Greedy Algorithm

1 Definition

A greedy algorithm is a simple, intuitive algorithmic approach that makes the best or most optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem. This means that it picks the best immediate option without considering the broader consequences and hopes that by choosing a local optimum at each step, a global optimum will be reached.

2 Interval Scheduling Problem

2.1 Problem Definition

Assume job j starts at s_j and finishes at f_j. Two jobs are compatible if they don't overlap. The goal is to find **maximum subset** of mutually compatible jobs.

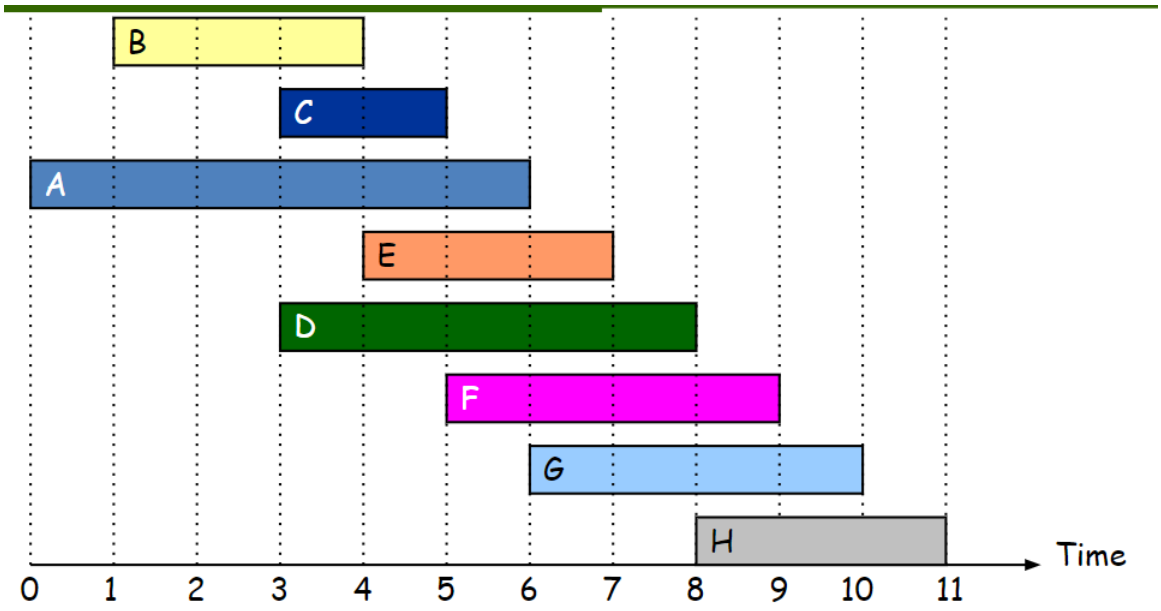

Figure 1: Interval Scheduling

The optimized solution is **choosing next job to add to solution as the one with earliest finish time that it is compatible with the ones already taken.**

2.2 Interval Definition

Let the set of all intervals be:

$$I = \{i_1, i_2, ..., i_n\} \tag{1}$$

where each interval i_j is represented by its start and finish times (s_j, f_j). Without loss of generality, assume these intervals are sorted by their **finish times**:

$$f_1 \leq f_2 \leq ... \leq f_n \tag{2}$$

2.3 Solution Definition

The set of intervals selected by the greedy algorithm, sorted by their finish times is defined as:

$$G = \{g_1, g_2, ..., g_k\} \tag{3}$$

The set of intervals in an optimal solution, also sorted by their finish times is defined as:

$$O = \{o_1, o_2, ..., o_m\} \tag{4}$$

2.4 Greedy Stays Ahead Argument

Need to prove that $f_{g_j} \leq f_{o_j}$ for all j where $1 \leq j \leq min(k, m)$. This implies that the j-th interval selected by the greedy algorithm finishes no later than the j-th interval in optimal solution.

