5SENG003W - Algorithms, Week 11

Dr. Klaus Draeger

The P vs NP problem: introduction

- One of the biggest open questions in CS
- P is the set of decision problems (i.e. problems with yes/no answers) that can be **solved** in polynomial time
  - That is, solved in $O(n^k)$ for any k
- NP is the set of decision problems for which **a solution can be verified** in polynomial time
- The question is if they are the same, i.e. $P = NP$.
  - Most people believe that the answer is "no".
  - Many incorrect proof attempts both for $P = NP$ and for $P \neq NP$
  - Would have important consequences in either case;
    For example, some encryption algorithms become easy to break if the answer is "yes"

NP-complete problems

- NP-complete problems are an important class
- The “hardest” problems in NP
- If a problem A is NP-complete, then any problem B in NP can be **reduced** to it in polynomial time:
  - From any instance of B, we can compute an instance of A in polynomial time
  - This instance has the same answer as the original
  - So we can solve B by using this translation together with a solver for A
- This means that if we can find a polynomial-time algorithm for **any** NP-complete problem, then $P = NP$

NP-complete problems: SAT

- A very important NP-complete problem is the **satisfiability problem (SAT)**
  - Input: a boolean formula Φ
    Example:
    $(a \vee \neg b) \wedge (b \vee \neg c \vee d) \wedge (\neg a \vee d \vee e) \wedge (\neg a \vee \neg d \vee \neg e) \wedge (b \vee c \vee e)$
  - Question: Are there true/false values for the variables in Φ that make it true?
- Many decision problems can be translated into SAT

Reducing problems to SAT

- Example: 4-colourability
- Given: a graph (V, E)
- Question: Can we **colour** all the vertices
  - using only 4 different colours
  - such that adjacent vertices don't have the same colour?
- One application: colouring countries on a map
  - Neighbouring countries should get different colours
  - Use one vertex per country, edges to represent neighbours

Reducing problems to SAT

We can solve 4-colourability like this:

- Represent the 4 available colours by the numbers 0,1,2,3
- For each vertex $v_j (j = 1, \ldots, n)$, introduce boolean variables a_j, b_j representing the binary representation of its colour
- For each edge $\{v_i, v_j\}$, introduce a formula saying that their colours differ (in at least one bit):
  $(a_i \wedge \neg a_j) \vee (\neg a_i \wedge a_j) \vee (b_i \wedge \neg b_j) \vee (\neg b_i \wedge b_j)$
- Give the conjunction of all these to a SAT solver
  – generally in **conjunctive normal form**:
  $(a_i \vee a_j \vee b_i \vee b_j) \wedge (a_i \vee a_j \vee \neg b_i \vee \neg b_j) \wedge$
  $(\neg a_i \vee \neg a_j \vee b_i \vee b_j) \wedge (\neg a_i \vee \neg a_j \vee \neg b_i \vee \neg b_j)$

Reducing problems to SAT

- Example: Sudoku
  - Need to fill a 9*9 grid with numbers $1 \ldots 9$
  - Some numbers already given
  - No duplicates within the same row/column/3*3 box
- Translation to SAT:
  - Introduce variables $x_{i,j,k}$
    meaning "the number in row i, column j is k"
  - Create a formula starting with the variables corresponding to the givens
  - Add conditions to represent the rules, e.g.
    - $x_{2,5,1} \vee \ldots \vee x_{2,5,9}$ (R2C5 has one of the values $1, \ldots, 9$)
    - $\neg x_{2,5,1} \vee \neg x_{2,5,2}$ (R2C5 cannot have two values)
    - $\neg x_{1,1,3} \vee \neg x_{1,6,3}$ (cannot have two 3s in row 1)

SAT solving

- SAT is NP-complete, so inherently hard.
- Still, there are some very effective solvers
- This is an active field of research
- Let's look at a basic algorithm known as DPLL.
- Consider this example from the beginning:
  $(a \lor \neg b) \land (b \lor \neg c \lor d) \land (\neg a \lor d \lor e) \land (\neg a \lor \neg d \lor \neg e) \land (b \lor c \lor e)$
- For each variable, we can check what happens if it is true or false

SAT solving

- We give names to the example formula and its clauses:
  $\Phi = C_1 \wedge C_3 \wedge C_3 \wedge C_4 \wedge C_5$, where
  $C_1 = a \vee \neg b$
  $C_2 = b \vee \neg c \vee d$
  $C_3 = \neg a \vee d \vee e$
  $C_4 = \neg a \vee \neg d \vee \neg e$
  $C_5 = b \vee c \vee e$
- With no obvious choice, we can
  - try one value for one of the variables
  - check for consequences
  - if this fails we know the variable must have the other value
  - if not, repeat

