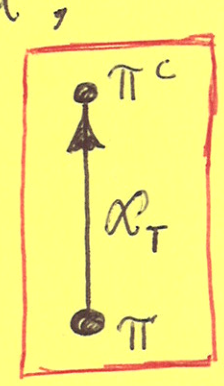
Lecture 38
ECE 664

Purdue Univ.
A.C. KAK

Complements and Optimization Versions of the NP-Complete Decision Problems

- When a decision problem Π is NP-Complete, its complement, denoted Π^c, is NP-Hard. This is for the simple reason that we can set up a Turing reduction from Π to Π^c. Stated simply, what that means is that we can immediately deduce the answer to Π if an oracle supplies us with the answer to Π^c.



- Consider, for example, the problem TS whose decision question is:

  "Is it true that there exists a tour of length less than B?"

  Now consider the decision question asked by TS^c:

  "Is it true that there exists no tour of length less than B?"

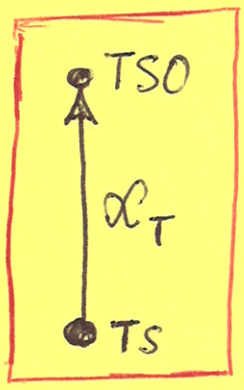
  Obviously, if the Oracle of Delphi would be willing to answer the TS^c question for us, we would immediately have the solution to TS also.

- While we can always establish a Turing reduction from Π to Π^c (and, for that matter, from Π^c to Π), we obviously cannot always establish a polynomial transformation from Π to Π^c because of the reversed roles of the 'yes' and 'no' answers.

- So with regard to the complements of problems, our conclusion is that if Π is NP-Complete or NP-Hard, Π^c must be NP-Hard.

Optimization Versions of NP-Complete Problems

- Consider the Traveling Salesman Optimization (TSO) problem. We say an algorithm solves TSO if for every instance $I \in D_{TSO}$ the algorithm returns at least one least cost path. [Note that TSO is fundamentally a search problem.]

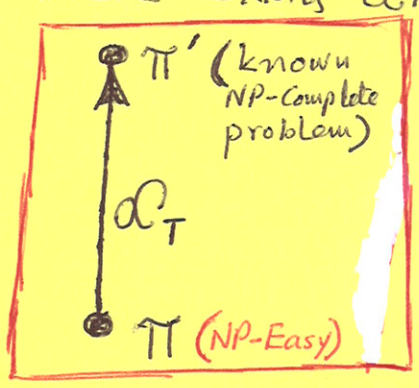


- We obviously have a Turing reduction from TS to TSO. If the Oracle of Delphi supplies us with an answer to TSO, all we need to do in order to solve TS is to compare the path-length in the solution returned by the oracle against the bound B specified in the instance of TS.

- Therefore, since TS is NP-Complete, TSO must be NP-Hard.
- In general, whenever there exists an optimization version of an NP-Complete problem, the optimization version will be NP-Hard.

NP-Easy Class of Problems

- A search/optimization problem is NP-Easy whenever there exists an NP-Complete problem Π' such that $\boxed{\Pi \propto_T \Pi'}$



- So if a new search/optimization problem Π Turing reduces to a known NP-Complete problem Π', we know that the new problem Π is no harder than Π'.
- We have already shown that TSO is NP-Hard because of $TS \propto_T TSO$. We will show next that TSO is also NP-Easy because $TSO \propto_T TS$.
- In order to prove $TSO \propto_T TS$, we need to first define a new intermediate problem that we will call Traveling Salesman Extension (TSE). Our proof of $TSO \propto_T TS$ will consist of demonstrating $\boxed{TSO \propto_T TSE \propto_T TS}$.

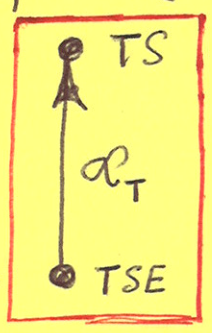
TRAVELING SALESMAN EXTENSION (TSE):

INSTANCE: A finite set $C = \{c_1, c_2, \ldots, c_m\}$ of cities, pairwise intercity distances $d(c_i, c_j)$ for every pair of cities, a bound $B \in Z^+$, a partial tour $\Theta = (c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(K)})$ of K distinct cities, $1 \le K \le m$.

$d(c_i, c_j) \in Z^+$

QUESTION: Can Θ be extended into a full tour of length B or less?

- It is obviously the case that $\boxed{TSE \in NP}$ since a guess at the extension of Θ into a full tour can be verified in polynomial time.
- For the purpose at hand, it is NOT important to know ~~that~~ whether TSE is also NP-Complete. However, because $TSE \in NP$, we can be certain that $TSE \propto TS$. [There is no need to supply a rigorous proof for this assertion since, theoretically speaking, every problem in NP can be polynomially transformed to any NP-Complete problem, even if that means going through SAT and working our way up the tree of the basic NP-Complete problems shown in Lecture 29.]
- It was stated in Lecture 37 that polynomial transformability is a special case of Turing reducibility and that $\Pi_1 \propto \Pi_2$ implies $\Pi_1 \propto_T \Pi_2$. Therefore, $TSE \propto TS$ implies $\boxed{TSE \propto_T TS}$.



- That completes one part of proving the overall assertion $\boxed{TSO \propto_T TSE \propto_T TS}$.
- We will next prove $TSO \propto_T TSE$.

PROOF OF TSO $\propto_T$ TSE :

- Let $\boxed{S(C, d, \Theta, B)}$ be a function that solves the TSE problem by consulting the Oracle of Delphi [although, with the approaching winter, that does create a difficulty for us since this oracle (who was actually the priestess Pythia in a trance in the Temple of Apollo in Delphi, Greece) did not work in the winter months].

