Greedy Scheduling Algorithms

1 Interval Scheduling

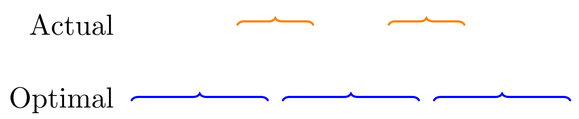
Input: A set of requests R for a resources, where $\forall r \in R$, there is a start time s_i and finish time f_i.

Output: A subset of R with no conflicts that maximizes the number of scheduled events.

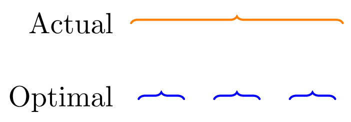
1.1 Greedy Criterion

A number of schemes are possible for the algorithm, including shortest request first, earliest start time first, and earliest finish time first.

Shortest request is suboptimal from the following counterexample, where it would schedule two requests instead of the optimal 3. By extrapolating this principle, shortest request could be off by a factor of 2 from the optimal schedule because one request can conflict with 2 optimal requests.



Earliest start time is actually a horrible criterion, as demonstrated by the following counterexample:



In the end, the optimal greedy criterion is actually sorting requests by earliest finish time, because it frees up the shared resource ASAP. Note this can be implemented in $\Theta(n)$ time by one walkthrough of the list - pick the first valid element, then walk along the list until the next valid start time, and so on.

1.2 Proof

Let O be some optimal schedule and G be the greedy schedule generated by the algorithm such that $O = G$ up to some index j. From our greedy criterion, we know that $f_j^G \leq f_j^O$. Therefore, we can swap O_j with G_j without introducing any more conflict, producing a new still-optimal schedule O'.

By induction, it is possible to transform any optimal schedule into our greedy schedule while maintaining optimality. Therefore, the greedy algorithm produces an optimal schedule.

2 Interval Coloring

Input: Set of requests $R(s_i, f_i)$.

Output: A minimum coloring of requests such that there is no conflict within a color.

2.1 Greedy Criterion

The greed criterion for this algorithm is actually quite straightforward - assign the smallest non-conflicting color to the next activity sorted by start time.

```
for $r_i$ $\in$ R:
    for $r_j$ $\in$ R | j < i:
        if $r_j$ conflicts with $r_i$:
            exclude color of $r_j$

    if $\exists$ color not excluded:
        assign color to $r_i$
    else:
        leave $r_i$ uncolored
```

2.2 Proof

Define d as the depth of the set of requests R, or the maximum number of conflicting requests at some point in time.

Claim: No interval ends up unlabeled, and only d colors are used.

Proof: Consider one of the requests r_i, and suppose there are t requests earlier in the sorted order that conflict with it. This means that the depth at some point in r_i is $t + 1$, which, given our definition of d, means $t + 1 \leq d$ and so $t \leq d - 1$. It follows then that if we use d labels, at least 1 will be free to color. Extending this analysis also proves that exactly d colors are used.

Because the algorithm by design never assigns conflicting requests to the same color and only uses d colors total, it produces the optimal coloring of requests.

3 Minimizing Lateness

Input: A set of tasks $T(t_i, d_i)$ where t_i is the duration of a task and d_i is the deadline.

Output: A schedule of tasks that minimizes the maximum amount of time that a task overshoots its deadline.

3.1 Greedy Criterion

Counterintuitively, we get the best result by sorting by earliest deadlines first. Even though half of the input is ignored, this scheme still produces an optimal schedule.

3.2 Proof

Claim: The greedy algorithm does not have any *idle time* - when there are jobs left but the machine is not working - and there exists an optimal schedule with no idle time. Both these points are obvious.

Now define an *inversion* in a schedule as a job i with deadline d_i is scheduled before a job j with deadline $d_j < d_i$. By definition, the greedy schedule has no inversions.

Claim: All schedules with no inversions and no idle time have the same maximum lateness.

Proof: If 2 schedules have no inversions or idle time, then they can only differ in the order in which jobs with identical deadlines are scheduled. Consider some deadline d such that in both schedules, the jobs with deadline d are all scheduled consecutively. Among the jobs with this deadline, the last one has the greatest lateness, the sum of all durations of these jobs, which does not depend on the order.

