Computational Complexity; slides 15, HT 2022
Search problems, and total search problems

Prof. Paul W. Goldberg (Dept. of Computer Science,
University of Oxford)

HT 2022

Overview

We noted that NP problems have "search" counterparts that are of equal difficulty.

(recall FSAT: find a satisfying assigment)

For search problems having guaranteed solutions, we'll see that a novel classification is needed...

Search versus decision

For NP-complete problems, e.g. SAT, suppose we want to compute a satisfying assignment, not just test for satisfiability. This is at least as challenging as SAT...

Search versus decision

For NP-complete problems, e.g. SAT, suppose we want to compute a satisfying assignment, not just test for satisfiability. This is at least as challenging as SAT...

If we had a SAT-oracle, proceed as follows.
For φ over variables $x_1, \ldots, x_n$, check if φ is satisfiable, if so, try φ with $x_1 \mapsto 0$ alternatively $x_1 \mapsto 1$, then proceed to x_2 etc.

Conclude that in a sense, computing a s.a. is no harder than SAT.

Complexity class **FNP**: functions checkable in poly-time.

Search versus decision

For NP-complete problems, e.g. SAT, suppose we want to compute a satisfying assignment, not just test for satisfiability. This is at least as challenging as SAT...

If we had a SAT-oracle, proceed as follows.
For φ over variables $x_1, \ldots, x_n$, check if φ is satisfiable, if so, try φ with $x_1 \mapsto 0$ alternatively $x_1 \mapsto 1$, then proceed to x_2 etc.

Conclude that in a sense, computing a s.a. is no harder than SAT.

Complexity class **FNP**: functions checkable in poly-time.

- FSAT is FNP-complete (via Cook-Levin)
- So are function versions of other NP-complete problems

Some apparently-hard *total* search problems in **FNP**

- many problems of local optimisation, e.g. LOCAL-MAX-CUT of a weighted graph.
- FACTORING
- NASH: the problem of computing a Nash equilibrium of a game (comes in many versions depending on the structure of the game)
- PIGEONHOLE CIRCUIT:
  Input: a boolean circuit with n input gates and n output gates
  Output: either input vector x mapping to $\mathbf{0}$ or vectors x, x' mapping to the same output
- many other problems associated with "non-constructive" existence results

Search problems as poly-time checkable relations

NP search problem is modelled as a relation $R(\cdot, \cdot)$ where

- $R(x, y)$ is checkable in time polynomial in $|x|$, $|y|$
- input x, find y with $R(x, y)$ (y as certificate)
- *total* search problem: $\forall x \exists y \quad (|y| = poly(|x|), R(x, y))$

SAT: x is boolean formula, y is satisfying bit vector.
Decision version of SAT is polynomial-time equivalent to search for y.

FACTORING: input (the "x" in $R(x, y)$) is number N, output (the "y") is prime factorisation of N. No decision problem!

contrast with "promise problems"

Reducibility among search problems

FP, FNP: search (or, function computation) problems where output of function is computable (resp., checkable) in poly time.

Any NP problem has FNP version "find a certificate".

Definition

Let R and S be search problems in FNP. We say that R (many-one) reduces to S, if there exist polynomial-time computable functions f, g such that

$$(f(x), y) \in S \implies (x, g(x, y)) \in R.$$

Observation: If S is polynomial-time solvable, then so is R. We say that two problems R and S are (polynomial-time) equivalent, if R reduces to S and S reduces to R.

Theorem: FSAT, the problem of finding a s.a. of a boolean formula, is FNP-complete.

Example

(To help motivate/understand that definition of reducibility)

Consider 2 versions of FACTORING: one using base-10 numbers, and the other version using base-2 numbers. Intuitively, these two problems have the same difficulty: there is a fast algorithm to factor in base 2, if and only if there is a fast algorithm to factor in base 10.

In trying to make that intuition mathematically precise, we get the definition of the previous slide.

TFNP

TFNP: "Total" function computation problems in NP

As we shall see, it looks like we really do need to introduce a new complexity class, in fact a collection of complexity classes...

Contrast with "promise problems", e.g. $\textsc{Promise SAT}$: SAT-instances where you've been promised there is a satisfying assignment. But such a promise isn't directly checkable.

