COMP36111: Advanced Algorithms I
Lecture 6: Propositional logic satisfiability

Ian Pratt-Hartmann

Room KB2.38: email: ipratt@cs.man.ac.uk

2017–18

Outline

- Let $\mathbf{P} = \{p_1, p_2, \ldots\}$ be a countably infinite set. We call the elements of $\mathbf{P}$ *proposition letters*.
- The set of formulas of propositional logic is defined recursively as follows:
  - every element of $\mathbf{P}$ is a formula
  - if φ_1 and φ_2 are formulas, then so are

$$(\neg\varphi_1), \quad (\varphi_1 \vee \varphi_2), \quad (\varphi_1 \wedge \varphi_2), \quad (\varphi_1 \to \varphi_2)$$

- For example:

$$(\neg(p_1 \to ((\neg p_2) \vee p_3)))$$
$$((p_1 \to (\neg p_1)) \wedge ((\neg p_1) \to p_1))$$

  are formulas
- We omit parentheses for clarity, using standard conventions:

$$\neg(p_1 \to (\neg p_2 \vee p_3))$$
$$(p_1 \to \neg p_1) \wedge (\neg p_1 \to p_1)$$

- An *assignment* is a function $\theta : \mathbf{P} \to \{T, F\}$.
- we extend θ to formulas by setting

$$\begin{aligned}
\theta(\neg\varphi_1) = T &\text{ iff } \varphi_1 = F \\
\theta(\varphi_1 \vee \varphi_2) = T &\text{ iff } \theta(\varphi_1) = T \text{ or } \theta(\varphi_2) = T \\
\theta(\varphi_1 \wedge \varphi_2) = T &\text{ iff } \theta(\varphi_1) = T \text{ and } \theta(\varphi_2) = T \\
\theta(\varphi_1 \to \varphi_2) = T &\text{ iff } \theta(\varphi_1) = F \text{ or } \theta(\varphi_2) = T
\end{aligned}$$

- A formula φ is *satisfiable* if there exists an assignment θ such that $\theta(\varphi) = T$.
- For example,

$$\neg(p_1 \to (\neg p_2 \vee p_3))$$

is satisfiable, but

$$(p_1 \to \neg p_1) \wedge (\neg p_1 \to p_1)$$

is not

- We then have the problem:

  PROPOSITIONAL SAT
  Given: a propositional logic formula φ;
  Return: Yes if φ is satisfiable, and No otherwise.

- Later on, we shall be interested in the complexity of the satisfiability problem.

Outline

- It turns out that the following special case is as general as we need.
- A *literal* is an expression p or $\neg p$, where p is an proposition letter
- A clause is an expression $\ell_1 \vee \cdots \vee \ell_k$, where the ℓ_i are literals. (We allow the *empty* disjunction, denoted $\bot$, which contains no literals.)
- Examples of clauses

$$p_1 \vee \neg p_2 \vee p_3$$

$$\neg p_1 \vee \neg p_4 \vee \neg p_7 \vee p_8$$

$$\neg p_{14}$$

$$p_1$$

$$\bot$$

- We extend any assignment θ to literals by setting

$$\theta(\neg p) = \begin{cases} F & \text{if } \theta(p) = T \\ T & \text{otherwise} \end{cases}$$

and to to clauses by setting

$$\theta(\ell_1 \vee \cdots \vee \ell_k) = \begin{cases} T & \text{if } \theta(\ell_i) = T \text{ for some } i \\ F & \text{otherwise} \end{cases}$$

A set of clauses is *satisfiable* if there exists an assignment θ such that $\theta(\gamma) = T$ for all $\gamma \in \Gamma$.

- Thus, the set of clauses

$$\{(p_1 \vee \neg p_2 \vee p_3), (\neg p_1 \vee \neg p_4 \vee \neg p_7 \vee p_8), \neg p_{14}\}$$

  is clearly satisfiable.
- By contrast,

$$\{(p_1 \vee p_2), (p_1 \vee \neg p_2), (\neg p_1 \vee p_2), (\neg p_1 \vee \neg p_2)\}$$

  is clearly unsatisfiable.

- We then have the problem:

> SAT
> Given: a set of clauses Γ;
> Return: Yes if Γ is satisfiable, and No otherwise.

- We are also intersted in the special case where there is a fixed bound on the length of each clause.
- For $k \geq 2$, we have the problem

> k-SAT
> Given: a set of clauses Γ, each with at most k literals;
>
> Return: Yes if Γ is satisfiable, and No otherwise.

