UCT

Space Bounds

Klaus Sutner
Carnegie Mellon University
Spring 2022

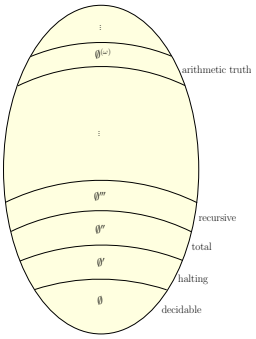
1 Space

2 Space Classes

3 Nondeterministic Space

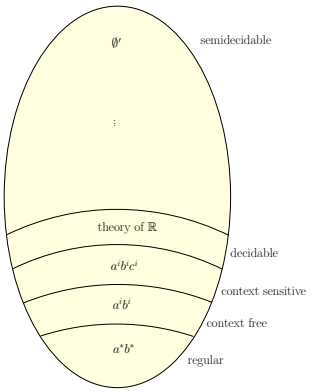
4 The Zoo

1930s



Well understood, but too far from feasible computation.

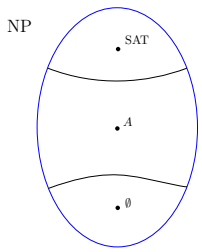
1950s



Chomsky's hierarchy, motivated by linguistics rather than effective computation.

1960/70s

Based on time complexity and the fundamental idea of nondeterminism, one of the key area in the vicinity of "easily decidable" looks like this:



Ladner's theorem provides the intermediate A, assuming $\mathbb{P} \neq \mathbb{NP}$.

Exponential Time Hypothesis

$\mathbb{P} \neq \mathbb{NP}$ is a reasonable, though not totally compelling, assumption.

Some (Impagliazzo, Paturi, 1999) prefer to be more direct: there is no polynomial (even: sub-exponential) time algorithm for satisfiability.

Exponential Time Hypothesis:
Every algorithm for SAT has running time $\Omega(2^{cn})$ for some $c > 0$.

Note that this would still allow for something like $\sqrt{2^n}$, but it rules out polynomial time solutions.

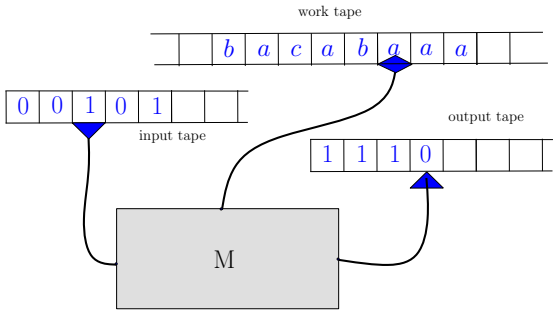
$\mathbb{P} \neq \mathbb{NP}$ can often be conveniently replaced by ETH, e.g., in the proof of Ladner's theorem.

How About Space? 6

Following Blum's idea of an abstract complexity measure, the second most important notion after time appears to be space–the memory required to carry out a computation.

- Space is a tighter bound than Time.
- Space, unlike Time, can be reused.
- In the RealWorld™, space is much less forgiving than time.

Recall: Transducer 7



Measuring Work Space 8

- We are using an off-line machine: there is a separate, read-only, two-way input tape. So the input can be read repeatedly.
- The machine also has a separate write-only output tape. This is not needed for acceptors, use states instead.
- The work tape is read/write in the usual manner.

The space complexity is measured only in terms of the work tape, the poor Turing machine is not responsible for input or output size.

Space Complexity 9

More precisely, let $(C_i)_{i<N}$ be the computation of TM $\mathcal{M}$ on input x. Write spc(C) for the number of tape cells used on the work tape in configuration C. Then the space complexity of $\mathcal{M}$ on x is defined as

$$S_{\mathcal{M}}(x) = \max\big(\,\mathsf{spc}(C_i) \mid 0 \le i < N\,\big)$$

In other words, we consider the largest configuration that appears during the computation. This makes perfect sense, it does not help much if most of the computation requires little space, but there is a brief phase where memory goes way up.

