Lecture 35
ECE 664

# Strong-Sense and Weak-Sense NP-Completeness for Number Problems

Purdue Univ
A. C. Kak

- We refer to a decision problem as a number problem if it is possible for the numbers involved in a generic instance to be arbitrarily large. That is, if max(I) is the value of the largest integer in the description of the instance I of a decision problem $\Pi$, we call $\Pi$ a number problem if max(I) can be arbitrarily large regardless of the size length(I). More formally, $\Pi$ is a number problem if there exists no polynomial p such that $\max(I) \leq p(\text{length}(I))$ for all $I \in D_\Pi$.

- Some examples of number problems: PARTITION, KNAPSACK, TS, etc.

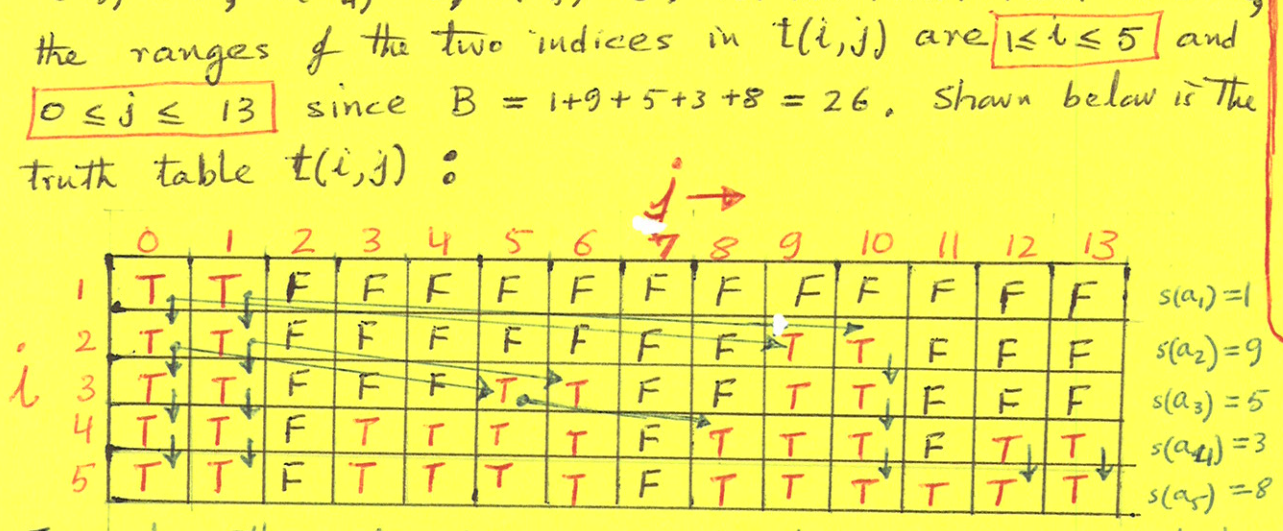
- You already know that PARTITION is an NP-Complete problem. Let's now examine the following dynamic programming based approach to solving the problem:

Let $A = \{a_1, a_2, \ldots, a_n\}$ and the sizes $s(a_1), s(a_2), \ldots$ constitute an arbitrary instance of PARTITION. Set $B = \sum_{a \in A} s(a)$. We assume B is even since otherwise the PARTITION instance has no solution. Now, for integers $1 \leq i \leq n$ and $0 \leq j \leq B/2$, let $t(i,j)$ denote the truthvalue of the statement:

> There is a subset of $\{a_1, a_2, \ldots, a_i\}$ for which the sum of the item sizes is exactly j.

(note)

EXAMPLE: $A = \{a_1, a_2, a_3, a_4, a_5\}$ and $s(a_1) = 1$, $s(a_2) = 9$, $s(a_3) = 5$, $s(a_4) = 3$, $s(a_5) = 8$. For this PARTITION instance, the ranges of the two indices in $t(i,j)$ are $1 \leq i \leq 5$ and $0 \leq j \leq 13$ since $B = 1+9+5+3+8 = 26$. Shown below is the truth table $t(i,j)$:




The algorithm for filling up the table can be stated in the following manner:

$i = 1$ (first row): $t(1, j) = T$ iff either $j = 0$ or $j = s(a_1)$

$i > 1$ : $t(i,j) = T$ iff $t(i-1, j - s(a_i)) = T$

The important thing to note here is that to figure out the entries in any row, we only need to examine the entries in the immediately previous row. This is supposed to be demonstrated by the arrows superimposed on the table. ▶ Once the entire table has been filled in, we have solved the given instance of PARTITION because the answer is 'yes' if $t(n, B/2) = T$.

