ORACLE MACHINES AND THE LIMITS OF DIAGONALIZATION

CS 721

Ashwin Paranjape

Dikkala Sai Nishanth

Vipul Singh

(slides adapted from Vaishali Athale CSE860, Michigan State Univ.)

Overview

- Introduction
- Concept of “Oracle”
- The Limits of Diagonalization
- Logical independence vs relativization

Introduction

- Revisiting question of NP=P?
- Diagonalization proof used to show that Halting Problem is undecidable
- Can we use it to prove that NP=P or NP $\neq$ P?

Diagonalization

- Existence of representation of TM by string
- Ability of a universal Turing Machine to simulate any other w/o much overhead in running time or space

Relativization & Oracle

- Turing Machine provided with some information for "free"
  - Concept of "Oracle" for a language
  - Black box that answers membership of a string in the given language in one step

- Information affects the outcome of TM
- Oracle TM solves some problems easier

Definition

- TM M with special r/w tape (oracle tape)
- 3 special states: q_{query}, q_{yes}, q_{no}
- Language O used as oracle for M

- On entering q_{query}, moves to q_{yes} if $q \in O$, else moves to q_{no}.
- Query counts as 1 step

P^O and NP^O

- Oracle Turing Machine M^A tells membership of given string in A in a single computation step.

- P^A
  - Class of languages decidable with a polynomial time TM M^A that uses oracle A.

- NP^A
  - Class of languages decidable with a nondeterministic polynomial time TM M^A that uses oracle A.

Examples of “Oracle”

- Consider an oracle for SAT
  - Solves SAT problem in single step, for any size Boolean formula.

- With the help of an oracle for SAT, a TM can solve any NP problem in polynomial time
  - Regardless of whether NP=P, every NP problem is polynomial time reducible to SAT

Examples of "Oracle"

- SAT' $\in P^{SAT}$
  - DTM makes 1 call to SAT oracle and inverts answer
- If the oracle $O \in P$, then $P^{O}=P$
  - Replace the oracle by its actual computations (will be poly-time), hence still a poly-time DTM

EXPCOM

- $\{ < M,x,1^n > : M$ outputs 1 on x within 2^n steps$\}$
- We show that $P^{EXPCOM} = NP^{EXPCOM}$
- $EXP \subseteq P^{EXPCOM}$ and $NP^{EXPCOM} \subseteq EXP$

(i) Exponential computation in single step

(ii) Enumerate all choices of NTM and answer queries, overall exponential time only

Relativizing Results

- Can represent oracle TM as string
- Use this to simulate on UTM with access to O
- So, any result about TMs or complexity classes that uses only diagonalization holds for all oracle TMs. These are called *relativizing results*.

Limits of Diagonalization

- Goal of BGS theorem(theorem 9.19) - to prove that Diagonalization technique is unlikely to resolve the P versus NP question.
- Key ideas
  - Diagonalization is simulation of one TM by another.
  - Theorem proved by TMs using the Diagonalization method would still hold if both the machines were given the same oracle.

Key ideas(contd.)

- If P$\neq$ NP is provable using Diagonalization method, then even if assistance of an oracle is given then they should be different.
  - Does not work because BGS theorem proves that there exists an oracle B such that $P^B = NP^B$

- If P = NP is provable using Diagonalization method, then even if assistance of an oracle is given then they should be same.
  - Does not work because BGS theorem proves that there exists an oracle A such $P^A \neq NP^A$

Proof

- Proof Idea
  - Oracle A exists whereby $P^A = NP^A$
  - Oracle B exists whereby $P^B \neq NP^B$
- Proof of existence of oracle A
  - Let A be EXPCOM
  - $P^{EXPCOM} = NP^{EXPCOM} = EXP$

Proof of existence of oracle A

- Goals
  - Design an oracle B such that certain language U_B in NP^B provably requires brute force search and hence U_B cannot be in $P^{B.}$

1. $L_B \in NP^B$

1. $L_B \notin P^B$
   - Construct B such that no polynomial time turing machine M_1, M_2……..solves L_B

Goal 1: Identifying Language U_B

- Let U_B be the unary language
  - $U_B = \{ 1^n :$ some string of length n is in B $\}$
  - i.e., a string is in L_A iff there exists some string of the same length that is in A.
  - Intuition:
    - There are 2^n strings of length n
    - For a large enough n (i.e. $2^n > n^i$) , a polynomial time deterministic Turing machine cannot check the status of all strings of length n.

