6.892: Algorithmic Lower Bounds, Spring 2019

Prof. Erik Demaine, Jeffrey Bosboom, Jayson Lynch

Problem Set 2 Solution

Due: Tuesday, February 19, 2019 at noon

Problem 2.1 [Problem Set Scheduling]. During the semester, you will have to plan when to work on problem sets from this and other courses, subject to constraints: you cannot start working on a problem set until it is released, you must finish a problem set before it is due, and you can work on only one problem set at a time. We can formalize these constraints in the following problems:

Sequencing with Release Times and Deadlines: Given a set T of tasks, where each task $t \in T$ has a positive integer length ℓ_t, cannot be started until a nonnegative integer r_t (its *release time*), and must be completed before a positive integer d_t (its *deadline*), is there a feasible one-processor schedule for T? In this problem, once the processor starts a task, it must finish that task before starting another task.

Preemptive Sequencing with Release Times and Deadlines is the same problem, except the processor is allowed to suspend work on its current task and start on another task at any time. A task t is completed when the processor has spent ℓ_t total time on the task.

(a) Prove that Sequencing with Release Times and Deadlines is weakly NP-hard by reduction from Partition.

Solution: Let $A = \{a_1, \ldots, a_n\}$ be an instance of Partition, and let $S = \sum_i a_i$. We construct an instance of Sequencing with Release Times and Deadlines as follows:

- For each number $a_i \in A$, we have a task t_i with $\ell_{t_i} = a_i$, $r_{t_i} = 0$, and $d_{t_i} = S + 1$.
- We have a 'separator' task s with $\ell_s = 1$, $r_s = \frac{1}{2}S$, and $d_s = \frac{1}{2}S + 1$.

Each task can be constructed in constant time, so this reduction takes linear time.

We need to show that A has a partition into two sets with sum $\frac{1}{2}S$ if and only if this set of tasks as a feasible schedule.

Suppose A has a partition into two sets A_1 and A_2 with sum $\frac{1}{2}S$. For each $a_i \in A_1$, we schedule t_i for some time in the interval $\left[0, \frac{1}{2}S\right]$, for each $a_i \in A_2$, we schedule t_i for some time in $\left[\frac{1}{2}S + 1, S + 1\right]$, and we schedule the separator task for $\left[\frac{1}{2}S, \frac{1}{2}S + 1\right]$. Since $\sum A_1 = \sum A_2 = \frac{1}{2}S$, there is enough time in each block to schedule all of the tasks, so this is a feasible schedule.

Conversely, suppose there is a feasible schedule for the tasks. Then the separator task must be scheduled during $\left[\frac{1}{2}S, \frac{1}{2}S + 1\right]$, since its length is the same as the time between its release and deadline. Each other task t_i is either scheduled for before or after the separator task. Let A_1 be the set containing a_i for each t_i scheduled before the separator task, and let A_2 be the set containing a_i for each t_i scheduled after the separator task. Since the total time to complete every task is $S + 1$, which is the same as the sum of the lengths of tasks, every time between 0 and $S+1$ must be occupied by a task. So the tasks scheduled before the separator task must have total length $\frac{1}{2}S$, and hence $\sum A_1 = \frac{1}{2}S$. Similarly $\sum A_2 = \frac{1}{2}S$, so A_1 and A_2 are a solution to the PARTITION problem.

An alternate solution is to pick one of the tasks t_i, and set its release time and deadline to $\frac{1}{2}S$ and $\frac{1}{2}S + a_i$, instead of using a separator. This forces t_i to be scheduled starting at time $\frac{1}{2}S$, which similarly requires the tasks to be partitioned into sets with total length $\frac{1}{2}S$.

(b) Prove that SEQUENCING WITH RELEASE TIMES AND DEADLINES is strongly NP-hard by reduction from 3-PARTITION.

Solution: Let $A = \{a_1, \ldots, a_n\}$ be an instance of 3-PARTITION, and let $S = \sum_i a_i$. We construct an instance of SEQUENCING WITH RELEASE TIMES AND DEADLINES as follows:

- For each number $a_i \in A$, we have a task t_i with $\ell_{t_i} = a_i$, $r_{t_i} = 0$, and $d_{t_i} = S + \frac{n}{3} - 1$.
- For each $j = 1, \ldots, \frac{n}{3} - 1$, we have a 'separator' task s_j with $\ell_{s_j} = 1$, $r_{s_j} = j(\frac{3}{n}S+1) - 1$, and $d_{s_j} = j(\frac{3}{n}S + 1)$.

