Algorithms and Complexity

Beyond NP-completeness:
Weak NP-completeness, coNP, and PSPACE

Julián Mestre

School of Information Technologies
The University of Sydney

THE UNIVERSITY OF SYDNEY

Subset sum

Input:

- Set of n number $A=\{a_1, a_2, ..., a_n\}$
- Target value T

Question:

- Is there a subset S of A that adds up to exactly T?

What do we know about this problem?

- A special case of Knapsack
- The DP algorithm for knapsack solves subset sum in $O(nT)$ time
- Today, we will show that subset sum is NP-complete

So is P=NP?

Our reduction from vertex cover (VC) to subset sum (SS) produced an instance of SS with $T = \Omega(2^m)$, where m is the number of edges in the VC instance

The DP solution for knapsack we saw in class is a $\Theta(n\,T)$ time algorithm for the subset sum problem.

Putting together the reduction and the DP solution only gives us a $\Omega(2^m)$ time algorithm for VC.

Weakly NP-hard problem

Def.: If an algorithm runs in time polynomial on the magnitudes of the input data, it is said to be a *pseudo-polynomial time* algorithm

Def.: If a problem admits a pseudo-polynomial time algorithm and is NP-hard, it is said to be *weakly NP-hard*

Examples of weakly NP-hard problems: subset sum, knapsack, partition (split set of number into two even sets)

Examples of strongly NP-hard problem: vertex cover, SAT, independent set, etc.

Asymmetric definition of NP

Recall that a problem is in NP if we can verify "yes instances" efficiently. Why not verifying "no instances"?

Def.: coNP is the class of problems that admit a polynomial time verifier for "no instances"

Examples of coNP problems:

- Tautology (Is this formula equivalent to True?)
- Graph expansion

Complementary complexity classes

Def.: For every decision problem X, its complement is $\overline{X}$ where yes-instances in X are labeled no-instances in $\overline{X}$ and vice-versa

Def.: A problem X belongs to coP if $\overline{X}$ belongs to P

Obs.: P = coP

If you can solve a problem efficiently, then you can solve its complement efficiently

What about NP? Is it true that NP = coNP?

- We don't know the answer, but most people think the answer is "no".

NP ∩ coNP

Def.: If problem X belongs to NP ∩ coNP then X is said to have a good characterization

Obs.: P ⊆ NP ∩ coNP

If you can solve a problem efficiently, then you can verify "yes" and "no" instances

Some problems with good characterization

- took some time to show they belong to P (e.g., linear programming, primes)
- are still not known to be solvable in polynomial time (e.g., factorization)

Is it true that P = NP ∩ coNP?

- We don't know the answer, but there is no overwhelming consensus.
- Why should we care?

Relationship of NP, coNP, and P

We show the contrapositive "If P = NP then NP = coNP"

Since P is closed under complement we have

$$X \text{ in } NP \Rightarrow X \text{ in } P \Rightarrow \overline{X} \text{ in } P \Rightarrow \overline{X} \text{ in } NP \Rightarrow X \text{ in } coNP$$

The opposite direction follows by a similar argument

What happens if NP=coNP? Is it true that P = NP?

- We don't know

Thm. If NP ≠ coNP then P ≠ NP

PSPACE

So far, we have been concerned only with time, but other resources are equally important: space, energy, etc.

Def.: PSPACE is the class of problems that can be solved by an algorithm that uses polynomial space. (No time constraints!)

