

# Extending the Complexity Theory to Search Problems

- All of our discussion so far was focused on decision problems. As you'll recall, a decision problem  $\Pi$  is a set  $D_\Pi$  of all instances and a subset  $Y_\Pi \subseteq D_\Pi$  of yes-instances. Solving a decision problem  $\Pi \in NP$  with a nondeterministic algorithm means returning 'yes' in polynomial time when  $I \in Y_\Pi$ . We don't expect anything from the algorithm when  $I \in D_\Pi - Y_\Pi$ .
- We will now define a more general class of problems called the **search problems**
- A **search problem**  $\Pi$  consists of a set  $D_\Pi$  of all instances and, for each instance  $I \in D_\Pi$ , a set  $S_\Pi(I)$  of solutions for  $I$ .
- An algorithm is said to solve a search problem  $\Pi$  if, given as input any instance  $I \in D_\Pi$ , it returns the answer 'no' whenever  $S_\Pi(I)$  is empty; otherwise, the algorithm must return one of the solutions in the set  $S_\Pi(I)$ .
- **EXAMPLE :** Consider the Traveling Salesman **search** problem. The solution set  $S_\Pi(I)$  for each instance  $I$  consists ~~is~~ of all tours having the minimum possible length. An algorithm that solves this search problem must find at least one such tour for each instance  $I$ . (The goal of the TS search problem is to find a tour of minimum length.)
- It is important to note that every decision problem can be cast as a search problem by defining:
 
$$\begin{cases} S_\Pi(I) = \{\text{"yes"}\} \text{ if } I \in Y_\Pi \\ \quad \quad \quad = \emptyset \quad \text{if } I \notin Y_\Pi \end{cases}$$

► Note that when a decision problem is solved as a search problem, the computational burden increases because now the solution must return an answer for both positive and negative instances. When  $I \notin Y_\Pi$ , since  $S_\Pi(I)$  is empty, the answer returned will be "no" in accordance with the fourth bullet above.

A decision problem restated as a search problem may not belong in class  $NP$  on account of the computational difficulties associated with recognizing 'no' instances.
- In Lecture 25, we said that a decision problem is best thought of as a **language** for the purpose of formally defining the classes  $P$  and  $NP$ . In that lecture I mentioned that solving a decision problem is akin to solving the word **acceptance** problem in language recognition.
- A search problem also has a formal counterpart — it is called a **string relation**.

In order to talk about the time complexity of algorithms that solve search problems, we need a computational model that our solutions can be based on. The computational model we will use employs the notion of a function that "realizes" a string relation. So what is a string relation and what do we mean by realizing a string ~~is~~ relation? We will take these up in what follows.

- A string relation over a finite alphabet  $\Sigma$  is a set  $R$  as defined by  $R \subseteq \Sigma^+ \times \Sigma^+$  where  $\Sigma^+ = \Sigma^* - \{\epsilon\}$ 
  - $\epsilon$  the empty string previously, we used '0' for this
  - the set of all strings over  $\Sigma$
- We say that a function  $f: \Sigma^* \rightarrow \Sigma^*$  realizes the string relation  $R$  if for every  $x \in \Sigma^+$  we have  $f(x) = \epsilon$  when there is no  $y \in \Sigma^+$  such that  $(x, y) \in R$ , and such that  $f(x) = y$  when for some  $y$  we have  $(x, y) \in R$ .
- In analogy with the solution to a decision problem being representable by the recognition of a language by a DTM, solving a search problem can also be represented by a DTM, except that now the DTM computes a function, as opposed to just accepting or rejecting strings.
- A DTM program  $M$  realizes the string relation  $R$  if the function  $f_M$  computed by  $M$  realizes  $R$ .
- Earlier we said that a search problem  $\Pi$  consists of a set  $D_\Pi$  of all instances and, for each  $I \in D_\Pi$ , a set  $S_\Pi(I)$  of the solutions for  $I$ . We will now use an encoding scheme  $e$  to represent a search problem by a string relation  $\circledast$

$$R(\Pi, e) = \{ (x, y) \mid \begin{array}{l} x \in \Sigma^+ \text{ is the encoding under} \\ e \text{ of an instance } I \in D_\Pi \text{ and} \\ y \in \Sigma^+ \text{ is an encoding under } e \\ \text{of a solution } s \in S_\Pi(I) \end{array} \}$$

- We say that a search problem  $\Pi$  (under encoding scheme  $e$ ) is solvable by a polynomial time algorithm if there is a polynomial time DTM program that realizes the relation  $R(\Pi, e)$ .
- What we have above is the search-problem equivalent of the class P for decision problems.

### AN INFORMAL DEFINITION OF NP-HARD :

- Consider the scenario shown at right where we have a presumably difficult decision problem  $\Pi$  whose solution can be deduced from the solution to some search problem  $\Pi'$ .
  - $\Pi'$  (this may be a search or an optimization problem)
  - $\Pi$  NP-complete
  - NP
  - means we can deduce the solution to  $\Pi$  from the solution to  $\Pi'$
- It stands to reason that  $\Pi'$  must be at least as hard as  $\Pi$  since otherwise we would be able to solve  $\Pi$  in polynomial time by making a subroutine call to  $\Pi'$  (assuming that the deduction of the solution to  $\Pi$  from the solution to  $\Pi'$  can also be carried out in polynomial time. [Note that easy solutions are polynomial time solutions and hard solutions are of complexity that is exponential or worse.]
- If the deduction of the answer to  $\Pi$  from the answer to  $\Pi'$  can be made in polynomial time, we say that the set of all such  $\Pi'$  constitute the class NP-Hard.