Goal 2: $U_B \in NP^B$

- Given a string 1^n,
  - Guess a string x of length n and verify that
  - Ask the oracle "Is the string x is in B"
    - Can be achieved in one step by the oracle for B
  - Note that NP^B can guess on all possible 2^n possible input words to B.
  - Result true for all languages B

Goal 3: $U_B \notin P^B$

- Construct B such that no polynomial time turing machine solves L_B in polytime (more generally in $o(2^n)$)

Goal 3: $U_B \not\in P^B$

- Construct B such that no polynomial time turing machine solves L_B in polytime (more generally in $o(2^n)$)
  - All possible oracle turing machines represented by M_i (M_i is binary expansion of integer i) for all $i \in N$

Goal 3: $U_B \notin P^B$

- Construct B such that no polynomial time turing machine solves L_B in polytime (more generally in $o(2^n)$)
  - All possible oracle turing machines represented by M_i (M_i is binary expansion of integer i) for all $i \in N$
  - Note that these turing machines are independent of Oracle

Goal 3: $U_B \notin P^B$

- Construct B such that no polynomial time turing machine solves L_B in polytime (more generally in $o(2^n)$)
  - All possible oracle turing machines represented by M_i (M_i is binary expansion of integer i) for all $i \in N$
  - Note that these turing machines are independent of Oracle
  - Construct B in stages where stage i ensures M_i^B does not decide U_B in time $o(2^n)$, WLOG $2^n/10$

Goal 3: $U_B \notin P^B$

- Construct B such that no polynomial time turing machine solves L_B in polytime (more generally in $o(2^n)$)
  - All possible oracle turing machines represented by M_i (M_i is binary expansion of integer i) for all $i \in N$
  - Note that these turing machines are independent of Oracle
  - Construct B in stages where stage i ensures M_i^B does not decide U_B in time $o(2^n)$, WLOG $2^n/10$
  - Start with B being empty, at each stage B determines the status of only finite number of strings
  - B has an underlying map from strings to yes, no undetermined. All undetermined strings are answered no.

Goal 3: $U_B \notin P^B$

- Stage i :

Goal 3: $U_B \not\in P^B$

- Stage i :
  - Let n be greater than the longest string determined by B

Goal 3: $U_B \not\in P^B$

- Stage i :
  - Let n be greater than the longest string determined by B
  - Give 1^n as input to M_i

`

Goal 3: $U_B \notin P^B$

- Stage i :
  - Let n be greater than the longest string determined by B
  - Give 1^n as input to M_i
  - M_i checks for $2^n/10$ strings all of which answer “No”

`

Goal 3: $U_B \notin P^B$

- Stage i :
  - Let n be greater than the longest string determined by B
  - Give 1^n as input to M_i
  - M_i checks for $2^n/10$ strings all of which answer “No”
  - Behaviour of M_i is different from B.

`

Goal 3: $U_B \notin P^B$

- Stage i :
  - Let n be greater than the longest string determined by B
  - Give 1^n as input to M_i
  - M_i checks for $2^n/10$ strings all of which answer "No"
  - Behaviour of M_i is different from B.
  - If (M_i answers "Yes"), set all length n strings in B as "No"

Goal 3: $U_B \notin P^B$

- Stage i :
  - Let n be greater than the longest string determined by B
  - Give 1^n as input to M_i
  - M_i checks for $2^n/10$ strings all of which answer "No"
  - Behaviour of M_i is different from B.
  - If (M_i answers "Yes"), set all length n strings in B as "No"
  - If (M_i answers "No"), set one length n string not checked by B as "Yes"

Goal 3: $U_B \notin P^B$

- Stage i :
  - Let n be greater than the longest string determined by B
  - Give 1^n as input to M_i
  - M_i checks for $2^n/10$ strings all of which answer "No"
  - Behaviour of M_i is different from B.
  - If (M_i answers "Yes"), set all length n strings in B as "No"
  - If (M_i answers "No"), set one length n string not checked by B as "Yes"
  - M_i answers opposite to U_B for 1^n, M_i does not decide U_B

Goal 3: $U_B \notin P^B$

- Stage i :
  - Let n be greater than the longest string determined by B
  - Give 1^n as input to M_i
  - M_i checks for $2^n/10$ strings all of which answer “No”
  - Behaviour of M_i is different from B.
  - If (M_i answers “Yes”), set all length n strings in B as “No”
  - If (M_i answers “No”), set one length n string not checked by B as “Yes”
  - M_i answers opposite to U_B for 1^n, M_i does not decide U_B
- True for all Oracle Tms

Goal 3: $U_B \notin P^B$

- Stage i :
  - Let n be greater than the longest string determined by B
  - Give 1^n as input to M_i
  - M_i checks for $2^n/10$ strings all of which answer "No"
  - Behaviour of M_i is different from B.
  - If (M_i answers "Yes"), set all length n strings in B as "No"
  - If (M_i answers "No"), set one length n string not checked by B as "Yes"
  - M_i answers opposite to U_B for 1^n, M_i does not decide U_B
- True for all Oracle Tms
- Generalizes to $DTIME^B(f(n))$, $f(n) = o(2^n)$

Significance of BGS theorem

- Given the above theorem, now we get back to the original question which was whether diagonalization can resolve P vs NP.

Significance of BGS theorem

- Given the above theorem, now we get back to the original question which was whether diagonalization can resolve P vs NP.
- Suppose a resolution of P vs NP is given using only the properties of diagonalization (properties I and II listed above), every statement about TM's in the resolution proof will also hold for oracle TM's (**for any oracle**).

