UNIT-5 NP-Hard and NP Complete Problems.

Topic-1 :- Introduction

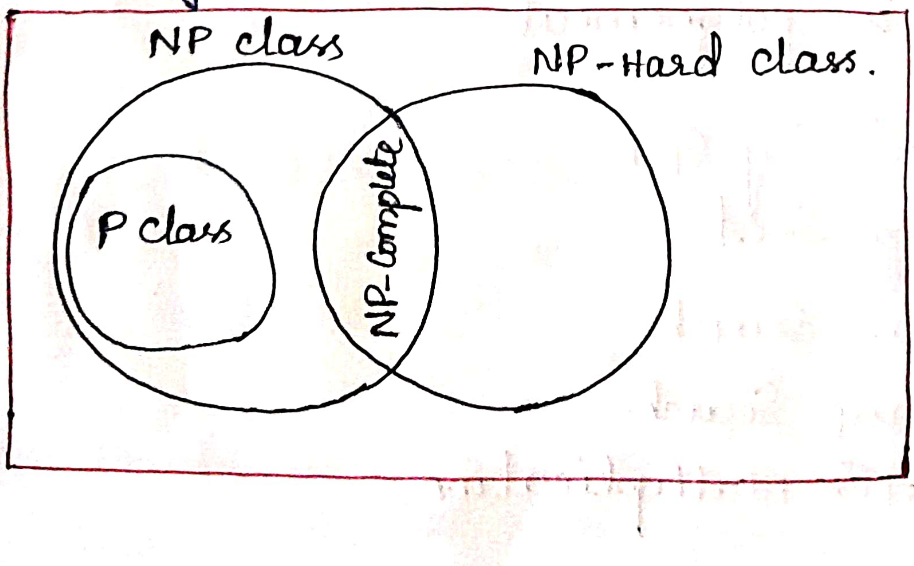
→ The problems are divided into classes known as Complexity classes.

→ Complexity class is a set of problems with related complexity

→ The time complexity of an algorithm is used to describe the number of steps required to solve a problem.

→ The space complexity of an algorithm describes how much memory is required for the algorithm to operate.

→ Complexity classes are useful in organizing similar type of problems.



Types of Complexity Classes

1. P class
2. NP class
3. NP Hard
4. NP Complete

① P class:-

The P in the P class stands for Polynomial Time.

→ It is the collection of decision problems that can be solved by a deterministic machine in polynomial time.

features:

* The solution to P problems is easy to find
* P is often a class of computational problems that can be solved in theory as well as in practice.
* This class contains all problems whose time complexity is polynomial.

Ex:- 1) Calculating GCD
2) Merge Sort
3) Linear Search
4) Binary Search
5) Matrix Multiplication,

...etc...

② NP class:-

→ The NP in NP class stands for Non-deterministic Polynomial Time.

→ It is the collection of decision problems that can be solved by a non-deterministic machine in polynomial time.

features

* The solutions of the NP class are hard to find since they are being solved by a non-deterministic machine but the solutions are easy to verify

* Problems of NP can be verified by a Turing machine in polynomial time.

Examples

1. Boolean Satisfiability Problem (SAT)
2. Hamiltonian Path Problem
3. Graph Coloring

→ For these problems, the answer is possible to find, but the solution is not known in polynomial time. If somebody can solve this then the problem comes into P class.
(Here, procedure to solve the problem in Polynomial time is known, but we don't know How to Solve ?)

③ NP- Hard class

An NP-hard problem is at least as hard as the hardest problem in NP and it is a class of problems such that every problem in NP reduces to NP-hard.

features

* All NP-hard problems are not in NP.
* It takes a long time to check them.
  → This means if a solution for an NP-hard problem is given then it takes a longer time to check whether it is right or not.
* A problem 'A' is in NP-hard if, for every problem L in NP, there exists a polynomial-time reduction from L to A.

Examples

1. Halting Problem
2. Qualified Boolean formulas
3. No Hamiltonian cycle.

④ NP - Complete class

A problem is NP complete if it is both NP and NP hard. NP-Complete problems are hard problems in NP.

features

→ NP-complete problems are special as any problem in NP class can be transformed (or) reduced into NP-Complete problems in polynomial time.

→ If one ~~and~~ could solve an NP-Complete problem in polynomial time, then one could also solve any NP problem in polynomial time.

Ex:- 1. Hamiltonian cycle
2. Satisfiability
3. Vertex cover.