2.5 Base Case

For $j = 1$, because g_1 is the interval with the earliest finish time, $f_{g_1} \leq f_{o_1}$

2.6 Inductive Step

1. Assume $f_{g_j} \leq f_{o_j}$ holds for particular j
2. Consider the next interval g_{j+1} chosen by Greedy, then g_{j+1} is the interval with the earliest finish time after g_j
3. Because we know job o_{j+1} is compatible with o_j, so we have:

$$f_{o_j} \leq s_{o_{j+1}} \leq f_{o_{j+1}} \tag{5}$$

Therefore we have:

$$f_{g_j} \leq s_{o_{j+1}} \tag{6}$$

Therefore job o_{j+1} is compatible with g_j, so it is an option for greedy algorithm. Based on step 2, we know g_{j+1} already has the earliest finish time, so:

$$f_{g_{j+1}} \leq f_{o_{j+1}} \tag{7}$$

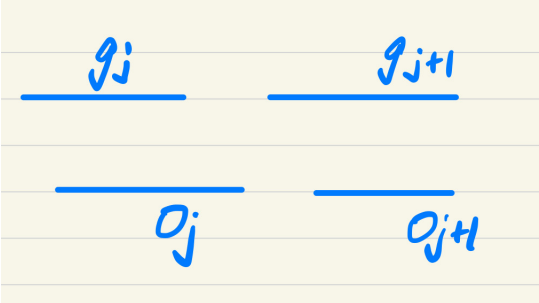

Figure 2: Inductive Step

2.7 Pseudo Code

$O(n \log n)$

$O(n)$

```
Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n.
        set of jobs selected
A ← φ
f_j* = 0
for j = 1 to n {
  if (job j compatible with A : s_j ≥ f_j*) {
    A ← A ∪ {j}
    f_j* = f_j
  }
}
return A
```

Running time: $O(n \log n)$

Figure 3: Interval Scheduling Pseudo Code

3 Minimizing Lateness Problem

3.1 Problem Definition

Suppose single resource processes on job at a time. Assume job j requires t_j units of processing time and is due at time d_j, as shown below:

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15

Figure 4: Processing Time and Due Time

If a job j starts at time s_j, then the finish time will be:

$$f_j = s_j + t_j \tag{8}$$

And the lateness will be expressed as:

$$l_j = max\{0, f_j - d_j\} \tag{9}$$

The final goal is to schedule **all jobs to minimize maximum lateness**. For example:

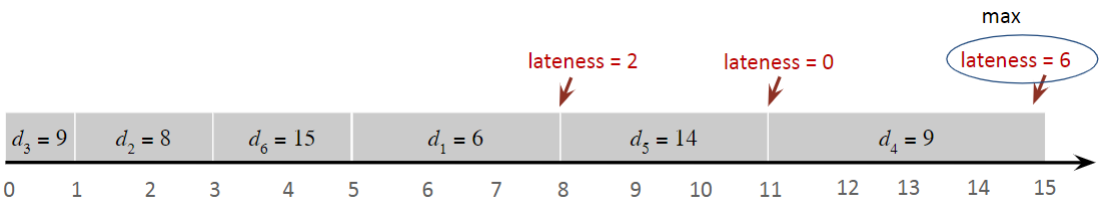

Figure 5: Max Lateness Example

3.2 Algorithm

The chosen algorithm is **earliest deadline first** algorithm:

```
Sort n jobs by deadline so that d_1 ≤ d_2 ≤ … ≤ d_n

t ← 0
for j = 1 to n
  Assign job j to interval [t, t + t_j]
  s_j ← t, f_j ← t + t_j
  t ← t + t_j
output intervals [s_j, f_j]
```

Figure 6: Earliest Deadline First Algorithm

Now we want to prove that this algorithm is the optimal solution, using **exchange argument** method.

3.3 Observations

Before the start of proof, we should first analyze the problem.

3.3.1 Observation 1

There exists an optimal schedule with no idle time.

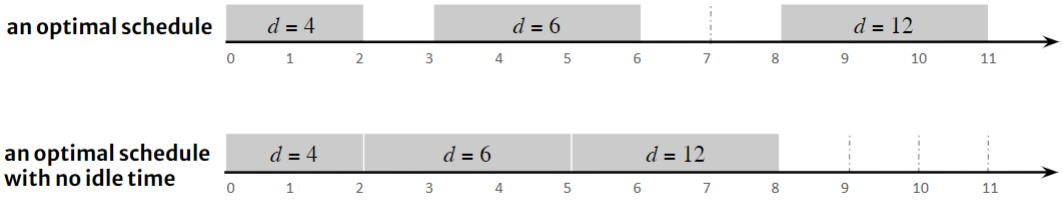

Figure 7: Observation 1

Because if an optimal schedule has idle time, we can always make it more optimal by reducing the idle time, so the all the tasks could be finished earlier.

3.3.2 Observation 2

The earliest-deadline-first schedule has no idle time. Similarly, if idle time exists, we can reduce it and make the schedule more optimal.

