Lecture 2: Solvers & Encoding

Rohan Menezes rohanmenezes@alumni.upenn.edu

Recap

Last week

- Intro to hard problems
- The SAT problem

This week

- Using SAT solvers in Python
- Factors that affect solver runtimes

Recall: SAT Problem

Given a formula φ of boolean variables, does there exist a truth assignment that makes the entire formula evaluate to True?

- Ex: $(x \vee y) \Rightarrow y$ is satisfiable with $\{x = T; y = T\}$
- Ex: $(x \wedge \overline{x})$ is unsatisfiable

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 SAT problem can be converted to a CNF-SAT problem**
  - New problem will only be linearly bigger
  - If curious, google the Tseitin transformation

Notation

- Positive (x) and negative ($\overline{x}$) literals
- Recall that we often consider a clause as a **set of literals**, and a CNF as a **set of clauses**
- Therefore might express CNFs in **compact notation**:

$$\{x\overline{z}, yz\overline{x}\} \quad \simeq \quad (x \vee \overline{z}) \wedge (y \vee z \vee \overline{x})$$

MiniSAT

- **MiniSat** is a minimal, open source SAT solver created by Niklas Eén & Niklas Sörensson
  - *An Extensible SAT-solver* (Eén & Sörensson, 2003)
- Silver medal in SAT Comp. 2005…
- …with only ~600 lines of C++ code!

Niklas Eén

Niklas Sörensson

PicoSAT

- **PicoSAT** is an open source SAT solver created by Prof. Armin Biere
  - *(Q)CompSAT and (Q)PicoSAT at the SAT'06 Race (Biere 2006)*
- Armin Biere: *Handbook of Satisfiability*
- Gold medalist in SAT Competition 2007
- Written in C, but bindings to Python, JS, R, etc.
  - We'll use this: can be easily installed with pip!

Armin Biere

Solving a CNF with PicoSAT

Let's solve the following formula:

$$\varphi = \left\{5, \overline{4}1, \overline{4}2, 4\overline{12}, \overline{5}43, 5\overline{4}, 5\overline{3}\right\}$$

```
cnf = [[5],[-4,1],[-4,2],[4,-1,-2],[-5,4,3],[5,-4],[5,-3]]
pycosat.solve(cnf) # That was easy!



[1, 2, -3, 4, 5]
```

So what's the satisfying assignment for φ?

SAT Encodings

Why do we care about SAT?

Can encode most problems as an instance of CNF-SAT. Let's see how!

Warmup: Boolean Encodings

$x \Rightarrow y$	$\{\overline{x}y\}$
$x \Leftrightarrow y$	$\{\overline{x}y, \overline{y}x\}$
$x \oplus y$	$\{xy, \overline{xy}\}$
if x then y else z	$\{\overline{x}y, xz\}$

First-Order Logic Encodings

$All(x_1, \ldots, x_n)$	$\{x_1, x_2, \ldots, x_n\}$
$ALO(x_1, \ldots, x_n)$	$\{x_1 x_2 \cdots x_n\}$
$AMO(x_1, \ldots, x_n)$	$\{\overline{x}_i \overline{x}_j \mid 0 \leq i < j \leq n\}$
$ExactlyOne(x_1, \ldots, x_n)$	$ALO(\ldots) \wedge AMO(\ldots)$

Complexity of Encoding

$AMO(x_1, \ldots, x_n)$ $\quad\quad$ $\{\overline{x}_i \overline{x}_j \mid 0 \leq i < j \leq n\}$

- Called the **binomial encoding**
- How many clauses in this encoding?
- Other encodings that use fewer clauses, but introduce extra variables

Binary AMO Encoding

$AMO(x_1, \ldots, x_n)$

- Retain original variables $x_1, \ldots, x_n$
- Let $m = \lceil \lg n \rceil$, and introduce new variables $b_1, \ldots, b_m$
  - b_j represents j^{th} bit of T, where x_T is the ($\leq$) one True var
- Clauses:

$$\left\{ \left(x_i \Rightarrow \begin{cases} b_j & \text{if } j^{th} \text{ bit of } i \text{ is } 1 \\ \overline{b_j} & \text{if } j^{th} \text{ bit of } i \text{ is } 0 \end{cases} \right) \middle| \, \forall x_i, b_j \right\}$$

- How many extra variables? How many clauses?

