IN4301 Advanced Algorithms: Programming Exercise 1

Deadline: November 20, 23:50

For this exercise we expect two products from you: an *implementation* of the exact dynamic programming algorithm and the approximation, including the requested conversions and tests, and a *report on the comparison* between these algorithms, and the algorithms provided via Blackboard. The implementation and the report should be submitted in one file through CPM. For this exercise, it is allowed to work in **groups of two students**. Note that the grade for the two programming exercises together weighs exactly the same as all the homework exercises.

Minimum tardiness scheduling

You are given a single processor that has to process n jobs $j_1 \dots j_n$. Every job j_i is specified with a processing length l_i and a due time d_i. The processing length is the amount of time necessary to finish the job. The due time is the time at which a job is supposed to be ready. You are asked for an algorithm that constructs a schedule that minimizes the total *tardiness* of these n jobs. The tardiness of a job is the amount of time that it is too late, i.e., if job j_i is completed at time c_i, then the tardiness of j_i is $t_i = \max\{c_i - d_i, 0\}$. Your schedule always starts at 0, and the jobs are processed sequentially and without preemption.

As an example, given two jobs j_1 and j_2 with length and deadlines: $l_1 = 3$, $l_2 = 4$, $d_1 = 5$, $d_2 = 4$, your schedule should first process j_2 and then j_1:

- j_2 starts at time 0, is being processed for length $l_2 = 4$, and finishes at time $c_2 = 4$, and
- j_1 starts at time 4, is being processed for length $l_1 = 3$, and finishes at time $c_1 = 7$.

The total tardiness is $\sum_{i \in \{1,2\}} t_i = \max\{c_1 - d_1, 0\} + \max\{c_2 - d_2, 0\} = 2$.

The problem of minimizing the total tardiness is NP-hard. You are asked to design and implement:

- an exact algorithm (using dynamic programming), and
- an FPTAS

for dealing with this problem.

A greedy and a best-first implementation for this problem can be downloaded from Blackboard. These implementations can be used as templates for your algorithm, and to test your implementations. You need to test the performance (quality and run-time) of all algorithms on a test-set that can also be downloaded from Blackboard.

Reading material

In order to implement the exact algorithm and the approximation algorithm, you have to read the papers listed below. The papers can be found on Blackboard. You are also allowed to consult any of the references in the given papers, but make sure that you include proper references in your report.

> Lawler, E. L. *A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness.* Annals of discrete Mathematics 1 (1977): 331-342.
>
> Lawler, E. L. *A fully polynomial approximation scheme for the total tardiness problem.* Operations Research Letters 1.6 (1982): 207-208.
>
> Koulamas, C. *The single-machine total tardiness scheduling problem: review and extensions.* European Journal of Operational Research 202.1 (2010): 1-7.

The paper by Lawler (1977) describes an exact dynamic programming solution for the minimum tardiness problem. In order to implement the exact dynamic programming algorithm, it is necessary to understand section 3 of the paper. Unfortunately, there are small errors: in Equation 3.1, the term $T(S(i, k + \delta, k'), t))$ should be $T(S(i, k' + \delta, k'), t))$, on the next page $C_{k'}(\delta)$ should be defined as $t + \sum_{i \leq j' \leq k' + \delta} p_{j'}$, and in the sentence "...where the summation is taken over all jobs j' in $S(i, k + 0, k')$" the first k should be k' too.

The paper by Lawler (1982) presents an approximation scheme for the same problem, based on the exact algorithm from Lawler (1977). Therefore, your implementation of the exact algorithm must be correct before you start implementing the FPTAS. The FPTAS should be based on the scaling technique that is described in the paper.

The paper by Koulamas (2010) provides a review of theoretical developments for the single machine minimum tardiness problem. This paper also contains some small errors, such as on page 2, left column, line 22, you should insert *"let j be such that"* before "$p_j = max_{i=1...n} p_i$". It is not required to read and understand everything that is described in the paper, but it contains valuable information that you may want to use.

