70068 Scheduling and Resource Allocation

Assessed Coursework: Image Processing Workflow

Issued: 1st November 2021
Working: Individual or pairs
Submission deadline: 24th November 2021 (submit on CATE)
Page limit: max TWO pages of A4, font size 11 or larger
To submit: 2-page PDF + ZIP archive for all the rest

1 Context

In this coursework, you are going to examine some scheduling problems dealing with a real workflow. You have been given access to an Azure account to be able to run some tests and record the performance of the schedules you propose to solve these problems.

The scheduling problems in this coursework arise in the context of serverless workflows. These are workflows where jobs are serverless function calls. A serverless function is a type software service that runs in the cloud and which are increasingly used, for example, in batch processing of images, videos and data. A scheduling problem arises when functions execute together hosted inside the same environment (e.g., a virtual machine) and we want to sequence their executions in order to optimize some performance metrics, such as minimizing the maximum lateness.

The functions considered in this coursework perform image processing. Each function invocation corresponds to the notion of job we have seen in class. The functions are, for example, filters that mix the content of an image with the style of another image. Some of the filters are based on machine learning and can take a relatively long time to execute. The inputs and outputs of the filters create precedences in their executions that can be described by means of a directed acyclic graph (DAG), which captures the workflow characteristics.

2 Goals

You will be given a pre-defined workflow and your goal is to apply methods to minimize its tardiness on a single machine. You will also learn in the process an optimal algorithm for minimizing a broad class of cost functions subject to precedences on a single machine.

3 Environment setup

You are asked to use an Azure virtual machine (VM) as working environment. Instructions on how to install and use the environment are available here:

`https://gitlab.doc.ic.ac.uk/gcasale/70068-cwk-ay2021-22`

All students who registered to 70068 have been awarded $100 on Azure. You should have received by now an email notification. If you did not or experienced other problems related to the subscription throughout (e.g., you run out of credits), please contact the lecturer, Runan Wang (*runan.wang19@imperial.ac.uk*). If you face technical problems related to Azure or to the tools used in this coursework, please contact the GTAs for the first part of the module: Shreshth Tuli (*s.tuli20@imperial.ac.uk*) and Runan Wang (*runan.wang19@imperial.ac.uk*).

EdStem: if you have difficulties in configuring your environment, you can also post questions on EdStem. It is also fine to use EdStem to ask clarifications about the spec wording. However, EdStem should not be used to discuss the coursework solution. Therefore, please remember not to include in your questions an explanation on how you are trying to solve the questions.

4 Assumptions

In the context of this coursework, we will take a number of assumptions similar to what seen in the lectures:

a) A *machine* is an Azure VM with a single CPU core (1 vcore, "Ubuntu 18.04 LTS Gen1").

b) *Single machine scheduling*: a single function can run at any given time.

d) Scheduling is *non-preemptive.*

e) Processing times should be approximately treated as *deterministic.* In reality, they depend on the processing sequence, since image sizes get modified along the way, but for the sake of the scheduling model you should treat them as usual, i.e., as sequence independent.

f) There are *no release times*, i.e., all jobs are ready at time 0.

g) A filter (e.g., *blur*) can be invoked multiple times in the same workflow, and each invocation for the sake of the theoretical model will have identical processing times, but will be indicated with a different index (e.g., *blur_1*, *blur_2*, ...). These invocations can be treated, by all means, as separate jobs.

5 Questions

5.1 Question 1 (10%)

Configure the environment. Include a table in your answer sheet given the job processing times obtained with the `gettimes.py` script in your VM. Include both the mean values and the standard deviations shown on screen. (If you recreate the environment at some point make sure to update your answer sheet with the new processing times.)

5.2 Question 2 (30%)

Consider an additive cost function $g(S) = \sum g_j(C_j)$ of the completion times of job $j = 1, \ldots, n$ and define $g^*_{max} = \max_j g_j(C_j)$. The *Least Cost Last* (LCL) rule[1], also called Lawler's algorithm, solves the cost minimization problem $1|prec|g^*_{max}$ as follows: the direct acyclic graph (DAG) describing the workflow is traversed bottom-up accounting for precedences and scheduling ready jobs in non-increasing cost order. Ties are broken arbitrarily. A pseudocode of the LCL rule is given in Algorithm 1.

[1]Chapter 3 (page 68 of the PDF): `https://arxiv.org/pdf/2001.06005.pdf` (optional reading, unassessed).