Topic-2 :- Cook's theorem

"Cook's theorem states that satisfiability is in P if and only if P = NP."

We now prove this important theorem.

→ We have already seen that satisfiability is in NP.

Hence, if P = NP, then satisfiability is in P.

→ To do this, we show how to obtain from any polynomial time nondeterministic decision algorithm A and input I

→ a formula Q(A, I)

* such that Q is satisfiable iff A has a succesful termination with I.

* If the length of I is n and the time complexity of A is p(n) for some polynomial p(), then the length of Q is $O(p^3(n)\log n) = O(p^4(n))$.

* The time needed to construct Q is also $O(p^3(n)\log n)$.

→ A deterministic algorithm Z to determine the outcome of A on any input I can be easily obtained

→ Algorithm Z simply computes Q and then uses a ~~non~~ deterministic algorithm for the satisfiability problem to determine whether Q is satifiable, then the complexity of Z is $O(p^3(n)\log n + q(p^3(n)\log n))$.

→ If satisfiability is in P, then $q(m)$ is a polynomial function of m and the complexity of Z becomes $O(T(n))$ for some polynomial $T()$.

→ Hence, if satisfiability is in P, then for every non deterministic algorithm A in NP we can obtain a deterministic Z in P.

→ So, the above construction shows that if satisfiability is in P, then P = NP.

NP Hard Graph Problems

Topic-3:- Clique Decision Problem (CDP)

Clique:- "A clique is a subgraph of a graph such that the subgraph is a complete graph."

Clique Decision Problem

"It is the problem of finding if a clique of size k exist in the given graph or not".

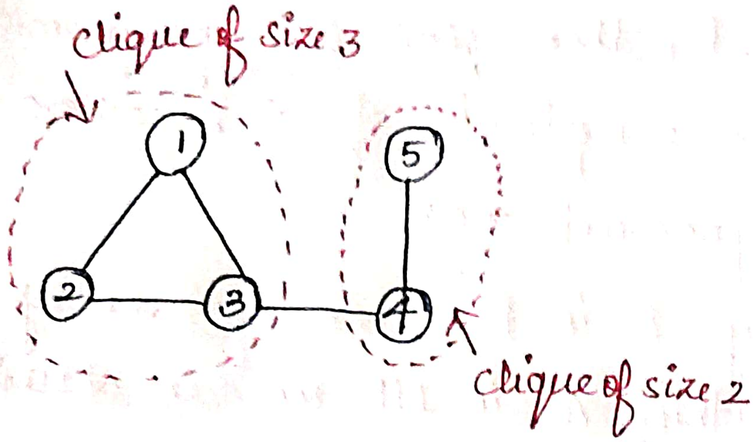
→ Every decision problem results in any of two solution i.e Yes or No, True or False, Positive or Negative, etc.

→ ∴ It is a decision problem.

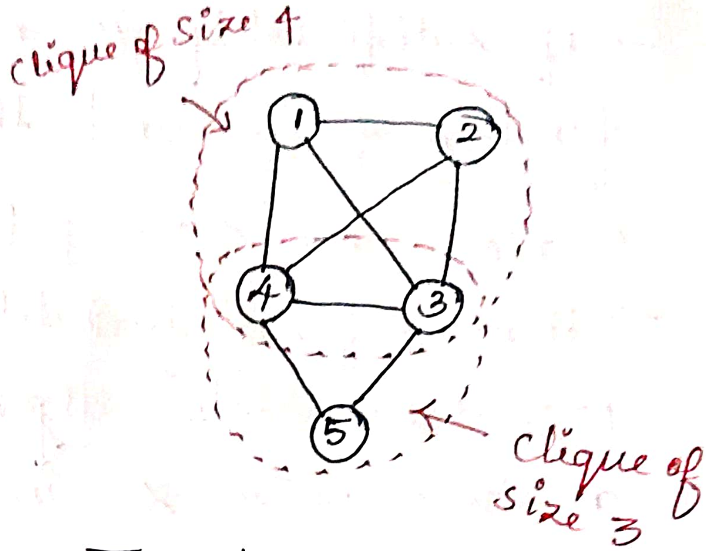
→ It is a NP hard graph problem.

→ CNF-Satisfiability $\propto$ CDP

→ To prove CDP as a NP hard we make use of CNF sat.



The above graph contains a maximum clique of size 3



The above graph contains a maximum clique of size 4.

