4.2 Complexity of NP Problems

We discussed the (equivalent) inversion, search, and NP types of problems. Nobody knows whether *all* such problems are solvable in P-time (i.e. belong to P). This question (called P=?NP) is probably the most famous one in Theoretical Computer Science. All such problems are solvable in exponential time but it is unknown whether any better algorithms generally exist. For many problems the task of finding an efficient algorithm may seem hopeless, while similar or slightly modified problems have been solved. Examples:

1. Linear Programming: Given integer $n \times m$ matrix A and vector b, find a rational vector x with $Ax < b$.
   Note, if n and entries in A have $\leq k$-bits and x exists then an $O(nk)$-bit x exists, too.

   Solution: The Dantzig's ***Simplex*** algorithm finds x quickly for many A.
   Some A, however, take exponential time. After long frustrating efforts, a worst case
   P-time Ellipsoid Algorithm was finally found in [Yudin and A.S. Nemirovsky 76].
2. Primality test: Determine whether a given integer p has a factor?

   Solution: A bad (exponential time) way is to try all $2^{\|p\|}$ possible integer factors of p.
   More sophisticated algorithms, however, run fast (see section 5.1).
3. Graph Isomorphism Problem: Are two given graphs G_1, G_2, isomorphic?
   I.e., can the vertices of G_1 be re-numbered so that it becomes equal G_2?

   Solution: Checking all $n!$ enumerations of vertices is impractical
   (for $n = 100$, this exceeds the number of atoms in the known Universe).
   [Luks 80] found an $O(n^d)$ steps algorithm where d is the degree. This is a P-time for $d = O(1)$.
4. Independent Edges (Matching):
   Find a given number of independent (i.e., not sharing nodes) edges in a given graph.

   Solution: Max flow algorithm solves a bipartite graph case.
   The general case is solved with a more sophisticated algorithm by J. Edmonds.

Many other problems have been battled for decades or centuries and no P-time solution has been found. Even modifications of the previous four examples have no known answers:

1. Linear Programming: All known solutions produce rational x.
   No reasonable algorithm is known to find integer x.
2. Factoring: Given an integer, find a factor. Can be done in about exponential time $n^{\sqrt{n}}$.
   Seems very hard: Centuries of quest for fast algorithm were unsuccessful.
3. Sub-graph isomorphism: In a more general case of finding isomorphisms of a graph
   to a part of another, no P-time solution has been found, even for $O(1)$-degree graphs.
4. Independent Nodes: Find k independent (i.e., not sharing edges) nodes in a given graph.
   No P-time solution is known.

Exercise: Restate the above problems as inverting easily computable functions.

We learned the proofs that Linear Chess and some other games have exponential complexity. None of the above or any other search/inversion/NP problem, however, have been proven to require super-P-time. When, therefore, do we stop looking for an efficient solution?

NP-Completeness theory is an attempt to answer this question.
See results by S.Cook, R.Karp, L.Levin, and others surveyed in [Garey, Johnson 79, Trakhtenbrot 84].

A P-time function f reduces one NP-predicate $p_1(x)$ to $p_2(x)$ iff $p_1(x) = p_2(f(x))$, for all x. p_2 is NP-complete if *all* NP problems can be reduced to it. Thus, each NP-complete problem is at least as worst-case hard as all other NP problems. This may be a good reason to give up on fast algorithms for it. Any P-time algorithm for one NP-complete problem would yield one for all other NP (or inversion, or search) problems. No such solution has been discovered yet and this is left as a homework (10 years deadline).

Faced with an NP-complete problem we can sometimes restate it, find a similar one which is easier (possibly with additional tools) but still gives the information we really want. We will do this in Sec. 5.1 for factoring. Now we proceed with an example of NP-completeness.