Assignment 7

Ayman Shahriar UCID: 10180260 Tutorial: T02

December 15, 2020

Question 1

If $P' \leq_p P$, and P' does not have a polynomial time solution, then it is impossible for P to have a polnomial time solution.

We will prove this by contradiction:

Suppose that P has a polynomial time solution. Consider the function $f'(x)$ that is defined in the question. It is stated that the functions **transform1** and **transform2** run in polynomial time. So their runtime is in $O(n^c)$ for some constant c. And since we assumed that p has a polynomial time solution, the runtime of $f(x)$ is in $O(n^d)$ for some constant d.

Then the runtime of $f'(x)$ is $O(n^c) + O(n^d) = O(n^e)$ for some constant e. Since $f'(x)$ runs in polynomial time, this means that P' has a polynomial time solution, which contradicts the assumption that P' does not have a polynomial time solution.

Thus, if $P' \leq_p P$, and P' does not have a polynomial time solution, then P will not have a polynomial time solution.

Question 2)

Yes.
a) We know that $Interval\ Scheduling \geq_p Independent\ Set$.
And from corollary 1.14 of the slides "CPSC413-8NP_ha", we know that $Vertex\ Cover =_p Independent\ Set$.
So $Independent\ Set \geq_p Vertex\ Cover$, and by the transitive property of $\geq_p$, we get $Interval\ Scheduling \geq_p Vertex\ Cover$

b) Unknown, because it would resolve the question of whether $P = NP$.

We know that Interval Scheduling $\in$ P. So if
$IndependentSet \leq_p IntervalScheduling$, then by definition of $\leq_p$ it would

mean that there is an algorithm that solves $IntervalScheduling$ in polynomial time.

And since $IndependentSet \in NPComplete$, it would mean that all $NPComplete$ problems would have a polynomial time algorithm, which would imply that $P = NP$.

And if $IndependentSet \nleq_p IntervalScheduling$, then since all problems in P are poly-time reducible to each other, it would mean that Independent Set does not have a polynomial time algorithm. And since $IndependentSet \in NP - Complete$, this would mean that all $NP - Complete$ problems do not have a polynomial time algorithm, implying that $P \neq NP$.

Question 3)

Proof that $Bin - Packing \in NP$**:**

1) Input of the verification algorithm:

Input to the problem: A sequence $S = s_1, s_2, \ldots, s_n$ of real numbers where each $s_i \in [0, 1]$, and a positive integer K.

Certificate: A sequence of sets of integers $B_1, \ldots, B_m$, where
$B_1 \cup \ldots \cup B_m = \{1, \ldots, n\}$
(We are assuming that there are no empty bins, the sets only contain 1 to n, and there no repeating numbers)

---

2) Show that certificate size is polynomial in size to remaining input:

Since $B_1 \cup \ldots \cup B_m = \{1, \ldots, n\}$ and there cannot be any empty bins, this means that $m \leq n$, and that each $|B_i| \leq n$.

So the length of our certificate $B_1, \ldots, B_m$ is in $\Theta(n)$, where n is the size of the input sequence.

---

3) Give the verification algorithm Note: We will assume that $K \leq n$ (according to Sarah)

```
Function CertifyBinPacking($S = (s_1, s_2, \ldots, s_n)$, $K$, $(B_1, \ldots, B_m)$)

    // check if there are too many bins
    if $m > K$: then
        return 0

    // check if any bin is overflowing
    for $\{i = 1, \ldots, m\}$: do
        if $\sum_{j \in B_i} s_j > 1$ then
            return 0

    // If none of these conditions occur, return true
    return 1
```

---

4) Prove that the verification algorithm is correct:

In order to prove that our certifier algorithm B is correct, we need to prove that:

$$x \in X \iff \exists C : B(x, C) = 1$$

(Where X is the decision problem, x is an instance of the decision problem, and C is a certificate of the decision problem)
(Also not that $x \in X$ means that "x is a yes-instance of decision problem X")

Proof that $\quad x \in X \longrightarrow \exists C : B(x, C) = 1$:

Suppose the input $(s_1, \ldots, s_n)$ and K return "yes" for the Bin Packing decision problem.

Then this means that we can partition $s_1, \ldots, s_n$ into $m < K$ sets such that the sum of each set is less than or equal to 1 (and there are no empty sets).

If we represent each s_i with i, then this will be the same as partitioning $\{1, \ldots, n\}$ into $m < K$ sets $B_1, \ldots, B_m$ such that $\sum_{j \in B_i} s_j \leq 1$

So if we use $B_1, \ldots, B_m$ as our certificate, then the two conditions checked in the certifier will not occur and then the certifier will return 1, as required.

