Computational Techniques Programming Exercise C

Adamenko Aliaksandr

Computational Techniques, ws2015/16

University of Vienna

M. Leitner

Table of Contents

1. Problem statement

Tool switching problem is a well-known problem in manufacturing: the core idea is to schedule jobs, which are performed on a single machine in such a way, that the resulting number of tool switches is kept to a minimum.

This problem can be formalized as follows[1]:

Let a ToSP instance be represented by a 4-tuple, I=(C, n, m, A) where,

- C denotes the magazine capacity (i.e., number of available slots),
- n is the number of jobs to be processed,
- m is the total number of tools required to process all jobs (it is assumed that C < m; otherwise the problem is trivial).
- A is a m x n Boolean matrix termed the incident matrix. This matrix defines the tool requirements to execute each job, i.e.,Aij=TRUE if, and only if, tool i is required to execute job j.

The solution to such an instance is a sequence J1, …, Jn determining the order in which the jobs are executed, and a sequence T1, …, Tn of tool configurations $(T_i \subset \{1, \cdots, m\})$ determining which tools are loaded in the magazine at a certain time.

Processing each job requires a particular collection of tools loaded in the magazine. It is assumed that no job requires a number of tools higher than the magazine capacity, i.e., $\sum_{i=1}^{m} \delta_{A_{ij},\mathrm{TRUE}} \leqslant C$ for all j, where δ_{ij} is Kronecker's delta. The objective function F(.) counts the number of switches that have to be done for a particular job sequence.

In this report, I've described a few algorithms, which were implemented in Java:

- Keep tools needed soonest on a predefined set of jobs
- Greedy construction heuristic
- Local search with two neighborhood structures (one with next and one with best improvement)
- Variable Neighborhood Descent - combination of two neighborhood structures

2. Predefined sequence of jobs - KTNS

This is the solution for the sub problem: we need to minimize the total amount of switches in a predefined job sequence. That means, that we don't need to alter the job position, but only choose what tools we need to switch at each step.

I've implemented the following algorithm:

1 As the first magazine filling is free, I'm putting the tools, which are needed for the first job. If there is free place after this, I'm taking the tools from the next job, which are not already included. I'm repeating this, until the first magazine is fully loaded with tools.
2 If there is difference between the current tools and the tools which are needed for next job, I'm calculating this difference and looking for a tools in magazine, which I can replace.
3 Then, for each tool, that can be replaced I'm calculating the first job position, where the tool Is needed.
4 I'm replacing the tool, which has the highest job position index.
5 Repeat the process, until all the jobs are executed.

Therefore, the principle can be summarized to following: whenever tools must be removed from the magazine in order to make room for the new tools required by the next job, the tools that needed last will be replaced.

If there are two tools to replace with the same job index, the following policy is applied - I'm just picking the last one.

I've received the following results for the test instances:

Instance (100 runs)	Amount of switches
$4\mathrm{C}_{10}^{10}$	14
$15\mathrm{C}_{30}^{40}$	159
$20C_{40}^{60}$	264

There is no difference in the results, because there is no any stochastic aspect in the algorithm.

3. Greedy construction

In the initial problem we need to create such sequence of jobs, that the amount of switches in the whole process will be minimized. The greedy approach assumes that on each step of our algorithm we will choose locally best solution. It doesn't mean that the whole solution will be optimal, but such an approach could produce relatively good results in a reasonable time.

As now we don't have an initial solution we need to create some tactic to determine it: I've suggested that the following algorithm can provide me with good initial solution.

1 I'm calculating the most commonly used tools: for each tool I'm calculating the total amount of usages.
2 For each job I'm calculating the following metric: summarizing the total amount usages of each tool. In such a way I can determine, which jobs consists of most widely used tools.
3 The highest ranked job is taken as an initial solution.

The first magazine is filled with tools, which are needed for the chosen job. If there is more places, I'm using a pretty naïve approach and just filling the free slots with a tools from the next jobs from the initial test instance.

Then, I'm using the greedy approach: starting from the initial solution I'm trying to find such a next job, for which I will need minimal amount of switches to perform it. For this purpose, I'm going through all possible jobs and calculating the amount of switches. So on each step I am comparing the current magazine load with the next job candidate.

When there exist several jobs with same amount of switches, I'm taking the last one. When I need to replace a tool, I'm just picking a random one from those, that are not needed more.

The results are:

Instance (100 runs)	Amount of switches
$4\mathrm{C}_{10}^{10}$	13
$15\mathrm{C}_{30}^{40}$	156
$20C_{40}^{60}$	281

4. Neighborhood structures and local search

4.1. First neighborhood approach - Next improvement

For this part of the task I've implemented the following neighborhood solutions structure: I'm splitting the solution provided by greedy into two parts and reverting the jobs in the first part - starting split procedure from the first element. After creation of each new neighborhood I'm calculating the total amount of switches in such sequence and stopping when this solution is better than the initial solution provided by greedy (next-improvement strategy). I'm using the same algorithm as on the first step to calculate the total amount of switches and to decide which tool I need to replace.