Outline

- The *Davis-Putnam* (*-Logemann-Loveland*) algorithm

begin `resolve`(Γ, ℓ)
    for each $\gamma \in \Gamma$
        if γ contains ℓ, remove γ from Γ
        if γ contains $\bar{\ell}$, remove $\bar{\ell}$ from γ

begin `DPLL`(Γ)
    if Γ is empty then return Yes
    if Γ contains the empty clause then return No
    while Γ contains any unit clause ℓ
        remove ℓ from Γ
        $\Gamma = \texttt{resolve}(\Gamma, \ell)$
        if Γ is empty then return Yes
        if Γ contains the empty clause then return No
    let ℓ be the first literal of the first clause of Γ
    if `DPLL`($\Gamma \cup \{\ell\}$) then return Yes
    if `DPLL`($\Gamma \cup \{\bar{\ell}\}$) then return Yes
    return No

- The DLLP algorithm (which is deterministic) can be seen to run in time bounded by $2^{p(n)}$, where p is some fixed polynomial, and n is the total size of Γ.
- It follows that SAT is in $\textsc{ExpTime}$.
- In fact, this algorithm is (close to) the best way of determining propositional clause satisfiability in practice.
- Nevertheless, from the point of view of the complexity classes seen in the last lecture, we can do 'better' ...

- Consider the following non-deterministic algorithm for SAT

```
begin NdSatTest(Γ)
    if Γ contains ⊥ then return No
    while Γ is non-empty
        Select some proposition letter p occurring in Γ
        Either
            Delete every clause containing the literal p
            Delete ¬p from all remaining clauses
        Or
            Delete every clause containing the literal ¬p
            Delete p from all remaining clauses
        if Γ contains ⊥ then return No
    return Yes
```

- Hence, SAT is in $\mathrm{NP}\textsc{Time}$.

- Notice the asymmetry involved in the notion of (non-deterministic) computation:

  M recognizes $L \subseteq \Sigma^*$ just in case, for each string $x \in \Sigma^*$, $x \in L$ if and only if there exists *a terminating run of M on input x.*

- This asymmetry prompts us to define the *complement* classes as follows.

  If $\mathcal{C}$ is a class of languages, then Co-$\mathcal{C}$ is the class of languages L such that $\Sigma^* \setminus L$ is in $\mathcal{C}$, where Σ is the alphabet of L.

- Trivially,

$$\begin{aligned} \textsc{Time}(G) &= \textsc{Co-Time}(G) \\ \textsc{Space}(G) &= \textsc{Co-Space}(G). \end{aligned}$$

- For non-deterministic classes, some of these equations are not known to hold:

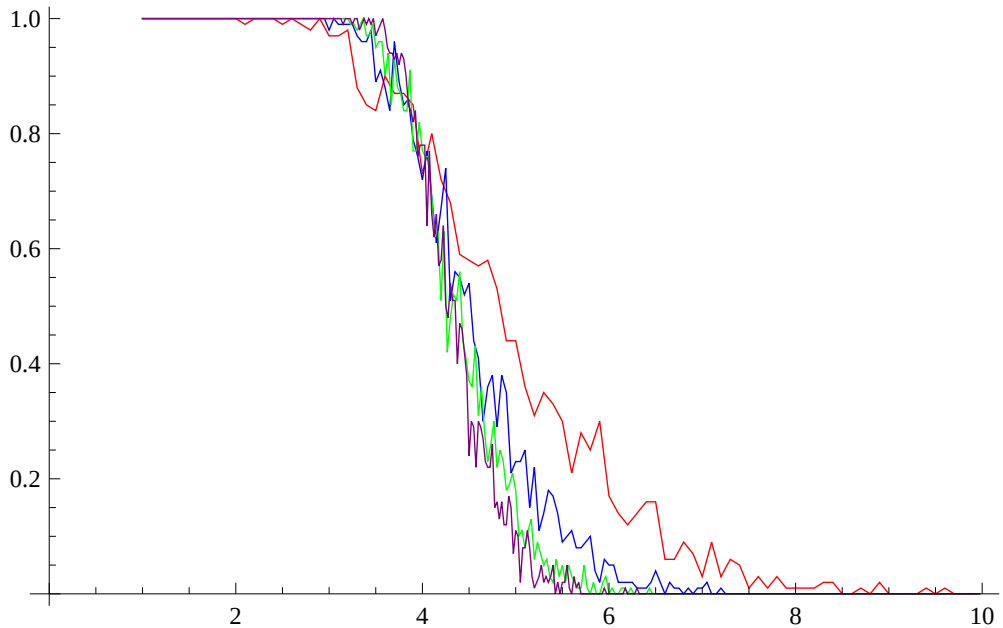
$$\textsc{NPTime} \quad ? = \quad \textsc{Co-NPTime}$$

- But there are some surprises to come ...