- Let's now examine how much work is required to fill up the $n \times \frac{B}{2}$ table:

> Ordinary recursion can involve an exponentially growing number of computational steps. Consider the classic example of calculating the Fibonacci sequence. If you directly implement the formula $f(n) = f(n-1) + f(n-2)$, you will end up evaluating the "leaf nodes" f(0), f(1), etc, exponentially growing number of times as n becomes larger and larger. Dynamic Programming, which strictly speaking has more to do with the organization of the calculation steps involving different memory/execution tradeoffs than with computer programming, allows us to control this exponential growth either through memoization (in top-down implementations) or through the translation of recursive calls into non-recursive executions.

Since the entries in each row depend on only the entries in the previous row, and that too with each entry depending on looking up a specific location in the previous row (that is dynamic programming in action), the amount of work required to fill up a row is some low-order polynomial in the number of columns (which is $B/2$). ➤ Therefore, we can claim that the total amount of work required to fill up the entire $n \times \frac{B}{2}$ table is a low-order polynomial in $nB$. This leads us to the conclusion:

$$\text{time complexity of PARTITION} < (nB)^k$$

for some value for the integer $k$. DOES THIS MEAN WE HAVE DISPROVED THE NP-COMPLETENESS OF PARTITION? No, not at all. The theory of NP-Completeness requires us to use efficient representations for problem instances. An instance of PARTITION is merely a sequence of $n$ integers. If we use $B$ as a generous upper bound on all $n$ integers, we should be able to represent an instance of PARTITION by a string of length $c \cdot n \cdot \log_2 B$ since each integer $s(a)$ can be represented by a string whose length is bounded by $\log_2 B$. The constant $c$ takes care of any marker symbols we may use to separate the consecutive $s(a)$'s. ➤ Given a concise representation of PARTITION, the theory of NP-Completeness says that for PARTITION to belong to class P, its time complexity function must be bounded by a polynomial in the length of that representation. In other words, for PARTITION to belong to class P, we must have

$$\text{time complexity of PARTITION} < (n \cdot \log B)^m$$

for some integer $m$. Now, for any given non-zero $k$, $(n \cdot B)^k$ cannot be bounded by $(n \cdot \log B)^m$ for any $m$ for arbitrary values of $n$ and $B$. Therefore, the dynamic programming based solution to PARTITION does not violate the NP-Completeness of this problem. ● Nonetheless, we can claim that if we assume an upper bound on the sizes of the numbers $s(a)$, PARTITION can be solved ~~be~~ in low-order polynomial time. That is,

$$\text{time complexity of PARTITION with a priori known upper bound on } B < n^k$$

for some integer $k$. What this tells us is that the exponential time complexity of PARTITION is less a consequence of any combinatorial considerations and more a consequence of the sizes of the numbers involved.

> Algorithms that solve in polynomial time those subproblems that are generated from the general problem by placing restrictions on the size of the numbers involved are called
>
> **Pseudo-Polynomial Time Algorithms**
>
> for solving the general problem.

● It is most important to note that a pseudo polynomial time algorithm can be useful even when there is no bound available on the numbers involved in instance representations. Such algorithms display exponential behavior only when confronted with instances containing exponentially large numbers. Instances of that sort might be rare.

● That sets us up to define NP-Completeness in the strong sense and in the weak sense.

- Definition of NP-Completeness in the strong sense:

For any decision problem $\Pi$ and any polynomial $p$ over the integers, let $\Pi_p$ denote the **subproblem of $\Pi$** obtained by restricting $\Pi$ to only those instances $I$ that satisfy:

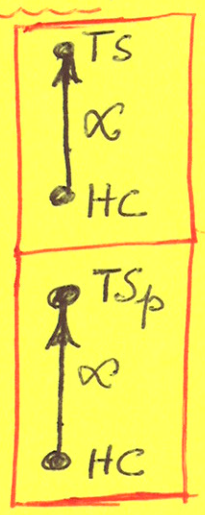
$$\max(I) \leq p(\text{length}(I))$$

where $\max(I)$ is the largest number in $I$ and $\text{length}(I)$ the size of $I$ under a reasonable encoding scheme.

A decision problem $\Pi$ is considered to be NP-Complete in the strong sense if $\Pi \in$ NP-Complete and there exists a polynomial $p$ over the integers for which $\Pi_p$ is also NP-Complete.

- A decision problem that is NOT NP-Complete in the strong sense is considered to be NP-Complete in a weak sense.
- A decision problem that is NP-Complete in the weak sense can be solved by a pseudo polynomial time algorithm.
- Obviously, PARTITION is NP-Complete in the weak sense. For contrast, the Traveling Salesman (TS) problem is NP-Complete in the strong sense.