As usual, quietly assume $S_{\mathcal{M}}(x) = \infty$ if the computation diverges and uses arbitrarily large configurations. But note: a computation can diverge and have limited space complexity.

White and Gray Blanks 10

One way of avoiding the search for the maximum tape use is to modify the definition of our Turing machines slightly: orginally the inscription uses *white blank* symbols, plus whatever the input requires.

Once the machine gets going, it never writes a white blank, only *gray blanks*. So in the end we can simply count the ordinary symbols plus the gray blanks on the tape.

In other words, we cannot cheat by using a huge amount of memory at some point, and then just erasing everything (with white blanks).

Worst Case Space Complexity 11

As before with time, we focus on all instances of size n:

$$S_{\mathcal{M}}(n) = \max\big(\,S_{\mathcal{M}}(x) \mid x \text{ has size } n\,\big)$$

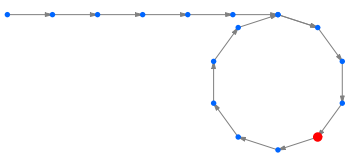
Again, this is worst case, average case space complexity can be defined similarly. As it turns out, average time complexity is very interesting, but average space complexity is less so (consider quick sort).

This is really a topic for an algorithms course.

Same Old? 12

Question: What are the similarities/differences between space and time complexity?

Here is a difference: a divergent computation obviously requires an unbounded amount of time. But a divergent computation may well only use bounded amount of space: the computation may be caught in a loop



Note that a loop cannot be too long: there are only exponentially many configurations (of a Turing machine) with a given space bound.

Loop Monitors 13

Unlike Halting, looping is decidable: given an arbitrary Turing Machine $\mathcal{M}$, we can construct a new machine $\mathcal{M}'$ that

- simulates $\mathcal{M}$, and
- keeps track of the history of the computation.

Say, we keep track of the history $(C_i)_{i<m}$ on an extra tape. Then we can easily check whether $\mathcal{M}$ has entered a loop.

Exercise

How much of a slow-down would this simulation cause?
How does the space complexity change?

1 Space

2 **Space Classes**

3 Nondeterministic Space

4 The Zoo

Space Complexity Classes 15

Definition

Let $f : \mathbb{N} \to \mathbb{N}$ be a function.

$$\mathrm{SPACE}(f) = \{\, \mathcal{L}(\mathcal{M}) \mid \mathcal{M} \text{ a TM}, S_{\mathcal{M}}(n) = O(f(n)) \,\}$$

A (deterministic) space complexity class is a class

$$\mathrm{SPACE}(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} \mathrm{SPACE}(f).$$

for some collection of functions $\mathcal{F}$.

Again, as in the time complexity case, one needs to be a bit careful with technical details.

Space Constructibility 16

In this case, the critical condition is the following: a function $s : \mathbb{N} \to \mathbb{N}$ is space constructible if there is a Turing machine that runs in $O(s(n))$ space and, for any input of size n, writes $s(n)$ on the tape as output (in binary).

Any function one encounters in the wild is indeed space constructible.

Without this assumption one has to jump through extra hoops in certain proofs, or the arguments may fail altogether. For example, one may have to try out possible values for $s(n)$ until we find one that works.

Some Important Classes 17

Here are some typical examples for deterministic space complexity classes.

- $\mathrm{SPACE}(1)$, constant space.
- $\mathrm{SPACE}(\log n)$, logarithmic space.
- $\mathrm{SPACE}(n)$, linear space.
- $\mathrm{PSPACE} = \mathrm{SPACE}(\mathrm{poly})$, polynomial space.

Note that unlike with polynomial time, polynomial space is already pushing the envelope of allegedly feasible computation quite a bit: a cubic memory requirement is much worse than just cubic running time.

Linear space is a much more reasonable constraint for feasible computation.

How Small Can It Get? 18

Recall that we assume that a Turing machine reads all its input, so for time complexity we have $n \leq T_{\mathcal{M}}(n)$.

Clearly, the analogous restriction $n \leq S_{\mathcal{M}}(n)$ would be totally inappropriate, it makes perfect sense to talk about sub-linear space complexities.