→ **The clique Decision Problem belongs to NPHard**

As we mentioned CNF Sat $\propto$ CDP.

$\therefore$ We prove CNF Sat is similar to CDP.

To prove that, lets take. CNF formula.

$$\boxed{F = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \wedge (x_1 \vee x_3)}$$

where $x_1, x_2, x_3 \leftarrow$ variable

$\wedge \leftarrow$ logical AND

$\vee \leftarrow$ logical OR.

$\bar{x} \leftarrow$ complement of x

Let expression with each paranthesis be a clause.

Hence we have 3 classes. $c_1 \leftarrow (x_1 \vee x_2)$

$c_2 \leftarrow (\bar{x}_1 \vee \bar{x}_2)$

$c_3 \leftarrow (x_1 \vee x_3)$.

→ Now consider the vertices

$C_1 \leftarrow \{ \langle x_1, 1 \rangle ; \langle x_2, 1 \rangle \}$

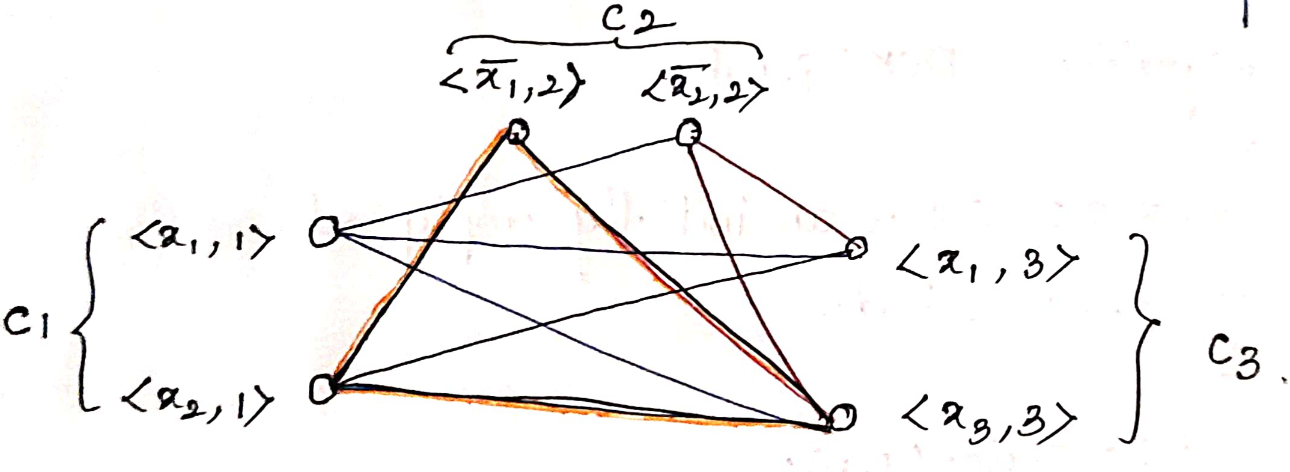
$C_2 \leftarrow \{ \langle \bar{x}_1, 2 \rangle ; \langle \bar{x}_2, 2 \rangle \}$

$C_3 \leftarrow \{ \langle x_1, 3 \rangle ; \langle x_3, 3 \rangle \}$

→ Where second term in each vertex denotes the clause number they belongs to.

→ We connect these vertices such that.

1. No two vertices belonging to the same clause
2. No variable is connected to its complement.



→ Thus, the graph $G(V,E)$ is constructed such that

$V = \{ \langle a, i \rangle \mid a \in C_i \}$ and $E = \{ (\langle a, i \rangle, \langle b, j \rangle) \mid$ i is not to j ; $b \neq a' \}$

Consider the subgraph of G with the vertices.

$\langle x_2, 1 \rangle, \langle \bar{x}_1, 2 \rangle, \langle x_3, 3 \rangle$

it forms a clique of size 3.

→ Corresponding to this $\langle x_1, x_2, x_3 \rangle = \langle 0, 1, 1 \rangle$
F = True.

∴ If we have k clauses in our satisfiability expression, we get a max clique of size k and for the corresping assignment of values,

→ the satisfiability problem evaluates to true.

→ Hence, for a particular instance, the satisfiability problem is reduced to the clique decision problem.

∴ Clique decision problem is NP-Hard.

