Production scheduling

Simon Sigurdhsson ⟨ssimon@student.chalmers.se⟩

Abstract This report discusses a solution to the second project of the Large scale optimization course (TMA521) given by the Applied mathematics department at Chalmers University of Technology. The problem considered is a production scheduling problem described by a time indexed model, which is solved using Danzig-Wolfe decomposition and column generation. The problem is thoroughly described by Thörnblad (2011).

1 Problem model and implementation

1.1 The time-indexed model

The time-indexed model is stated by the project description as

$$\min \sum_{j\in\mathcal{J}}\sum_{k\in\tilde{\mathcal{K}}}\sum_{u\in\mathcal{T}}\left(A_j\left(u+\tilde{p}_j^{pm}\right)+B_j\left[u+\tilde{p}_j^{pm}-\tilde{d}_j\right]_+\right)x_{jku} \tag{1}$$

$$\text{s.t.}\sum_{k\in\tilde{\mathcal{K}}}\sum_{u\in\mathcal{T}}x_{jku}=1,\quad j\in\mathcal{J}, \tag{1a}$$

$$\sum_{u\in\mathcal{T}}x_{jku}\le\lambda_{jk},\quad j\in\mathcal{J},k\in\tilde{\mathcal{K}}, \tag{1b}$$

$$\sum_{j\in\mathcal{J}}\sum_{v=\left[u-\tilde{p}_j+1\right]_+}^{u}x_{jkv}\le 1,\quad k\in\tilde{\mathcal{K}},u\in\mathcal{T}, \tag{1c}$$

$$x_{jku}=0,\quad j\in\mathcal{J},k\in\tilde{\mathcal{K}},u=0,1,\ldots,\max\left\{\tilde{r}_j^m,\tilde{a}_k\right\}-1, \tag{1d}$$

$$x_{jku}\in\{0,1\},\quad j\in\mathcal{J},k\in\tilde{\mathcal{K}},u\in\mathcal{T}, \tag{1e}$$

where $\mathcal{J}$ is the set of jobs to be completed during the planning period, $\mathcal{T}$ is a set of time intervals of the planning period and $\tilde{\mathcal{K}}$ is the set of resources that the multitask cell consists of. Additionally, the constants λ_{jk} represent the availability of operation j on resource k and the constants in the objective function can be interpreted as a weighted sum of job finishing times ($A_j\cdots$) and tardiness ($B_j\cdots$). Finally, $\tilde{r}_j^m$ is the release date for job j, $\tilde{a}_k$ is the time when resource k is first available and $\tilde{d}_j$ is the due date of job j.

1.1.1 Danzig-Wolfe decomposition

Following Lasdon (1970, pp. 148–155), the time-indexed model of the machining problem can be decomposed using the Dantzig-Wolfe method and solved by column generation. Identifying (1a) as the coupling constraint and (1b)-(1e) as forming independent blocks, a master program can be formulated:

$$\min \sum_{i=0}^{n} \sum_{j\in\mathcal{J}} \sum_{k\in\tilde{\mathcal{K}}} \sum_{u\in\mathcal{T}} \left(A_j \left(u + \tilde{p}_j^{pm} \right) + B_j \left[u + \tilde{p}_j^{pm} - \tilde{d}_j \right]_+ \right) x_{jku}^i \alpha_j^i \tag{2}$$

$$\text{s.t.} \sum_{u\in\mathcal{T}} \sum_{k\in\tilde{\mathcal{K}}} \sum_{i=0}^{n} x_{jku}^i \alpha_k^i + s = 1, \quad j \in \mathcal{J} \tag{2a}$$

$$\sum_{i=0}^{n} \alpha_k^i = 1, \quad k \in \tilde{\mathcal{K}} \tag{2b}$$

$$\alpha_i^k \geq 0, \quad k \in \tilde{\mathcal{K}}, i = 0,1,\ldots,n \tag{2c}$$

