formula φ can be expressed in CNF**
  - Rewrite in terms of $\wedge, \vee, \neg$
  - Apply distributive & DeMorgan's laws until formula is in CNF

CNF-SAT: a loss of generality?

- **Issue:** How large is the resulting CNF formula?
- **Ex:** $(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$

  $((x_1 \wedge x_2) \vee x_3) \wedge ((x_1 \wedge x_2) \vee x_4)$

  $((x_1 \vee x_3) \wedge (x_2 \vee x_3)) \wedge ((x_1 \vee x_4) \wedge (x_2 \vee x_4))$
  - In general, the CNF of $(x_1 \wedge x_2) \vee (x_3 \wedge x_4) \vee \cdots \vee (x_{2n-1} \wedge x_{2n})$ has 2^n clauses
- This **exponential blowup** will make solving arbitrary non-CNF formulas very difficult… can we do better?

The Tseitin Transformation

- Two boolean formulas are **equisatisfiable** if they are either both satisfiable or both unsatisfiable
  - No small *equivalent* CNF, but we only need to find a small *equisatisfiable* CNF
- For each subformula $\psi = \psi_1 \circ \psi_2$, introduce a new variable x_ψ
  - Here, "subformula" includes the formula φ itself, but excludes all literals
  - The operator $\circ$ represents a boolean connective; i.e., $\wedge$ or $\vee$
- Conjoin (AND) together x_φ with $(x_\psi \Leftrightarrow x_{\psi_1} \circ x_{\psi_2})$ for each ψ
- Convert $(x_\psi \Leftrightarrow x_{\psi_1} \circ x_{\psi_2})$ into an *equivalent* CNF
  - Helpful fact: $(x \Rightarrow y)$ is equivalent to $(\overline{x} \vee y)$

The Tseitin Transformation

- **Ex:** Find an equisatisfiable CNF for $\varphi = (1 \wedge 2) \vee 3$.
  - Make new variables: 4, corresponding to $(1 \wedge 2)$; and 5, corresponding to the entire formula φ

$$5 \wedge [4 \Leftrightarrow (1 \wedge 2)] \wedge [5 \Leftrightarrow (4 \vee 3)]$$

$$5 \wedge \big[(4 \Rightarrow (1 \wedge 2)) \wedge (4 \Leftarrow (1 \wedge 2))\big] \wedge \big[(5 \Rightarrow (4 \vee 3)) \wedge (5 \Leftarrow (4 \vee 3))\big]$$

$$5 \wedge (\bar{4} \vee (1 \wedge 2)) \wedge \big(4 \vee \overline{(1 \wedge 2)}\big) \wedge \big(\bar{5} \vee 4 \vee 3\big) \wedge \big(5 \vee \overline{(4 \vee 3)}\big)$$

$$5 \wedge \big(\bar{4} \vee (1 \wedge 2)\big) \wedge (4 \vee \bar{1} \vee \bar{2}) \wedge \big(\bar{5} \vee 4 \vee 3\big) \wedge (5 \vee (\bar{4} \wedge \bar{3}))$$

$$5 \wedge (\bar{4} \vee 1) \wedge (\bar{4} \vee 2) \wedge (4 \vee \bar{1} \vee \bar{2}) \wedge \big(\bar{5} \vee 4 \vee 3\big) \wedge (5 \vee \bar{4}) \wedge (5 \vee \bar{3})$$

  - This is longer than the equivalent CNF, so is it better?