[Proof: The NP-Completeness proof of TS consists of the HC $\propto$ TS transformation that was presented in Lecture 27. That transformation shows how any arbitrary instance of HC can be transformed into an instance of TS. But note that the mapped instance of TS thus obtained corresponds to the $TS_p$ subproblem of TS. [In accordance with the definition presented above, the subproblem $TS_p$ is obtained by restricting TS to only those instances that satisfy $\max(I) \leq p(\text{length}(I))$.] So, in effect, the HC $\propto$ TS transformation in Lecture 27 is actually a HC $\propto$ $TS_p$ transformation. Thus, $TS_p$ is NP-complete. This proves, TS is NP-Complete in the strong sense.]




## A Strong-Sense NP-Complete Number Problem That Serves as a Base Problem for Proving Strong-Sense NP-Completeness of Several Other Number Problems:

- So far you have seen one weak-sense NP-Complete number problem, PARTITION, and one strong-sense NP-Complete number problem, TS.
- To prove strong-sense NP-Completeness of a general number problem, it is best done if you can construct a polynomial transformation from a previously known strong-sense NP-complete problem. But it is not convenient to use TS for that purpose. The number problem that has proved to be more useful in serving such a role is 3-PARTITION that is presented

next. (For that reason, 3-PARTITION is also referred to as the seventh basic NP-Complete problem.)

**3-PARTITION :**

INSTANCE : A finite set $A$ of $3m$ elements, a bound $B \in Z^+$, and a size $s(a) \in Z^+$ for each $a \in A$ that satisfies $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = mB$.

QUESTION : Can $A$ be partitioned into $m$ disjoint set $A_1, A_2, \ldots, A_m$ such that $\sum_{a \in A_i} s(a) = B$ ?

Note that the constraint on $s(a)$ implies that, when a solution exists, each set $A_i$ will contain exactly 3 elements. Since $s(a) < B/2$, two elements will never suffice, and since $s(a) > B/4$, four elements will be too many.

- To prove that 3-PARTITION is NP-Complete in the strong sense, we must first prove that a related problem, 4-PARTITION, is NP-Complete in the strong sense.
- 4-PARTITION is identical to 3-PARTITION except that the set $A$ now contains $4m$ elements and that each $s(a)$ must satisfy : $B/5 < s(a) < B/3$
- The NP-Completeness of 4-PARTITION is established by : → 4-PARTITION $\propto$ 3DM

  As you'd expect, this proof consists of showing how an arbitrary instance of 3DM can be transformed into an instance of 4-PARTITION.
- As it turns out, the instances of 4-PARTITION constructed in the above transformation can be shown to obey the $\max(I) \leq p(\text{length}(I))$ constraint. Therefore, the 3DM $\propto$ 4-PARTITION transformation also proves that 4-PARTITION is NP-Complete in the strong sense.
- The NP-Completeness in the strong sense of 3-PARTITION is now established by the following polynomial transformation : → 3-PARTITION$_{p'}$ $\propto$ 4-PARTITION$_p$

  Note that this transformation takes us directly from the 4-PARTITION instances that satisfy $\max(I) \leq p(\text{length}(I))$ for some polynomial $p$ to the 3-PARTITION instances that satisfy $\max(I) \leq p'(\text{length}(I))$ for some other polynomial $p'$. This therefore constitutes a proof of strong-sense NP-Completeness of 3-PARTITION.

---

- Ordinarily, in order to prove that a problem $\Pi$ is strong-sense NP-Complete, you must establish that $\Pi_p$ is NP-Complete in the regular sense. A more convenient way to do the same is through what is known as a pseudo polynomial transformation
- A pseudo polynomial transformation from problem $\Pi_1$ to problem $\Pi_2$ is a function $f : D_{\Pi_1} \to D_{\Pi_2}$ such that : ① for all $I \in D_{\Pi_1}$, $I \in Y_{\Pi_1}$ iff $f(I) \in Y_{\Pi_2}$ ; ② $f$ can be computed in time polynomial in $\max(I)$ and $\text{length}(I)$ ; ③ for all $I \in D_{\Pi_1}$, there exists a polynomial $q_1$ such that $q_1(\text{length}(f(I))) \geq \text{length}(I)$ ; and ④ for some two-variable polynomial $q_2$ we have $\max(f(I)) \leq q_2(\max(I), \text{length}(I))$.
- LEMMA : If $\Pi_1$ is NP-Complete in the strong sense, $\Pi_2 \in NP$, and there exists a pseudo polynomial transformation from $\Pi_1$ to $\Pi_2$, then $\Pi_2$ is NP-Complete in the strong sense.