3.3.3 Observation 3

First we define **inversion** as a pair of jobs i and j such that $d_i < d_j$, but j is scheduled before i in a given schedule S.

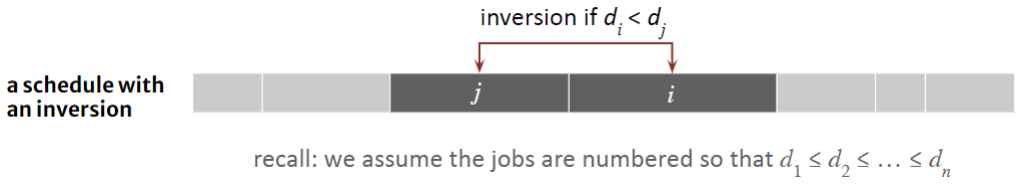

Figure 8: Inversion Definition

Therefore if all deadlines are different, **the earliest-deadline-first schedule is the unique idle-free schedule with no inversions**, because this algorithm arrange the task based on the deadlines order.

3.3.4 Lemma

All schedules with no inversions and no idle time have the same lateness. The possible difference is that we have several jobs with the same deadline, but we arrange them in different ways. But all of these arrangements will give the same lateness, which could be proven:

1. All jobs with same deadline d come in a block of consecutive jobs in any schedule with no inversions.
2. In any reordering of these jobs, the last job of the block has the same finish time and it is the worst finish time among the jobs in the block.
3. They all have same deadline, so latest job in the block is always last in the block and its lateness is always $f - d$ no matter reordering.

3.3.5 Observation 4

If an idle-free schedule has an inversion, then it has an adjacent inversion (two inverted jobs scheduled consecutively). This could be proven by contradiction.

Figure 9: Proof

1. Let i, j be a **closest but not adjacent (otherwise it is proved already)** inversion ($d_i < d_j$)
2. Now assume k be the element immediately to right of j
3. If $d_j > d_k$, then j, k is an adjacent inversion
4. If $d_j < d_k$, then i, k is a closer inversion, which is a contradiction.

3.3.6 Key Claim

Exchanging two adjacent, inverted jobs i, j reduces the number of inversions by 1 but does not increase the max lateness.

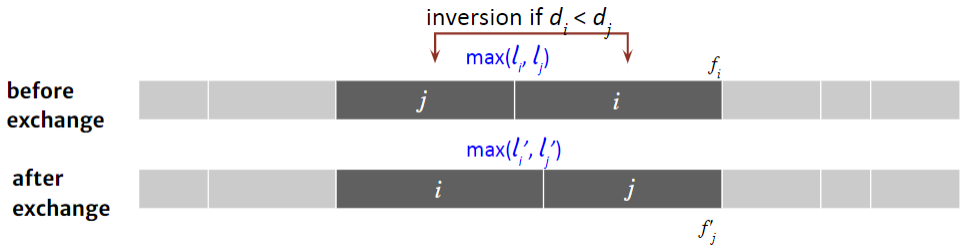

Figure 10: Flip Inversion

This could also be proven:

1. Let l be the lateness before the swap, and let l' be it afterwards.
2. For all $k \neq i, j$, flipping the adjacent inversion will not affect other elements, so $l'_k = l_k$.

3. Because we move i forwards, so we have $l'_i \leq l_i$.

4. Now if job j is late, then based on the definition we have:

$$l'_j = f'_j - d_j \tag{10}$$

Because i, j are adjacent inversion, so we have:

$$f_i = f'_j \tag{11}$$

$$l'_j = f_i - d_j \tag{12}$$

Recall the assumption:

$$d_i \leq d_j \tag{13}$$

Therefore we have:

$$l'_j \leq f_i - d_i = l_i \tag{14}$$

3.4 Theorem Proof

3.4.1 Summary

1. **Observation 1**: There exists an optimal schedule with no idle time.
2. **Observation 2**: The earliest-deadline-first schedule has no idle time.
3. **Observation 3**: The earliest-deadline-first schedule is the unique idle-free schedule with no inversions, if all deadlines are different.
4. **Lemma**: All schedules with no inversions and no idle time have the same lateness.
5. **Observation 4**: If an idle-free schedule has an inversion, then it has an adjacent inversion.
6. **Key Claim**: Exchanging two adjacent, inverted jobs i and j reduces the number of inversions by 1 and does not increase the max lateness.

