Programming Exercises Advanced Algorithms

Exact and approximation algorithms

September 28, 2009

Expected results

For this exercise we expect two products from you: An *implementation* of the requested algorithms in a programming language of your choice, and a report on the *comparison* of these algorithms. Both 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 this programming exercise weighs **three times** as strong as the homework exercises. The constraints for the implementations are as follows:

- It is allowed to make use of (both standard and algorithmic) libraries, you should indicate which ones you use.
- The main loop should be written by yourself and fit on one A4 paper.
- It meets the requirements for neat programming from earlier courses.

Your report on the comparison of the different algorithms should meet the following constraints:

- It should be at most six sides (three double-sided pages) of an A4 (excluding an optional front page, space used by diagrams, and in PDF.
- It should be written in clear, readable, and correct **English**.
- It should contain clearly readable graphs of the performance of the different algorithms (use for example gnuplot).
- It should contain your name, student number, the exercise and version number, and
- for each question a clear explanation about how you obtained the answer.

Submit your implementation and report through CPM before the deadline has passed. The most recent document submitted before the deadline will be graded.

Assessment and feedback

We will check both your report as well as your code on

- correctness,
- clarity,
- quality, and
- completeness with respect to the requirements above.

Your grade and the feedback on your answers will be given through CPM.

1 Minimum tardiness scheduling

You are given a single computer processor for 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 have to construct 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 $t_i = \max\{c_i - d_i, 0\}$. The computing time of your schedule always starts at 0, and the processes processes the jobs 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 $c_2 = 4$,
- 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 two advanced algorithms for dealing with this problem:

1. An exact algorithm (using dynamic programming).
2. An approximation algorithm (that combines scaling with the DP algorithm).

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

For the implementation of the advanced algorithms we are going to use some theoretical results that bound the number of solutions of the tardiness problem. These results can be found in the following paper:

> C. Koulamas, *The single-machine total tardiness scheduling problem: Review and extentions*, European Journal of Operations Research, *article in press*, 2009, Elsevier

You should read this paper and implement a dynamic programming approach that uses the rightmost decomposition assumption by Lawler. In addition, you should use the scaling technique described in the paper to construct an FPTAS for the problem. You are allowed to consult any of the references in the given paper.

The resulting advanced 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. We expect from you a small report on these tests that contains at least the following:

1. A small description (including pseudo-code and run-time analysis) of the exact algorithm.
2. A small description of the intuition behind the approximation algorithm.
3. Your expectations before running the algorithms on all test problems.
4. A clear comparison of both the run-times and the tardiness scores of all four algorithms on each of the test-set problems, including graphs.
5. A discussion of some important results of this comparison.
6. A motivation for which approach you would be most likely to use in practice.