Outline

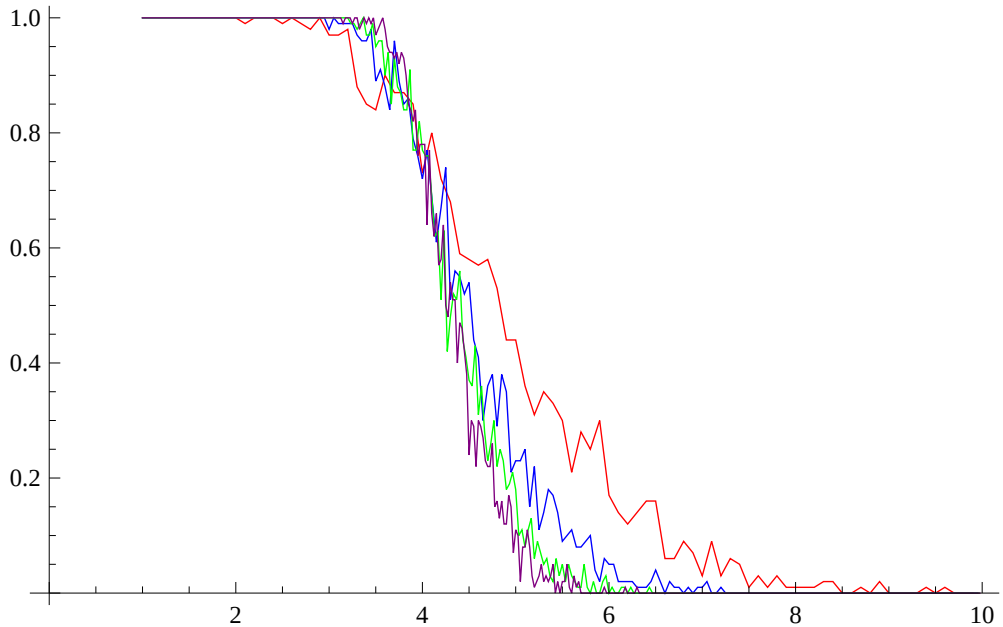
- Suppose we fix integers $m > 0$, $n > 0$ and $k > 1$.
- There is a finite number of (multi-) sets of m k-literal clauses over n proposition letters.
- Some of these will be satisfiable, others not. So *how many* are satisfiable (as a function of m, k and n)?
- Immediately, we see that, for fixed k:
  - if m/n is small, then the probability of satisfiability is high;
  - if m/n is large, then the probability of satisfiability is low.
- But what does the relationship look like in detail?
- In practice, we must solve this problem by generating a *sample* of sets of clauses at random, and then running an algorithm such as DPLL.

- Here is a graph I obtained by running my own implementation on large, randomly generated sets of 3-literal clauses.



- Probability of satisfiability is plotted against m/n where m is number of clauses and n is number or proposition letters
- Graphs are given for $n = 20$, $n = 30$, $n = 40$, $n = 50$.

- The 50% satisfiability point seems to be achieved at around $m/n = 4.3$



- As $n \to \infty$, the 50% threshold value seems to approach a limit; moreover, the transition seems to get steeper with increasing n.
- This phenomenon is known as a *phase transition*: it still has the status of a conjecture.

Outline

- A clause $\ell_1 \vee \cdots \vee \ell_k$ is *Horn* if all but at most one of the literals are negative.
- For example,

$$\neg p_1 \vee p_2, \qquad \neg p_1 \vee \neg p_2 \vee p_3, \qquad p_1, \qquad \neg p_1$$

are all Horn, while

$$p_1 \vee p_2, \qquad p_1 \vee \neg p_2 \vee p_3$$

are not.

- Note that a Horn clause

$$\neg p_1 \vee \cdots \vee \neg p_{k-1} \vee p_k$$

can be written as an implication

$$(p_1 \wedge \cdots \wedge p_{k-1}) \rightarrow p_k.$$

- The problem *Horn-SAT* may now be defined as follows:

  Given: A set of Horn clauses Γ

  Return: Yes if Γ is satisfiable, and No otherwise.

- The following modification of DPLL decides *Horn-SAT*.

```
begin Horn-DPLL(Γ)
    if Γ contains the empty clause then return No
    while Γ contains any unit clause ℓ
        remove ℓ from Γ
        Γ = resolve(Γ, ℓ)
        if Γ contains the empty clause then return No
    return Yes
end Horn-DPLL
```

- Horn-DPLL is easily seen to run in time $O(n^2)$.