The Tseitin Transformation

- Can write output of Tseitin transformation with scary formula:
- Clearly the formula ρ output by Tseitin's procedure is in CNF

$$\rho = x_\phi \wedge \left(\bigwedge_{\substack{\psi = \psi_1 \circ \psi_2 \\ \text{subformula of } \phi}} \text{CNF}(x_\psi \iff x_{\psi_1} \circ x_{\psi_2}) \right)$$

- Key idea: If φ has n literals, then ρ has $O(n)$ literals.
  - 7 literals in each $\text{CNF}(x_\psi \Leftrightarrow x_{\psi_1} \circ x_{\psi_2})$
  - $n - 1$ subformulas in total
  - $7 * (n - 1) = O(n)$ literals

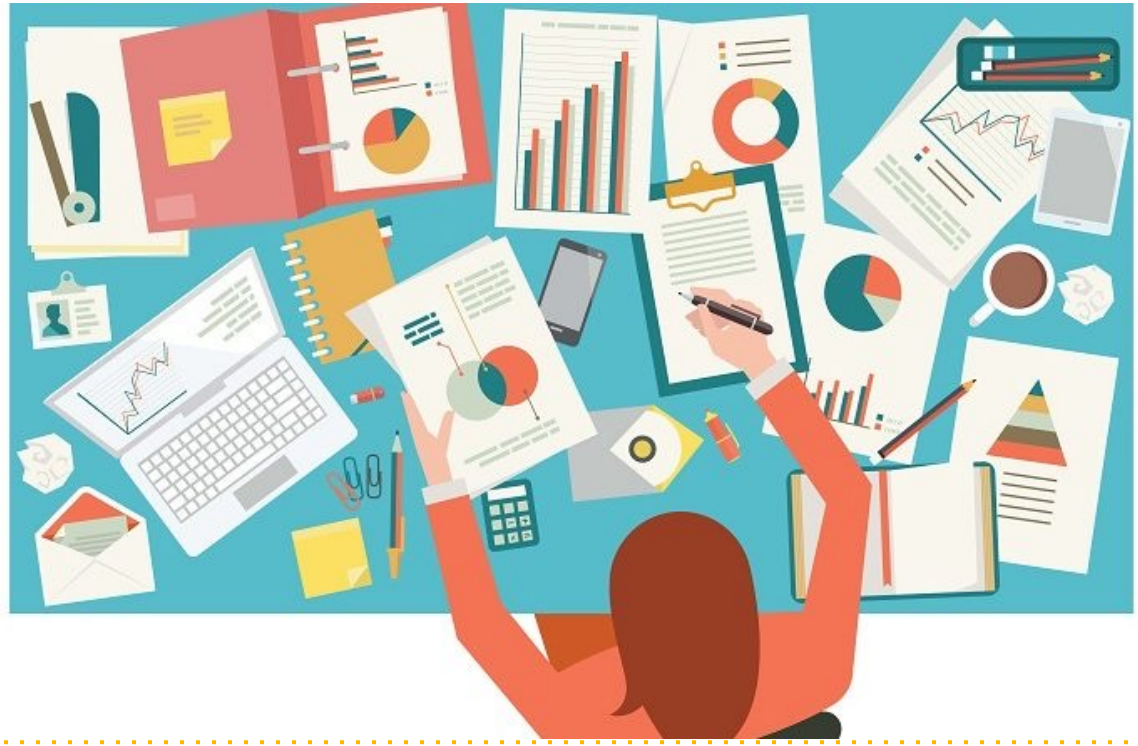
The Tseitin Transformation

- Key idea: φ and ρ are equisatisfiable

$$\rho = x_\phi \wedge \left(\bigwedge_{\substack{\psi = \psi_1 \circ \psi_2 \\ \text{subformula of } \phi}} \text{CNF}(x_\psi \iff x_{\psi_1} \circ x_{\psi_2}) \right)$$

- If ρ has a satisfying assignment β:
  - Can use the same assignment to satisfy φ
  - Just ignore new vars
- Can run SAT solver on ρ and use the result as the answer for φ

A couple past final projects...



