Greedy algorithms for scheduling problems
(and comments on proving the correctness of some greedy algorithms)

Vassos Hadzilacos

1 Interval scheduling

For the purposes of the interval scheduling problem, a ***job*** is an interval $i = [s, f]$, where s and f are real numbers such that $s < f$; s and f are, respectively, the ***start*** and ***finish*** times of job i. We say that two jobs ***overlap*** if their intersection contains more than one real number; that is, if each starts before the other has finished. (By this definition if a job starts exactly when the other finishes, the two jobs do not overlap.)

The interval scheduling problem is defined as follows:

INPUT: A finite set I of jobs.

OUTPUT: A maximum cardinality set of jobs in I, no two which overlap.

Following is a greedy algorithm for the interval scheduling problem.

```
    IntervalScheduling(I)
1   sort the intervals in I by increasing finish time
2   A := ∅
3   for each interval i in sorted order do
4       if i overlaps no interval in A then A := A ∪ {i}
5   return A
```

We now describe two ways to prove that this algorithm is correct, i.e., that it returns a maximum cardinality subset of the given set of intervals in which no intervals overlap.

1.1 The "greedy-stays-ahead" proof

Consider the set of intervals A constructed by the algorithm. By the test in line 4, this set is feasible: no two intervals in it overlap. Let $j_1, j_2, \ldots, j_k$ be the list of these intervals in left-to-right order (this is well defined since the intervals do not overlap). Note that this is the order in which the algorithm adds these intervals to A. Let A^* be any optimal set of intervals and $j_1^*, j_2^*, \ldots j_m^*$ be the list of intervals in A^* listed left-to-right (this is also well defined, since A^* is certainly feasible and so its intervals do not overlap).

Let $s(j)$ and $f(j)$ denote, respectively, the start and finish times of interval j.

Lemma 1 ("Greedy-stays-ahead" lemma) *For every t, $1 \leq t \leq k$, $f(j_t) \leq f(j_t^*)$.*

PROOF. By induction on t. The basis $t = 1$ is obvious by the algorithm (the first interval chosen by the algorithm is an interval with minimum finish time).

For the induction step, suppose that $f(j_t) \leq f(j_t^*)$. We will prove that $f(j_{t+1}) \leq f(j_{t+1}^*)$. Suppose, for contradiction, that $f(j_{t+1}^*) < f(j_{t+1})$. This means that j_{t+1}^* was considered by the greedy algorithm before j_{t+1}. We have

$$f(j_t) \leq f(j_t^*) \leq s(j_{t+1}^*) < f(j_{t+1}^*) < f(j_{t+1}). \qquad (*)$$

(The first inequality is by the induction hypothesis, the second because A^* is feasible and the jobs in A^* are indexed left-to-right, the third because every interval starts before it finishes, and the fourth is by the

assumption we made for the sake of contradiction.) Thus, the algorithm considered the interval j^*_{t+1} after j_t and before j_{t+1}, but did not add it to A (otherwise, j^*_{t+1} and not j_{t+1} would have been the $(t+1)$-th interval added to A). So j^*_{t+1} starts before j_t finishes, contradicting $(*)$. Therefore $f(j_{t+1}) \leq f(j^*_{t+1})$, which completes the induction step. □

Using this lemma, we can prove that the greedy algorithm is correct.

Theorem 2 *The set of intervals A produced by the greedy algorithm is optimal.*

PROOF. Since A is feasible, $k \leq m$. Suppose, for contradiction, that A is not optimal; i.e., $k < m$. So A^* contains an interval j^*_{k+1}. By Lemma 1, $f(j_k) \leq f(j^*_k)$. Since A^* is feasible and its intervals are indexed left-to-right, $f(j^*_k) \leq s(j^*_{k+1}) < f(j^*_{k+1})$. So (a) the greedy algorithm considers interval j^*_{k+1} after interval j_k, and (b) j^*_{k+1} does not overlap any of the intervals of A. Thus, the greedy algorithm should add it to A, contradicting that A has only k intervals. □

1.2 The "promising set" proof

There is an alternative proof of a similar nature that uses what is sometimes called the "promising set" argument. Specifically, we prove that:

Lemma 3 ("Promising set" lemma) *For every iteration i of the loop there is an optimal set A^*_i of intervals that contains the set of intervals A_i in variable A at the end of that iteration.*

PROOF. By induction on i. The basis $i = 0$ is obvious, since at the end of the "0-th iteration" of the loop, i.e., just before we enter the loop for the first time, A_0 is empty.

For the induction step, assume (by induction hypothesis) that A_i is a subset of some optimal set A^*_i of intervals. We will prove that A_{i+1} is a subset of some optimal set A^*_{i+1} of intervals. (Note that A^*_{i+1} may be a different optimal set than A^*_i.) This is obvious if no interval is added to A in iteration $i+1$, or if the interval j added to A in iteration $i+1$ happens to be in A^*_i: In both of these cases, we take $A^*_{i+1} = A^*_i$ and we are done. So, suppose that some interval j is added to A in iteration $i+1$ and $j \notin A^*_i$.