Again each task takes constant time to construct, and there are a linear number of them, so the reduction takes linear time.

To show that the 3-PARTITION instance has a solution if and only if the SEQUENCING WITH RELEASE TIMES AND DEADLINES instance does, first observe that each separator s_j must be scheduled during $\left[j(\frac{3}{n}S + 1) - 1, j(\frac{3}{n}S + 1)\right]$, since there is just enough time to complete this tasks. This leaves $\frac{n}{3}$ gaps of size $\frac{3}{n}S$ in which to schedule the remaining tasks.

Suppose we can partition A into $\frac{n}{3}$ sets A_j $(j = 0, \ldots, \frac{n}{3} - 1)$ of size 3 with sum $\frac{3}{n}S$. For such a set $A_j = a_{i_1}, a_{i_2}, a_{i_3}$, we schedule t_{i_1}, t_{i_2}, and t_{i_3} to the interval $\left[(j-1)(\frac{3}{n}S + 1), j(\frac{3}{n}S + 1) - 1\right]$, i.e. between the $j-1$st separator and the jth separator. This interval has length $\frac{3}{n}S$, so there is just enough space for the three tasks; thus we have a feasible schedule.

Conversely, suppose there is a feasible schedule. Since the total length $S + \frac{n}{3} - 1$ of tasks is the same as the time available, the entire time must be filled. In particular, each gap between seperators of length $\frac{3}{n}$ must be entirely occupied by tasks, whose lengths must sum to $\frac{3}{n}$. We can assume $\frac{1}{4} < a_i < \frac{1}{2}$ and still have 3-PARTITION strongly NP-hard; this implies there are exactly three tasks scheduled in each gap, and their lengths sum to $\frac{n}{3}$. The elements of A corresponding to these three tasks sum to $\frac{n}{3}$, so we partition A based on which gap the corresponding task is scheduled in. The sets in the resulting partition have 3 elements which sum to $\frac{3}{n}S$.

It is also possible to design the reduction with $\frac{n}{3}$ or $\frac{n}{3} + 1$ separator tasks (instead of $\frac{n}{3} - 1$).

(c) Prove that PREEMPTIVE SEQUENCING WITH RELEASE TIMES AND DEADLINES is in P by giving a polynomial-time algorithm that solves it. Explain why your proofs for the non-premptive case do not hold under preemption.

Solution: Iterate through the tasks, sorted by deadline. For each task t, schedule it in the earliest times possible; it takes the first cumulative ℓ_t time after r_t not already taken by another task. If this results in t not being completed by d_t, output NO. Otherwise, after going through all tasks, output YES.

Suppose that this algorithm fails to schedule t before its deadline. In order to finish t on time, we must push work on another task to either before r_t or after d_t. Since tasks are scheduled to be completed as soon as possibly, we can't push it to before r_t. Since tasks scheduled before t have deadline before t, if we push it to after d_t that task will miss its deadline. So this algorithm finds a feasible schedule if one exists.

Each step of this algorithm takes polynomial time: we sort n tasks in $O(n \log n)$ time, and for each of n tasks, we divide it into at most n chunks and perform $O(n)$ arithmetic operations. So Preemptive Sequencing with Release Times and Deadlines is in P.

Another algorithm constructs a schedule in temporal order, where at each time we are working on the task with the earliest deadlines among released unfinished tasks. In order to do this in polynomial time, we must repeatedly skip ahead to the next 'event,' which is either a release time, a deadline, or the completion of task currently being worked on. Since there are at most $3n$ tasks, this can be done in polynomial time.

To show that this algorithm finds a feasible schedule if one exists, we consider the first time a feasible schedule differs from the schedule found by the algorithm. Suppose at this time the feasible schedule is working on task t_1 and the algorithm is working on task t_2. Then $d_{t_2} \leq d_{t_1}$, and the feasible schedule must work on t_2 later. We can swap this future work on t_2 and the current work on t_1 to obtain a new feasible schedule which matches the algorithm's schedule for longer. Repeating this, we eventually obtain a feasible schedule identical to that found by the algorithm; in particular, the algorithm finds a feasible schedule if one exists.

The hardness proofs above do not hold under preemption because we can now split a task across one of the separators, making separators do essentially nothing. This would be analogous to solving (3-)Partition where we can split numbers apart before adding them, which is much easier.

If an algorithm constructs a schedule by iterating through all times (working on the most urgent task at each one), the algorithm takes pseudopolynomial time. The lengths, release times, and deadlines of tasks may be exponentially large in the length of the input.