Now suppose there exists some optimal schedule O with no idle time but with inversions - that is, there is a pair of jobs i, j such that $j = i + 1$ and $d_j < d_i$. The greedy schedule G is identical to O up to the first task in this pair, but G has no inversions.

Let l_i^O, l_j^O be the lateness of O_i, O_j, and l_i^G, l_j^G be lateness of G_i, G_j. Let t be the common start time between the two schedules for tasks i and j. It's easy to see that, given a non-negative lateness,

$$l_i^O = t + t_i - d_i \tag{1}$$
$$l_j^O = t + (t_i + t_j) - d_j \tag{2}$$

and similarly for G,

$$l_i^G = t + (t_i + t_j) - d_i \tag{3}$$
$$l_j^G = t + t_j - d_j \tag{4}$$

By definition of an inversion, we know that $d_j < d_i$, and because a task must have non-negative duration,

$$l_j^O = t + (t_i + t_j) - d_j > t + t_i - d_i = l_i^O \tag{5}$$

Therefore, the maximum lateness of O for this inversion is $L^O = l_j^O$.

A similar analysis for lateness in G shows that

$$l_i^G = t + (t_i + t_j) - d_i < t + (t_i + t_j) - d_j = L^O \tag{6}$$
$$l_j^G = t + t_j - d_j \leq t + t_i + t_j - d_j = L^O \tag{7}$$

Therefore, no matter if $L^G = l_i^G$ or $L^G = l_j^G$, $L^G \leq L^O$, so tasks i and j in O can be exchanged without increasing the maximum lateness. By induction then we can turn any optimal schedule O with inversions into a greedy schedule G without inversions by flipping all inversions in O, so therefore the algorithm produces an optimal schedule.

4 Cache Scheduling

Input: A list of jobs j each with a weight w_j and length l_j.

Output: An ordering of jobs that minimizes the weighted sum of completion times, where completion time is the amount of wall time it takes from the beginning to the completion of job j.

$$C_j = \sum_1^j l_j$$

$$min(\sum_1^n w_j C_j)$$

4.1 Greedy Criterion

4.1.1 Special Cases

Identical lengths In this case, we choose the larger-weight jobs first to minimize the term in the sum (when n is smaller).

$$\sum_1^n w_j C_j = \sum_1^n w_j \cdot n \cdot L = L \cdot \sum_1^n w_j \cdot n \tag{8}$$

Identical weights In this case, we choose the shortest jobs first to minimize the sum over completion times.

$$\sum_1^n W C_j = W \sum_1^n C_j \tag{9}$$

4.1.2 Resolving Conflicting Advice

The key idea is to assign each job a score that is directly proportional with w and indirectly proportional with l. Taking two guesses at such a score, there are

- $s_j = w_j - l_j$
- $s_j = w_j / l_j$

To distinguish between the two, find the simplest example that produces different outputs. In this case, for the two jobs $j_1(w = 3, l = 5), j_2(w = 1, l = 2)$, the second score yields the lower weighted sum, so it is better than the first.

5 Proof

Claim: sorting by decreasing ratio w_j/l_j is always correct

Proof: by exchange argument and contradiction.

Assume: no ties in ratios (doesn't change algorithm, only proof)

Fix an arbitrary input of n jobs with σ as the greedy schedule from the algorithm and σ^* being a theoretically optimal algorithm. Next, rename the jobs $j_1, j_2, \ldots, j_n$ such that $s_1 > s_2 > \ldots > s_n$, so that σ just involves ordering the jobs in numerical order.

Assuming $\sigma^* \neq \sigma$, there must exist two consecutive jobs $(i, j) \in \sigma^* \mid i > j$. Because σ is a unique schedule (properly ordered), the only way σ^* can be different is if at least one pair of jobs is flipped.

Suppose we exchange the order of these two jobs in σ^*, leaving all other jobs unchanged. Jobs other than these two have no impact on completion time after this operation, while C_i increases and C_j decreases.

$$i > j \rightarrow \frac{w_i}{l_i} < \frac{w_j}{l_j} \tag{10}$$

$$w_i l_j < w_j l_i \tag{11}$$

From the definition of weighted sum, it's clear that the cost associated with this exchange is outweighed by the benefit. Therefore, it is impossible for σ^* to be optimal, violating out assumption in the first step, so σ is an optimal schedule.