Advanced Analysis of Algorithms

K. Subramani, LCSEE, West Virginia University

November 17, 30 2013

1. Theory of **NP-completeness**.
2. What is a problem? A question to be answered over several parameters with unspecified values and a constraint on the type of solution that is desired.
3. Instance of a problem - Instantiating the parameters of the problem.
4. Size of input. Length of input string.
5. An algorithm - Step by step decision procedure that takes as input an instance and returns the correct answer.
6. TSP problem example.
7. Time complexity function.
8. Efficient algorithms and intractable problems. Exponential time worst-case algorithms may actually work well in practice. Simplex.
9. Problem classification I: Decision, function, search, optimization.
10. Every decision problem Π consists of a set of instances D_Π, Y_Π and N_Π.
11. The motive behind this class. The Garey-Johnson story.
12. Problem classification II: Tractable, Intractable, evidence of intractability.
13. Reasons for intractability $\rightarrow$ definitional, undecidable, Presburger.
14. Problem as a formal language. Alphabet, Strings, language, problem and encodings. Binary encodings. Everything can be transformed based on encoding. Give graph shortest paths, example.
15. Problem is membership question. Use even numbers and prime numbers as languages. Membership question is interesting when language is infinite.
16. Associated with each decision problem Π is the language L_Π, which is the set of strings in the encoding, such that $x \in Y_\Pi$.
17. Algorithm as decider. What does it mean for an algorithm to decide L? Associated with an algorithm $\mathcal{A}$ is its language $L_\mathcal{A}$.
18. Deterministic algorithms and the class **P**.
19. Non-deterministic algorithms and the class **NP**. Guess and check algorithm. Easy verifiability. The tree of computations method to illustrate non-determinism. You count the depth of the tree and not the total number of computations.
20. Time complexity functions associated with both types of algorithms.

21. Without loss of generality, assume that degree of non-determinism is 2.
22. Main idea is from logic and theorem proving.
23. Simple problems. SAT, Circuit SAT, Vertex cover, clique, independent set, Independent set, hamilton path, hamilton circuit, TSP, Max cut, 2-Partitioning, scheduling on identical parallel machines, subset-sum, 0/1 knapsack.
24. Non-deterministic algorithms for simple problems.
25. Relationship between **P** and **NP**.
26. Notion of transformations. $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$ A function $f : \Sigma_1^* \rightarrow \Sigma_2^*$, such that $\forall x \in \Sigma_1^*$, $x \in L_1 \leftrightarrow f(x) \in L_2$. Also called reductions. f is called a transducer. Denoted by $L_1 \leq L_2$.
27. HC to TSP example.
28. Limits on f. Why needed? Polynomial time transformations. Mention log-space. Mention many-to-one.
29. If $L_1 \leq_p L_2$ and L_2 is in **P**, then L_1 is in **P**.
30. Transitivity of reductions.
31. Definition of NP complete. NP-hard. Optimization problems. TSP example.
32. If $L_1 \leq_p L_2$ and L_1 is **NPC**, then L_2 is in **NPC**.
33. **NP-complete** and **P** refer to sets of languages. Languages are sets.
34. Another way of thinking: How can I use an algorithm for problem L_2 to solve problem L_1?
35. Reductions order problems just like $\leq$ orders numbers.
36. Relation between **P** and **NP**.
37. **NP-completeness** is for decision problems only. For optimization problems, use a target.
38. Some common reductions. Graph-coloring to SAT. HP to SAT.
39. We need the first NPC problem. Cook's theorem.
40. Steps to show a problem is **NPC**:
    (a) Show that it is in **NP**.
    (b) Start with a good, **NP-complete** problem, say $\mathbf{P_1}$,
    (c) Find a suitable, polynomial-time transducer function f.
    (d) Reduce P_1 to our problem, using f.
41. 3SAT, 0/1 Integer Programming, Circuit-SAT.