Significance of BGS theorem

- Given the above theorem, now we get back to the original question which was whether diagonalization can resolve P vs NP.
- Suppose a resolution of P vs NP is given using only the properties of diagonalization (properties I and II listed above), every statement about TM's in the resolution proof will also hold for oracle TM's (**for any oracle**).
- Since BGS theorem shows the existence of oracles A and B such that $P^A = NP^A$ and $P^B \neq NP^B$, we can say that P = NP can neither be proved nor disproved using diagonalization arguments.

Significance of BGS theorem

- Given the above theorem, now we get back to the original question which was whether diagonalization can resolve P vs NP.
- Suppose a resolution of P vs NP is given using only the properties of diagonalization (properties I and II listed above), every statement about TM's in the resolution proof will also hold for oracle TM's (**for any oracle**).
- Since BGS theorem shows the existence of oracles A and B such that $P^A = NP^A$ and $P^B \neq NP^B$, we can say that P = NP can neither be proved nor disproved using diagonalization arguments.
- Hence any resolution of the P vs NP problem must use a non-relativizing fact.

Relativizing in Complexity Theory

- Many results in complexity theory relativize.

Relativizing in Complexity Theory

- Many results in complexity theory relativize.
- However, there do exist non-relativizing results.
- Examples: PCP theorem and IP = PSPACE

Relativizing in Complexity Theory

- Many results in complexity theory relativize.
- However, there do exist non-relativizing results.
- Examples: PCP theorem and IP = PSPACE
- Not yet known how to use these non-relativizing techniques to resolve P vs NP.
- BGS just tells us that if you ever hope to resolve P vs NP, you had better use a non-relativizing fact in your attempt to resolve.

What is behind relativization ?

- A deeper look at the concept of relativization.

What is behind relativization ?

- A deeper look at the concept of relativization.
- Notion of relativization was inspired from **independence results.**

What is behind relativization ?

- A deeper look at the concept of relativization.
- Notion of relativization was inspired from **independence results.**
- Independence results in mathematical logic are results which showed that certain natural statements can neither be proved nor disproved in a particular set of axioms.

Independence Results

- Independence results showed that certain natural mathematical statements can neither be proved nor disproved in a particular set of axioms.
- Two well known examples of independence results.
- Eg 1: Euclid’s 5^{th} postulate is independent from the first four.

Euclid’s Postulates

1. A straight line segment can be drawn joining any two points.
2. Any straight line segment can be extended indefinitely in a straight line.
3. Given any straight lines segment, a circle can be drawn having the segment as radius and one endpoint as center.
4. All Right Angles are congruent.
5. If two lines are drawn which intersect a third in such a way that the sum of the inner angles on one side is less than two Right Angles, then the two lines inevitably must intersect each other on that side if extended far enough. **This postulate is equivalent to what is known as the Parallel Postulate.**

Non-Euclidean Geometries

- The fifth postulate is independent from the first four. i.e it can neither be proved nor disproved using the first four postulates.
- If we assume the fifth postulate, we get a certain set of axioms – Euclidean Geometry
- If we assume the fifth postulate is not true, we get a different set of axioms – **Non-Euclidean geometries.**

Another Example

- Continuum hypothesis is independent from axioms of Zermelo-Fraenkel set theory.

Logical Independence vs Relativization

- BGS theorem shows that the statement P = NP can neither be proved nor disproved using relativizable facts.
- Consider a system of axioms which consists of all and only those facts about P which relativize.
- P = NP is independent from this set of axioms.
- Such an set (axiomatic system) was actually given by Arora, Impagliazzo and Vazirani [AIV93].

Set of non-relativizing facts ?

- How do we extend the above axiomatic system to allow it to prove non-relativizing results ?

Set of non-relativizing facts ?

- How do we extend the above axiomatic system to allow it to prove non-relativizing results ?
- One idea: Whenever a non-relayivizing result is encountered, assume it is true. (Like we did in constructing Euclidean geometry)
- A more conservative approach would ask the following question.

- Is there a single non-relativizing fact which is general enough so as to imply all known non-relativizing results ?

The Cook-Levin Theorem!

- Surprisingly, it turns out that there is such a non-relativizing fact.

- This is essentially the Cook-Levin Theorem

Crux of Cook-Levin: Computation is Local

- Locality of computation: Each basic step of a Turing machine only examines and modifies a constant number of tape locations.
- The article mentioned above [AIV93] describes a few ways to formalize the fact that computation if local.
- This means that any resolution of P vs NP will in some way of the other use the fact that TM computations are local.
- How close does knowing this fact take us to resolving P vs NP ?

How big a leap have we made towards resolving P vs NP ?

No bigger than the leap made by basic axioms of arithmetic towards proving Fermat's last theorem!!

Conclusion

References

- Arora, Sanjeev, and Boaz Barak. Computational complexity: a modern approach. Vol. 1. Cambridge, UK: Cambridge University Press, 3(4): 65-68 (2009)
- T. P. Baker, J. Gill, R. Solovay. Relativizatons of the P =? NP Question. SIAM Journal on Computing, 4(4): 431-442 (1975)