Algorithm 1 Least Cost Last rule

```
1: $V = \{1, \dots, n\}$
2: $n_j :=$ number of immediate successors of job $j \in V$
3: $p(V) := \sum_{j \in V} p_j$
4: for $k = n$ to 1 do
5:     Find job $j$ in $V$ with $n_j = 0$ and minimal $g_j(p(V))$
6:     Schedule $j$ in $k$-th position of the optimal sequence $S$
7:     Set $n_i = n_i - 1$ for all jobs $i \in V$ that have $j$ as immediate successor
8:     Remove $j$ from $V$
9: end for
10: return the optimal sequence $S$
```

The question asks you to apply the LCL rule to the workflow given in Appendix A, using as processing times the mean values collected in question Q1 and as cost function the tardiness $g_j(C_j) = T_j = max(0, C_j - d_j)$ with due dates also given in the Appendix. Using a programming language of your choice write a code to apply the LCL rule. Give details in your PDF answer sheet about the intermediate results obtained for each value of the variable k, i.e., the partial schedule S at each iteration and the values g_j for all the jobs still in set V.

Include your code in the ZIP archive, inclusive of a `README.txt` file that explains how to launch it. Note that, for this question, you are not asked to conduct experiments on Azure, other than those reported in question Q1 to acquire the processing times.

5.3 Question 3 (30%)

Suppose now that we wish to solve a total tardiness problem for the same workflow studied in question Q2. The problem is now NP-hard and since the measure $\sum T_j$ is no longer expressible as a maximum of cost functions, the LCL rule is no longer optimal.

Using a programming language of your choice, implement a tabu search algorithm to obtain solutions to the $1|prec|\sum T_j$ problem for the workflow given in Appendix A. Compared to the tabu search algorithms seen in class you will therefore need to introduce a modification to account for job precedences. As seen in the exercises in problem sheet 3, write your code so that it explores the local neighborhood in lexicographic order.

Include in the ZIP archive your code as the answer to this question, inclusive of a `README.txt` file that explains how to launch it.

5.4 Question 4 (30%)

In this question, you are asked to run both LCL and tabu search on the workflow in Appendix A. Note again that in this case LCL cannot be optimal, since it is designed for other problems. The parameters for tabu search are as follows. Initialize your algorithm using the following basic schedule obtained via topological sorting (referred in the Github instructions as `s_init`):

$$S_{init} = (33, 32, 23, 31, 30, 22, 20, 29, 21, 19, 28, 18, 27, 17, 26, 14, 16, 10, 25, 13, 15, 9, 24, 12, 11, 8, 5, 7, 4, 2, 6, 3, 1, 34)$$

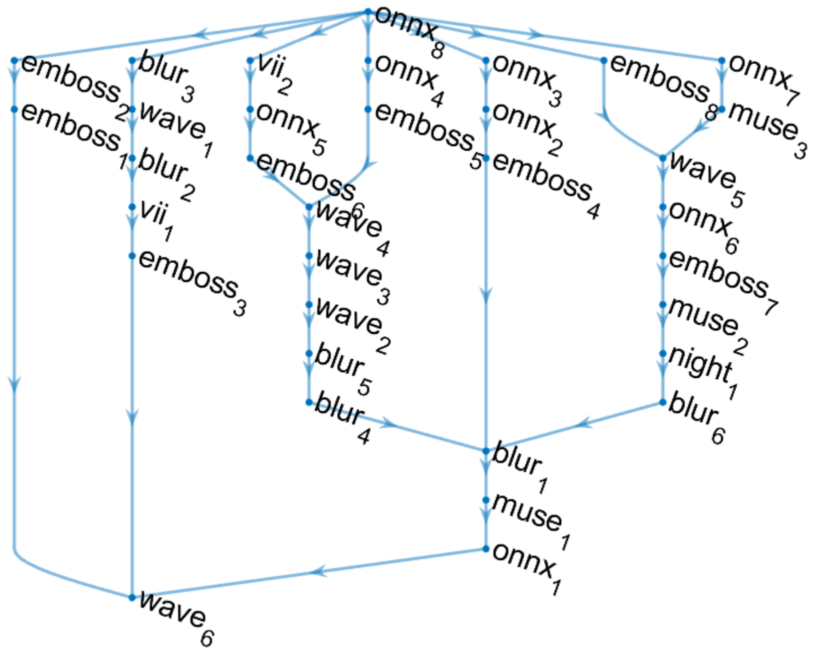
Assume a tabu list length $L = 5$. Obtain the tabu search schedules with $K = 10$, $K = 100$, and $K = 1000$ iterations. Set the tolerance to $\gamma = 30$. Run the best schedules found in this way in the Azure VM environment and record the results.

