Assignment - 5

(a) Yes

→ Internal scheduling can be sorted in polynomial time and so it can also be solved in polynomial time with access to a black box for vertex cover.

→ Internal scheduling is NP and anything in NP can be reduced to vertex cover.

→ Internal scheduling $\leq_P$ independent set.
Independent scheduling $\leq_P$ vertex corner.
So, the use of transitive property gives the set.

(b) If P = NP independent set can be solved in polynomial time and so independent set $\leq_P$ interval scheduling.

If independent set $\leq_P$ internal scheduling then since internal scheduling can be solved n polynomial time, so can independent set. But independent set is NP-complete so solving it in polynomial time would imply P = NP.

Proof of each case is shown below:

(a) Problem in NP

→ The given problem is in NP.

→ A polynomial time algorithm exists for given problem.

→ If a set of k processes are present in polynomial time algorithm exists for given problem. It could be checked that resource requested by one process is not needed by another.

→ For each process pair loop over all resources to check condition.

→ The solution is valid if there is no such resource, otherwise invalid.

⇒ Problem is NP hard

→ The resources can be considered as a set of vertices.

→ Each edge pair is then checked for common pair of vertexes.

→ If vertices are common, then there is no valid solution.

→ If no vertices get a common solution, then solution is valid.

→ The given details reduce an NP problem independent set to given problem.

→ Hence, given problem is NP hard.

→ The given problem is in NP and is NP hard.

→ Hence, it is NP complete.

(b) If there are only two processes, then it's a subset of the previous algo. The resources requested by one process should not be needed by another. Each resource should be checked for this condition. The problem is in NP and is reduced from an NP problem.

→ Hence, it is NP complete.

(c) It is a limited case of previous algorithm. Here, only two types of resources are checked with the condition. The given problem is in NP as it has a polynomial time an NP problem.

→ Hence, it is NP complete.

(d) If each resource is requested by at most two processes. It denotes a special case of previous algorithm. All set of resources are been checked with the request list. If a particular resource is requested by more than one process, then it is invalid. The given case is in NP and could be reduced from an NP problem.

→ Hence, it is NP - complete.

Q.3

This solution needs to exhibit that the set H one can easily verify in polynomial time if H is of size k and intersects each of the sets $B_1, \ldots B_m$.

→ We reduce from vertex cover.

→ Consider an instance of the vertex - cover problem graph $G = (V, E)$ and a positive integer.

→ We map it to an instance of the hitting set problem as follows.

→ The set A is of vertices V. For every edge e belongs to E we have a set S consisting of two end - points of e.

→ It is easy to see that a set of vertices S is a vertex cover of G, iff the corresponding elements from a hitting set in the hitting set instance.

Q-4 The claim by dimensioned as 3 of matching by $\leq p$ of collection of path.

→ The instance of consider as matching of 3 dimensional with set by X, Y, Z size of 'n' & triples order by $T_1 \ldots T_m$ from $X \times Y \times Z$.

→ We construct a directed graph $G=(V,E)$ on the node set $X \cup Y \cup Z$. For each triple $= T_i (x_i, y_j, z_k)$ we add edges (x_i, y_j) and (y_j, z_k) to G.

→ Finally, for each $i = 1, 2, \ldots m$, we define a path P_i that passes through the nodes $\{x_i, y_j, z_k\}$ where again $T_i = (x_i, y_j, z_k)$.

→ Note that by our definition of edges each P_i is a valid path in G.
Also the reduction takes polynomial time.

→ Now, we claim that there are n paths among $P_1 \ldots P_m$ sharing no nodes if and only if there exist n disjoint triples among $T_1 \ldots T_m$.

→ For if there do exist n paths sharing no nodes, then the corresponding triples must each contain a different element from X, a different element from Y. and a different element from Z- they form a perfect 3D matching.

→ Conversely, if there exist n disjoint triples, then the corresponding paths will have no nodes in common.

→ Since, Path selection is in NP and we can reduce an NP-complete problem to it, it must be NP-complete.

5 (a) Consider a ship enter with n containers of weight $w_1, w_2, \ldots w_n$.

Every truck in a set is capable of hold k units of weights.

→ Each & every truck is able to be stacked by many containers by a weight limit of k unit. Then the objective is to reduce the no. of trucks which are necessary to take every the containers.

→ Consider $\{w_1, w_2, w_3\} = \{1, 2, 1\}$ & assume $k=2$ Afterwards using the greedy algorithm we can apply 3 trucks & while there is a method to apply only 2 trucks.

(b) From the greedy algorithm, for any set of weights & for any k value, the no. of truck is surrounded by a factor of 2.

→ Assume that w is the total weight loaded with each truck which will be computed by $w = \Sigma w_i$. Each truck is able to carry max. k units of weight, therefore the least no. of

trucks required is represented by $\frac{w}{k}$.

