Department of Mathematics and Computer Science
University of Southern Denmark, Odense

Issued: March 27, 2009
Marco Chiarandini

DM204 - Scheduling, Timetabling and Routing

Exam Project, Fall 2009 Problem A

---

1 Practical Notes

Note 1 The project consists in modeling a real-life problem, designing a solution algorithm, implementing a solver, experimenting with some real or synthetic instances of the problem and documenting the work in a written report.

The evaluation of the project is based on the written report only, which may be in English or in Danish. However, a program that implements the solver described in the report must also be delivered. The program can be used to check the correctness of the solutions discussed.

The grade of the project will contribute to 40% of the final grade of the course. There is no external censorship on the project.

Note 2 Additional material to the project description, like input data, is available on the web at:
`http://www.imada.sdu.dk/~marco/DM204/`. Corrections or updates to this project description or to the web page above will be posted in the announcement list of the Blackboard system.

Note 3 *Submission*. An archive containing the electronic version of the written report and the source code of the program must be handed in through the Blackboard system **before 14:00 of Monday, June 19, 2009**. This is the procedure:

- go at the course page in the Blackboard system;
- choose "Exam Project Submission" in the menu on the left;
- fill the form and conclude with submit;
- print and keep the receipt (there will be a receipt also per email).

See Appendix C for details on how to organize the electronic archive. Reports and codes handed in after the deadline will generally not be accepted. System failures, illness, etc. will not automatically give extra time.

---

2 Problem Description

The following problem arises in several related variants in the manufacturing industry, whenever items are to be moved by robots around a set of machines to receive some kind of treatment. In the anodizing plant of the Danish company Anodize Efficiently, items are metal components, robots are cranes that move the components and machines

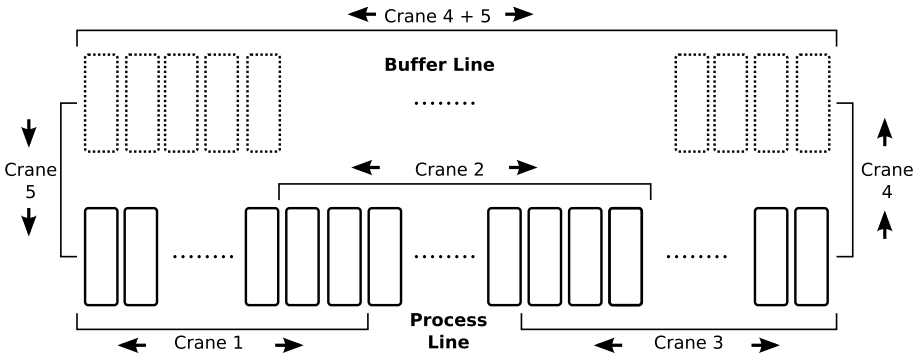

Figure 1: The layout of the anodizing plant at Anodize Efficiently.

are tanks filled with different fluids, at different acidity and temperatures. The plant of Anodize Efficiently consists of 57 separate *tanks* of equal dimensions arranged in sequence in a process line and of a buffer line.

Each aluminum component has to go through a predefined set of tanks in a particular order to acquire the desired characteristics. Each visit of a component to a tank defines a *stage*, and a series of stages in some order defines a *recipe* or a *treatment program*. Each stage is determined by the tank number, the minimum and maximum treatment times, and other data concerning the specific processing occurring in the tank.

Components with the equal recipe are grouped together and mounted on *bars* that fit into the tanks. When a bar is ready to be processed it is placed in a *buffer line* next to the *process line* where tanks are disposed. Figure 1 gives an overview of the overall anodizing plant.

Bars are moved around the plant by a set of *cranes*, or *hoists*. There are two cranes operating the buffer line. One takes new bars from the buffer line into the process line, and the other takes processed bars out of the process line and back to the buffer line. There are three cranes operating the process line, each handling about one third of the line and overlapping on some tanks with the cranes next to it. The tanks where they overlap are called *exchange tanks*. In the beginning of the operations each crane stays in a starting position that is the same where it ended in the previous operating period.

Anodize Efficiently wants to automate the control of the anodizing plant. More precisely, there are two kinds of decisions that may be controlled: (i) the *dispatching* that indicates which is the next bar to be moved from the buffer line into the system, and (ii) the *hoist scheduling* that determines the lifting, sinking and carrying times of the bars through the process line. These decisions have strong interdependencies and hence they must be considered together. The focus can be restricted to the process line and to the three cranes that operate it, since the other two cranes are for most of the time unused and can always accomplish the service requirements of the process line. Thus, the operations of the system can be controlled by deciding a *schedule* from which both dispatching and hoist scheduling can be deduced. It is enough that the schedule indicate the starting time for each operation of the cranes. A crane operation must then be defined by indicating: the type, lift, sink or move; the tanks involved (initial and final); and the specific bar involved. In a schedule, the dispatching decision can be determined by the bar that has to be moved away from the first tank into the process line.

In order to be *feasible* a schedule must satisfy a number of constraints. They can be

distinguished in two types: constraints that affect the cranes and constraints that affect the bars.

