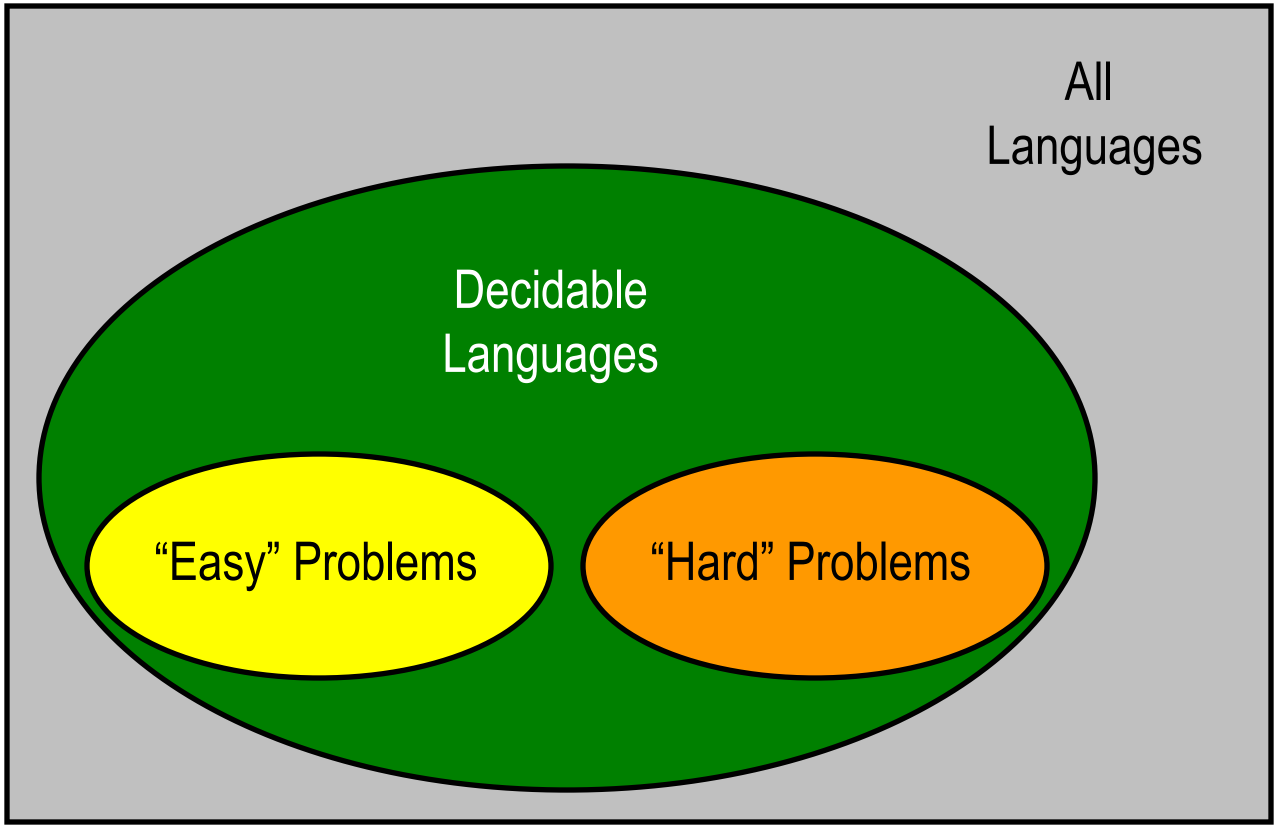
Complexity of Problems

Classifying Problems

- We have seen that decision problems (and their associated languages) can be classified into decidable and undecidable.
  This result was obtained by Turing and others in the 1930' — before the invention of computers.

- After the invention of computers, it became clear that it would be useful to classify decidable problems, to distinguish harder problems from easier problems
  This led to the development of computational complexity theory in the 1960's and 1970's

The Aim


All
Languages
Decidable
Languages
“Easy” Problems
“Hard” Problems

Question

Which of these decision problem is hardest?

1. For a given n, is n prime?
2. For a given n, is n equal to the sum of 3 primes?[1]
3. For a given n, does the nth person in the Vancouver telephone directory have first initial **J**?

---

[1]see **`http://www.faber.co.uk/faber/million_dollar.asp`**

Complexity Measures

Every decidable problem has a set of algorithms (= TMs) that solve it.

What property of this set of algorithms could we measure to classify the problem?

- The difficulty of constructing such an algorithm?
- The length of the shortest possible algorithm?

  (Giving a static complexity measure[2].)
- The efficiency of the most efficient possible algorithm?

  (Giving a dynamic complexity measure.)