For the submission:

- Give in the PDF answer sheet the measured total completion time and tardiness obtained in the Azure VM experiments for LCL, for S_{init}, and for the three tabu search schedules for different values of K. This should display some differences compared to the theoretical model due to the sequence dependent processing times, but still demonstrate the value of local search. Comment on whether the tabu search is still capable of improving the total tardiness in practice compared to LCL and the initial schedule S_{init} based on topological sorting (it should!).
- Include in the PDF the first few iterations and notable intermediate solutions (i.e., those where a new optimum is found) as the search progresses. Use a level of detail similar to what seen in the tabu search example in the lecture notes.
- Include in the ZIP archive the input.json and output.json files you have used for the experiments.

A Workflow

The figure below shows the considered image processing workflow, consisting of 34 nodes. Despite its size may come across as large at first, in reality this is of moderate size for typical workflows used in industry.



A.1 Incidence matrix of the direct acyclic graph

Element `G(i,j)=1` if and only if there exists an edge from node i to j.

MATLAB-like format (indices start at 1):

```
G(33,4)=1;
G(33,10)=1;
G(33,14)=1;
G(33,20)=1;
G(33,23)=1;
G(33,30)=1;
G(33,32)=1;
G(4,3)=1;
G(32,31)=1;
G(31,29)=1;
G(30,29)=1;
G(29,28)=1;
G(28,27)=1;
G(27,26)=1;
G(26,25)=1;
G(25,24)=1;
G(24,5)=1;
G(14,13)=1;
G(13,12)=1;
G(12,5)=1;
G(23,22)=1;
G(22,21)=1;
G(21,18)=1;
G(20,19)=1;
G(19,18)=1;
G(18,17)=1;
G(17,16)=1;
G(16,15)=1;
G(15,11)=1;
G(11,5)=1;
G(10,9)=1;
G(9,8)=1;
G(8,7)=1;
G(7,6)=1;
G(5,2)=1;
G(2,1)=1;
G(3,34)=1;
G(1,34)=1;
G(6,34)=1;
```

Python-like format (indices start at 0):

```
G[0,33]=1
G[1,0]=1
G[2,33]=1
G[3,2]=1
G[4,1]=1
G[5,33]=1
G[6,5]=1
G[7,6]=1
G[8,7]=1
G[9,8]=1
G[10,4]=1
G[11,4]=1
G[12,11]=1
G[13,12]=1
G[14,10]=1
G[15,14]=1
G[16,15]=1
G[17,16]=1
G[18,17]=1
G[19,18]=1
G[20,17]=1
G[21,20]=1
G[22,21]=1
G[23,4]=1
G[24,23]=1
G[25,24]=1
G[26,25]=1
G[27,26]=1
G[28,27]=1
G[29,28]=1
G[30,28]=1
G[31,30]=1
G[32,3]=1
G[32,9]=1
G[32,13]=1
G[32,19]=1
G[32,22]=1
G[32,29]=1
G[32,31]=1
```

A.2 Node types

The following list gives the node numerical index used in the previous incidence matrix declarations and the corresponding filter type. For example, nodes 3 and 12 are both of `emboss` type, so they should be treated as having the same processing time.

```
1,onnx_1
2,muse_1
3,emboss_1
4,emboss_2
5,blur_1
6,emboss_3
7,vii_1
8,blur_2
9,wave_1
10,blur_3
11,blur_4
12,emboss_4
13,onnx_2
14,onnx_3
15,blur_5
16,wave_2
17,wave_3
18,wave_4
19,emboss_5
20,onnx_4
21,emboss_6
22,onnx_5
23,vii_2
24,blur_6
25,night_1
26,muse_2
27,emboss_7
28,onnx_6
29,wave_5
30,emboss_8
31,muse_3
32,onnx_7
33,onnx_8
34,wave_6
```

A.3 Due dates

For each of the 34 nodes, the following list reports the corresponding due date:

```
1,172
2,82
3,18
4,61
5,93
6,71
7,217
8,295
9,290
10,287
11,253
12,307
13,279
14,73
15,355
16,34
17,233
18,77
19,88
20,122
21,71
22,181
23,340
24,141
25,209
26,217
27,256
28,144
29,307
30,329
31,269
32,102
33,285
34,51
```