In this case, first we note that A^*_i contains some interval j^* that overlaps with j: for, otherwise, we could add j to A^*_i and obtain a feasible set with more intervals than A^*_i, contradicting that A^*_i is optimal. Second, we note that A^*_i does not contain two intervals that overlap with j: for, otherwise, both of them would start after all the intervals in A_i finish (because A_i is, by induction hypothesis, a subset of A^*_i, which is optimal and hence feasible), and one of them, call it j', would finish before j finishes; so the greedy algorithm would have added j', rather than j, to A in iteration $i+1$. So, there is exactly one interval j^* in $A^*_i - A_i$ that overlaps with j. Let A^*_{i+1} be the set obtained from A^*_i by replacing j^* with j; i.e., $A^*_{i+1} = (A^*_i - \{j^*\}) \cup \{j\}$. The set A^*_{i+1}, (a) is feasible (since j^* is the only interval in A^*_i that overlaps with j); (b) has the same number of intervals as the optimal set A^*_i; and (c) contains $A_i \cup \{j\} = A_{i+1}$ (by construction). So, A^*_{i+1} is an optimal set that contains all the intervals in A_{i+1}, as wanted. □

We now prove Theorem 2 using Lemma 3 instead of Lemma 1.

PROOF. By Lemma 3, there is an optimal set $A^* \supseteq A_n$. We claim that, in fact, $A_n = A^*$. Suppose, for contradiction, that there is some interval $j \in A^* - A_n$. Since A^* is optimal (and hence feasible), j does not overlap with any interval in A^* other than itself. Since $A_n \subseteq A^*$ and $j \notin A_n$, j does not overlap with any interval in A_n. So, when j was considered for inclusion in A by the algorithm, it would have be added to it, contradicting that $j \notin A_n$. Thus, $A_n = A^*$, and so the greedy algorithm returns an optimal set. □

2 Minimum lateness scheduling

For the purposes of this problem, a ***job*** i is associated with a ***duration*** $t(i) > 0$ (how long it takes to execute the job) and a ***deadline*** $d(i)$ (the time by which it must finish). The ***release time*** r specifies the earliest time when any job can start. A ***schedule*** S for a set of jobs specifies, for each job i in the set, a start time $S(i) \geq r$ so that for any two distinct jobs i and j the intervals $[S(i), S(i) + t(i)]$ and $[S(j), S(j) + t(j)]$ do not overlap (one job finishes no later than the other starts). The ***lateness*** $\ell(i, S)$ ***of job*** i ***in schedule*** S is the amount of time by which i misses its deadline in S (zero, if i meets its deadline) — i.e., $\ell(i, S) = \max\big(0, S(i) + t(i) - d(i)\big)$. The ***lateness of schedule*** S is the maximum lateness of any job in the schedule, i.e., $\max_{1 \leq i \leq n} \ell(i, S)$.

The ***minimum lateness scheduling problem*** is specified as follows:

INPUT: A set of n jobs, $i = 1, 2, \ldots, n$, with their durations $t(i)$ and deadlines $d(i)$, and a release time r.

OUTPUT: A schedule of these jobs with minimum lateness.

```
    MinLateness(t[1..n], d[1..n], r)
1   sort the jobs in increasing deadline
2   last := r
3   for each job i in sorted order do
4       S(i) := last
5       last := S(i) + t(i)
6   return S
```

We say that a schedule S ***has no gaps*** if there is a permutation $i_1, i_2, \ldots, i_n$ of $1, 2, \ldots, n$ (intuitively, the order in which the jobs are done in S) so that $S(i_1) = r$ and for every k, $1 \leq k < n$, $S(i_{k+1}) = S(i_k) + t(i_k)$. That is, the first job starts at the release time, and every other job starts immediately when the previous one ends. Thus, the above greedy algorithm produces a schedule with no gaps in which jobs appear in increasing deadline. We will prove that such a schedule is optimal.

Lemma 4 *There is an optimal schedule with no gaps.*

PROOF. Given any schedule S with gaps, by "pushing" all jobs as far to the left as possible, but keeping them in the same order as in S, we construct a schedule S' with no gaps whose lateness is no greater than the lateness of S. □

Theorem 5 *The greedy algorithm produces an optimal schedule.*

PROOF. For this proof we define an ***inversion*** between schedules S_1 and S_2 to be a pair of jobs i and j, $i \neq j$, that appear in opposite orders in S_1 and S_2; i.e., $S_1(i) < S_1(j)$ if and only if $S_2(j) < S_2(i)$.

Let S be a schedule produced by the greedy algorithm, and let S^* be an optimal schedule ***that has the smallest possible number of inversions relative to*** S. (This is well-defined by the well-ordering principle, since the number of inversions of a schedule relative to S is a non-negative integer.) By Lemma 4 we can assume, without loss of generality, that S^* has no gaps.

If the number of inversions between S^* and S is zero, then $S^* = S$ and we are done: S is optimal.