SAT solving using DPLL

- We give names to the example formula and its clauses:
  $\Phi = C_1 \wedge C_3 \wedge C_3 \wedge C_4 \wedge C_5$, where
  $C_1 = a \vee \neg b$
  $C_2 = b \vee \neg c \vee d$
  $C_3 = \neg a \vee d \vee e$
  $C_4 = \neg a \vee \neg d \vee \neg e$
  $C_5 = b \vee c \vee e$
- Suppose a is false. Then
  - C_3 and C_4 are true
  - C_1 reduces to just $\neg b$
  - So we still need to satisfy $(\neg b) \wedge (b \vee \neg c \vee d) \wedge (b \vee c \vee e)$

SAT solving using DPLL

- If a is false, we stilll need to satisfy
  - $C_1' = \neg b$
  - $C2 = b \vee \neg c \vee d$
  - $C5 = b \vee c \vee e$
- The first clause is a **unit clause.**
  It **forces** b to be false.
- We are left with
  - $C_2' = \neg c \vee d$
  - $C_5' = c \vee e$

SAT solving using DPLL

- After trying $a = \mathit{false}$ we had to also set $b = \mathit{false}$.
- Then we still need to satisfy
  - $C_2' = \neg c \vee d$
  - $C_5' = c \vee e$
- Now d and d are **pure literals**
  i.e. they only occur in one form (negated or un-negated)
  in this case it is the latter for both
- So it is safe to choose d and e to be true
- And we have a solution:
  $a = \mathit{false}, b = \mathit{false}, d = \mathit{true}, e = \mathit{true}$ (c arbitrary)

SAT solving using DPLL

- The naive(brute force) algorithm for SAT would try to guess values for all variables
  – Up to 2^n tries for n variables
- DPLL cannot always avoid this
  – but improves it by only guessing when there is no inference
- If a contradiction is found we need to backtrack
  - We know that the last choice was wrong (assuming the ones before it were right)
  - So we invert it

SAT solving using DPLL

- Suppose we have these clauses:
  - $D_1 = a \vee \neg b \vee e$
  - $D_2 = a \vee d \vee \neg e$
  - $D_3 = \neg a \vee c \vee \neg e$
  - $D_4 = b \vee \neg c$
  - $D_5 = b \vee \neg d$
  - $D_6 = c \vee d$
- There are no unit clauses or pure literals, so we make an initial guess $a = \mathit{false}$. This leaves
  - $D_1' = \neg b \vee e$
  - $D_2' = d \vee \neg e$
  - $D_4 = b \vee \neg c$
  - $D_5 = b \vee \neg d$
  - $D_6 = c \vee d$
- where there are no unit clauses or pure literals.

SAT solving using DPLL

- After guessing $a = false$ we are left with
  - $D_1' = \neg b \vee e$
  - $D_2' = d \vee \neg e$
  - $D_4 = b \vee \neg c$
  - $D_5 = b \vee \neg d$
  - $D_6 = c \vee d$
- The next guess is so we pick $b = false$ leaving
  - $D_2' = d \vee \neg e$
  - $D_4' = \neg c$
  - $D_5' = \neg d$
  - $D_6 = c \vee d$

SAT solving using DPLL

- After guessing $a = \mathit{false}$ and then $b = \mathit{false}$ we have
  - $D_2' = d \vee \neg e$
  - $D_4' = \neg c$
  - $D_5' = \neg d$
  - $D_6 = c \vee d$
- Now c and d must be true to satisfy D_4' and D_5' but this makes D_6 unsatisfiable
- So we backtrack: we have learned that (still assuming $a = \mathit{false}$) b must be *true*.

DPLL algorithm

- Input: a boolean formula $f = C_1 \wedge \ldots \wedge C_n$
- If there are **no more clauses**, return true (we are done)
- If f contains an **empty clause**, return false (unsatisfiable)
- If f contains a **unit clause** v (or $\neg v$), set v to true (or false) by
  - removing all clauses containing v (or $\neg v$)
  - removing $\neg v$ (or v) from all clauses that contain it
- If f contains a **pure literal**, remove all clauses containing it
- Repeat until none of the above apply; then we need to make a choice:
  - Pick a variable v and truth value b
  - Recursively call DPLL on f with v set to b
  - If the result is true, return true
  - Otherwise, set v to $\neg b$ as above and continue