15.7 Exponential Solutions to $\mathcal{NP}$ Problems

In the previous section, we have said that it is currently not know whether the classes $\mathcal{P}$ and $\mathcal{NP}$ are different. While we know that $\mathcal{P} \subset \mathcal{NP}$, we do not know whether it is a *proper* subset. It is widely believed that there are problems that are in the class $\mathcal{NP}$, but not in the class $\mathcal{P}$, but this has so far not been proven. This is much harder problem than determining, for example, whether the classes of partially decidable and totally decidable languages are different, which we were able to solve pretty easily.

So, we do not know whether there exist efficient, polynomial-time deterministic solutions to some of the problems from the class $\mathcal{NP}$. But can we say anything about the efficiency of deterministic solutions to the problems in the class $\mathcal{NP}$? One thing that we know is that they have at least an exponential-time deterministic solution. That is a consequence of the following theorem.

Theorem 15.3: Exponential Solutions to $\mathcal{NP}$ Problems

Let L be a language accepted by a standard NDTM M with time complexity $p(n)$, which halts on every input string. Then there exists a deterministic Turing Machine M' that decides the language L, and which is of time complexity $2^{p(n)}$.

Proof Outline. Recall that a standard NDTM halting on every input string means that for every input string there is at least one branch of computation that halts in an accepting or non-accepting state. Recall also that the time it takes to compute an answer "yes" or "no" for a string is defined to be the minimal time over all the branches of computation that halt for that string. Therefore, the time complexity of M being $p(n)$ means that for each input string w, the branch that takes the minimal time to decide on that string takes at most $p(n)$ time, where n is the size of the string w.

The proof of this theorem is a matter of devising a way to simulate the computation of a standard NDTM M by some deterministic Turing Machine M', and showing that the number of steps taken by M for an input of size n is at most $2^{p(n)}$, if the number of steps in the quickest (in terms of the number of steps) branch of computation for M for that same string is at most $p(n)$. How can we simulate M by some deterministic Turing Machine M? We said that the computation of M can be seen as executing every branch of computation in parallel, and halting as soon as one of the branches halt. Therefore, M' has to somehow simulate all of these branches (as was the case with a DFA simulating NFA) and halt as soon as the first one of them halts. As M can only have a finite number of different branches, one idea that we can entertain is of M' simulating each of the branches of M in turn, as each individual branch is deterministic. However, this might not work, since some of the branches of M may not terminate. Therefore, if M' simulates a non-halting branch before a halting one, it will never halt, whereas M doing these two branches in parallel would halt. So, we have to devise some other strategy. The strategy is to enumerate all the different branches of M by positive integer numbers (which we can do easily), then to generate an infinite sequence of ordered pairs (i, j) and finally for each pair to simulate the i-th branch of M for j moves. If M halts in one of these simulations, then M' also halts and accepts a string if M' accepts, and rejects otherwise. Thus, M' first generates a pair $(1, 1)$, then simulates the first branch of M for one move, then generates a pair $(1, 2)$ and simulats the first branch of M for one move, then generates a pair $(2, 1)$ and simulates the second branch of M for two moves and so on. The sequence of pairs is, thus,

$$(1, 1), (1, 2), (2, 1), (1, 3), (2, 2), (3, 1), (1, 4), (2, 3), (3, 2), (4, 1), \ldots$$

Since we know that for every input string w, M halts in some branch k after l moves, and we know that the pair (k, l) will at some point be generated, we know that M' will also halt on every string and accept exactly the strings that M accepts.

How many moves will M' constructed in such way make? Let us first consider how many possible branches there are for any input string with M. In M, for each state and each tape symbol, there exists only finitely many possible next moves. Furthermore, since the number of states and the number of tape symbols is finite, there exists a positive integer k such that for each state and each tape symbol, M has at most k possible next moves. Therefore, in the absolutely the worst case, at each step of computation of M, there might be k possible next moves, each corresponding to one new branch of computation. Since for a string of size n, each branch will take at most $p(n)$ steps before the first one of them halts, the theoretical upper limit on the number of branches for such string is $k^{p(n)}$. The total number of (i, j) pairs that M' has to simulate is then $k^{p(n)} \cdot p(n)$, and it is certain that for one of these (i, j) pairs, M' will halt. Each (i, j) pair needs to simulate at most $p(n)$ steps. Therefore, the total number of steps M' may need to

simulate is at most $k^{p(n)} \cdot p(n) \cdot p(n) \leq k^{3p(n)}$. Since the simulation of each step of M by M' takes a constant time c, the total number of steps that M' makes for any input string of length n is certainly at most $ck^{3p(n)} = c2^{\log_2 kp(n)} = O(2^{q(n)}$, where $q(n) = \log_2 kp(n)$ is a polynomial. Therefore, the theorem is proven. □

What we have just shown is that if a language/problem L belongs to the class $\mathcal{NP}$, then there exists a deterministic Turing Machine that decides it in time $O(2^{p(n)})$ for some polynomial $p(n)$. Therefore, every problem in the class $\mathcal{NP}$ has at least exponential deterministic solution. So, we at least know that every problem in the class $\mathcal{NP}$ is deterministically solvable, though we do not know whether all of them are *efficiently* deterministically solvable.

We have expressed the Theorem 15.3 in terms of standard NDTMs, but a similar proof can be used if we consider verifier NDTMs. Thus, the same statement holds for verifier NDTMs.

Solving a problem using a verifier NDTM is, essentially, trying out all possible solutions in parallel and stopping as soon as one is found. In the HAMPATH problem, trying out all possible solutions is, essentially, trying out every sequence of n nodes in a n-node graph, where the first node in the sequence is our starting node and the end node is the ending node, and checking whether this sequence is a solution. Checking whether a sequence of nodes is a solution (i.e. a Hamiltonian path) can be done in polynomial time. But if we were to convert this verifier NDTM into a deterministic Turing Machine, that Turing Machine would essentially test each candidate solution in turn. Therefore, the time it takes for such a machine to decide on an instance of the problem is, in the worst case, the number of all different sequences of n nodes in a graph times the time it takes to check whether a sequence is a Hamiltonian Path. There are $n!$ different sequences of n nodes in a graph, therefore this is most definitely not a polynomial-time deterministic solution. This solution can obviously be improved, but no known improvement of it reduces its time complexity to some polynomial time.