In fact, even constant space complexity makes sense.

Claim: $\mathrm{SPACE}(1)$ is the same as regular.

To see this, think of the state set as $Q \times [c] \times \Sigma^c$ (keep track of state, work tape head position, work tape inscription). Then use the theorem by Rabin/Scott that two-way FSMs can be simulated by ordinary one-way FSMs.

A Little Exercise 19

Suppose a directed graph is represented by its adjacency list, written out as a string:

$$a_{11}|a_{12}|\ldots|a_{1d_1} \# a_{21}|a_{22}|\ldots|a_{2d_2} \# \ \ldots\ \# a_{n1}|a_{n2}|\ldots|a_{nd_n}$$

where, say, $\Sigma = \{0, 1, |, \#\}$ and $a_{ij} \in \mathbf{2}^\star$.

Exercise

Find a natural graph problem that is in

- *linear space*
- *logarithmic space*
- *doubly logarithmic space*

Falling off a Cliff 20

The last example is already a bit contrived. Certainly it is not at all clear how to come up with a $\log\log\log n$ algorithm, never mind $\log^4 n$ or some such.

As mentioned, $\mathrm{SPACE}(1)$ is the class of regular languages.

Innocent question: is there anything between $\log\log n$ and 1?

Theorem (Hartmanis, Lewis, Stearns, 1965)

Let $f(n) = o(\log\log n)$. Then $\mathrm{SPACE}(f)$ is the same as constant space.

Note the little-oh, there are languages recognizable in space $\log\log n$ that are not recognizable in constant space.

Proof Sketch 21

Suppose $\mathcal{M}$ accepts some non-regular language in space f. Then there is an unbounded sequence of numbers

$$\ell_k = \min\big(\, \ell \mid \exists\, x \in \Sigma^\ell (\mathcal{M} \text{ uses} \geq k \text{ space on } x) \,\big)$$

Let $x \in \Sigma^\star$, $|x| = \ell_k$ be a witness. We use crossing sequences:[†] when the input head moves between positions i and $i+1$, record the configuration in a crossing sequence $\mathcal{X}_i$.

The number of all possible configurations of $\mathcal{M}$ is bounded by $2^{cf(\ell_k)}$, where c is some constant.

We have $f(n) \in o(\log\log n)$, so the number of configurations is $o(\log \ell_k)$. On the other hand, the number of crossing sequences in linear in n. Hence $\mathcal{X}_i = \mathcal{X}_j$ for some $i \neq j$, and we can do surgery on x and produce a shorter word x' that requires $f(\ell_k)$ space, a contradiction.

□

[†] You may have seen these in the proof that 2-way DFAs are no more powerful than 1-way DFA, or in the argument that a 1-tape TM requires quadratic time to recognize palindromes.

Space vs. Time 22

It is clear from the definitions that

$$\mathrm{TIME}(f) \subseteq \mathrm{SPACE}(f).$$

For a bound going in the opposite direction we can count the number of instantaneous descriptions of a Turing machine of given space complexity to obtain a bound on the length of any accepting computation of such a machine.

Theorem

Let $f(n) \geq \log n$. Then

$$\mathrm{SPACE}(f) \subseteq \mathrm{TIME}(2^{O(f(n))}).$$

Recall: Time Hierarchy Theorems 23

It is intuitively clear that $\mathrm{TIME}(f)$ will be larger than $\mathrm{TIME}(g)$ provided that f is sufficiently much "larger" than g. But as the example of log-log space bounds shows, one has to be a bit careful.

Theorem (Hartmanis, Stearns 1965)

Let f be time constructible, $g(n) = o(f(n))$.

Then $\mathrm{TIME}(g(n)) \subsetneq \mathrm{TIME}(f(n)\log f(n))$.

Keeping It Simple

In the following, we assume that time/space functions are always reasonable (say, time-constructible and space-constructible).

Remember Poincaré: don't insult the fathers (insulting mothers was presumably OK around 1900).

Also, on occasion it will be helpful to assume that the functions in question are not too small, something along the lines of $f(n) \geq \log n$.

