Local exchange lemmas
An application to correctness of greedy algorithms

Periklis A. Papakonstantinou

York University

Greedy algorithms are elegant, conceptually simple and computationally efficient. In general there may be many ways to prove correctness of an algorithm. Students are responsible for exposing themselves to as many ways of proving correctness of an algorithm as possible. Most of these ways presumably share a great deal of ideas and arguments. Your textbook for 3101 is ideal for this kind of study. You are advised to study these handouts together with your notes from the lectures and in conjunction with the text. Note that exposing yourself to very similar arguments that prove the same thing considerably strengthens the ability of a person to understand but more importantly to devise proofs of correctness of algorithms. This is because there is no "cookbook" for proofs of correctness. Despite this, the reason why we study algorithmic paradigms (as opposed to a "free-style" study of combinatorial algorithms) is because we want to classify, to the extend possible, similar algorithmic solutions. And it is the case that for these "similar algorithms" we can also apply similar proofs of correctness. Our goal is to provide general frameworks that make this possible. In this set of notes we give a model example for proving correctness of greedy algorithms.

1 The problem

Problem: Unweighted Interval Scheduling

Input: A finite set of intervals S.

Output: A feasible schedule $S' \subseteq S$ of maximum cardinality (i.e. maximum number of elements) over all feasible schedules from S.

Side remark: Unlike other problems such as the stable-marriage problem or the sorting problem, here we could have relaxed the requirement of a correct output in a natural way. Note that there is no natural way that relaxes the requirements e.g. for sorting. In contrast, for Interval Scheduling it seems natural to say that we are happy with feasible schedules that have many but not the maximum number of intervals. Therefore, proving correctness for an algorithm for the above problem we prove that this algorithm works optimally; in the sense that we maximize a quantity. Hence in this course, unless we say something else the terms "correct" and "optimal" coincide.

2 A correct algorithm for Unweighted Interval Scheduling

We will present a greedy algorithm correct for *Unweighted Interval Scheduling*.

Intuition: When one first encounters this problem it is not immediately obvious that there exists a greedy algorithm which is correct for this problem. One reason for this is that when

one first attempts to devise a greedy algorithm for this problem many "natural" greedy criteria (e.g. choose the interval of the smallest length) fail to be correct. Recall that a greedy algorithm: (i) has a very local view of the input and (ii) makes locally optimal and *irrevocable* decisions. That is, the way the algorithm constructs the output is such that in each step the already processed part of the input corresponds to a sub-output which is (i) optimal for the already processed part of the input and (ii) can be extended to an optimal/correct output for the given input. Surprisingly, there is a greedy criterion that satisfies these requirements.

In what follows EFTF stands for "Earliest Finishing Time First"

EFTF[S]
(where S is a set of n intervals of the form $I_i = (s_i, f_i)$)
1 Order the intervals in non-decreasing finishing time order
that is, $Order : f_1 \leq f_2 \leq \ldots \leq f_n$.
2 $Schedule \leftarrow \emptyset$
3 **for** $i \leftarrow 1$ **to** n
do
4 **if** the interval I_i can be scheduled without overlapping an interval in $Schedule$
then $Schedule \leftarrow Schedule \cup \{I_i\}$
5 Output $Schedule$

Exercise 1. When we describe algorithms we abstract implementation details for the sake of clarity (focus on the main task of the algorithm). Implement efficiently the test in line 4.

Exercise 2. Provide details for an efficient implementation of EFTF and analyze its worst-case running time.

3 Proof of correctness

It is not clear at all why the above algorithm optimally computes a feasible schedule. Perhaps without much effort we can prove a loop-invariant (for every iteration) which states that in every iteration (at the end of every iteration) the intervals in $Schedule$ do not overlap. We will not prove this separately since we can get feasibility for free when proving optimality/correctness of EFTF for the problem.

Proof idea: We give a model-proof of correctness for greedy algorithms. The "heart" of this proof (as in every inductive proof) is the construction involved in the inductive step. *Very important remark: For reasons that we will explain later we cannot argue inductively that there is a single schedule that optimally extends every schedule constructed in every iteration of the algorithm.* In an inductive proof we use the objects that exist from the induction hypothesis in order to construct the objects we want in the inductive step. In

the inductive hypothesis we will assume that there exists an optimal schedule that extends the schedule constructed by the algorithm at the (end of) the k-th iteration. A priori there is no guarantee that the same schedule extends the schedule constructed by the algorithm at the $(k+1)$-th iteration. The non-trivial part of the construction in the inductive step is to show how to modify the optimal extension we have from the inductive hypothesis to an optimal extension for the schedule constructed in the $(k+1)$-th iteration.

Remark 1. As mentioned in class, for every greedy algorithm that we will see we will take for granted that we are provided with a "black-box" that *correctly* implements a sorting algorithm. The reason is that when studying greedy algorithms we want to focus on the main task of the algorithm rather than provide distracting details for implementation and correctness issues regarding a sorting procedure. It is the case that we study sorting procedures and their correctness separately. When studying greedy algorithms we take one step of abstraction that hides these issues.