Some total search problems seem hard. (F)NP-hard?

Theorem

There is an FNP-complete problem in TFNP if and only if NP=co-NP.

Proof: "if": if NP=co-NP, then any FNP-complete problem is in TFNP (which is F(NP$\cap$co-NP)).

"only if": Suppose X$\in$TFNP is FNP-complete, and R is the binary relation for X.

Consider problem FSAT (given formula φ, find a satisfying assignment.) We have $\mathrm{FSAT}\leq_p$X.

Any unsatisfiable φ would get a certificate of unsatisfiability, namely the string y with $(f(\varphi), y) \in R$ and $g(y) =$"no" (or generally, anything other than a satisfying assignment).

N. Megiddo and C.H. Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, **81**(2) pp. 317–324 (1991).

So what can we say about the hardness of FACTORING, and others?

FACTORING (for example) cannot be NP-hard unless NP = co-NP. Unlikely! So FACTORING is in strong sense "NP-intermediate".

So what can we say about the hardness of FACTORING, and others?

FACTORING (for example) cannot be NP-hard unless NP $=$ co-NP. Unlikely! So FACTORING is in strong sense "NP-intermediate".

$\rightsquigarrow$ task of classifying "hard" NP total search problems. FACTORING and PIGEONHOLE CIRCUIT are important in cryptography; other important problems include local optimisation

OK can we have, say, FACTORING is TFNP-complete?

So what can we say about the hardness of FACTORING, and others?

FACTORING (for example) cannot be NP-hard unless NP $=$ co-NP. Unlikely! So FACTORING is in strong sense "NP-intermediate".

$\rightsquigarrow$ task of classifying "hard" NP total search problems. FACTORING and PIGEONHOLE CIRCUIT are important in cryptography; other important problems include local optimisation

OK can we have, say, FACTORING is TFNP-complete? Good question! TFNP-completeness is as much as we can hope for, hardness-wise

TFNP doesn't (seem to) have complete problems (which needs syntactic description of "fully general" TFNP problem). (Similarly, RP, BPP, NP$\cap$co-NP don't have complete problems)

Try to describe "generic" problem/language X in NP$\cap$co-NP as pair of NTMs that accept X and $\overline{X}$: what goes wrong?

FACTORING, NASH, and the others, as "NP-intermediate"

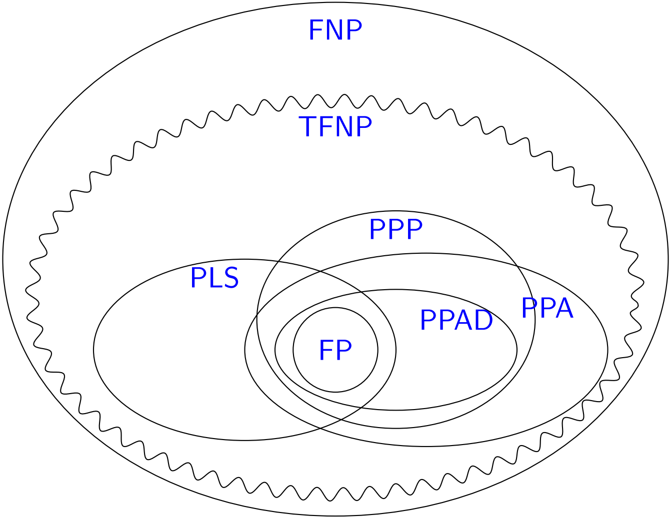
Advantage of (problems arising in) Ladner's theorem: you just have to believe P$\neq$NP, to have NP-intermediate. For us, we have to believe that FACTORING (say) is not in FP, also that NP$\neq$co-NP.