We'll just call these functions reasonable.

Space Hierarchy

The analogous result for space is actually a bit easier to state.

Theorem (Hartmanis, Lewis, Stearns 1965)

Let f, g be reasonable, $g(n) = o(f(n))$.
Then $\mathrm{SPACE}(g(n)) \subsetneq \mathrm{SPACE}(f(n))$.

As before for time, these results are proved by diagonalization and do not produce natural examples of hard problems.

Given a concrete combinatorial problem it is usually very, very hard to find a lower bound for its concrete space complexity, we have to make do with hardness results, just as for time complexity.

Proof

There is a universal Turing machine $\mathcal{U}$ that has space complexity $f(n)$ and simulates all $g(n)$-space machines $\mathcal{M}_e$ in the sense that there is a constant c_e such that for $n = |x|$:

- $\mathcal{U}(e\#x)$ uses space $c_e \cdot g(n)$,
- $\mathcal{U}(e\#x) = \mathcal{M}_e(x) \in \mathbf{2}$ provided that $c_e \cdot g(n) \leq f(n)$,
- $\mathcal{U}(e\#x)$ aborts if $c_e \cdot g(n) > f(n)$.

Note that c_e cannot be avoided, the simulated machine may have a larger tape alphabet.

So $\mathcal{U}$ has space complexity f by brute force (analogous to using a clock to enforce time complexity; this time, we clip the tape).

Proof II

Now consider a new machine $\mathcal{M}$ such that

$$\mathcal{M}(e\#x) = 1 - \mathcal{U}(e\#e\#x)$$

So we flip the output bit if there is one, and abort otherwise. Note that $\mathcal{U}$ simulates $\mathcal{M}_e$ on input $e\#x$, the usual diagonalization mumbo-jumbo.

Then $\mathcal{M}$ also has space complexity $f(n)$, but $\mathcal{M}(e\#x)$ disagrees with $\mathcal{M}_e(e\#x)$ for all sufficiently large x.

Thus $\mathcal{M}$ is not in $\mathrm{SPACE}(g)$, as required.

□

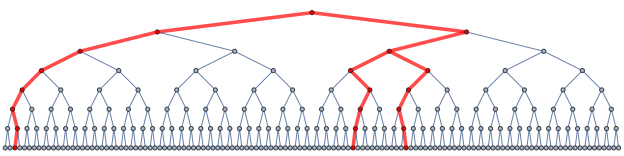
1 Space

2 Space Classes

3 **Nondeterministic Space**

4 The Zoo

Nondeterministic Machines



Space complexity also makes sense for nondeterministic machines: pick the (accepting) branch that requires the least memory.

This is directly analogous to our definition of nondeterministic time complexity.

Nondeterministic Space

We can handle space complexity $S_{\mathcal{M}}(x)$ in a similar manner:

$$S_{\mathcal{M}}(x) = \min\big(\,|\beta| \mid \beta \text{ accepting branch in } \mathcal{T}_x\,\big)$$

Here $\mathcal{T}_x$ is the computation tree of the machine on input x, and $|\beta|$ stands for the largest amount of memory used along branch β. Note that this does not necessarily mean the the branch shorter than all others.

Similarly define $S_{\mathcal{M}}(n)$ in the usual worst-case manner. We will not be concerned with average-case space complexity here.

Weak versus Strong

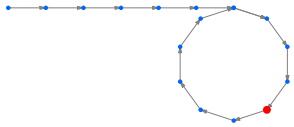
One might object that from an algorithms point of view our definition is a bit weak: arguably we would prefer a bound on all branches, not just on the accepting ones. One should not distinguish between Yes and No instances when it comes to resource bound.

This alternative notion is called strong space complexity, Similarly we could use strong time complexity.

All true, but most papers in complexity theory use the weak model rather than the strong: the difference only comes into play when dealing with low classes, otherwise we can clean up the machines to make them behave properly. And weak is easier to deal with than strong.

Trade-Offs

Note that one can sometimes trade space for time: a computation can be made to use less memory by re-computing values rather than storing them. Floyd's cycle detection algorithm is a nice example.