Theorem 1. EFTF *is correct for Unweighted Interval Scheduling.*

Proof. We have to prove that if the input is in the correct form (i.e. the preconditions hold) then (i) EFTF terminates and (ii) if it terminates then the output is in the correct form (i.e. the postconditions hold).

Lemma 1. *If the input is in the correct form then the algorithm terminates.*

Proof. This is an iterative algorithm with only one for-loop. Therefore, termination follows by standard arguments. Here are the details. Suppose that the input is in the correct form (i.e. the algorithm terminates). We have assumed that the sorting procedure involved in the first line terminates. Therefore, we have to show that the loop terminates. In each iteration we use variable i as an indication of a quantity that strictly increases in the natural numbers. By the description of the algorithm this quantity cannot get above n. Therefore, the sequence $b_i = n - i$ is strictly decreasing in the natural numbers. This is because $i < i+1 \Rightarrow -i > -(i+1) \Rightarrow n - i > n - (i+1) \Rightarrow b_i > b_{i+1}$ and we also have that $i \leq n$ therefore $b_i = n - i \geq n - n = 0$.

Now it comes the more interesting part where we show partial correctness.

Terminology and notation: Say that the algorithm has read k intervals (i.e. it has completed k iterations). We denote by S_k the set of scheduled intervals *Schedule* at the end of the k-th iteration. When we say that S_k can be extended to an optimal schedule S_{OPT} we mean that *among the remaining intervals* $\{I_{k+1}, I_{k+2}, \ldots, I_n\}$ *there exist some of them which together with* S_k *can form an optimal schedule.* We also call S_{OPT} an optimal extension (or extension to a correct output) of S_k. In order to emphasize between (possibly) different optimal extensions for every iteration we will denote the optimal extension of S_k as S^k_{opt}.

Lemma 2. *For every iteration of the algorithm there exists an optimal extension among the remaining (i.e. not-yet-processed) intervals.*

Proof. We prove the lemma by induction on the iterations of the algorithm. Therefore, we show that the predicate $P(i)$ holds in every iteration of the algorithm, where $P(i)$: "at the end of the i-th iteration there exists an optimal extension S^i_{OPT} of the set of scheduled intervals S_i". (Therefore by induction we will show that for every i $P(i)$ holds - *note that the optimal extension S^i_{OPT} depends on the iteration and thus it may be different for every iteration*).

(Basis) For $i = 0$ we have that $S_0 = \emptyset$, and the empty set is a subset of every set, thus it is also a subset of an optimal schedule.

(Induction Step) Suppose that $P(k)$ holds (I.H). Let me rewrite what this means (sometimes it helps to explicitly see once more what you have from the induction hypothesis). We have that there exists an optimal extension S^k_{OPT} of S_k (hence $S_k \subseteq S^k_{OPT}$). We want to show that $P(k+1)$ is also true. Say that I_{k+1} is the interval read by the algorithm in the $(k+1)$-th iteration.

If I_{k+1} is rejected then by the way the algorithm works (now "we are on the $(k+1)$-th iteration") this means that I_{k+1} overlaps an interval from S_k. Therefore, $I_{k+1} \notin S^k_{OPT}$. We have that $S_{k+1} = S_k$ and since the interval overlaps S_k we have that the same extension $S^k_{OPT} = S^{k+1}_{OPT}$ optimally extends S_{k+1}.

Else, I_{k+1} is scheduled by the algorithm; i.e. $I_{k+1} \in S_{k+1}$.

If[1] $I_{k+1} \in S^k_{OPT}$ then again the previous (from the I.H.) optimal extension also extends $S_{k+1} \subseteq S^k_{OPT} = S^{k+1}_{OPT}$. (recall that we that the predicate asks to prove existence of an optimal extension; note that we prove existence of S^{k+1}_{OPT} by just showing it.)

Else, $I_{k+1} \notin S^k_{OPT}$.

If I_{k+1} does not overlap S^k_{OPT} then $(S^k_{OPT} \cup \{I_{k+1}\})$ is also a feasible schedule and this contradicts the induction hypothesis, since we have assumed that S^k_{OPT} is optimal (i.e. a feasible schedule of maximum number of elements).