Good Encoding Matters

- The performance of a SAT solver can vary **hugely** depending on the encoding.
- Different encodings work better in different cases: **no "universal best encoding."**
- **Start simple** and be more clever if necessary.

Encoding Graph Coloring

Can we color $G = (V, E)$ with $\leq k$ colors?

- Direct encoding: one boolean variable for each assignment of variable to value
  - Typically the first choice of variables to try, if possible
  - Good enough for many problems, but may need to experiment with smarter options for best results
- For this problem:
  - x_{ic}: if we assign v_i color c

Encoding Graph Coloring

Can we color $\boldsymbol{G = (V, E)}$ with $\leq \boldsymbol{k}$ colors?

- x_{ic}: if we assign v_i color c
- Constraint 1: every vertex has ≥ 1 color

$$\{x_{i1}x_{i2} \ldots x_{ik} \mid \forall\, v_i \in V\}$$

Encoding Graph Coloring

Can we color $\boldsymbol{G = (V, E)}$ with $\leq \boldsymbol{k}$ colors?

- x_{ic}: if we assign v_i color c
- C2: if $(u, v) \in E$, then u, v have different colors

$$\left\{\overline{x_{ic}x_{jc}} \;\middle|\; \forall\, (v_i, v_j) \in E, c \in [1..k]\right\}$$

Encoding Graph Coloring

Can we color $G = (V, E)$ with $\leq k$ colors?

- x_{ic}: if we assign v_i color c
- ~~C3: every vertex has ≤ 1 color~~
- If we can color G with multi-color vertices, of course we can color G with one-color vertices, so this constraint is unnecessary

From Coloring to Min-Coloring

- What if we wanted to find the $\chi(G)$, the minimum number of colors to color G?
- Binary search
  - Check if G can be colored with 1, 2, 4, 8, 16, … colors
  - Stop at smallest value 2^k such that G is 2^k-colorable
  - Binary search for $\chi(G)$ in range $[2^{k-1},\ 2^k]$
- Requires $O(\lg \chi(G))$ runs of solver