- Another special case is 2-SAT
- Terminology: a clause is *Krom* if it contains at most two literals.
- For example,

$$\neg p_1 \vee p_2, \qquad \neg p_1 \vee \neg p_2 \qquad p_1, \qquad \neg p_1$$

  are all Krom, while $\neg p_1 \vee \neg p_2 \vee p_3$ is not.
- The problem *2-SAT* just asks for the satisfiability of Krom clauses.

- Recall that, in the context of propositional logic, a clause is Krom if it contains at most two literals.
- Let us write the opposite of any literal ℓ as $\bar{\ell}$.
- Note that (non-unit) Krom clauses may be regarded as implications:

$$\ell \vee m \equiv \bar{\ell} \rightarrow m.$$

- We have the problem

KROM-SAT
Given: A set Γ of Krom clauses.
Return: Yes if Γ is satisfiable, and No otherwise.

- and its complement

KROM-UNSAT
Given: A set Γ of Krom clauses.
Return: Yes if Γ is unsatisfiable, and No otherwise.

Theorem

The problem KROM-UNSAT is in NLogSpace.

Proof.

Let a set of clauses Γ be given. We may assume there are no unit clauses, since these can be eliminated by unit propoagation. Also, we may assume $\bot \notin \Gamma$ and $\Gamma \neq \emptyset$. So the clauses in Γ are all of the form $\ell \to m$.

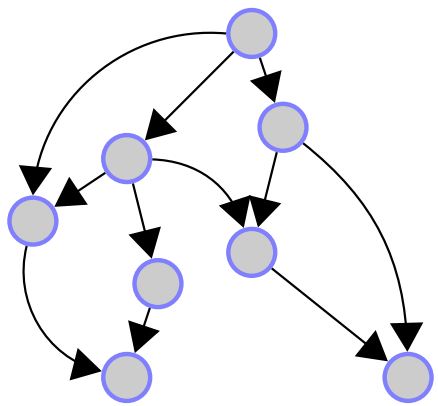
Define a relation $\succeq$ on the literals in Γ by $\ell \succeq m$ iff there is a sequence of literals $\ell = \ell_0, \ldots, \ell_k = m$ $(k \geq 1)$ such that $\ell_i \to \ell_{i+1} \in \Gamma$ for each i $(0 \leq i < k)$. Thus, $\succeq$ is a pre-order (reflexive and transitive). Write $\ell \sim m$ if $\ell \succeq m$ and $m \succeq \ell$.

It suffices to prove that Γ is satisfable iff there exists no literal ℓ such that $\ell \sim \bar{\ell}$, determining $\ell \succeq m$ is essentially a 'graph search'.

Proof.

It is obvious that Γ is unsatisfable if there exists a literal ℓ such that $\ell \sim \bar{\ell}$.

To prove the converse, consider the prartial order (reflexive and transitive and anti-symmetric) induced by $\succeq$ on the equivalence classes of $\sim$

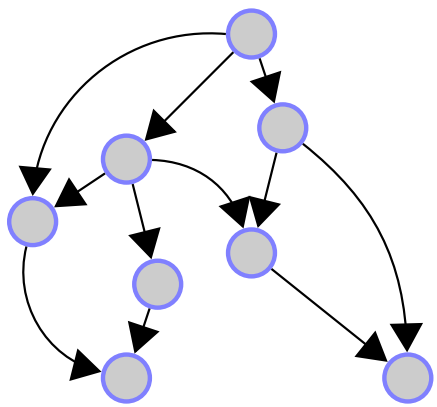


Note that if ℓ and m are in the same equivalence class, then so are $\bar{\ell}$ and $\bar{m}$. So equivalence classes come in 'opposite pairs'. □

Proof.

Suppose that ℓ is never equivalent to $\bar{\ell}$.

Start with some (undecided) equivalence class C lowest in the partial order, and make all its literals true (no contraditions). Make all its literals in the opposite equivalence class, say $\bar{C}$, false. (no contraditions).



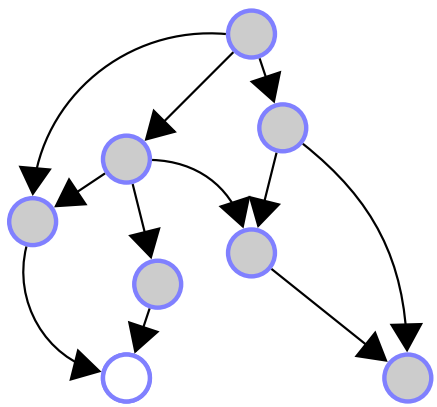
Make all literals false in any D such that $\bar{C}$ is reachable from D in the partial order (no contraditions). Continue until all literals have been given a truth value. Easy to see that Γ is satisfied.