Sample instance is looking like this:

```
[[2, 0, 7, 3], [6, 3], [0, 6, 7], [7, 4, 6], [0, 4, 1], [7, 2, 1], [8, 2, 7], [4, 6, 8, 1], [5, 3, 9], [0, 9, 2, 4]] 13
[[6, 3], [2, 0, 7, 3], [0, 6, 7], [7, 4, 6], [0, 4, 1], [7, 2, 1], [8, 2, 7], [4, 6, 8, 1], [5, 3, 9], [0, 9, 2, 4]] 13
[[0, 6, 7], [6, 3], [2, 0, 7, 3], [7, 4, 6], [0, 4, 1], [7, 2, 1], [8, 2, 7], [4, 6, 8, 1], [5, 3, 9], [0, 9, 2, 4]] 13
[[7, 4, 6], [0, 6, 7], [6, 3], [2, 0, 7, 3], [0, 4, 1], [7, 2, 1], [8, 2, 7], [4, 6, 8, 1], [5, 3, 9], [0, 9, 2, 4]] 13
[[0, 4, 1], [7, 4, 6], [0, 6, 7], [6, 3], [2, 0, 7, 3], [7, 2, 1], [8, 2, 7], [4, 6, 8, 1], [5, 3, 9], [0, 9, 2, 4]] 12
Best f(x): 12
```

And I've received the following results:

Instance (100 runs)	Amount of switches
$4\mathrm{C}_{10}^{10}$	12
$15\mathrm{C}_{30}^{40}$	131
$20C_{40}^{60}$	230

As we can see, the result are slightly better than in simple greedy - that means that even with such a naïve approach and without any additional techniques to deal with tools/jobs with same metrics there can exist better solutions within the picked neighborhood.

4.2. Second neighborhood approach - Best improvement

The second tactic to create neighborhood solutions that I've chose was to swap nearest jobs and apply best improvement strategy: I'm creating the whole set of neighbor solutions and then picking the best one. I'm swapping two nearest jobs from the solution provided by greedy, leaving the rest unaltered. The idea is the same - to find a better solution within the neighbors.

I've received the following results:

Instance (100 runs)	Amount of switches
$4\mathrm{C}_{10}^{10}$	11
$15\mathrm{C}_{30}^{40}$	130
$20C_{40}^{60}$	227

This strategy seems to perform a little bit better: I'm suggesting that it can depend on the data set.

5. Variable Neighborhood Descent

In all of the previous approaches I've used the initial solution as a start point and didn't change it during the execution of the algorithm. In the VND I'm taking the solution, which is provided by the first neighborhood structure local search (with next improvement) and passing it as an initial solution for the second local search(with best improvement). Then I'm taking the solution provided by the second algorithm and passing it back to the first one. In such a way I can explore a new neighborhoods in order to find better solutions: the each step will theoretically make solution closer to optimal. As it is recursive procedure, I'm stopping after N iterations.

Instance (100 runs)	Amount of switches
$4\mathrm{C}_{10}^{10}$	10
$15\mathrm{C}_{30}^{40}$	124
$20C_{40}^{60}$	214

6. Results

All the results are merged into the final table:

Instance	KTNS	Greedy	First Structure	Second Structure	VND
$4\mathrm{C}_{10}^{10}$	14	13	12	11	10
$15\mathrm{C}_{30}^{40}$	159	156	131	130	124
$20C_{40}^{60}$	264	281	230	227	214

As a result, mixed approach performed the best: it is not very surprising, because we can analyze the wider amount of neighborhood solutions. The neighborhood building structure, when we are reverting the part of the job sequence performed better than simple greedy, but not as good as 2-swap. However, difference Is not that big, only a few switches.

There is still room for improvement: maybe it is possible to receive better results by applying additional strategies to the tools replacement procedure when we have tools with same priority metric. In addition, maybe another strategy for a first magazine load for a greedy can be applied: the results on the last data set are worse than on the initial sequence. As I'm using greedy result for the next algorithms it can happen that the strategy for the first magazine load is also affecting them.

7. References

1. http://www.unet.edu.ve/~jedgar/ToSP/ToSP.htm
2. https://moodle.univie.ac.at/pluginfile.php/2061758/mod_resource/content/1/HeuristicOptimization.pdf
3. http://perso.uclouvain.be/daniele.catanzaro/SupportingMaterial/JSTSP.pdf
4. http://feb.kuleuven.be/public/NDBAE03/papers/ijfms1994paper.pdf
5. http://www.lcc.uma.es/~ccottap/papers/amaya08memetic.pdf