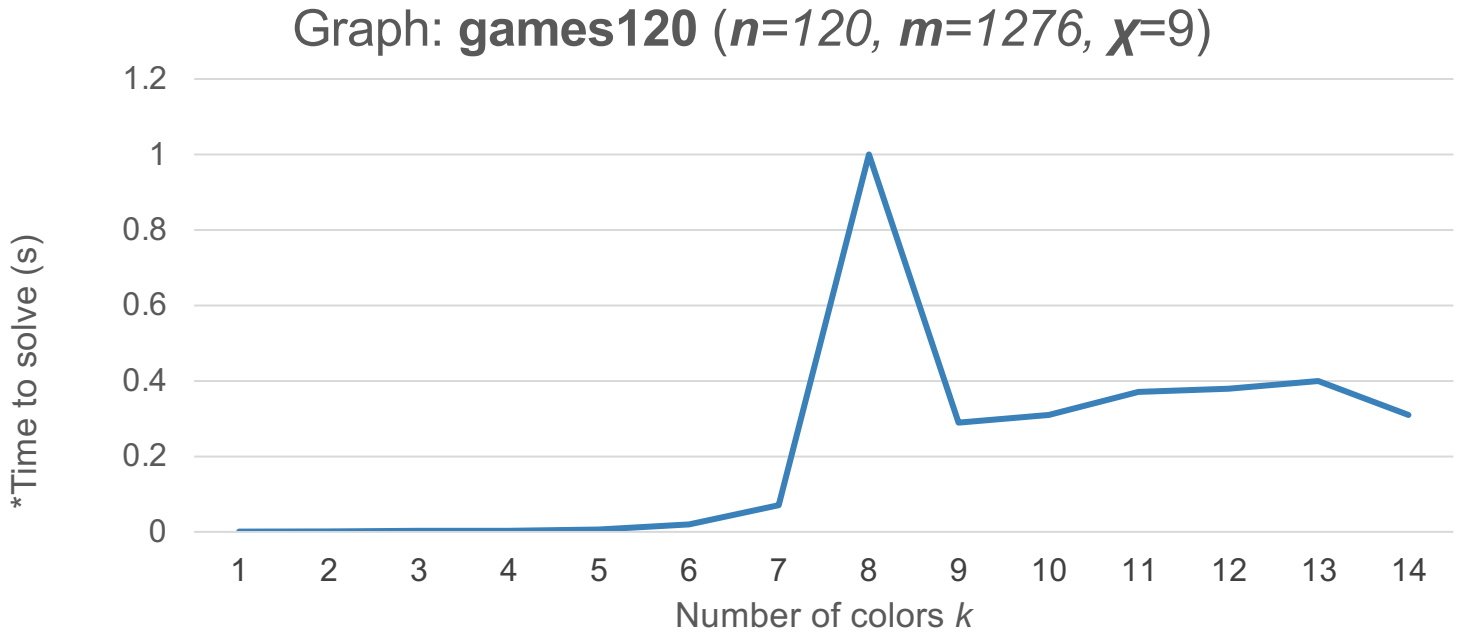
From Coloring to Min-Coloring

- UNSAT can be **way** harder than SAT
  - Finding just one satisfying assignment vs. showing that none exists, in a massive search space
- Instead of starting with $k = 1$ and working up, can start with $k = n$ and work down
  - Even better: start with $k = \Delta + 1$
  - Sometimes better to descend linearly, rather than binary

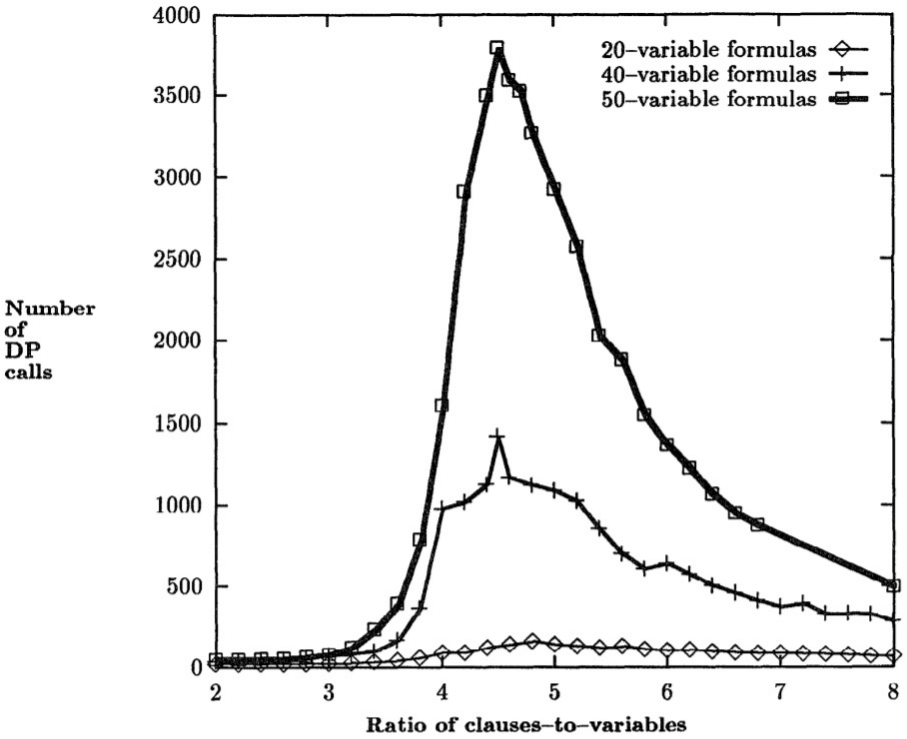
Does Size Matter?

- Weak correlation between number of variables/clauses and “hardness” of formula
- Just because a constraint is **unnecessary** doesn’t mean it’s **useless**
- In fact, adding extra constraints rules out more infeasible options and is typically **critical** to solving quickly

PicoSAT Coloring Benchmarks



*Running on my Dell XPS laptop with 16GB of RAM, in a Jupyter notebook.