---

[2]This has proved useful for classifying the complexity of strings, where it is called Kolmogorov complexity. See "Introduction to Kolmogorov Complexity and its Applications", Li and Vitani, 1993

Dynamic Complexity Measures

A dynamic complexity measure is a numerical function that measures the maximum resources used by an algorithm to compute the answer to a given instance

To define a dynamic complexity measure we have to define for each possible algorithm M a numerical function φ_M on the same inputs

Blum's Axioms

Blum proposed[3] that any useful dynamic complexity measure should satisfy the following properties:

- $\varphi_M(x)$ is defined exactly when $M(x)$ is defined
- The problem: for given M, x, r, does $\varphi_M(x) = r$? is decidable

---

[3]see "A machine independent theory of the complexity of recursive functions", Blum, *Journal of the ACM* 14, pp 322-336, (1967)

Time Complexity

The most critical computational resource is often time, so the most useful complexity measure is often time complexity

If we take Turing Machine as our model of computation, then we can give a precise measure of the time resources used by a computation

Definition The time complexity of a Turing Machine T is the function Time_T such that $\mathrm{Time}_T(x)$ is the number of steps taken by the computation $T(x)$

(Note that if $T(x)$ does not halt, then $\mathrm{Time}_T(x)$ is undefined.)

Space Complexity

Another important computational resource is amount of "memory" used by an algorithm, that is space. The corresponding complexity measure is space complexity

As with time, if we take Turing Machine as our model of computation, then we can easily give a measure of the space resources used by a computation

Definition The space complexity of a Turing Machine T is the function Space_T such that $\mathrm{Space}_T(x)$ is the number of distinct tape cells visited during the computation $T(x)$

(Note that if $T(x)$ does not halt, then $\mathrm{Space}_T(x)$ is undefined.)

Time Complexity of Problems I

Now it seems that we could define the time complexity of a problem as the time complexity of the most efficient Turing Machine that decides the corresponding language, but there is a difficulty …

It may be impossible to define the most efficient Turing Machine, because:

- It may be possible to speed up the computation by using a bigger alphabet
- It may be possible to speed up the computation by using more tapes

Linear Speed-up Theorem

Theorem For any Turing Machine T that decides a language **L**, and any $m > 0$, there is another Turing Machine T' that also decides **L**, such that

$$\mathrm{Time}_{T'}(x) \leq \frac{\mathrm{Time}_{T}(x)}{m} + 2\,|\,x\,|$$

(see Papadimitriou, Theorem 2.2)

Proof

The machine T' has a larger alphabet than T, many more states, and an extra tape.

The alphabet includes an extra symbol for each possible k-tuple of symbols in the alphabet of T

T' first compresses its input by writing a symbol on a new tape encoding each k-tuple of symbols of the original input. It then returns the head to the leftmost cell. This takes $2|x|$ steps in total

T' then simulates T by manipulating these more complex symbols to achieve the same changes as T. T' can simulate k steps of T by reading and changing at most 3 complex symbols, which can be done in 6 steps

Choosing $k=6m$ gives a speed-up by a factor of m

Table Look-Up

We can do a similar trick to speed up the computation on any finite set of inputs.

Theorem For any Turing Machine T that decides a language **L**, and any $m > 0$, there is another Turing Machine T' that also decides **L**, such that for all inputs x with $|x| \leq m$,

$$\mathrm{Time}_{T'}(x) \leq |x|$$

Proof Idea

The machine T' has additional states corresponding to each possible input of length at most m

T' first reads to the end of the input, remembering what it has seen by going into the corresponding state. If it reaches the end of the input in one of its special states it then immediately halts, giving the desired answer.

Otherwise it returns to the start of the input and behaves like T

Time Complexity of Problems II

Given any decidable language, and any TM that decides it, we have seen that we can construct a TM that

- Decides it faster by any linear factor
- Decides it faster for all input up to some fixed length

So we cannot define an exact time complexity for a language, but we can give an asymptotic form …

Math Prerequisites

Let f and g be two functions $f, g : \mathrm{N} \rightarrow \Re^{+}$. We say that $f(n)=O(g(n))$ if there exist positive integers c and n_0 such that for every $n > n_0$

$$f(n) \leq cg(n)$$

Examples

- A polynomial of degree k is $O(n^k)$

  $$5n^3 - 7n^2 + 3n - 110 = O(n^3)$$

- $\log(n^k) = O(\log n)$

- $\log_a n = O(\log_b n)$

Math Prerequisites

Let f and g be two functions $f, g : \mathrm{N} \to \Re^{+}$. We say that $f(n)=o(g(n))$ if

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=0$$

In other words, $f(n)=o(g(n))$ means that, for any real number c, there is n_0 such that for every $n > n_0$

$$f(n) < cg(n)$$

Examples

- $n^{k-1} = o(n^k)$ $\qquad n = o(n^2), \quad n^2 = o(n^3), \ldots$
- $\log^{k-1} n = o(\log^k n)$
- $\log^k n = o(n^m)$
- $n^k = o(a^n), \quad a > 1$

Definition

For any function f, we say that the time complexity of a decidable language `L` is in $O(f)$ if there exists a Turing Machine T which decides `L`, and constants n_0 and c such that for all inputs x with $|x| > n_0$

$$\mathrm{Time}_T(x) \leq cf(|x|)$$

Complexity Classes

Now we are in a position to divide up the decidable languages into classes, according to their time complexity

Definition

The time complexity class $\mathrm{TIME}[f]$ is defined to be the class of all languages with time complexity in $O(f)$

(Note, it is sometimes called $\mathrm{DTIME}[f]$ — for Deterministic Time)

Examples