Disadvantage of Ladner's theorem: the NP-intermediate problems are unnatural (did not arise independently of Ladner's thm; problem definitions involve TMs/circuits)

Next: subclasses of TFNP that have complete problems

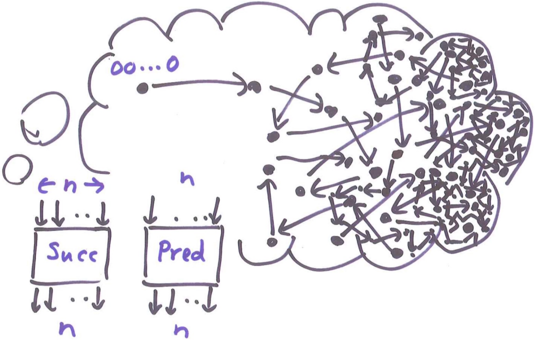
General idea: define classes in terms of "non-constructive" existence principles

Some syntactic classes



Johnson, Papadimitriou, and Yannakakis. How easy is local search? *JCSS*, 1988.
C.H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *JCSS*, 1994.

PPAD "given a source in a digraph having in/outdegree at most 1, there's another degree-1 vertex"



The END-OF-LINE problem

given Boolean circuits S, P with n input bits and n output bits and such that $P(0) = 0 \neq S(0)$, find x such that $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0$.

PPAD

A problem X belongs to PPAD if $X \leq_P$ End-of-Line.

X is PPAD-complete if in addition, End-of-Line$\leq_P X$

Work by myself and others: Nash equilibrium computation is PPAD-complete

...which is taken to indicate it's a hard problem! Let's see why we believe "PPAD is hard"

PPA, PPP

PPA: Like PPAD, but the "implicit" graph is undirected.

The LEAF problem

given boolean circuit C with n inputs, $2n$ outputs. regard input as one of 2^n vertices, output as 2 neighbouring vertices.
If $\mathbf{0}$ has degree 1, find some other degree-1 vertex.

PPP ("polynomial pigeonhole principle"): defined in terms of:

The PIGEONHOLE CIRCUIT problem

given boolean circuit C with n inputs, n outputs.
Find *either* a bit-string that is mapped to $\mathbf{0}$, *or* two bitstrings that are mapped to the same bit-string

We have

- END-OF-LINE $\leq_p$ LEAF (hence PPAD$\subseteq$PPA)
- END-OF-LINE $\leq_p$ PIGEONHOLE CIRCUIT

PPAD is a subset of PPP

END-OF-LINE reduces to PIGEONHOLE CIRCUIT:

Given S, P, circuits representing an END OF LINE instance, build a circuit C_{PPP} that does the following:

C_{PPP} uses S, P to identify any neighbours of a vertex v in the END-OF-LINE graph, then

- If v has no outgoing edge in the END-OF-LINE graph, C_{PPP} maps v to itself.
  (so all isolated vertices are mapped to themselves)
- Otherwise, let (v, w) be a directed edge in the END-OF-LINE graph.
  C_{PPP} maps v to w

Evidence of hardness

- Failure to find poly-time algorithms for most of these problems, indeed even sub-exponential algorithms.
- cryptographic hardness

- **Separation oracles**

Circuits viewed as proxies for unrestricted boolean functions: the search problems stay total even if the circuits in the defs are allowed to be any functions (not necessarily having small circuits)

Warm-up: in the context of END OF LINE/PPAD, if the circuits S and P were replaced with unrestricted boolean functions allowing "black-box access", the problem becomes impossible.
Call this "oracle PPAD"

Now define a "PPAD machine" to be a notional machine that, given black-box access to S and P, identifies a solution...
Such a machine can't used to solve oracle PPA!

Paul Beame, Stephen A. Cook, Jeff Edmonds, Russell Impagliazzo, Toniann Pitassi: The Relative Complexity of NP Search Problems. *JCSS* (1998)

Conclusion

This is ongoing work! The hardness of some of these complexity classes has been derived from various cryptographic assumptions (that are stronger than P$\neq$NP).

Further question include: do we "need" any other as-yet undefined classes of TFNP problems?
Can we base the hardness of (say) PPAD on weaker assumptions, ideally P$\neq$NP?

Conclusion

This is ongoing work! The hardness of some of these complexity classes has been derived from various cryptographic assumptions (that are stronger than P$\neq$NP).

Further question include: do we "need" any other as-yet undefined classes of TFNP problems?
Can we base the hardness of (say) PPAD on weaker assumptions, ideally P$\neq$NP?

Thanks!