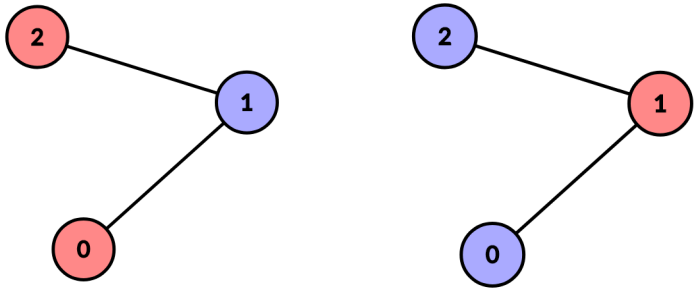


- Relationship between clause-to-variable ratio and solving time on random hard 3-CNF formulas
- **Underconstrained**: easy to find satisfying assignment while avoiding conflicts
- **Overconstrained**: many conflicts quickly rule out unsatisfying assignments
- High difficulty “sweet spot”

Mitchell, Selman & Levesque, 1992

Symmetry Breaking

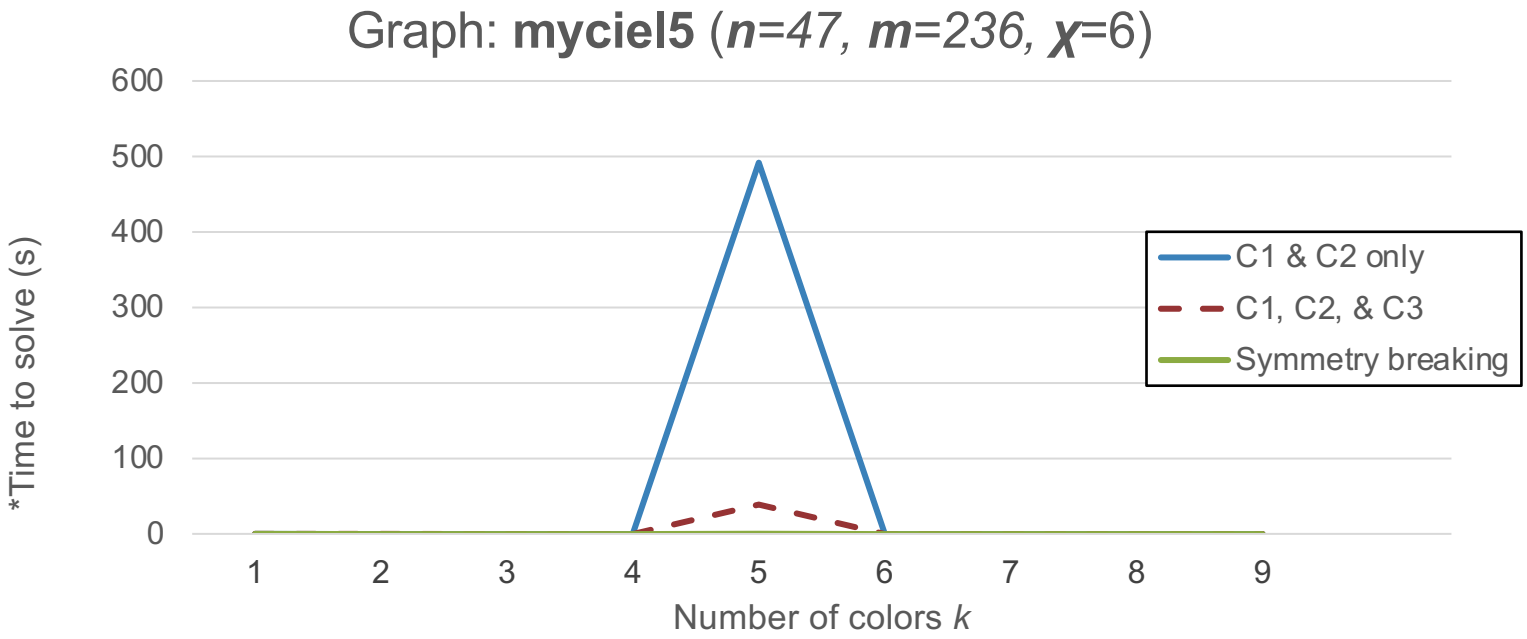
- Solving UNSAT graph coloring problems takes a very long time... why?
- Must rule out every symmetric coloring
- Ex: these equivalent colorings are technically different