□

Proof that if $\exists C : B(x, C) = 1$, then $x \in X$:

Suppose the inputs $(s_1, \ldots, s_n)$, K and $(B_1, \ldots, B_m)$ return 1 on the certifier algorithm.

Let us partition $s_1, \ldots, s_n$ into bins $B'_1, \ldots, B'_m$ where $B'_i = \{s_j \mid j \in B_i\}$.

Since the certifier algorithm returned 1, then it must mean that the first condition checked in the algorithm did not occur, so $m \leq K$ (ie, we do not have too many bins).

And also, it also means that the second condition did not occur, so $\sum_{j \in B_i} s_j \leq 1$, which in turn means that $\sum_{j \in B_i'} j \leq 1$.
This means that none of our bins are overflowing.

Then since the bins are less than K and are not overflowing, it means that $((s1, \dots, s_n), K)$ is a yes instance of the Bin Packing decision problem, as required.

$\square$

---

5) Show that the verification/certification algorithm runs in polynomial time:

In our algorithm, we check two conditions:

First we check if there are too many bins by comparing m with k, which takes constant time $O(1)$.

Second, we check if each bin is overflowing by checking if there exists any B_i such that $\sum_{j \in B_i} s_i > 1$. So the runtime of this step will depend on the number of bins as well as the number of elements within each bin. From the description of the input, we know that $B_1 \cup \dots \cup B_m = \{1, \dots, n\}$, This means that the total number of elements in all the bins will always be n.

And since there cannot be any empty bins, this also means that the maximum number of bins cannot exceed n (ie, in the worst case, $m = n$).

So in the worst case, the runtime of this step will be in $O(n^2)$

So the runtime of our algorithm will be $O(1) + O(n^2) = O(n^2)$, where n is the size of $s_1, \dots, s_n$.

Thus, the certification/verification algorithm runs in polynomial time

Proof that PARTITION $\leq_P$ BIN PACKING:

1) Give an algorithm to transform an input to PARTITION into an input to BIN PACKING:

Function Transform$(m_1, \dots, m_n)$:

Let $M = \sum_{i \in \{1, \dots, n\}} m_i$ (so M is the sum of the sequence given as input)

```
For $i = 1, \dots, n$:
if $m_i > \frac{M}{2}$
// return an arbitrary "no" for bin packing
return ((1, 1), 1)
```

```
Let $k = 2$
Initialize empty sequence S

For $i = 1, \dots, n$:
$s_i = \frac{2m_i}{M}$
Add $s_i$ to end of S

return $(S, k)$
```

2) Prove that the transformation algorithm runs in polynomial time.

The first thing we do in the algorithm is to calculate the sum of the sequence, whose runtime is dependent on the length of the sequence. Since the length of the sequence is n, then this step runs in $O(n)$ time.

In the second step, we check if there is any element in the sequence whose value is greater than $\frac{M}{2}$. Again, this step is dependent on the length of the sequence, so the runtime of this step is in $O(n)$

And in the final step, we multiply each element in the sequence with $\frac{2}{M}$ and add it to our newly created set S. This step is also dependent on the length of the input sequence, so the runtime of this step is in $O(n)$

So the runtime of the transformation algorithm is in
$O(n) + O(n) + O(n) = O(n)$.

Thus, the transformation algorithm runs in polynomial time

3)

Suppose $(m_1, \dots, m_n)$ is an input to our transformation algorithm and $((s_1, \dots, s_n), k)$ will be the transformed output produced by the algorithm. We will prove the correctness of our transformation algorithm by showing that $(m_1, \dots, m_n)$ is a "yes" instance of PARTITION if and only if $((s_1, \dots, s_n), k)$ is a "yes" instance of BIN PACKING.

Proof of "$\Rightarrow$":

Suppose $(m_1, \dots, m_n)$ is a "yes" instance of PARTITION.
Then there must not exist any m_i such that m_i is greater than $M/2$ (where M is the sum of the sequence). So for "yes" instances of PARTITION, our algorithm will not return an arbitrary "no" input for BIN PACKING.

Our algorithm transforms each m_i into s_i by dividing m_i by $M/2$. And since $(m_1, \dots, m_n)$ is a "yes" instance of PARTITION, this means that we can partition $(m_1, \dots, m_n)$ into P_1 and P_2 such that the sum of P_1 = sum of P_2 = $M/2$.

Then this means that the sum of $P_1/(M/2)$ = sum of $P_2/(M/2)$ = $(M/2)/(M/2) = 1$. In other words, we can partition the transformed input

$(s_1, \ldots, s_n)$ into two partitions such that the sum of each partition will be 1. Then since the size of a bin in the BIN PACKING problem is 1, we will be able to allocate all of our s_i into two bins (where all the elements from a partition will be put into one bin).