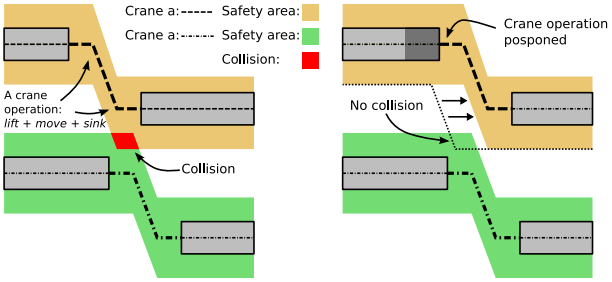

Figure 2: Left: Two colliding cranes. Right: The collision is avoided by postponing one of the crane operations, extending the bars stay in a tank.

Crane Constraints The following are physical constraints imposed on the cranes.

- *Blocking* In the process line there are no intermediate buffers. Therefore bars can be moved out of a tank only if the next tank is free.
- *Crane speed* Cranes have a limited speed to travel from one tank to another. This speed takes into account accelerations and decelerations.
- *Lift and sink time* Once arrived at the tank, the cranes take a certain amount of time to lift a bar from the tank and a certain amount of time to sink the bar into the tank. These times may vary according to bar and tank as they may include times for dripping the bar or for heating and cleaning color tanks.
- *Waiting* A crane cannot wait when handling a bar. Once a crane has been assigned to pick up a bar and move it to the next tank it has to do so without interruption. Once a crane has lowered a bar into its tank it is finished with that bar, and it is in a lowered position ready to lift another bar, or the same bar again. This means that it does not lift empty, when moving without a bar, it stays down, ready to lift the new bar as soon as it reaches its position.
- *Operating area and Safety distance* The three cranes operate about a third of the process line each and there are few tanks that can be served by two cranes (see Figure 1). In this area, cranes cannot pass each others and must maintain a distance above a safety limit (see Figure 2).

Bar Constraints The following are the constraint is imposed to the bars by the recipe.

- *Unique processing and no preemption* A bar can only be in one tank at a time and no more than one bar can be in one tank at the same time. A treatment cannot be interrupted and restarted before it is finished.
- *Minimum and Maximum time* The duration of the stay in a tank must strictly adhere to the minimum and maximum duration defined in the recipe, otherwise the part is damaged and must be discarded.

All these constraints are *strict*, in the sense that a violation of any of them causes the waste of the bar.

The anodizing plant of Anodize Efficiently operates without interruptions from Sunday night to Friday night. During this time there is a constant flow of bars and the system is prepared to work with on line decisions, that is, a different schedule can be issued and put into operation each time a new bar arrives in the buffer line. The interest of the managers is in *maximizing the throughput* of bars, that is, the output rate.

There are two sets of data available for the problem described. The `snapshot` instance collects data of an off-line problem with 10 bars waiting in the buffer line at the beginning and no more bars arriving. This instance should be used to validate a solver and check its feasibility. Observations at the plant revealed that this is about the number of bars that are in the buffer whenever a new decision has to be taken.

The `one-week` instance provides, instead, data for the whole week with bars arriving at different times.

3 Your Task

Your task is the following:

- Model the problem as a scheduling problem, using the terminology and the classification presented in the course, and defining mathematically the variables, the constraints and the objectives. If this is possible, find a mixed integer programming (MIP) formulation for the off-line problem, that is, the one defined by the `snapshot` instance.
- If a MIP model is produced, write a ZIMPL program for it and solve the `snapshot` instance.
- If the MIP formulation turns out to be infeasible either in modelling terms or in terms of computation times, try using other approaches of your choice (e.g., constraint programming, heuristics) to achieve at least a feasible solution.
- If a feasible solution has been obtained solve the `one-week` instance.
- Analyze and discuss experimental results on the `one-week` instance.

It is not necessary to accomplish all the points above to achieve the highest grade in the project.

4 Remarks

Remark 1 The following criteria will be kept in consideration in the evaluation:

- Good model structure, keeping the model as simple as possible.
- The quality of the solution found and the solution time.
- Clear reporting with focus on the relevant key performance.

Remark 2 If you find issues that are not specified in full detail make your own assumptions, i.e., try to guess what is probably happening in practice, describe the way how you suggest the issue should be handled, and formulate the models according to your version of the anodizing plant problem.[1]

Few issues have been purposely left out from the description but are present in practice. Tanks are grouped according to type of treatment and a stage in the recipe can be accomplished in any of the tanks that offer the treatment required. Maximizing the throughput is not the only criterion to evaluate a schedule. There might be priorities assigned to the bars or due dates and deadlines. Finally, it might be necessary to avoid that some bars remain too long in the buffer line. These issues should not be considered in the project.

Remark 3 In case the solution is found by exact methods, the source code of the model must be delivered and it must be stated which program is used to solve it (e.g., CPLEX, SCIP, GECODE, etc.). Only freeware software is allowed. For heuristic algorithms, the programs must output the solution in a reasonable amount of time. Solutions must be written in a file in text format as described in Appendix B.

Remark 4 The total length of the report should not be less than 10 pages and not be more than 15 pages, appendix included (lengths apply to font size of 11pt and 3cm margins). Although these bounds are not strict, their violation is highly discouraged. Do not include source code in the report.

