Exercise Sessions on Programming

Problem definition

Consider single machine scheduling problem. The aim is to determine the best possible order for N jobs that minimizes the weighted sum of total tardiness (tt) and total setup time (ts).

$$Min\, z = \lambda\ tt + \sigma\ ts$$

The following notation is used for every job j:

- Starting time: s_j
- Completion time: c_j
- Release time: r_j
- Processing times: p_j
- Due date: d_j
- Setup times between job i and j: $setup_{i,j}$

where tardiness t_j of job j is defined as $t_j = max\{0, c_j - d_j\}$.

Session I

Download "OMSession1.zip" from Toledo. Make sure you extracted all the files from the compressed folder. Then, either put all the files in your Matlab directory or open Matlab and change the directory to where the files are.

Datasets

a. Chemicals
A reactor is producing 3 types of polyethylene where each type is also divided in 3 different grades. Setup times between jobs are dependent on the type and grade of the jobs. Two different instances (CHEM-S and CHEM-L) are provided with the following assumptions:

- Start date = 01/01/2020 00:00:00,
- Machine is available 24/7,
- All jobs are released at the start date,
- Objective: Minimize total tardiness in $\underline{\text{weeks}}$ + setup time in $\underline{\text{hours}}$.

CHEM-S: Chemicals1.xlsx → 67 jobs for particular types and grades of chemicals,
CHEM-L: Chemicals2.xlsx → 670 jobs for particular types and grades of chemicals.

b. Consumer goods
A company is processing several products on a machine where each product requires several different jobs. Setup times between jobs are dependent on the product types of the jobs. Two different instances (CG-S and CG-L) are provided with the following assumptions:

- Start date = 02/02/2019 00:00:00,
- All jobs are released at the start date,
- Machine is available 24/7,
- Objective: Minimize total tardiness in $\underline{\text{days}}$ + setup time in $\underline{\text{hours}}$.

CG-S: Consumergoods1.xlsx → 57 jobs for 57 products,
CG-L: Consumergoods2.xlsx → 228 jobs for 57 products.

Matlab Files

Matlab files include following functions and a main script to run the algorithm.

- $[r, p, d, setup, jobinfo, N, familycode]$ = inputexcelfile(inputfilename)

  This function returns all the necessary information in the input excel files and assigns a family code to each job for the setup requirements according to the properties of jobs. See the file 'inputexcelfile.m'.

- $[schedule]$ = solnevaluation(jobid, p, d, setup, familycode, jobseq)

  This function returns required setup time, starting time, completion time, and tardiness of each job according to the given job sequence in the array *schedule*. See the file 'solnevaluation.m'.

- $[schedule, tt, ts]$ = swap(jobid, p, d, setup, familycode, schedule, pos1, pos2)

  This function swaps the jobs in the positions pos1 and pos2 in a given sequence and returns the new *schedule* array including setup time, starting time, completion time, and tardiness of each job after the move. See the file 'swap.m'.

- $[schedule, tt, ts]$ = insert(jobid, p, d, setup, familycode, schedule, L, pos1, pos2)

  This function removes a part of the given sequence and inserts it to another position. The sequence to be removed starts from $pos1$ and goes back until L jobs are included (or less if $pos1 < L$), then it is inserted just before $pos2$. It then returns the new *schedule* array including setup time, starting time, completion time, and tardiness of each job after the move. See the file 'insert.m'.

- $[schedule, tt, ts]$ = exchangeseq(jobid, p, d, setup, familycode, schedule)

  This function randomly selects two parts of a given sequence and exchanges them. It then returns the new *schedule* array including setup time, starting time, completion time, and tardiness of each job See the file 'exchangeseq.m'.

- solutioncheck(jobid, schedule)

  This function checks if the returned solution is feasible i.e. each job is included in the sequence exactly once. It displays a message if something is wrong. See the file 'solutioncheck.m'.

