Tractable problems and P

We want to identify which problems are *tractable/feasible*, or *efficiently computable*, i.e. those problems whose solutions can be computed in a reasonable amount of time.

We focus on worst-case analysis, and we want the worst-case complexity $W(n)$ to be not too large, where n is the input size.

Example 1: **Sorting a list by comparisons**

We have seen $W(n) = n \log n$ comparisons where n is the length of the list (using MergeSort). We can agree that sorting is tractable, since comparing two elements in a list can be done easily.

Example 2: **Given an undirected graph G, does G have a Euler path?**

Suppose G has n nodes and m arcs. Input size $|G|$ depends on the representation of G:

- Adjacency matrix has size n^2
- Linked list has size $n + m$

G has an Euler path iff G has 0 or 2 nodes of odd degree.

This is tractable, since we can count the odd degree nodes by making a single pass through either representation, keeping track of the degree count for the current node and the number of odd degree nodes found so far.

By contrast:
Example 3: **Given an undirected graph G, does G have a Hamiltonian path?**

Seeing whether a graph with n nodes has a Hamiltonian path seems to require looking at $n!$ different paths - too long.

Thus this problem is apparently intractable.

Decision problems are those problems which have a yes/no answer, such as Examples 2 or 3.

A decision problem D is decided by an algorithm A if for any input x, A returns ‘yes’ or ‘no’ depending on whether D(x) (and in particular A always terminates)

A decision problem D is *decidable in polynomial time* iff it is decided by some algorithm A which runs within polynomial time.

We abbreviate polynomial time to **poly time** or **p-time**

Example 2 is decidable in p-time, but Example 3 is (apparently) not.

A problem is tractable if it can be computed within polynomially many steps in worst case ($W(n) \leq p(n)$ for some polynomial $p(n)$).

More succinctly, **'tractable = polynomial time'**.

According to this, sorting a list is tractable, as is Example 2, but Example 3 is (apparently) not.

We have been talking about $p(n)$ steps in a computation on input size n. Clearly the notions of input size and computation step have different meanings depending on the model we are using:

- For sorting a list, we took input size to be the number of list items, and we counted comparisons, ignoring other computation steps (swaps, copying, recursive procedure calls, etc.)
- For Example 2 we measured input size either using adjacency matrices or adjacency lists, and computation steps would involve inspecting the input and incrementing counters

However, all models of computation and measures of input size give approximately the same results:

- If a problem can be solved in polynomial time $p(n)$ in some model, then if we change the model the problem can still be solved in $p - timeq(n)$
- We may get a different polynomial, but the concept of p-time is robust

If a problem can be solved in p-time in some reasonable model of computation, then it can be solved in p-time in any other reasonable model of computation.

Thus for sorting a list, a different model would be to take the sizes of the items to be sorted into account in measuring input length, and to take into account the fact that comparisons can take different amounts of time depending on the sizes of the items and the length of the list.

We would still get p-many steps but not necessarily $O(n \log n)$

A decision problem $D(x)$ is in the complexity class P (polynomial time) if it can be decided within time $p(n)$ in some reasonable model of computation.

By the Invariance Thesis this definition is model-independent.

Note that sorting a list does not belong to the class P, since it is not a decision problem. It is convenient to define complexity classes for decision problems only, at least to start with.

What models would be unreasonable? We give two unreasonable models:

- Superpolynomial parallelism is unreasonable. If we could carry out more than polynomially many operations in parallel in a single step, then we might be able to solve exponential time problems in p-time. So this model is unreasonable.
- Unary numbers (writing e.g. 11111 for 5) are unreasonable. When dealing with numbers we do not allow unary representation (use base 2 or greater). This is because unary gives input size which is exponentially larger than binary, and so an exponential time algorithm can appear to be p-time

Unary Example: With a unary representation we can check whether a number n is prime by looking at all $m < n$ and seeing whether m divides n. This takes n divisions. This is p-time if input size is n. However the true input size is actually $\log n$ and so we have an exp-time algorithm.

The usual arithmetical operations (addition, subtraction, multiplication, division) are p-time using the usual binary or decimal representations, i.e. they are polynomial in $\log n$, rather than in n

The next result shows that polynomial time is well-behaved.

Suppose that f and g are functions which are p-time computable. Then the composition $g \circ f$ is also p-time computable.

Proof. Suppose that $f(x)$ is computed by algorithm A within time $p(n)$ where $n = |x|$, while $g(y)$ is computed by algorithm B within time $q(m)$ where $m = |y|$.

Take input x with $|x| = n$. We compute $g(f(x))$ by first running A on x to get $f(x)$ and then running B on $f(x)$ to get $g(f(x))$. Running A on x takes $\leq p(n)$ steps.

To see how long running B takes, we need a bound on the size of the input $f(x)$. But A runs for $\leq p(n)$ steps to build $f(x)$. So $|f(x)|$ must be polynomially bounded in n — there is no time to build a larger output. Say $|f(x)| \leq p'(n)$ for some polynomial $p'(n)$. Then B runs within $q(p'(n))$ steps.

The total running time (A followed by B) is $p(n) + q(p'(n))$. This is polynomial in n. Hence result.