Acknowledgments This document is based on the Master thesis of Jacob Skov, IMADA, SDU, Odense, 2007.

[1] This situation is very typical in practice. In the meetings with the company, the engineers might have forgotten to tell some details that then become necessary while modelling the problem. Often for reasons of time, these issues must be resolved by making some additional reasonable assumptions. Hopefully, then practitioners agree. Otherwise something must be changed.

Appendix A Instance Format

There are 2 instances, a small one for testing purposes and a large one taken from real life data. In the instances bars are identified by an identity number from 1 to n and tanks by an identity number from 1 to m. Tank 1 and tank m are the entering tank and the leaving tank. All times are expressed in seconds starting from 0 when the first bar arrives in the system. Each instance is composed by 4 text files. In the following we describe the content and the format of these files.

`cranes.txt` for each of the three cranes, a line indicating in the order:

- the crane identifier number
- the first tank covered
- the last tank covered
- the tank on which the crane is positioned at the beginning of the schedule

followed by a matrix indicating the time distances between each pair of the tanks covered. In each matrix, the tanks in the columns and in the rows run from the first tank to the last tank covered by the crane.

Example:

```
1 1 17 5
 0  6  8  9 10 10 11 12 13 13 14 15 15 16 17
 6  0  6  8  9 10 10 11 12 13 13 14 15 15 16
 8  6  0  6  8  9 10 10 11 12 13 13 14 15 16
 9  8  6  0  6  8  9 10 10 11 12 12 13 14 15
...
```

`bars.txt` the list of bars with arrival time in the buffer and with output time from the last tank of its recipe that occurred in practice. Each row of data comprises:

- bar number;
- time of arrival in the buffer;
- output time (this is the time recorded by Anodize Efficiently. It can be used for comparisons, although they are not exactly based on the same problem specifics);

Example:

```
1 0 4835
2 423 5046
3 697 8312
4 1034 9550
5 1283 9941
6 1744 12600
7 2349 7513
8 2826 11317
...
```

`recipes.txt` for each bar a recipe is reported indicating the itinerary of the bar through the process line. Each line reports:

- the bar number;
- the stage number going from 1 to a_i, where a_i is the number of tanks the bar has to visit;
- the identifier of the tank to be visited in the corresponding stage;
- the minimal duration of the stay at the tank;
- the maximal duration of the stay at the tank;
- the sink time;
- the lift time;

Examples:

```
1 1 1 0 0 14 20
1 2 3 2 6 14 13
1 3 3 5 10 14 13
...
1 18 57 0 0 14 20
2 1 1 0 0 14 19
2 2 5 3 14 14 13
2 3 3 3 14 14 13
2 4 4 5 55 14 13
...
```

Note that the recipe includes the arrival buffer and the exit buffer. The minimal and relative times in this cases are to be ignored as well as the sink and the lift times respectively.

The following data must be assumed implicitly:

Tanks: there are 57 tanks, ordered from 1 to 57. Some tanks are never used.

Minimal safety distance between cranes: 2617 mm.

Originally the large instance is in XML format. A JAVA application for reading this format and preprocessing data is made available together with the instance files mentioned above. The application makes it possible to change easily the format of the files described above, in case needed.

Appendix B Solution Format

In order to check the validity of the results reported, the program submitted must output the solution in a text file when finishing.

The file must report the operations of the cranes. Each change in the status of one of the cranes is documented by a line. Precisely, each line of the solution file reports:

- the identifier of the crane;

- one among UP, DOWN, MOVE to indicate which operation is performed
- the bar that is handled (-1 if the operation does not involve a specific bar);
- the stage of the bar handled (-1 if the operation does not involve a specific bar);
- the starting tank;
- the final tank;
- the starting time of the operation.

Example

```
1 MOVE -1 -1 5 1 0
1 UP 1 1 1 1 10
...
2 MOVE -1 -1 22 16 30
2 UP 1 1 16 16 41
...
```

Appendix C Electronic Submission

The electronic submission must be organized in an archive that expands in a main directory named with your full CPR number and with content

```
<CPR-NUM>/README
<CPR-NUM>/report/
<CPR-NUM>/src/
<CPR-NUM>/zimpl/
```

The file `README` contains the instructions for compiling and running the solver. The directory `src` contains the sources which may be in C, C++, JAVA or other languages. If needed a Makefile can be included either in the root directory or in `src`. After compilation the executable must be placed in `src`. For JAVA programs, a jar package can also be submitted. The directory `zimpl` contains models to be interpreted by ZIMPL.

Programs must work on IMADA's computers under Linux environment and with the compilers and other applications present on IMADA's computers. Students are free to develop their program at home, but it is their own responsibility to transfer the program to IMADA's system and make the necessary adjustments such that it works at IMADA.[2]

[2]Past issue: make sure the JAVA compiler and virtual machine are the ones you intend to use (check the path: `ls -l /etc/alternatives/javacpath`); in C, any routine that uses subroutines from the math.c library should be compiled with the `-lm` flag – eg, `cc floor.c -lm`.