

Problem analysis of Greedy Algorithm(1)

YAO ZHAO

A Task-scheduling Problem

- ▶ A unit-time task is a job, such as a program to be run on a computer, that requires exactly one unit of time to complete. Given a finite set S of unit-time tasks, a schedule for S is a permutation of S specifying the order in which to perform these tasks. The first task in the schedule begins at time 0 and finishes at time 1, the second task begins at time 1 and finishes at time 2, and so on.
- ▶ The problem of **scheduling unit-time tasks with deadlines and penalties for a single processor** has the following inputs:
 - a set $S = \{a_1, a_2, \dots, a_n\}$ of n unit-time tasks;
 - A set of n integer **deadlines** d_1, d_2, \dots, d_n , such that each d_i satisfies $1 \leq d_i \leq n$ and task a_i is supposed to finish by time d_i ; and
 - a set of n nonnegative weights or **penalties** w_1, w_2, \dots, w_n , such that we incur a penalty of w_i if task a_i is not finished by time d_i , and we incur no penalty if a task finishes by its deadline.
- ▶ We wish to find a schedule for S that minimizes the total penalty incurred for missed deadlines.

Sample Input:

	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Sample output:

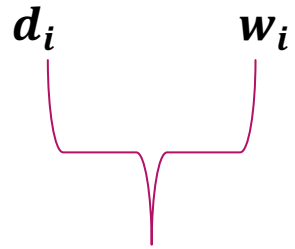
Time slot:	0	1	2	3	4	5	6	7
	3	2	4	1	7	6	5	

The answer is not unique. There are many other possible solutions.
For example:

Time slot:	0	1	2	3	4	5	6	7
	1	2	4	3	7	5	6	

Target: minimizes the total penalty

task a_i must be finished **by time** d_i , or will incur a **penalty** w_i



Which condition should be given priority?

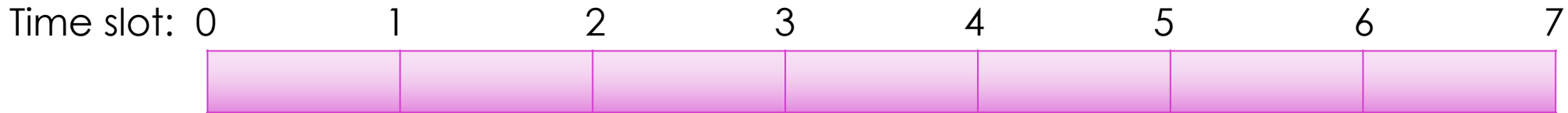
	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Let's start with **deadline**

Analysis process based on deadline priority

Consider a given schedule. We say that a task is **late** in this schedule if it finishes after its deadline. Otherwise, the task is **early** in the schedule. The problem of minimizing the sum of the penalties of the late tasks is the same as **the problem of maximizing the sum of the penalties of the early tasks**.

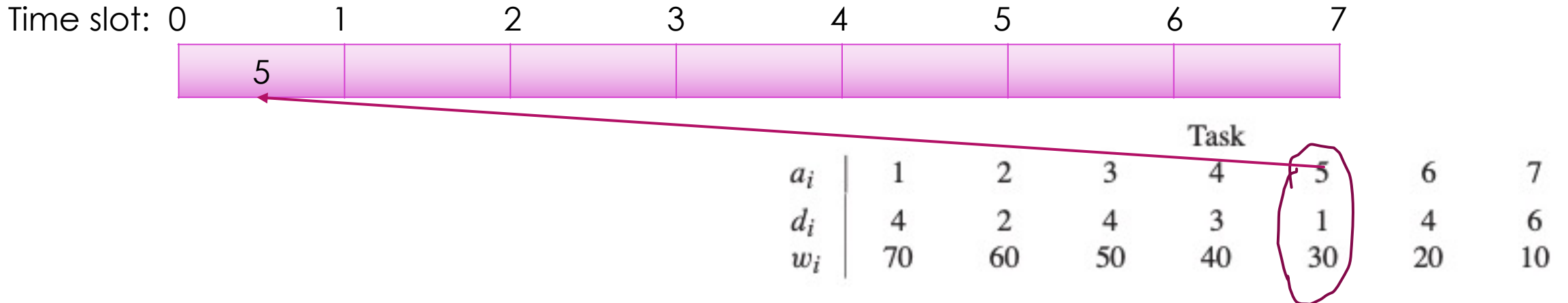
Initial State(S_0): No task in time slots.