The metaheuristic algorithm is based on "Variable Neighbourhood Search" with two local search moves, *insert* and *swap*; and a shake operator, *exchange sequence.* It applies insert and swap until a local minimum is found, then applies exchange sequence to escape from local minimum.

Algorithm 1: VNS

```
Set the initial solution,
Set the incumbent solution,
Set the global best solution,
while stopping condition is not met do
    Insert - First Improvement
    Randomly select a position from the sequence, $pos1$
    $L$ jobs including $pos1$ will be removed and can be inserted forward or later.
    Insert the part of the sequence forward just before $pos2$, where $pos2 < pos1 - L + 1$.
    if the new solution is better then
        Update the incumbent solution, check the global best.
        Go to adjacent swap.
    else
        Go to the next $pos2$.
    end
    If no better solution is found inserting forward,
    Insert the part of the sequence later just before $pos2$, where $pos2 > pos1 + 1$.
    if the new solution is better then
        Update the incumbent solution, check the global best.
        Go to adjacent swap.
    else
        Go to the next $pos2$.
    end
    Adjacent swap - First Improvement
    Swap the job on $pos1$ with the job on its right, $pos2 = pos1 + 1$
    Start with pos1 = 1;
    Swap the job at $pos1$ and $pos2$
    if the new solution is better then
        Update the incumbent solution, check the global best.
    else
        Go to the next $pos1$ and $pos2$.
    end
    If in local search, no better solution is found, local optima is reached.
    Exchange Sequence
    Randomly select two positions $pos1$ and $pos2$ and exchange two parts of the best sequence.
    Update the incumbent solution, and the global best if better.
end
```

1. Improving the Algorithm

In this step, you will use only CHEM-S *"Chemicals1.xlsx"* dataset until you cannot further improve your algorithm. We will provide you instructions to modify the algorithm. Then, based on your observations, you will choose the best structure for your metaheuristic to continue to the next steps.

2. Evaluation on different dataset

In this step, you are going to test the algorithm on larger dataset for chemicals, CHEM-L *"Chemicals2.xlsx"* and on different type of dataset, CG-S *"Consumergoods1.xlsx"* and CG-L *"Consumergoods2.xlsx"*.

3. Summarizing your results

In this step, you are going to make different trials to see the effect of stopping condition on the final solution quality for all 4 datasets, CHEM-S, CHEM-L, CG-S, and CG-L.

Do not forget to save your own codes and results so that you can continue with them next session.

Session II

1. Modifying the Algorithm

Download "Consumergoods3.xlsx" from Toledo. In this step, you are going to modify your algorithm such that now it works for the consumer goods dataset (CG-ADV) with its special constraints. This dataset has the following special constraints:

1 All jobs has their distinct release times.

This means that the setup of a job cannot start before its release time. You should first make the changes related to this constraint. Once you make sure that your algorithm works for this constraint, then you should include the next one.

2 Machine is available all day during weekdays, but closed the entire day during weekends. Jobs cannot be split across machine downtime. Also, idle time is not allowed between the setup and the job itself.

These together mean that once the setup of a job starts, the job must be executed immediately after and completed before anything else happens on the machine, i.e. another setup, production, or downtime.

2. Summarizing your results

In this step, you are going to make different trials to see the effect of stopping condition on the final solution quality for CG-ADV.

Benchmark Results

Table 1: Benchmark Results

	VNS[1]		CP[2]	SA[3]	$GRASP$[4]
instance	TS + TT	Comp time	TT + TS	TT + TS	TT + TS
CHEM-S	62.80	15s	43.22	47.95	52.90
CHEM-L	1385.03	180s	374.22	364.20	486.0
CG-S	47.95	15s	36.61	36.61	36.61
CG-L	423.41	100s	101.27	100.5	106.7
CG-ADV	1363.53	50s	515.40	510.0	540

[1]VNS Results - Variable Neighbourhood Search in 10,000 iterations or 180s

[2]CP Results - Exact method 3600s

[3]SA Results - Simulated Annealing in 180s

[4]GRASP Results - GRASP in 180s