Symmetry Breaking

- Key idea: add constraints that rule out equivalent symmetric colorings
- Basic way to do this: pick some vertices (ideally a dense subgraph) and fix their colors

PicoSAT Coloring Benchmarks



*Running on my Dell XPS laptop with 16GB of RAM, in a Jupyter notebook.

Encoding Stable Matchings

We have n men and n women. Each man and woman submits a preference list ranking everyone of the opposite sex (descending).

Goal: find a **matching** of men to women.

A man and woman who both prefer each other to their matched partners are a **blocking pair**.

A matching is **stable** if it has no blocking pairs.

Encoding Stable Matchings

m_{ip}: if man i is matched to p^{th} woman or later on his list

w_{ip}: if woman i is matched to p^{th} man or later on her list

$$[W_1 > W_2]\ M_1 \longleftrightarrow W_1\ [M_1 > M_2]$$
$$[W_1 > W_2]\ M_2 \longleftrightarrow W_2\ [M_1 > M_2]$$

m_{11}	m_{12}	m_{21}	m_{22}	w_{11}	w_{12}	w_{21}	w_{22}
T	F	T	T	T	F	T	T

Encoding Stable Matchings

m_{ip}: if man i is matched to p^{th} woman or later on his list

w_{ip}: if woman i is matched to p^{th} man or later on her list

- C1: every man is matched

$$\{m_{i1} \mid 1 \leq i \leq n\}$$

(plus symmetric constraints for women for this and the following constraints)

Encoding Stable Matchings

m_{ip}: if man i is matched to p^{th} woman or later on his list

w_{ip}: if woman i is matched to p^{th} man or later on her list

- C2: if a man gets his p^{th} or later choice, it's also his $(p-1)^{th}$ or later choice

$$\{m_{ip} \Rightarrow m_{i(p-1)} \mid 1 \leq i \leq n, 2 \leq p \leq n\}$$

Encoding Stable Matchings

m_{ip}: if man i is matched to p^{th} woman or later on his list

w_{ip}: if woman i is matched to p^{th} man or later on her list

- C3: if man i is matched to woman j, then she is matched to him also

$$\left\{ m_{ip} \wedge \overline{m_{i(p+1)}} \Rightarrow w_{jq} \wedge \overline{w_{j(q+1)}} \;\middle|\; 1 \leq i, j \leq n \right\}$$

- p = position of woman j in man i's list
- q = position of man i in woman j's list

Encoding Stable Matchings

m_{ip}: if man i is matched to p^{th} woman or later on his list

w_{ip}: if woman i is matched to p^{th} man or later on her list

- C4: if man i is matched to someone worse than woman j, her match must be better than him

$$\left\{ m_{i(p+1)} \Rightarrow \overline{w_{jq}} \;\middle|\; 1 \leq i, j \leq n \right\}$$

- p = position of woman j in man i's list
- q = position of man i in woman j's list

Why Stable Matchings?

- Gale-Shapley algorithm solves SM problem in linear time. Why use SAT solvers?
- **SMTI:** stable matching problem where preference lists may be incomplete and contain ties
- **SM-C:** stable matching problem with couples
- Our encoding easily generalizes to SMTI, SM-C
- **Theorem:** SMTI and SM-C are NP-complete.

Next Week

- Start learning how SAT solvers work
- Homework 1 due on 2/7
  - Encoding Sudoku into SAT
  - Extra credit challenge!

References

A. Biere, *Handbook of satisfiability*. Amsterdam: IOS Press, 2009.

J. Burkardt, "CNF Files," *John Burkardt's Home Page*. [Online]. Available: https://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html.