Memoizing is a trick that goes in the opposite direction: avoid recomputation at a cost of an additional hash table.

Or space measure ignores these issues, we go for the least memory-intensive branch, no matter how long the computation is. Obviously, it would be of interest to combine both measures.

Nondeterministic Complexity Classes

We can now lift the definitions of deterministic space complexity classes to nondeterministic ones.

Definition

Let $f : \mathbb{N} \to \mathbb{N}$ be a (reasonable) function.

$$\mathrm{NSPACE}(f) = \{\, \mathcal{L}(\mathcal{M}) \mid \mathcal{M} \text{ a nondeterministic TM}, S_{\mathcal{M}}(n) = O(f(n)) \,\}$$

As before, this generalizes easily to $\mathrm{NSPACE}(\mathcal{F})$ for some class of functions $\mathcal{F}$.

As usual, this is for acceptors (decision problems) only, we don't yet know how to handle nondeterministic transducers.

Some Results

From the definitions

$$\mathrm{TIME}(f) \subseteq \mathrm{NTIME}(f) \subseteq \mathrm{NSPACE}(f)$$

$$\mathrm{SPACE}(f) \subseteq \mathrm{NSPACE}(f)$$

More interesting is the question how much deterministic time/space is required to capture a nondeterministic class.

Theorem

Assume that $\log n \leq g(n)$. *Then*

$$\mathrm{NTIME}(f) \subseteq \mathrm{SPACE}(f)$$

$$\mathrm{NSPACE}(g) \subseteq \mathrm{TIME}(2^{O(g)})$$

Savitch's Theorem

Theorem (Savitch 1970)

Assume that $\log n \leq g(n)$. *Then*

$$\mathrm{NSPACE}(g(n)) \subseteq \mathrm{SPACE}(g(n)^2).$$

Sketch of proof.

This is again deterministic simulation, but keeping an eye on space requirements.

The trick is to use a divide-and-conquer approach: a computation of length t is broken up into two subcomputations of length $t/2$.

Since the target is a space class and space, unlike time, can be reused, one can show that the divide-and-conquer algorithm requires no more than $g(n)^2$ space.

Savitch Proof

Let L be in $\mathrm{NSPACE}(g(n))$. Suppose $\mathcal{M}$ is a non-deterministic Turing machine that accepts L and has space complexity $S_{\mathcal{M}}(n) \leq g(n)$.

Clearly we are dealing with yet another path-existence problem in the computation graph of $\mathcal{M}$, this time for paths of limited length.

Correspondingly define a path predicate that bounds the number of steps to get from one configuration to the other (uniformly in n). Here C_1, C_2 are configurations of $\mathcal{M}$ of size at most $g(n)$.

$$\mathrm{path}(C_1, C_2, k) \iff \exists\, s \leq 2^k \left(C_1 \vdash^s_{\mathcal{M}} C_2\right)$$

Proof Contd

Note that $x \in L$ iff $\mathrm{path}(C^{\mathsf{init}}_x, C^{\mathsf{halt}}_Y, O(f(n)))$, so to test membership in L we only need to compute path.

Claim

$\mathrm{path}(C_1, C_2, 0)$ *iff* $C_1 = C_2$ *or* $C_1 \vdash^1_{\mathcal{M}} C_2$.

$\mathrm{path}(C_1, C_2, k+1)$ *iff* $\exists\, C\ (\mathrm{path}(C_1, C, k) \wedge \mathrm{path}(C, C_2, k))$.

The claim is obvious from the definition, but note that it provides a recursive definition of path.

Also observe that the recursion involves a search for the intermediate configuration C.

Implementing the Recursion

Implementing a recursion requires memory, typically a recursion stack: we have to keep track of pending calls and the required memory information.

In this case the recursion stack will have depth $O(f(n))$: the length of a computation is bounded by $2^{O(f(n))}$.

Each stack frame requires space $O(f(n))$ for the configurations.

The search for C can also be handled in $O(f(n))$ space.

Thus the total space complexity of the algorithm is $O(f^2(n))$: stack size times frame size.