Nondeterministic clique pseudocode.

```
Algorithm DCK(G, n, k)
{
S := φ; // S is an initially empty set.
for i := 1 to k do
{
    t := Choice(1, n);
    if t ∈ S then Failure();
    S := S ∪ {t} // add t to set S
}
// At this point S contains k distinct vertex indices
for all pairs (i, j) such that i ∈ S, j ∈ S,
and i ≠ j do
if (i, j) is not an edge of G then Failure();
    Success();
}
```

Topic-4 :- Chromatic Number Decision Problem (CNDP)

A coloring of a graph $G = (V, E)$ is a function $f: V \rightarrow \{1, 2 \ldots k\}$ defined for all $i \in V$.

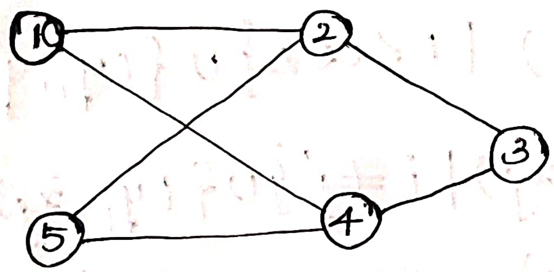
If $(u, v) \in E$, then $f(u) \neq f(v)$. The chromatic number decision problem is to determine whether G has a coloring for a given k.

Example:- A possible 2-coloring of the graph of following figure is

$$f(1) = f(3) = f(5) = 1$$

$$f(2) = f(4) = 2.$$

$\therefore$ clearly; this graph has no 1-coloring



→ To find the chromatic number of any graph, find the cycle with maximum nodes covered in it.

→ Now, if the maximum cycle has odd no. of vertices then the chromatic number is 3.

→ if the maximum cycle has even no. of vertices then the chromatic number is 2.
(This rule is not valid for complete graphs)

To prove CNDP as NP hard, we use

Theorem

$$\boxed{SATY \propto CNDP}$$

Proof:-

→ Let F be a CNF-formula with n variables $(x_1, x_2, \dots x_n)$ and the formula have at most three literals in each of r clauses, $c_1, c_2 \dots c_r$.

→ For $n >= 4$, we can construct a graph $G=(V,E)$ defined as follows

$$V = \{x_1, x_2 \dots x_n\} \cup \{\bar{x_i} : 1 \leq i \leq n\} \cup \{y_i : 1 \leq i \leq n\} \cup \{c_i : 1 \leq i \leq r\}$$

$$E = \{(x_i, \bar{x_i}) \mid 1 \leq i \leq n\} \cup \{(y_i, y_j) \mid i \neq j\} \cup \{(y_i, x_j) \mid i \neq j\} \cup \{(y_i, \bar{x_j}) \mid i \neq j\} \cup \{(x_i, c_j) \mid x_i \in c_j) \cup \{\bar{x_i}, c_j) \mid \bar{x_i} \in c_j\}$$

(OR).

Lets consider 3-coloring problem is NP-Hard.

In order to prove that the 3-coloring problem is NP-Hard, perform a reduction from a known NP-Hard problem to this problem.

→ carry out reduction from which the 3-SAT problem can be reduced to the 3-coloring problem.

→ Let us assume that the 3-SAT problem has a 3-SAT formula of m clauses on n variables denoted by $x_1, x_2 \ldots, x_n$.

→ The graph can then be constructed from the formula in the following way.

1. For every variable x_i Construct a vertex v_i In the graph and a vertex $\bar{v_i}$, denoting the negation of the variable x_i.

2. For each clause c in m, add 5 vertices correspo-nding to values c1, c1, ...c5.

3. Three vertices of different colors are additionally added to denote the values True, False, and Base (T, F, B) respectively.

4. Edges are added among these three additional vertices T, F, B to form a triangle.

5. Edges are added among the vertices v_i and $\bar{v_i}$. and Base (B) to form a triangle.

The following constraints are true for graph G:

1. For each of the pairs of vertices V_i and $\overline{V_i}$, either one is assigned a TRUE value and the other, FALSE.

2. For each clause c in m clauses, at least one of the literal has to hold TRUE value for the value to be true.

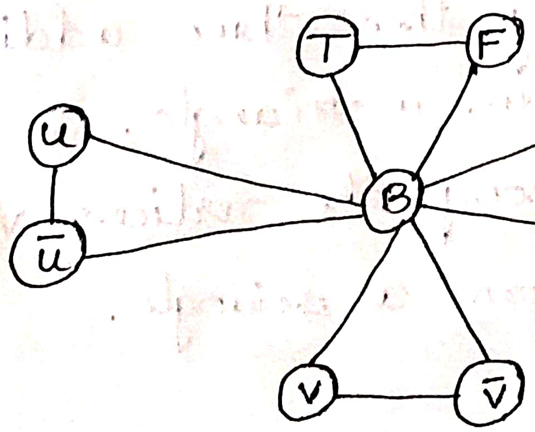
→ A small OR - gadget graph therefore can be constructed for each of the clause

$$c = (u \vee V \vee w).$$

in the formula by input nodes u, v, w and connect output node of gadget to both False and Base special nodes.