So if we set $k = 2$ and ask BIN PACKING about inputs k and $(s_1, \ldots, s_n)$, it will return yes since it's possible to allocate all $(s_1, \ldots, s_n)$ into two bins without overfilling.

Thus, $((s_1, \ldots, s_n), k)$ is a yes instance of BIN PACKING

Proof of "$\Leftarrow$":

Suppose this output $((s_1, \ldots, s_n), k)$ is a "yes" instance of BIN PACKING. We want to prove that $(m_1, \ldots, m_n)$ is a "yes" instance of PARTITION

Now, since the transformed output is a "yes" instance of PARTITION, this must mean that the algorithm skipped over the step where it returns an output that is an arbitrary "no" for PARTITION.

This in turn means that for all $i \in \{1, \ldots, n\}$, $m_i \leq \frac{M}{2}$ (where M is the sum of the input sequence). Also, this means that the algorithm assigned $k = 2$ since it skipped returning the arbitrary "no" output.

Since each $s_i = \frac{2m_i}{M}$, this means that the sum of all elements $(s_1, \ldots, s_n)$ is 2. And since $((s_1, \ldots, s_n), k)$ is a "yes" instance of BIN PACKING and $k = 2$, this means that we can partition $(s_1, \ldots, s_n)$ into S_1 and S_2 such that the sum of each partition is 1.
Then since each $m_i = s_i * \frac{M}{2}$, it follows that we can divide $(m_1, \ldots, m_n)$ into two partitions P_1 and P_2 (whose elements correspond to the transformed elements in S_1 and S_2) such that the sum of each partition P_1 and P_2 is $1 * \frac{M}{2} = \frac{M}{2}$

Thus, $(m_1, \ldots, m_n)$ is a "yes" instance of PARTITION

Question 4)

Proof that $MAGNETS \in NP$:

1) Input of the verification algorithm:

Input to the problem: A set $M = \{m_1, \ldots, m_l\}$ of l magnets, where each magnet $m_i \in S = \{s_1, \ldots, s_n\}$ (one of the available symbols)
A set $W \subseteq S^*$ (a set of words)

Certificate: A multiset/sequence of words $w_1, \ldots, w_k$, where each $w_i \in W$

2) Show that certificate size is polynomial in size to remaining input:

In order for the verification algorithm to return 1, the certificate cannot contain more letters than the set of magnets.

So this means that the number of letters in the sequence of words $w_1, \dots, w_k$ has to equal the number of letters available with the magnets $\{m_1, \dots, m_l\}$.

So it must be the case that the number of letters in $w_1, \dots, w_k \leq l$. So the length of our certificate will be in $O(l)$.

---

3) Give the verification algorithm

```
Function CertifyMagnets($M = \{m_1, \dots, m_l\}$, $S = \{s_1, \dots, s_n\}$, $W$, $w_1, \dots, w_k$)

    // Check if the number of letters in $M$ is not equal to the number of letters in $w_1, \dots, w_k$
    if $|M|$ != $\sum_{w \in W} |w|$: then
        return 0

    // At this point we know that number of letters is same in $M$ and $w_1, \dots, w_k$
    // Check if all the letters in $w_1, \dots, w_k$ make up all the letters in $M$

    Create arrays $A = \{0, \dots, 0\}$ and $B = \{0, \dots, 0\}$ each of size $l$

    for $i = \{1, \dots, l\}$ do
        Let $s_j$ be the letter that $m_i$ corresponds to
        A[$j$]++

    for each $w \in (w_1, \dots, w_k)$ do
        for $i = \{1, \dots, |w|\}$ do
            Let $s_j$ be the $i^{th}$ letter of $w$
            B[$j$]++

    if A != B: then
        return 0

    return 1
```

---

4) Prove that the verification algorithm is correct:

In order to prove that our certifier algorithm B is correct, we need to prove that:

$$x \in X \iff \exists C : B(x, C) = 1$$

(Where X is the decision problem, x is an instance of the decision problem, and C is a certificate of the decision problem)
(Also note that $x \in X$ means that "x is a yes-instance of decision problem X")

Proof that $\quad x \in X \longrightarrow \exists C : B(x, C) = 1$:

Suppose the input $M = \{m_1, \ldots, m_l\}$, $S = \{s1, \ldots, s_n\}$, W return "yes" for the Magnets decision problem.

Then there must exist a set of words $w_1, \ldots, w_k \in W$ such that the letters that make up all these words is equal to M.

This means that the number of letters that make up M is equal to that of $w_1, \ldots, w_k$, and that the number of occurrences of each letter in M is equal to that of $w_1, \ldots, w_k$.

So if we use $w_1, \ldots, w_k$ as the certificate, the two conditions checked by the certifier/verifier algorithm will not occur, and so the certifier will return 1, as required.