□

Recall: The Graph Perspective

Given a Turing machine $\mathcal{M}$ (deterministic or nondeterministic), we can define the configuration graph $\mathfrak{C}(\mathcal{M})$ as follows:

- vertices are all configurations of $\mathcal{M}$,
- there is an edge (C, C') if the machine can go from C to C' in one step.

Usually we are only interested in the subgraph $\mathfrak{C}(\mathcal{M}, x)$, for some input x: the part of $\mathfrak{C}(\mathcal{M})$ accessible from the initial configuration C^{init}_x.

The size of the nodes in $\mathfrak{C}(\mathcal{M}, x)$ is $O(s(|x|))$ where s is the space complexity of $\mathcal{M}$ (we can ignore larger nodes as far as acceptance goes; we could also use strong complexity).

For a deterministic machine, each node has out-degree at most 1. Recall from HW that we may assume the out-degree is 2 for nondeterministic machines (tame Turing machines).

Computation Tree

On occasion it is preferable to think of the computations of a Turing machine as describing a tree: C' is a child of C if the machine can move from C to C' in one step. Technically, in order to ensure a tree structure, we need to distinguish between nodes on different branches. For example, we could use finite sequences of configurations

$$C_x, C_1, C_2, \ldots, C_{k-1}, C_k$$

as nodes, rather than just configurations: we keep track of the history of the computation.

Trees are useful for example to define time and space complexity.

1 Space

2 Space Classes

3 Nondeterministic Space

4 **The Zoo**

Some Classes

- $\mathrm{SPACE}(1)$, constant space, regular languages
- $\mathbb{L} = \mathrm{SPACE}(\log)$, logarithmic space
- $\mathbb{NL} = \mathrm{NSPACE}(\log)$, nondeterministic logarithmic space
- $\mathbb{P} = \mathrm{TIME}(\mathrm{poly})$, polynomial time
- $\mathbb{NP} = \mathrm{NTIME}(\mathrm{poly})$, nondeterministic polynomial time
- $\mathrm{SPACE}(n)$, linear space
- $\mathrm{NSPACE}(n)$, nondeterministic linear space
- $\mathrm{PSPACE} = \mathrm{SPACE}(\mathrm{poly})$, polynomial space
- Classical stuff: CFL, CSL, primitive recursive, decidable, semidecidable.

The Chain

Theorem
$\mathbb{L} \subseteq \mathbb{NL} \subseteq \mathbb{P} \subseteq \mathbb{NP} \subseteq \mathrm{PSPACE} = \mathrm{NPSPACE}$.

Proof.

The sticky points are

- $\mathbb{NL} \subseteq \mathbb{P}$: see next slide
- $\mathbb{NP} \subseteq \mathrm{PSPACE}$: search for all witnesses
- $\mathrm{PSPACE} = \mathrm{NPSPACE}$: Savitch's theorem

Proof

To show that $\mathbb{NL} \subseteq \mathbb{P}$, suppose we have a logarithmic space acceptor $\mathcal{M}$.

The nodes in the computation graph $\mathfrak{C}(\mathcal{M}, x)$ can be specified in $O(\log n)$ bits: state constant, head position $\log n$, tape inscription $O(\log n)$.

Hence the size of the graph is polynomial in n. Moreover, we can construct the graph in polynomial time (say, write down the adjacency lists).

But $\mathcal{M}$ accepts x iff there is a path in $\mathfrak{C}(\mathcal{M}, x)$ from C_x^{init} to C_Y^{halt}. We can use any standard graph reachability algorithm to check this.

□

Separation

The time and space hierarchy results of Hartmanis, Lewis and Stearns (from 1965, no less) give us some rather weak separation results:

Theorem
$\mathbb{L} \neq \mathrm{PSPACE}$ *and* $\mathbb{P} \neq \mathrm{EXP}_1$.

Surprisingly, that's it for separation results: nothing else is known at this point.

At the core of all these results is old-fashioned diagonalization: with just a bit of exaggeration one could say that nothing has happened since Cantor (well, let's say, since Turing).