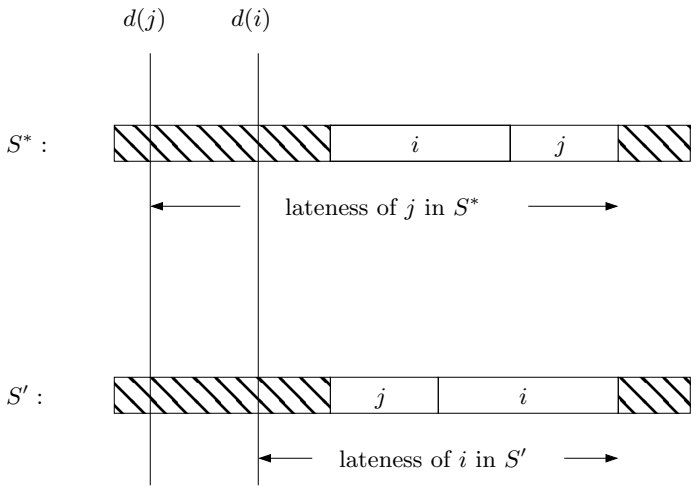
We will now prove that the number of inversions between S^* and S cannot be positive. Suppose the contrary. Then there are jobs i' and j' that cause an inversion between S^* and S. Without loss of generality, assume that i' appears before j' in S^*. This implies that there are two jobs i and j that cause an inversion between S^* and S ***and*** j ***is the right neighbour of*** i ***in*** S^*, i.e., $S^*(j) = S^*(i) + t(i)$. To see this, start with i' and its right neighbour in S^*. If these cause an inversion between S^* and S, we are done; if not, continue with the right neighbour of i' and its right neighbour. At some point, by the time the right neighbour becomes j', we must find a pair of successive jobs in S^* that cause an inversion relative to S:

otherwise, all jobs between i' and j', including these two, are in the same order in S^* as in S, contradicting the definition of i' and j'. Since i and j cause an inversion between S^* and S and i appears before j in S^*, we have that j appears before i in S. By the order in which the greedy algorithm puts jobs in the schedule S,

$$d(j) \leq d(i). \qquad (*)$$

Let S' be the schedule that is identical to S^* but with jobs i and j swapped — see the figure below. Note that S' has one inversion less than S^* relative to S,[1] so

$$\text{there are fewer inversions between } S' \text{ and } S \text{ than between } S^* \text{ and } S. \qquad (\dagger)$$



The lateness of each job other than i or j is the same in S' as in S^*, and the lateness of j is no greater in S' than in S^* (since j was moved to an earlier position). The lateness of i may have increased in S' compared to its lateness in S^* (because i was moved to a later position), ***but the lateness of*** i ***in*** S' ***is at most the lateness of*** j ***in*** S^*: This is because $d(j) \leq d(i)$ (by $(*)$) and therefore $\big(S'(i)+t(i)\big)-d(i) = \big(S^*(j)+t(j)\big)-d(i) \leq \big(S^*(j)+t(j)\big)-d(j)$. Therefore, the lateness of S' is no greater than the lateness of S^*. Since S^* is optimal, so is S'.

Therefore S' is an optimal schedule with fewer inversions than S^* relative to S (by $(\dagger)$), which contradicts the definition of S^*. So, the number of inversions between S^* and S cannot be positive, as wanted. □

This proof is an example of a technique known as the "exchange argument" that is used to prove the correctness of some greedy algorithms. Intuitively we show that any optimal solution can be transformed into the solution produced by the greedy algorithm in a finite number of ***optimality-preserving steps***, each of which involves swapping two elements — in the above proof, the jobs i and j that are out-of-order in the optimal schedule S^* under consideration relative to their order in the solution S produced by the algorithm.

3 Remarks on "greedy-stays-ahead", "promising-set", and "exchange" proofs

The "greedy-stays-ahead" and "promising set" arguments both capture the intuition that a greedy algorithm builds an optimal solution incrementally so that the partial solution constructed at each stage is "optimal so far". In the "greedy-stays-ahead" argument we do induction on the number of "pieces"

[1] It is for this reason that we wanted i and j to be successive in S^*.

that make up the optimal solution. In the "promising set" argument we do induction on the number of iterations that the greedy algorithm performs.

The "exchange" (or "switching") argument we saw in the proof of the greedy algorithm for minimum-lateness scheduling has a different flavour. There we prove that the greedy algorithm produces an optimal solution by showing how to transform an arbitrary optimal solution to the one constructed by the greedy algorithm through a series of "exchange" (or "switching") steps, each of which preserves optimality. This style of proof also uses induction (or its first cousin, the well-ordering principle), but in a different way. We don't do induction on the number of steps through which the algorithm builds its solution. Rather, we do induction on the number of optimality-preserving steps required to gradually "massage" the arbitrarily chosen optimal solution into the solution produced by the greedy algorithm.

The promising set argument bears some resemblance to the exchange argument in that the optimal solution that extends the partial solution may need to be changed in some iterations, and the change involves replacing (i.e., exchanging) some elements of an optimal solution with some elements of the solution constructed by the greedy algorithm. For instance, in the promising-set proof of the interval scheduling greedy algorithm, we had to change A_i^* to A_{i+1}^* by replacing j^* with j.

Of course, not all greedy algorithms' correctness is proved using one of these methods! They are, however, common enough that they deserve to be well understood.