→ Here, $m = 2q + 1$ (where m is even) which is for if the add no. of trucks utilized by this algorithm.

→ Then no. of truck is partitioned into two consecutive set providing a total of $2q + 1$ trucks in a set.

→ In each set of truck apart from the total weight of final containers have to be larger than k or else the 2nd truck of the set will not have been established then.

→ Therefore, it can be suppossed that $w > qk$ and so $\frac{w}{k} > q$. That is from above method, the optimal result uses at the min. "$q+1$" trucks which is within a factor of two of "m" which is specified by $m = 2q + 1$.

2-6 In this problem, we are given with an integer B which is positive.

→ The subset B_S of A need to be considered of which the sum of numbers would not be more than B. the example set A is given as $A = \{8, 2, 4\}$ and $B = 11$.

(4) The algorithm is given by:

```
S = ϕ
T = 0
For i = 1, 2, .... n
    If  T + qi ≤ B  then
        S ← S ∪ {qi}
        T ← T  T + qi
    End if
End for.
```

→ Considering an instance of problem for which the overall sum of elements in subset S, which the algorithm returns would be lesser then the half of total sum of any other possible feasible subset of A.

→ We can consider the particular instance B = 11 and A = {1, 11}. The algorithm would add only 1 to T, the optimal subset can be T* = {11}

(b) The idea of algorithm is given as:
We are adding integers q_j to set S, one by one in descending order until an unfeasible set is obtained by adding an integer.

The algorithm is shown below:

```
Subset feasible (A, B)
sortin items in A in descending order
S ← { }
sum = 0
```

for a_j from a_1 to a_n in descending order
do
if $a_j > B$ then
continue
else if $a_j + sum \leq B$ then
$S \leftarrow S \cup \{a_j\}$
$S \leftarrow sum + a_j$
else
break
end if
end for
return S

The algorithm subset feasible sums n
O (nlogn).

2.1 //

Consider the following LP.

$$\min \sum_{i=1}^{n} w_i x_i$$

where,

$$0 \leq x_j \leq 1, \quad j = 1 \ldots n,$$

$$\sum_{a_j \in B_j} x_j \geq 1, \quad j = 1, \ldots m$$

Let x be the solution of this problem and $w_1 P$ Ps the optimal value.

Now, define set S to be all these elements where $x_i \geq 1/b$, ie $S = \{a_i / x_i \geq 1/b\}$

(1) S is a hitting set. In fact, we know that the sum of all x_i where $a_i \in B_j$ is at least 1.

The set B_j contains at most b elements. Therefore some $x_i \geq 1/b$, for some $a_i \in B_j$. By definition of S, the element $a_i \in B$. So B_j intersects with S by a_i.

(2) The total weight of all elements in S in at most bw_{LP}, for each $a_i \in S$, we know that $x_i \geq 1/b$. i.e $1 \leq b x_i$, therefore $w(S) = \sum_{a_i \in S} w_i \leq \sum_{a_i \in S} w_i b x_i \leq b \sum_{i=1}^{n} w_i x_i = b w_{LP}$

(3) S^* is the optional set and let $x_i^* = 1$, if $a_i \in S^*$ and $x_i^* = 0$ otherwise.

Then the vector x^* satisfied the LP above. Thus,

$$w_{LP} \leq \sum_{i=1}^{n} w_i x_i^* = \sum_{a_i \in S} w_i = w(S^*)$$

Therefore, we have a hitting set S, such that $w(S) \leq b w(S^*)$

2.8 // Suppose T^* represents the optimal makespan t_j is the job size which is allocated to the machine.

and

T represents makespan.

Suppose that $T_i - t_j$ is the minimum load hold by every machine.

So, the load on the entire 10 machines will be:

$$\Sigma_k T_k \geq m(T_i - t_j)$$

$$\Rightarrow \quad m(T_i - t_j) \leq \Sigma_k T_k$$

$$\Rightarrow \quad T_j - t_j \leq \frac{1}{m} \Sigma_k T_k$$

But $\Sigma_k T_k$ is the entire load of all the jobs. $\Sigma_j T_j$ allocated to the machines.

Hence, $T - t_j \leq T^*$

So, the avg. load for optimal makespan is calculated which is given below:

$$T^* \geq \frac{1}{m} \Sigma t_j$$

$$\frac{1}{m} \Sigma t_j \geq \frac{1}{10} \times 3000 - 300$$

It is also known that $t_j \leq 50$.

Thus, the difference between optimal makespan and the makespan of user will be at most.

$$\therefore \frac{T - T^*}{T^*} \leq \frac{t_j}{T^*}$$

$$\therefore \frac{t_j}{T} \leq \frac{50}{300} \leq \frac{1}{6}$$

which is $\leq 20\%$

So, the difference between optimal makespan and the makespan of the user will be at most 20%.