- The parameters C, d, Θ, and B in the function header $S(C, d, \Theta, B)$ are the same as in the instance description for TSE.

- Our proof will consist of two separate parts :

① The algorithm for TSO will first consult the oracle to determine the length B^* of the shortest tour. This will require multiple consultations with the oracle. However, the number of times the oracle will be consulted will be bounded by the log of the maximum possible value for B^*. [The important point to remember is that, for Turing reducibility, the total amount of work required to fetch information from the oracle must be bounded by a polynomial in the size of the base problem. As a consequence, if multiple consultations with the oracle are required, the number of such consultations must be bounded by a polynomial. A lograrithm can always be bounded by a polynomial, but not the other way around.]

② After finding the length B^* of the shortest tour, the TSO algorithm will again consult the TSE oracle, but only a polynomial number of times, to find the tour of the cities for which the total length equals B^*. This will be the optimum tour.

- Consultations with the Oracle for Finding B^* :

  - We can always assume that the optimum tour starts at city c_1. Note that any full tour of the cities can be cyclically permuted, without altering the length of the tour, so that c_1 is the start city of the tour.

  - Since we assumed $d(c_i, c_j) \in Z^+$, the length B^* of the optimal tour must lie between $\boxed{B_{MIN} = m}$ and $\boxed{B_{MAX} = m \cdot d_{MAX}}$ where $\boxed{d_{max}}$ is the largest value of $d(c_i, c_j)$.

  - The TSO algorithm will now consult the TSE oracle to compute B^* through the following binary search procedure :

STEP 1 : Set $B_{MIN} \leftarrow m$ and $B_{MAX} \leftarrow m \cdot d_{MAX}$

STEP 2 : If $\boxed{B_{MAX} - B_{MIN} = 1}$, set $B^* = B_{MIN}$ and HALT

STEP 3 : Set $\boxed{B \leftarrow \frac{B_{MIN} + B_{MAX}}{2}}$ and call $\boxed{S(C, d, (c_1), B)}$.

If the oracle returns 'yes', that means it is possible to extend c_1 to a full tour of length B or less. So we set $\boxed{B_{MAX} \leftarrow B}$ and go back to step 2.

If the oracle returns 'no', that means it is NOT possible to extend c_1 to a full tour of length B or less. So we set $\boxed{B_{MIN} \leftarrow B}$ and ~~g~~ go back to step 2.

The above binary search procedure will determine the exact value of B^* by calling the TSE oracle at most $\lceil \log_2 B_{MAX} \rceil$ number of times where $B_{MAX} = m \cdot d_{MAX}$.

- With the value of B^* known, the TSO algorithm again consults the TSE oracle a multiple number of times to construct the optimum tour. The TSO algorithm uses the notion of extendible partial tour for this purpose. A partial tour is an $\boxed{\text{extendible partial tour}}$ if it can be extended into a full tour of length B^*. The TSO algorithm works in the following fashion:

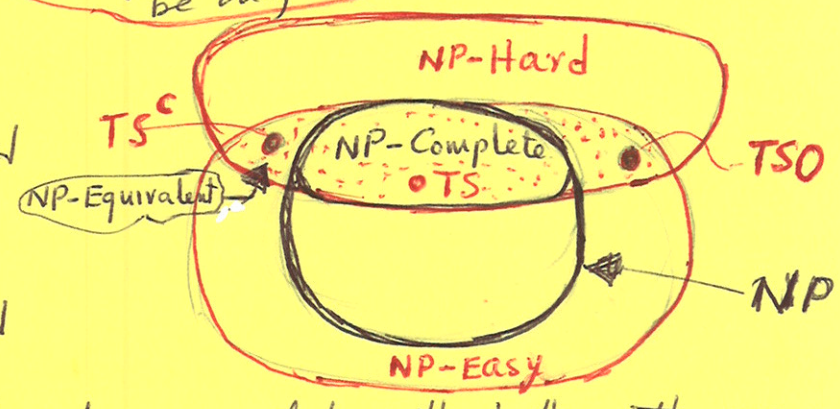
  Start with (c_1) as an extendible partial tour. Next, find a city c_j from the remaining cities so that (c_1, c_j) is an extendible partial tour. In the worst case, this will require $m-2$ calls to the oracle of type $\boxed{S(C, d, (c_1, c_j), B^*)}$.

  [There are obviously $m-1$ choices for c_j. But, if the first $m-2$ choices do not work out, the only remaining choice must.] After the selection of the second city, now find the third city so that (c_1, c_j, c_k) is an extendible partial tour. In the worst case, this will require $m-3$ calls to the oracle; and so on. Reasoning in this manner, we conclude that the total number of calls to the oracle for constructing the full tour will be

$$(m-2) + (m-3) + \cdots\cdots + 1 = \frac{(m-1)(m-2)}{2}$$

For the $(m-1)^{th}$ city. No calls need be made for the last city on the full tour since there will be only choice left at that point

- We have therefore proved $\boxed{TSO \propto_T TSE}$. Since earlier we established that $TSE \propto_T TS$, it follows from the transitivity of $\propto_T$ that $\boxed{TSO \propto_T TS}$, implying that TSO is NP-Easy. But previously we showed that $\boxed{TS \propto_T TSO}$, implying that TSO is NP-HARD.



- Because TSO is both NP-Easy and NP-Hard vis-a-vis TS, we say TS and TSO are NP-Equivalent. That means, the difficulty level associated with both is the same