Assume that we should find a schedule for time slot $[0, 1)$, and make the penalty as high as possible for tasks finished before time 1. We can do the following:

- (1) Find all tasks a_i , if task a_i is finished by time 1.
- (2) If the number of the satisfied task > 1 , only select the one with largest w

Step 1(S_1):



Step 2 (S_2):

Time slot: 0



	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

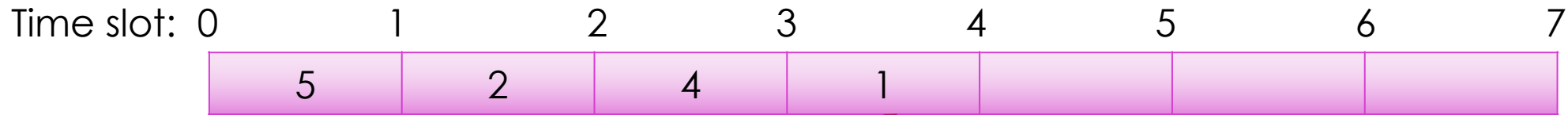
Step 3 (S_3):

Time slot: 0



	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Step 4 (S₄):



- (1) 3 tasks found : 1, 3, and 6
- (2) Task 1 has the maximum penalty.
- (3) But task 3 and 6 should be abandon?
- (4) Observe that:

$$w_3 > w_5 \text{ and } w_3 > w_4$$

Task 3 is better than task 4 and task 5;

$w_4 > w_5$, task 4 is better than task 5.

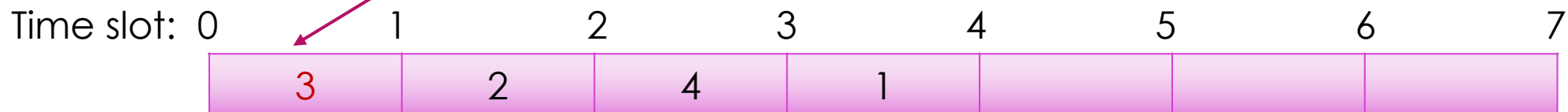
So, Task 3 can replace Task 5

w_6 is the smallest one, abandon task 6

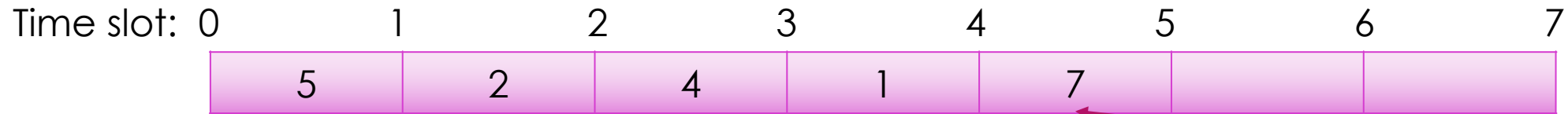
	Task 1	2	3	4	5	6	7
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

$$70 > 50 > 20$$

Step 5 (S₅):



Step 6 (S_6):



- (1) No new task found by time 5
- (2) One new task found by time 6

Task							
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

If more than one new tasks found by time 6, how to do?

Solution 1:

$S \leftarrow \emptyset$

$F \leftarrow \emptyset$

For $t = 1$ to n {

 add all tasks to S whose deadline is t or earlier

 if the size of S $s > t$, move the $(s-t)$ tasks with the minimum penalty from S to F

}

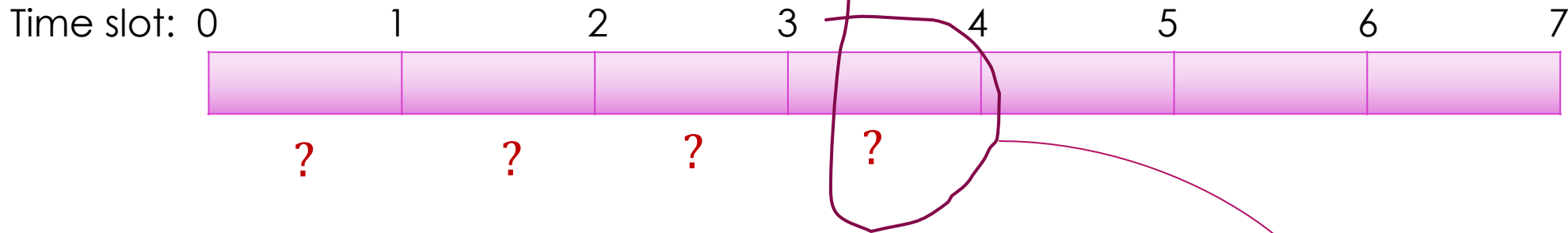
Order the tasks in S by deadline, then following the tasks in F in arbitrary way.

Analysis process based on penalty priority

	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Order tasks by w_i

Initial State(S_0): No task in time slots.