3.4.2 Proof (less formal)

Now we want to prove that **there is an optimal schedule with no inversions and no idle time.**

1. There is an optimal schedule O with no idle time (Observation 1).
2. If O has an inversion then it has a consecutive inversion (Observation 4). We will swap the consecutive inversion, and this does not increase the maximum lateness (Key Claim)
3. We repeat these swaps until no inversion exists. We will get an optimal solution without inversions.

4. Finally we need to show that the total number of exchange operations. For a list of n elements, in the worst case, every pair of elements is an inversion. Then, the first element of the list can pair with $n-1$ other elements to form an inversion, second $n-2$, third $n-3$ and so on. Therefore, the total cases are:

$$(n-1)+(n-2)+...+1=\frac{n(n-1)}{2} \tag{15}$$

Then, we want to prove **greedy is optimal**:

1. There is an optimal schedule with no idle time and no inversions.
2. Greedy has no idle time and no inversions (Observation 2, 3).
3. All schedules with no idle time and no inversions have the same lateness (Lemma)
4. Greedy has optimal lateness.

3.4.3 Exchange Argument Introduction

The idea of exchange argument is to show that you can iteratively transform any optimal solution into the solution produced by the greedy algorithm without worsening the cost of the optimal solution, thereby proving that the greedy solution is optimal.

The basic steps include:

1. **Define your solutions**: Comparing the greedy solution G to an optimal solution O.
2. **Compare Solutions**: Show that if $G \neq O$, then they must differ in some way. Some classical assumptions include there is a piece of G that is not in O, or that two elements of G that are in a different order in O.
3. **Exchange Pieces**: Show how to transform O by exchanging some piece of O for piece of G. Then prove that by doing so, you did not worsen the score of O and therefore have a different optimal solution.
4. **Iterate**: Argue that you have decreased the number of differences between G and O by performing the exchange, and that by iterating this process for a finite number of times you can turn O into G without impacting the quality of the solution. Therefore, G must be optimal.

3.4.4 Proof (formal)

1. **Define the solutions**: Denote greedy solution as G, and an optimal solution O. Pick the optimal solution which has no idle time (Observation 1).
2. **Compare Solutions** Show that if $G \neq O$, then they must differ in some way. From Observation 3, G has no inversions and no idle time.
   (a) If O has no inversions: then G has the same lateness as O (Lemma)

(b) If O has inversions: move to exchange argument

3. **Exchange Pieces**: show that O can be gradually converted into G without hurting the quality of O. Because O has inversions, it must have an adjacent inversion (Observation 4). We can invert the adjacent inversion without hurting the quality of O (Key Claim)

4. **Iterate**:The previous operation reduces the number of inversions by 1. There are at most $\frac{n(n-1)}{2}$ inversions. After this, O is turned into a solution with no inversions and no idle time, same as G. Then, from Lemma, G has the same lateness as O. Therefore G is optimal.

4 Interval Partitioning

4.1 Problem Definition

Assume the lecture j starts at s_j and finishes at f_j. Then the goal is to find **minimum number** of classrooms to schedule **all lectures** so that no two occur at the same time in the same room.

An example is shown below (not optimal):

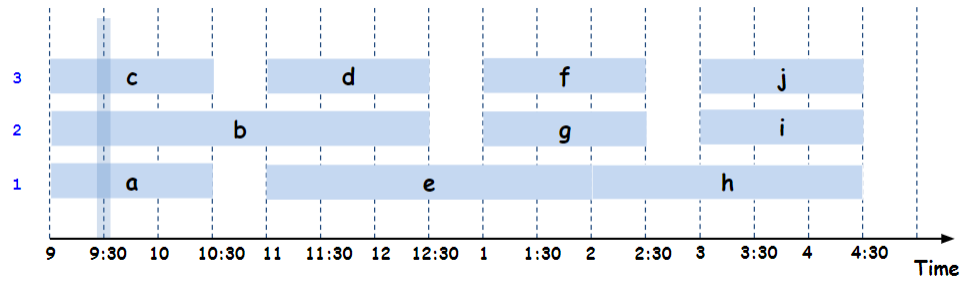

Figure 11: Interval Partitioning

4.2 Observation

First we define the **depth** of a set of intervals is the maximum number that pass over any single point on the time-line. The main observation in this case is:

$$\text{Number of classrooms needed} \geq \text{depth} \tag{16}$$

Then we apply greedy algorithm to consider lectures in increasing order of start time, and assign lecture to any compatible classroom. The algorithm is shown below:

```
Sort intervals by starting time so that s_1 ≤ s_2 ≤ ... ≤ s_n.
d ← 0        number of allocated classrooms

for j = 1 to n {
  if (lecture j is compatible with some classroom k)
    schedule lecture j in classroom k
  else
    allocate a new classroom d + 1
    schedule lecture j in classroom d + 1
    d ← d + 1
}
```