→ Let us consider the formula

$$f = (\bar{u} \vee V \vee \bar{w}) \wedge (u \vee V \vee \bar{w})$$



Now the reduction can be proved by the following two propositions:

→ Let us assume that the 3-SAT formula has a satisfying assignment, then in every clause, at least one of the literals x_i has to be true, therefore, the corresponding v_i can be assigned to a TRUE color and $\overline{v_i}$ to FALSE.

→ Now, extending this, for each clause the corresponding OR-gadget graph can be 3-colored. Hence, the graph can be 3-colored.

→ Let us consider that the graph G is 3-colorable, so if the vertex v_i is assigned to the true color, correspondingly the variable x_i is assigned to true.

→ This will form a legal truth assignment. Also, for any clause $C_j = (x \vee y \vee z)$, it cannot be that all the three literals x, y, z are False.

→ Because in this case, the output of the OR-gadget graph for C_j has to be colored False.

→ This is a contradiction because the output is connected to Base and False. Hence, there exists a satisfying assignment to the 3-SAT clause.

∴ 3-coloring is ~~and~~ NP-Hard problem.

Topic-5:- Travelling Salesperson Decision Problem

Travelling Salesperson problem:-

In a given n cities, the task of Travelling Sales person is to visit each city exactly once and should come back to source city with minimum cost.

TSDP:-

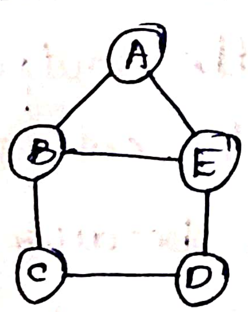
Travelling Salesperson Decision problem is to determine whether a complete graph $G=(V,E)$ with edge cost $c(u,v)$ has to tour of cost at most M.

To prove TSDP an NP-Hard problem we use Directed Hamiltonian Cycle (DHC) α TSDP.

Proof:-

1. Consider an Hamiltonian Graph

$G(V,E) \rightarrow$



$\rightarrow$ Hamiltonian cycle.

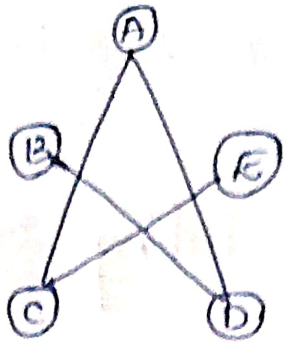
$\downarrow$

A - B - C - D - E - A

2. Now construct a complete graph for the above considered graph using

$$G'(V,E') + G(V,E)$$

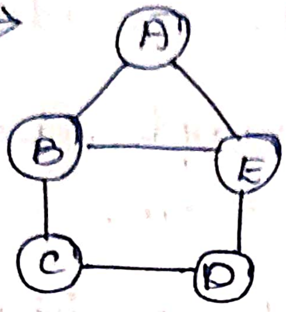
i.e $G'(V, E') \longrightarrow$



↗

Complementad graph of $G^{\circ}(V, E)$

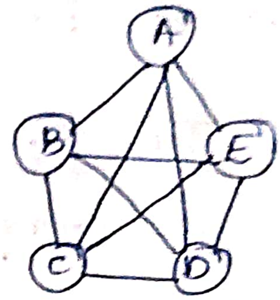
$G(V, E) \rightarrow$



$G'(V,E') + G(V,E)$

+

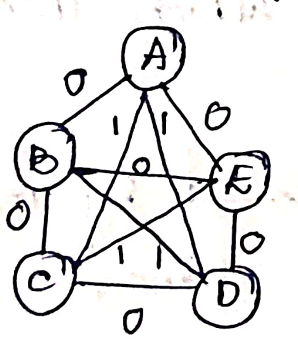
→



3) Now update the edges cost using

$$c(i,j) = \begin{cases} 0 & \text{if } i,j \in E \\ 1 & \text{if } i,j \in E' \end{cases}$$

4) The graph with edges of their weights is



5) The minimum tour of TSP for above graph is A – B – C – D – E – A. with the cost 0.