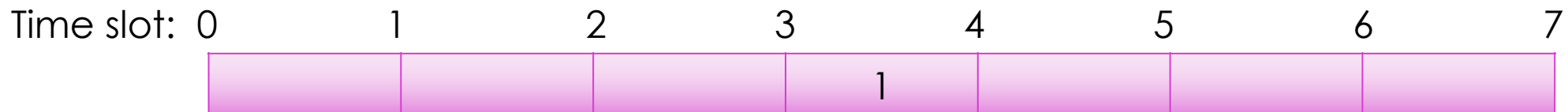


The deadline of task 1 is 4

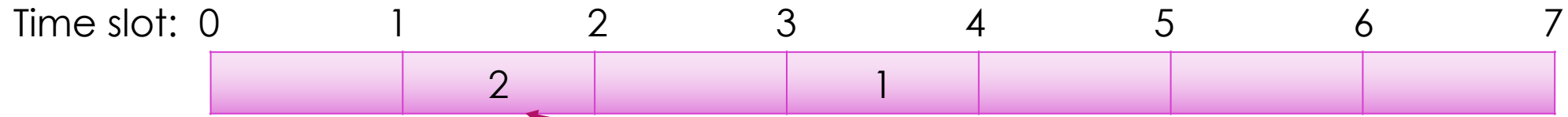
Which slot is best for task 1 ?

The latest one is the best one

Step 1 (S_1): Place task 1

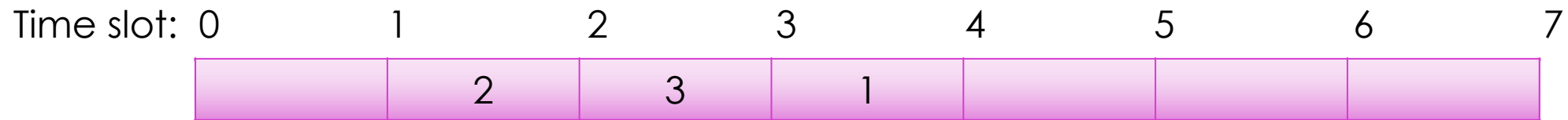


Step 2(S_2): Place task 2



	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Step 3 (S_3): Place task 3



Go ahead, find a idle slot

conflict

	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Step 4(S₄): Place task 4

Time slot: 0 1 2 3 4 5 6 7



Go ahead, find a idle slot

conflict

	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Step 5 (S₅): Place task 5

Time slot: 0 1 2 3 4 5 6 7



Go ahead, **fail** to find a idle slot

conflict

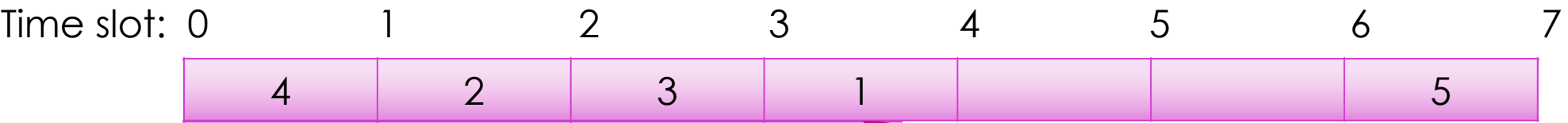
	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Time slot: 0 1 2 3 4 5 6 7



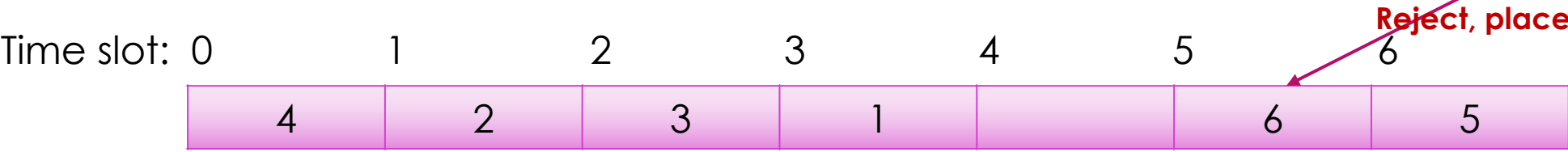
Reject, place it to the last slot

Step 5 (S₅): Place task 5



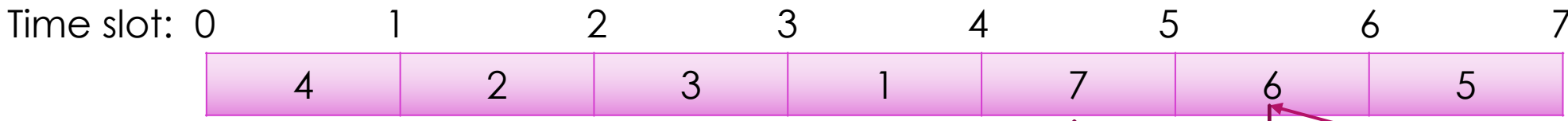
conflict

	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10



Reject, place it to the last 2 slot

Step6 (S₆): Place task 6



Go ahead, find a idle slot

conflict

	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10

Solution 2

Sort the tasks by finish time so that $w_1 > w_2 > w_3 > w_4 \dots > w_n$

For $a = 1$ to n {

 if (ts[a] is idle{

$ts[d[a]] \leftarrow a$

 } else{

 for $t = d[a]-1$ to 1 , n to $d[a]+1$ {

 if ts[t] is idle $ts[t] \leftarrow a$

 }

}