Else, I_{k+1} does overlap S^k_{OPT}. Note that this is single non-trivial case. It is non-trivial because we cannot simply use the same set we have from the I.H. to extend S_{k+1}. We have that I_{k+1} overlaps at least one interval in S^k_{OPT}. Note that if it is possible to overlap more than one then we have just failed to prove correctness of this algorithm! But fortunately this is not the case. Suppose that I_{k+1} overlaps more than one intervals from S^k_{OPT}. Therefore, one of them must have smaller finishing time than I_{k+1}. (note that the intervals I_{k+1} overlaps in S^k_{OPT} are from $S^k_{OPT} \backslash S_k$ since we are in the case that I_{k+1} is scheduled by the algorithm and by the I.H. we know that S_k extends optimally to S^k_{OPT}). But by the way the algorithm works this is not possible since the algorithm schedules intervals in non-decreasing finishing time. Therefore, it should have considered this interval before I_{k+1}, thus a contradiction. Hence, I_{k+1} overlaps only one interval I from S^k_{OPT}. Therefore, the set $(S^k_{OPT} \backslash \{I\}) \cup \{I_{k+1}\}$ is also a feasible schedule (**here we performed a local exchange in the hypothesized optimal solution**) and thus by choosing $S^{k+1}_{OPT} = (S^K_{OPT} \backslash \{I\}) \cup \{I_{k+1}\}$, we have that $S_{k+1} \subseteq S^{k+1}_{OPT}$ which proves the lemma since $|S^{k+1}_{OPT}| = |S^k_{OPT}|$ (recall that one way of proving existence of an object is by just presenting it).

[1] In our argument all "ifs" are "nested" - this way we progressively narrow the possibilities.

Corollary 1. EFTF *is partially correct.*

Why is this corollary true? In class many times we just proved a loop-invariant (as in the above lemma) by not mentioning why does this loop-invariant (predicate) directly implies partial correctness. Here we proved that at the k-th iteration the constructed schedule can be extended to an optimal one (i.e. using intervals among the ones that have not yet been processed). Therefore, the above lemma implies that at the last iteration the scheduled intervals can be extended to optimal schedule. But we are already at the end (no more remaining intervals to process)! Therefore, at the last iteration the algorithm has already constructed an optimal schedule!

4 Remarks

In this set of notes we provide a general methodology of proving (partial) correctness of greedy algorithms. The whole methodology sums up to the following: we prove by induction on the iterations of the algorithm that the so far constructed solution extends to **an** optimal/correct solution.

Observe that in the induction step the single interesting case is when the constructed schedule does not extend to the optimal extension we have available from the I.H. Note that every algorithm (even an incorrect one) that simply maintains feasibility of the schedule would be able to successfully "pass" every case in the induction step but the last one! Therefore, our case analysis is essential in narrowing the possibilities and focusing on the important part of the algorithm that gives it credits for being the one that is correct. You are advised to carefully review the proof of correctness and understand why everything else is just the "environment" in which we prepare ourselves in order to finally being able to check what really this algorithm does and why it is correct.

Exercise 3. Change the description of the algorithm such that the intervals are ordered in non-decreasing starting time. Prove that this variation of the algorithm is not correct for Unweighted Interval Scheduling. Identify the exact part where the above proof of correctness fails for this algorithm.

Also, note that as always we have separated the proof of correctness into two parts. In one part we prove termination and in the other we prove partial correctness. This separation cleans things up since it helps us to isolate parts of the proof. Lets restate what is partial correctness: "If the input is in the correct and if the algorithm terminates then the output is in the correct form". There are a few cases where an algorithm may output the correct output but it does not terminate. For example, consider the EFTF algorithm and at the end (after the "Output" instruction) add an infinite loop. According to our definition of correctness this algorithm is not correct (correct algorithms always stop running at some point). But there are also cases where we make use of the fact that the algorithm terminates inside our proof of partial correctness. Recall that when we were dealing with the stable marriage problem we had to make use of the fact that the algorithm terminates inside our

proof of partial correctness. We had lemmas where at some point we had arguments of the form "since it terminates then this means...". Note that technically we didn't make use of the "full-power" of the hypothesis when we proved partial correctness of EFTF. But this is fine since we show something somehow stronger[2].

There is one subtle point in the above discussion that usually confuses people. Here is the question:

Observe that the decisions of the algorithm are irrevocable and the set constructed at the last iteration, clearly, can be used as an extension of every other iteration. But in the proof we did not prove by assuming the existence of a specific optimal extension. Instead we argued that every iteration may has its own and possibly different optimal extension. Of course at the end of the proof we can infer the existence of this single extension; but we cannot infer this before completing the induction! This is very awkward! Why we cannot give a more "direct" argument? Why does this happen? Unfortunately, there is no satisfactory intuitive answer to this. It has to do with the mathematics themselves. This is the "limitation" of an inductive argument. By induction we can only prove that $\forall i P(i)$ is true. Note that the universal quantifier comes as an implication of establishing the induction. When proving by induction there is absolutely no way to move something before this quantifier. Therefore, in case of our model-example the best we can do is to say that there exists an optimal extension for iteration i. But this "exists" is bounded by the "for all" that the inductive argument proves.

[2] Well, not really. We need the fact that the algorithm terminates in order to know that the "Output" instruction will be executed. But if we change the algorithm such that it outputs one by one the elements in the loop then we would have proved something stronger.