Figure 12: Greedy Algorithm

And the result is shown below:

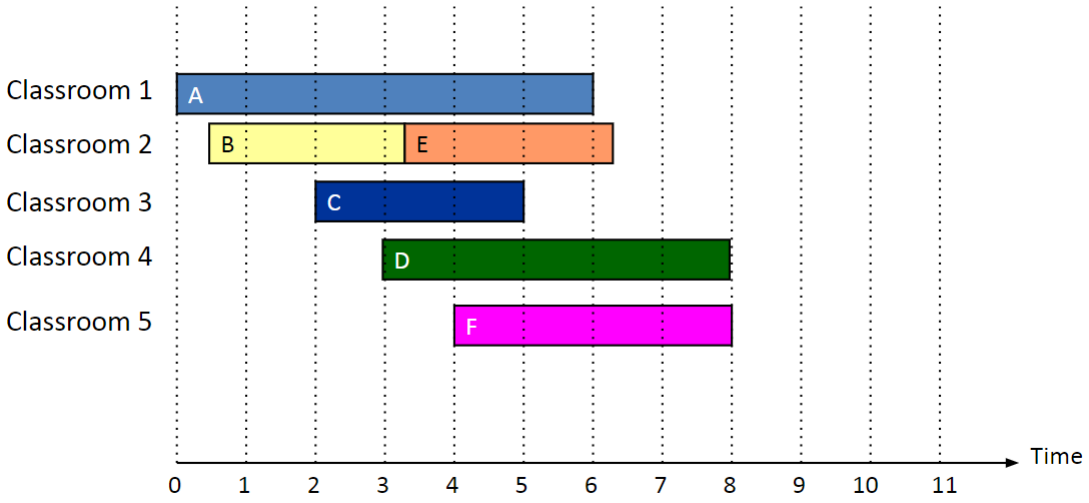

Figure 13: Greedy Algorithm Result

Here comes another important observation: **Greedy algorithm never schedules two incompatible lectures in the same classroom.**

4.3 Proof

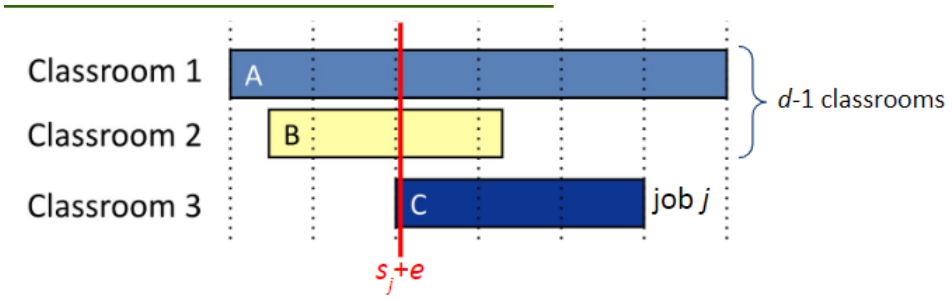

Figure 14: Greedy Algorithm Proof

The proof includes the following steps:

1. Define d as the number of classrooms that the greedy algorithm allocates, which is also the index of last classroom.
2. The last classroom d is opened because we need to schedule a job, say j, that is incompatible with all $d-1$ other classrooms.
3. In **each of the** $d-1$ **classrooms**, there exists a job which starts no later than s_j and ends after s_j, otherwise there will be space in other classrooms for job j, no need classroom d.
4. Thus, we have d lectures overlapping at time $s_j + e$, which is the time point which just passes s_j.
5. Which means depth needs to be greater or equal to d.
6. Recall the previous observation, $d \geq$ depth.
7. Therefore we have $d =$ depth and the greedy solution must be optimal.

5 Summary

There are three main methods to prove greedy algorithm:

1. **Greedy algorithm stays ahead**: Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's. (Interval Scheduling)
2. **Exchange argument**: Gradually transform any optimal solution to the one found by the greedy algorithm without hurting its quality. (Minimizing Maximum Lateness)
3. **Structural**:Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound. (Interval Partitioning)