Obs.: $P \subseteq PSPACE$

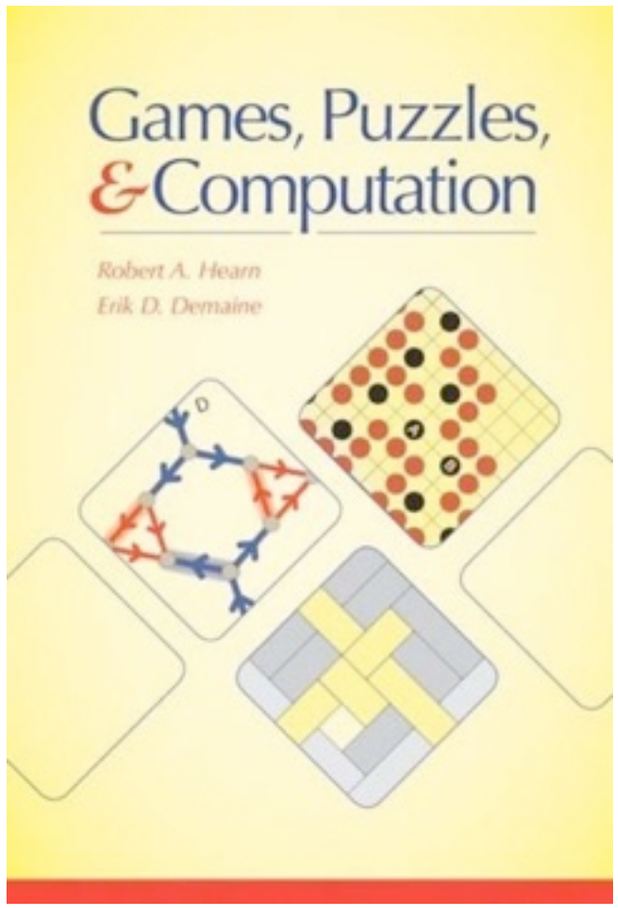
Thm. $NP \subseteq PSPACE$

PSPACE-hard problems

Def.: A problem X is PSPACE-hard if every problem Y in PSPACE is polynomial-time reducible to X; that is, $Y \leq_P X$

Some examples of PSPACE-hard problems:

- Planning: How to achieve a goal by executing a sequence actions with pre-conditions
- Quantified boolean formulas:
  $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \cdots \exists x_n \varphi(x_1, x_2, ..., x_n)$
- Games: Many generalizations of standard two-player board games



Quantified boolean formulas

Input:

- A quantified formula $\exists\, x_1 \forall\, x_2 \exists\, x_3 \forall\, x_4 \cdots \exists\, x_n\ \varphi(x_1, x_2, ..., x_n)$

Question:

- Is this formula true?

Examples:

- Suppose $\exists\, x_1 \forall\, x_2 \exists\, x_3 : (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$?
  The answer is "no": If $x_2 = x_1$, the formula is False, no matter the value of x_3
- Suppose $\exists\, x_1 \forall\, x_2 \exists\, x_3 : (x_1 \vee x_2) \wedge (\neg x_3 \vee x_2) \wedge (\neg x_2 \vee x_3)$?
  The answer is "yes": x_1 = True and $x_3 = x_2$

Quantified boolean formulas

We can think of SAT as the question:

$$\exists\, x_1\ \exists\, x_2 \cdots \exists\, x_n\ \varphi(x_1, x_2, ..., x_n)?$$

Thus, SAT is a special case of QSAT and so QSAT is NP-hard.

It turns out that QSAT is a much harder problem.

Today we just show that QSAT belongs to PSPACE

Thm. QSAT is PSPACE-complete

Geography

Geography is a two-player game often played by children:

- One player starts by naming some city.
- The other player has to name another such that the last letter of the previous city and the first letter of the next city match
- You are not allowed to repeat a city.
- If you cannot name the next city, you lose.

Here is an example using Sydney suburbs:

- Darlington, Newtown, Neutral Bay, Yarramundi, Illawong, Glebe, Enmore, Eveleigh, Haymarket, The Rocks, Summer Hill, Lewisham, etc.

Even if you had the complete list of suburbs, it is not clear how to play optimally!

Generalized geography

Generalized geography (GG) is played on a graph by two players:

- There is a distinguished starting vertex s.
- Player 1, chooses an edge (s,u) and moves to u
- Player 2, choose an edge (u,v) and moves to v
- And so on. If you re-visit a vertex, you lose

The GG problem is to decided whether it is possible to force a win on a given input instance.

Thm.	GG is PSPACE-complete

Bird's eye view