Constrained Style Sheets

Kaan Erdogmus & Shriyash Upadhyay

```
c(p == min(h, w))
c(l * 10 == p)
c(2l == l * 2)

box-v(p)-neuv(inset) {
  box-shadow:
      v(inset) v(l)px v(l)px
      v(2l)px #bebebe,
      v(inset) -v(l)px -v(l)px
      v(2l)px #ffffff;
}
```

Optimal Asset Portfolios

Soham Dharmadhikary & Nikhil Kokra

An Optimization Problem

maximize return, minimize covariance and shorting cost

Maximize:

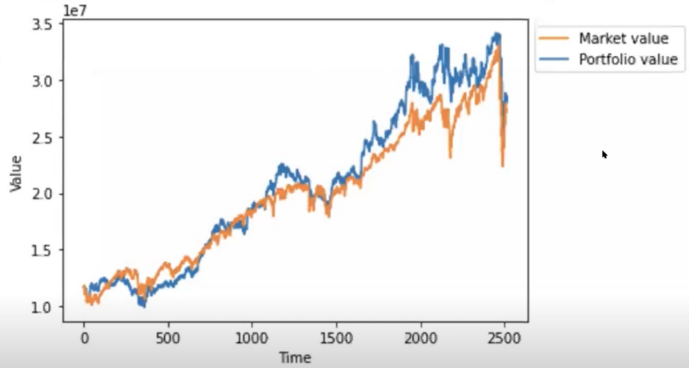
$$(V \cdot \vec{r})^T \vec{w} - A||\vec{w}V|| - \sum_{i=1}^{M} k * 1(w_i < 0)$$

Subject To:

$$\sum_{i=1}^{M} w_i = 1$$

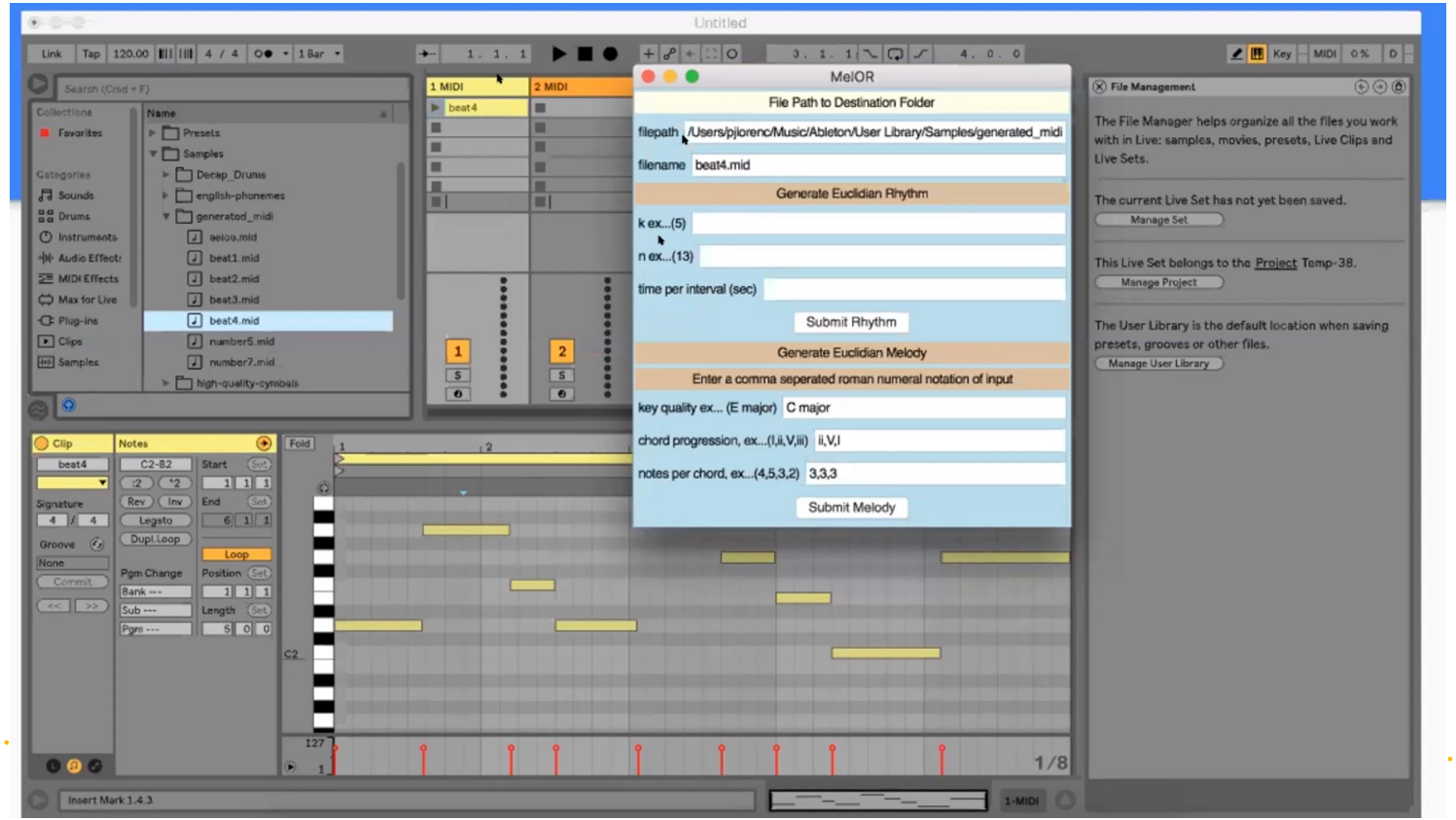
Collecting and Evaluating Results

```
Portfolio return (2010 - 2020): 2.3963905327367754
Portfolio variance (2010 - 2020): 0.01115399257558882
Market return (2010 - 2020): 2.3295003556295586
Market variance (2010 - 2020): 0.010533059982624654
```



