MAST90098 - Approximation Algorithms and Heuristics

Project - 2016

Due date: 11am Monday 17th October 2016

In this project you will be asked to design various heuristics for the Minimum Makespan Scheduling Problem. The problem is defined as follows:

Makespan Scheduling Problem (MS)

Schedule n jobs with designated processing times on m identical machines in such a way that the whole processing time is minimised.

Input: Positive integers $p_1, p_2, ..., p_n$ and an integer $m \geq 2$ for some $n \in \mathbb{N} - \{0\}$. Each p_i is the processing time of the ith job on any of the m available machines.

Constraints: For every input instance $(p_1, ..., p_n, m)$ of MS, $\mathcal{M}(p_1, ..., p_n, m) = \{S_1, S_2, ..., S_m | S_i \subseteq \{1, 2, ..., n\}$ for $i = 1, ..., m, \bigcup_{k=1}^{m} S_k = \{1, 2, ..., n\}$, and $S_i \cap S_j = \emptyset$ for $i \neq j\}$.

Cost: For each $(S_1, ..., S_m) \in \mathcal{M}(p_1, ..., p_n, m)$,
$cost((S_1, ..., S_m), (p_1, ..., p_n, m)) = \max\left\{\sum_{\ell \in S_i} p_\ell | i = 1, ..., m\right\}$.

Goal: *minimum*

The project tasks are as follows:

1. **Design, implement and experimentally test a *greedy local search* (GLS) algorithm for MS**. This is simply a local search algorithm where a lowest cost neighbour is picked at any iteration. The following subtasks are required

   (a) Define a suitable k-exchange neighbourhood for MS.

   (b) Implement the GLS algorithm with this neighbourhood in Matlab or R, where k is one of the input parameters.

   (c) Perform an experimental study on randomly generated instances of MS across a range of n and m values in order to select an appropriate value for k. The tradeoff between solution quality and the running time of your algorithm is the main consideration here.

   (d) Using the value of k you found in the previous task, perform an experimental study to test the performance (in terms of both speed and quality of solutions) of your algorithm. Once again, you should test your algorithm for a range of randomly generated instances, across a range of n and m values. It is important to test on small instances, as well as trying push the limit of your algorithm, i.e., try to what the size of the largest input instances are that your algorithm can handle in "reasonable" time. *Save these input instances, as they will be used again later.*

   (e) Write a report detailing the solutions to the above four tasks. Graphs and tables should be used, for both Task (c) and (d), to report on the speed and accuracy of your algorithm. Note that a suitable output for your algorithm will be a description of some feasible solution (i.e., the list of jobs to be processed on each machine and the makespan of the solution). *Your algorithm should also output the processing time required to find the solution.*

2. **Design, implement and experimentally test a *variable depth search* (VDS) algorithm for MS**. The same k-exchange neighbourhood should be used as in the previous task. The following subtasks are required

   (a) Write detailed pseudo-code for your algorithm in the report, and provide justification that conditions (i) and (ii)(a),(ii)(b) in Section 3.6.2 of the textbook are satisfied by your algorithm.

   (b) Implement your algorithm in Matlab or R.

   (c) Perform an experimental study for your algorithm on the same instances you used in Task 1(d). Draw graphs comparing the performance of your algorithm to your GLS algorithm.

   (d) Once again, the above task should be detailed in the form of a report.

3. **Design, implement and experimentally test a heuristic of your choice (eg., genetic algorithm, simulated annealing, etc.)**. Detail all aspects of your algorithm, and the experimental results in the report. The same test instances used in Tasks 1(d) and 2(c) should be used. NB: As in the selection of k in Task 1(c), *experimental testing should be used in order to justify the choice of any free parameters in the heuristic* (eg., the cooling schedule for simulated annealing). Experimentally compare your heuristic to your GLS and VDS algorithms, and draw graphs to show the results.

Important Notes

- Students must work in **groups of at most three**. It is the groups responsibility to ensure that everyone contributes their fair share to the project. All members of the group will receive the same mark.
- It is recommended that members of the group share a Dropbox folder for their work. This is so that all group members can have access to all parts of the project at any time.
- This project contains an extensive programming component. It is assumed that all students are able to program. The lecturer will **not** assist in issues relating to the debugging of code. The code will not be directly marked. However, marks may be deducted if code is untidy. Remember to place sufficient comments within your code for readability.
- Students may research ideas on the web, but full credit to the relevant authors must be given if the students use any of these ideas. Code may NOT be copied from existing sources. Collaboration between groups is NOT allowed.
- A complete submission will consist of a typed report with three parts (one for each task) and your computer code. The code should be sent in a zip file via email to your lecturer before the due date.
- All code must be in Matlab or R. An exception can be made for students proficient in C,C++, or Python.
- The most important thing is that your algorithms are described in a rigorous manner, and that your experimental results are thorough and presented clearly in the form of graphs and tables. Marks will be awarded for the choice and justification of free parameters, and for the originality of ideas in Task 3. Some marks will also be awarded to the groups that design the fastest and the most accurate heuristic. Keep in mind that algorithms written in Matlab are sometimes slower than identical algorithms written in R.
- An example of the range of test instances used in the experimental testing is the following: suppose that your algorithm "maxes out" (eg. takes more than 20 mins to solve instances) on, say, $n = 1000$. Generate random instances with $n = 100, 200, ..., 900, 1000,$

where for each n you choose a range of m values, eg. $m = n/10, 2n/10, ..., 9n/10$. For each such n, m combination, you then generate at least 10 random instances of that size. The performance (time, accuracy) of your algorithm for each n, m combination will be the average over these 10 instances. Store your test instances in data files for later use.

- The lecturer will provide a fixed set of test instances towards the end of the semester (these test instances are separate to the ones generated by the groups in Task 1(d)). Groups will describe their algorithm to the class in the last week of the semester, and then describe the performance of their algorithm on the lecturer's test instances.

I hope I thought of everything here guys. Please come to me if you have any questions. Have fun, and good luck!