$\square$

Proof that if $\exists C : B(x, C) = 1$, then $x \in X$:

Suppose the inputs $M = \{m_1, \ldots, m_l\}$, $S = \{s1, \ldots, s_n\}$, W, $w_1, \cdots, w_k$ return 1 on the certifier algorithm.

Then this means that the two conditions check by the certifier algorithm did not occur.

So both M and the certificate $w_1, \ldots, w_k$ have the same number of letters, as well as the same number of occurrences of each letter.

This means that the magnets of M can make up all the words of $w_1, \ldots, w_k$ (with no magnet left over), so the inputs $M = \{m_1, \ldots, m_l\}$, $S = \{s1, \ldots, s_n\}$, W will return yes for the decision problem Magnets, as required.

$\square$

---

5) Show that the verification algorithm runs in polynomial time

In our algorithm, we check two conditions:

First we check if the number of letters in M and $w_1, \ldots, w_k$ are the same. So the runtime will depend on the number of letters in M plus $w_1, \ldots, w_k$. We know that the number of letters in M is equal to l and we also know from part 2) that the number of letters in $w_1, \ldots, w_k$ is in $O(l)$.

So the runtime of this step will be in $O(l^2)$

Checking the second condition involves two loops and a comparison. In the first loop we access each element of M and update the array A each time. Since there are l elements in M, the runtime of this loop will be in $O(l)$.

The second loop consists of an inner and outer loop. In the outer loop, we access each word of $w_1, \ldots, w_k$, and in the inner loop we access each letter of the word accessed in the outer loop and update the array B.

In the worst case, $k = l$ (every word will be a single letter), and the length of each word will be l. So the runtime of this loop will be in $O(l^2)$

In the comparison after the second loop, we check if the two arrays A and B contain the same elements. In the worst case, we will have to access all the l elements of A and B to check if they are the same, so the runtime of this comparison will be in $O(l)$

So the runtime of the algorithm will be in
$O(l) + O(l^2) + O(l) + O(l^2) + O(l) = O(l^2)$

Thus, the certification/verification algorithm runs in polynomial time

---

Proof that 3D-Matching $\leq_P$ Magnets:

1) Give an algorithm to transform an input of 3D-Matching to an input of Magnets:

```
Function Transform((X, Y, Z), T)

    Let $S = X \cup Y \cup Z$

    Let $M = m_1, \ldots, m_l$, where $l = |S|$ and each $m_i = s_i$

    Let $W = \{xyz \mid (x, y, z) \in T\}$

    return $(M, S, W)$
```

---

2)

Suppose (X, Y, Z), $T \subseteq X \times Y \times Z$ is an input to our transformation algorithm. Suppose $M = \{m_1, \ldots, m_l\}$, $S = \{s_1, \ldots, s_n\}$, $W \subseteq S^*$ will be the transformed output produced by the algorithm.

We will prove the correctness of our transformation algorithm by showing that (X, Y, Z), T is a "yes" instance of 3D-Matching if and only if M, S, W is a yes instance of Magnets

Proof of "$\Rightarrow$":

Suppose (X, Y, Z), T is a "yes" instance of 3D-Matching.
Then T is composed of n triplets such that each element of $X \cup Y \cup Z$ is contained in exactly one of these triplets.

So if each $(x, y, z) \in T$ was a "word" and $X \cup Y \cup Z$ was a set of magnets, then we could use up all the magnets by creating all the words in T

So we can create a set of magnets and letters by defining $S = X \cup Y \cup Z$, $M = m_1, \ldots, m_l$ where each $m_i = s_i$ and $l = |S|$. And we can define a set of words by defining $W = \{xyz \mid (x, y, z) \in T\}$.

And as mentioned before, all the magnets would be used up by creating all the words. So M, S, W would be a yes instance of Magnets

Thus, M, S, W is a "yes" instance of Magnets.

$\square$

Proof of "$\Leftarrow$":

Suppose $M = \{m_1, \ldots, m_l\}$, $S = \{s_1, \ldots, s_n\}$, $W \subseteq S^*$ is a yes instance of Magnets.
Note that $S = X \cup Y \cup Z$ and $W = \{xyz \mid (x, y, z) \in T\}$
Also note that $|M| = |S|$ and each magnet of M corresponds to a unique symbol.

Because of these reasons, this means that all the magnets in M will be used up when creating all the (three lettered) words in T.

Since each word is three lettered, and the set of magnets contains exactly the symbols in $S = X \cup Y \cup Z$, this means that each element of $X \cup Y \cup Z$ is contained in exactly one one of the triplets of T

Thus, (X, Y, Z), T is a yes instance of 3D Matching.

$\square$