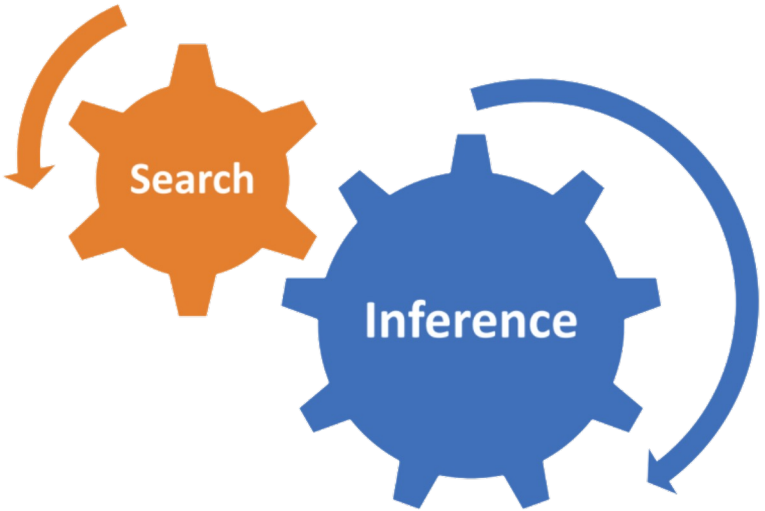
Generative Melody Creator

Paul Lorenc & Leonardo Nerone

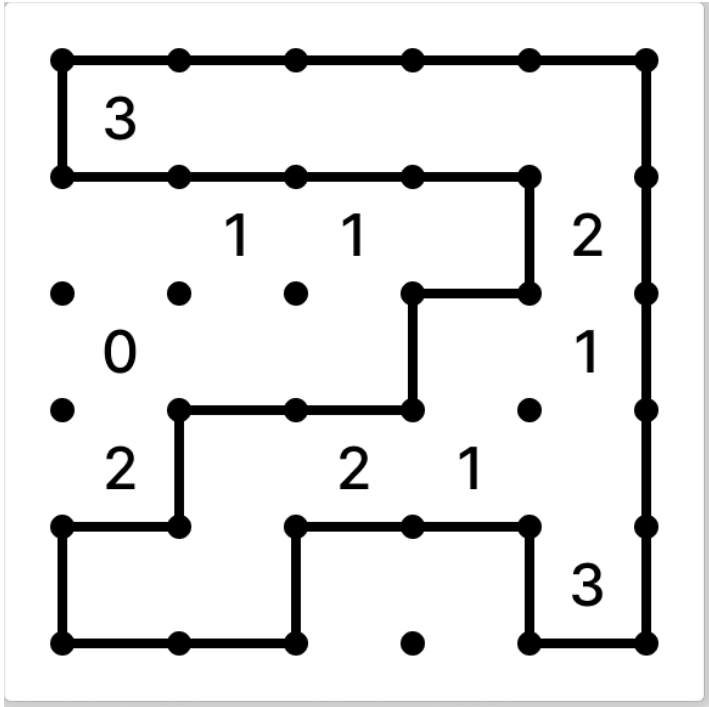


Let's play a game

Search & Inference



Solution



Next week: learn how to **use** SAT solvers ourselves!

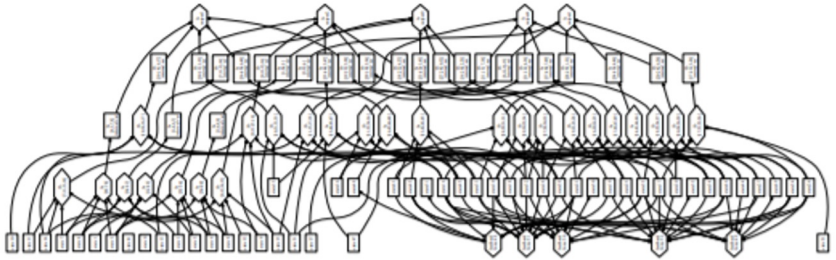
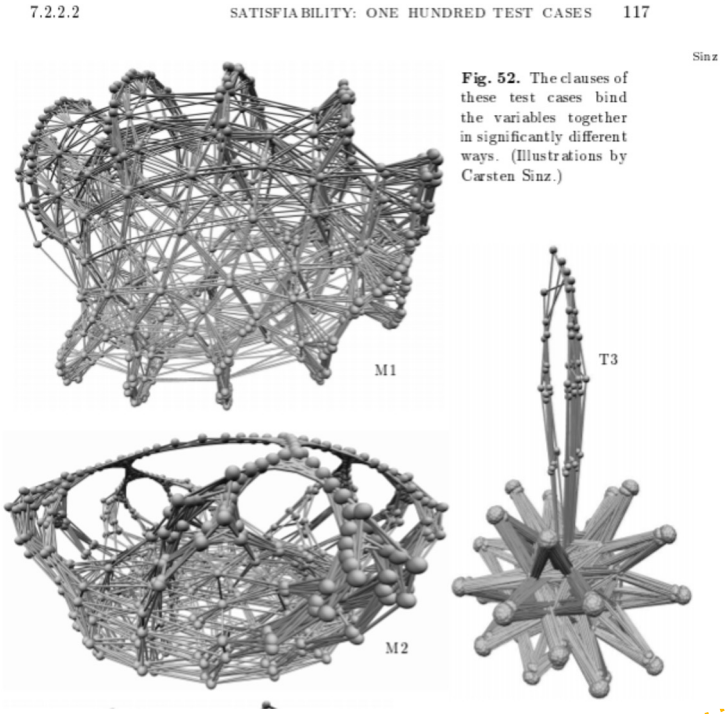


Fig. 4. Clause- and variable-dependency graph of HiTag2. Clause groups are represented as hexagons, and variables as boxes. The known keystream bits are the 5 final filter functions at the top, and the feedback functions are the 5 hexagons at the bottom right.



Our language of choice...

Python!

- Pros:
  - Easy to learn and use
  - Concise
  - Don't need to spend time worrying about low-level details
- Cons:
  - Slow (in practice, C++ is used to develop solver systems)

But I don't know Python...

Don't worry!

- HW0: Finger Exercises will bring you up to speed
- Very easy syntax, low learning curve
- Don't need to be a Python expert to succeed in 189
- If you are comfortable with any OOP language (e.g. Java) you'll be fine