Part I: exact dynamic programming algorithm

First you need to design and implement an algorithm that is based on the dynamic programming solution from Lawler (1977). Do not forget to account for the errata. The solution by Lawler (1977) is unable to deal with jobs having the same processing time, but you can redefine the set S to support such problem instances. It is not strictly necessary to implement an *iterative* dynamic programming algorithm, but your algorithm should store and reuse solutions to subproblems to reduce the number of computations. For more information regarding dynamic programming you can consult chapter 6 from the book Algorithm Design by Kleinberg and Tardos.

To test and debug your implementation, you can define small testcases for which you can compute a solution manually. It is important that the test cases include corner cases that may trigger problems. Additional testcases will be provided on Blackboard. Since the correctness of the FPTAS depends on the correctness of the exact algorithm, you can ask the assistants to verify your implementation before starting with the second part of the programming exercise.

Part II: approximation scheme

After implementing the exact dynamic programming algorithm, you need to implement an approximation algorithm that uses the scaling technique described in Lawler (1982). This involves scaling of the problem instance, running the exact algorithm and computing the (approximate) tardiness for the unscaled problem instance.

In the implementation of your approximation algorithm, it is crucial that the approximate tardiness is computed using the unscaled jobs. First you need to derive a schedule using the scaled jobs, and then you can compute the approximate tardiness using the unscaled jobs based on the schedule that you obtained.

Implementation requirements

The two algorithms should be tested against the given standard algorithms. You are free and encouraged to adapt and improve any of these implementations. However, you have to include the standard implementations in your tests as well. It is recommended that you implement the algorithms in Java. Otherwise, you would have to re-implement the given algorithms in the language of your choice (to enable a fair running time comparison). If you want to use another programming language, please discuss this with the assistants during one of the lab sessions. It is allowed to make use of (both standard and algorithmic) libraries, but you should indicate which ones you use. The main loop of your algorithm should be written by yourself.

Automated tests will be used to verify the correctness of the implementations. Therefore, your program should meet the following requirements:

- The executable should accept two arguments: an integer representing the epsilon value of the FPTAS, and the path to an instance file. This path should be the last argument.
- The executable should print two integers: the optimal tardiness and the approximation (in this order, separated by a space).

Report

After implementing both part I and part II, we expect a brief report that contains at least the following:

1. A brief description (including pseudo-code and run-time analysis) of the exact (dynamic programming) algorithm.
2. A brief description (including pseudo-code and run-time analysis) of the FPTAS.
3. Your expectations before running the algorithms on all test problems.
4. A comparison of run times and tardiness scores of all four algorithms on each of the test-set problems, including the following plots.
    (a) run time (y-axis) versus problem sizes (x-axis)
    (b) run time (y-axis) versus Relative Due Date (RDD) (x-axis)

(c) run time (y-axis) versus Tardiness Factor (TF) (x-axis)

(d) furthermore, think of and include some plots to visualize the combined effect of RDD and TF on tardiness

Note that the parameters RDD and TF are used in the filenames of the instances. Make sure that the graphs in the report are readable and easy to understand.

5. A discussion of some important results of this comparison.
6. A motivation for which approach you would most likely use in practice.

Your report on the algorithm should meet the following constraints:

- It should be at most five sides of an A4 (excluding an optional front page, and space used by diagrams, pseudocode, or figures), and in PDF.
- It should be written in clear, readable, and correct **English**.
- It should contain your name(s), student number(s), the exercise number.
- It should contain clearly readable graphs of the performance of the algorithm (use for example gnuplot). When discussing your graphs it is not necessary to explain things that can be seen in the graphs (e.g., "the red line increases"). Instead, please try to explain **why** results have been observed (e.g., "We think that Algorithm A performs better than B in our experiments because ...").
- When presenting experimental results, it is important that the reader is able to verify how the experiments have been executed, such that the results are reproducible.

We will check both your report as well as your code on correctness, clarity, quality, and completeness with respect to the requirements. Grades and feedback will be given through CPM.

Good luck!