Alright, alright, that's grossly unfair: $\mathsf{MIP}^\star = \mathsf{RE}$ is truly mindboggling.

Oracles and Relativization

Alas, it is not so easy to come up with other promising approaches to separation. E.g., oracles probably won't help at all.

It is a labor of love to check that all the basic results of computability theory (which are phrased in terms of computable functions $\{e\}$) also hold for functions computable relative to an arbitrary oracle, $\{e\}^A$.

The classical theory really is nothing but the study of $\{e\}^\emptyset$.

This means that for every theorem in classical computability theory there is a relativized version that uses an oracle. In particular all arguments involving diagonalization carry over to the oracle world.

Oracle Classes

By fixing a particular oracle $A \subseteq \Sigma^\star$ and attaching it to all machines in a certain class we can relativize the class.

For example, $\mathbb{P}^A$ is the collection of all decision problems decidable by a polynomial time Turing machine with oracle A.

Clearly, $\mathbb{P}^A = \mathbb{P}$ whenever $A \in \mathbb{P}$, but for "smarter" oracles we get more interesting problems.

Exercise
What would $\mathbb{P}^{\mathsf{SAT}}$ *look like*[†]*?*

[†]This is perfectly practical, just think of the oracle as being given by a state-of-the-art SAT solver. What can you do with it?

Oracles Are Monsters

Theorem (Baker, Gill, Solovay 1975)

There is an oracle A for which $\mathbb{P}^A = \mathbb{NP}^A$.

There is an oracle B for which $\mathbb{P}^B \neq \mathbb{NP}^B$.

In conjunction with the relativization claim from above, this means that standard techniques from computability theory are not going to help to resolve the $\mathbb{P} = \mathbb{NP}$ question: all these arguments are invariant under oracles.

New tools are needed, and to-date it is not really clear which line of attack might ultimately succeed in separating the classes. Not to mention that a group of researchers think that the answer really is $\mathbb{P} = \mathbb{NP}$.

Proof Sketch

For the identity $\mathbb{P}^A = \mathbb{NP}^A$ we can give a fairly simple definition of A:

$$A = \{\, e\#x\#0^n \mid \mathcal{M}_e \text{ accepts } x \text{ in at most } 2^n \text{ steps} \,\}$$

This is a padded version of the Halting set for plain exponential computations. With this oracle it is not too hard to check that

$$\mathbb{P}^A = \mathbb{NP}^A = \mathrm{EXP}_1$$

The oracle "absorbs" the potential advantages of nondeterministic computation.

Incidentally, we could also use a PSPACE-complete oracle (say, quantified Boolean formulae).

Proof Sketch II

To obtain $\mathbb{P}^B \neq \mathbb{NP}^B$, first define the length set of $B \subseteq \Sigma^\star$ as

$$L_B = \{\, 0^{|x|} \mid x \in B \,\} \subseteq 0^\star$$

and note that the tally language L_B is in $\mathbb{NP}^B$.

Hence it suffices to construct B such that $L_B \notin \mathbb{P}^B$.

This time, we won't be able to give an explicit definition of B, instead we resort to the standard trick in CRT: B is constructed in stages.

Initially, $B_0 = \emptyset$. At each stage σ, we will put some strings into B, or block them from ever entering B. As usual, B_σ always determines only finitely many strings, the whole construction is effective.

Stage σ

Let $n = \max\big(\, |x| \mid x \text{ determined by } B_{<\sigma} \,\big) + 1$.

Run $\mathcal{M}_\sigma^{B_{<\sigma}}$ on input 0^n for $2^n/42$ steps[†].

Here we assume that queries $z \in B_{<\sigma}?$ are handled as follows:

- If the membership status of z has not been settled, return the answer No, and block z from entering B.
- Otherwise, return the correct answer.

Now if $\mathcal{M}_\sigma^{B_{<\sigma}}$ accepts, then block all strings in $\mathbf{2}^n$ from B.

Otherwise, pick a free string $z \in \mathbf{2}^n$ and place it in B_σ, block the rest.

□

[†] Ponder deeply: why 42?

Immerman's Zoo