∴ TSP is similar to Hamiltonian cycle.

Hence we proved $DHG \propto TSDP$ and TSDP is an NP Hard problem.

Part-C :- NP Hard Scheduling Problems.

Topic-1 :- Scheduling Identical Processors.

→ Let P_i, $1 \le i \le m$, be m identical processors (m/cs).

→ The P_i could, for example, be line printers in a computer output room.

→ Let J_i, $1 \le i \le n$, be n jobs.

→ Job J_i requires t_i processing time.

→ A schedule S is an assignment of jobs to processors.

→ For each job J_i, S specifies the time intervals and the processor(s) on which this job is to be processed.

→ A job cannot be processed by more than one processor at any given time.

→ Let f_i be the time at which the processing of job J_i is completed.

→ The mean finish time (MFT) of schedule is

$$\boxed{MFT(S) = \frac{1}{n} \sum_{1 \le i \le n} f_i}$$

where $S \leftarrow$ schedule
$n \leftarrow$ no. of jobs
$f_i \leftarrow$ finish time
$i \leftarrow$ i^{th} job
MFT ← Mean finish time.

→ Let W_i, be a weight associated with each job J_i. The weighted mean finish time (WMFT) of schedule S is

$$\boxed{WMFT(S) = \frac{1}{n} \sum_{1 \le i \le n} W_i f_i}$$

→ Let T_i be the time at which P_i finishes processing all jobs (or job segments) assigned to it.

The finish time (FT) of S is

$$\boxed{FT(S) = \max_{1 \le i \le m} \{T_i\}}$$

→ Scheduling S is a nonpreemptive schedule if and only if each job J_i is processed continuo-usly from start to end on the same processor.

<u>Theorem</u>:- Partion α minimum finish time Non Preemptive schedule.

<u>Proof</u>:- We prove this for $m=2$. The extension to $m>2$ is trivial. Let a_i, $1 \le i \le n$, be an instance of the partition problem. Define n jobs with processing requirements $t_i = a_i$, $1 \le i \le n$. There is a nonpreemptive schedule for this set of jobs on two processors with finish time at most $\sum t_i/2$.

Topic-2 :- Job shop Scheduling

→ A Job shop has m different processors.

→ The 'n' jobs to be scheduled require the completion of several tasks.

→ The time of the j^{th} task for J_i is $t_{k,i,j}$.

→ The Task j is to be performed on processor P_k.

→ The tasks for any job J_i are to be carried out in the order 1, 2, 3, ..., and so on.

→ Task j can not begin until task j-1 (if j>1) has been completed.

→ Note that it is quite possible for a job to have many tasks that are to be performed on the same processor.

→ Obtaining either a MFT preemptive schedule (or) a MFT nonpreemptive schedule is NP-hard even when m=2.

→ The proof for the nonpreemptive case is very simple.

→ We present the proof for the preemptive case.

→ This proof will also be valid for the nonpreemptive case, but will not be the simplest proof for this case.

Theorem:- Partition $\propto$ Minimum Finish Time Preemptive job shop schedule $(m>1)$.

Proof:- We use only two processors.

→ Let $A=\{a_1, a_2, \ldots a_n\}$ define an instance of the partition problem.

→ Construct the following job shop instance JS, with $n+1$ jobs and $m=2$ processors.

Job $1, \ldots n$: $t_{1,i,1} = t_{2,i,2} = a_i$ for $1 \leq i \leq n$.

Job $n+1$: $t_{2,n+1,1} = t_{1,n+1,2} = t_{2,n+1,3}$

$= t_{1,n+1,4} = T/2$

where $T = \sum_{i=1}^{n} a_i$

We show that the job shop problem has a preemptive schedule with finish time at most $2T$ if and only if S has a partition.

$\{t_{1,i,1} \mid i \in u\}$	$t_{1,n+1,2}$	$\{t_{1,i,1} \mid i \notin u\}$	$t_{1,n+1,4}$
$t_{2,n+1,1}$	$\{t_{2,i,2} \mid i \in u\}$	$t_{2,n+1,3}$.	$\{t_{2,i,2} \mid i \notin u\}$

$0 \qquad T/2 \qquad T \qquad 3T/2 \qquad 2T$