□

Proof.

Suppose that ℓ is never equivalent to $\bar{\ell}$.

Start with some (undecided) equivalence class C lowest in the partial order, and make all its literals true (no contraditions). Make all its literals in the opposite equivalence class, say $\bar{C}$, false. (no contraditions).



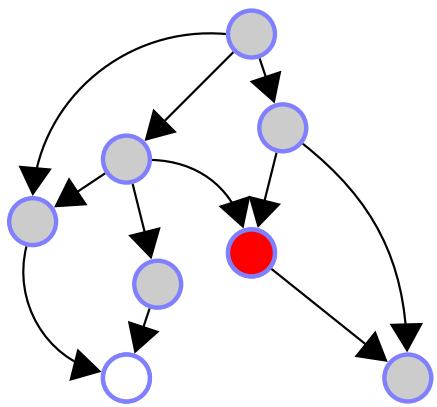
Make all literals false in any D such that $\bar{C}$ is reachable from D in the partial order (no contraditions). Continue until all literals have been given a truth value. Easy to see that Γ is satisfied.

□

Proof.

Suppose that ℓ is never equivalent to $\bar{\ell}$.

Start with some (undecided) equivalence class C lowest in the partial order, and make all its literals true (no contraditions). Make all its literals in the opposite equivalence class, say $\bar{C}$, false. (no contraditions).



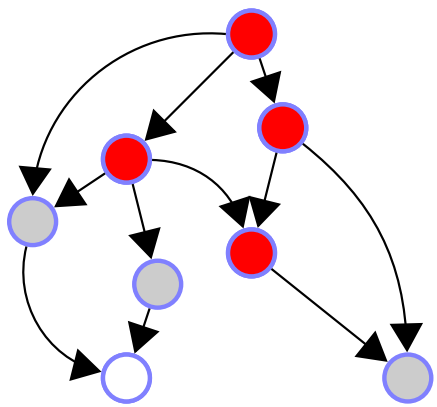
Make all literals false in any D such that $\bar{C}$ is reachable from D in the partial order (no contraditions). Continue until all literals have been given a truth value. Easy to see that Γ is satisfied.

□

Proof.

Suppose that ℓ is never equivalent to $\bar{\ell}$.

Start with some (undecided) equivalence class C lowest in the partial order, and make all its literals true (no contraditions). Make all its literals in the opposite equivalence class, say $\bar{C}$, false. (no contraditions).



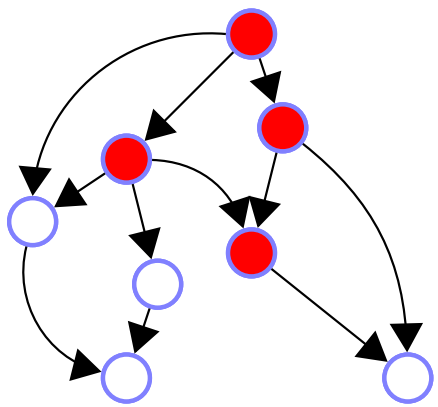
Make all literals false in any D such that $\bar{C}$ is reachable from D in the partial order (no contraditions). Continue until all literals have been given a truth value. Easy to see that Γ is satisfied.

□

Proof.

Suppose that ℓ is never equivalent to $\bar{\ell}$.

Start with some (undecided) equivalence class C lowest in the partial order, and make all its literals true (no contraditions). Make all its literals in the opposite equivalence class, say $\bar{C}$, false. (no contraditions).



Make all literals false in any D such that $\bar{C}$ is reachable from D in the partial order (no contraditions). Continue until all literals have been given a truth value. Easy to see that Γ is satisfied.

□

Outline

- We defined the problems PROPOSITIONAL SAT, SAT, k-SAT ($k \geq 1$) and HORN-SAT.
- We presented the DPLL algorithm for SAT, and saw that it runs in exponential time.
- We showed that SAT is in $\mathrm{NP}\textsc{Time}$.
- We presented a modified version of the DPLL algorithm for Horn-SAT, and saw that it runs in polynomial time. Thus, Horn-SAT is in $\mathrm{P}\textsc{Time}$.
- We presented a non-deterministic logarithmic space algroithm for KROM-UNSAT. Thus KROM-SAT is in $\textsc{Co-NLogSpace}$. (Warning: you will have to wait for several more lectures to hear the end of the story on this.)