Here, a slack variable s has been added to the coupling constraint. The independent blocks can be treated either as one subproblem or as k different subproblems. Quick tests seemed to indicate that fewer iterations were needed when using multiple subproblems, and as such this is what was used. The k subproblems are of the form

$$\min \sum_{j\in\mathcal{J}} \sum_{u\in\mathcal{T}} \left(c_{ju} - \pi_j \right) x_{jku} - \pi_{0k} \tag{3}$$

$$\text{s.t.} \sum_{u\in\mathcal{T}} x_{jku} \leq \lambda_{jk}, \quad j \in \mathcal{J}, \tag{3a}$$

$$\sum_{j\in\mathcal{J}} \sum_{\nu=[u-\tilde{p}_j^{pm}]_+}^{u} x_{jk\nu} \leq 1, \quad u \in \mathcal{T}, \tag{3b}$$

$$x_{jku} = 0, \quad j \in \mathcal{J}\ u = 0,1,\ldots,\max\left\{ \tilde{r}_j^m, \tilde{a}_k \right\} - 1, \tag{3c}$$

$$x_{jku} \in \{0,1\}, \quad k \in \tilde{\mathcal{K}}, u \in \mathcal{T}, \tag{3d}$$

where $c_{ju} = \left(A_j \left(u + \tilde{p}_j^{pm} \right) + B_j \left[u + \tilde{p}_j^{pm} - \tilde{d}_j \right]_+ \right)$.

1.1.2 Column generation

The column generation procedure used to solve the system consists of two (or three) phases. In the first phase, a basic feasible solution to the problem is found. In the second phase, columns are added until an optimal solution is found.

The first phase replaces the objective function (2) with the slack variable, and sets the costs c_{ju} to zero. As long as $s \geq 0$, the subproblems are solved and new columns are added to the master program to form a new restricted master program. When

$s = 0$ (or in practice, $s < \epsilon$ for some small ϵ), the algorithm moves on to the second phase.

In the second phase, the value of s is fixed and the original objective functions are restored. Again, the subproblems are solved, columns are added and the restricted master program is solved again. When $\min_{k \in \tilde{\mathcal{K}}} z_k \geq 0$ (*i.e.*, all subproblems have positive objective values), the algorithm has found an optimal solution Lasdon 1970, p. 153 and the loop is terminated.

In the third and final 'phase', an optimal solution is calculated from the α_j^i and x_{jku}^i given by the algorithm. This is done by solving the optimization problem

$$\min \sum_{k \in \tilde{\mathcal{K}}} \sum_{j \in \mathcal{J}} \sum_{u \in \mathcal{T}} c_{ju} x_{jku} \tag{4}$$

$$\text{s.t.} \sum_{u \in \mathcal{T}} \sum_{k \in \tilde{\mathcal{K}}} \sum_{i=0}^{n} x_{jku}^i \alpha_k^i + s = 1, \quad j \in \mathcal{J} \tag{4a}$$

$$\sum_{u \in \mathcal{T}} x_{jku} \leq \lambda_{jk}, \quad j \in \mathcal{J}, \tag{4b}$$

$$\sum_{j \in \mathcal{J}} \sum_{\nu = [u - \tilde{p}_j^{pm}]_+}^{u} x_{jk\nu} \leq 1, \quad u \in \mathcal{T}, \tag{4c}$$

$$x_{jku} = 0, \quad j \in \mathcal{J}, u = 0,1,\ldots, \max\left\{\tilde{r}_j^m, \tilde{a}_k\right\} - 1, \tag{4d}$$

$$\sum_{k \in \tilde{\mathcal{K}}} \sum_{u \in \mathcal{T}} x_{jku} = o_j, \quad j \in \mathcal{J}, \tag{4e}$$

$$x_{jku} \in \{0,1\}, \quad k \in \tilde{\mathcal{K}}, u \in \mathcal{T}, \tag{4f}$$

where α_j^i are fixed and $o_j = \sum_{i=0}^{n} \sum_{u \in \mathcal{T}} \sum_{k \in \tilde{\mathcal{K}}} x_{jku}^i \alpha_j^i$. This results in a solution x_{jku} which my be transformed into variables z_{ijk} and y_{ijpqk}, to be used as input to the feasibility problem.

1.2 Implementation

The entire algorithm was implemented in AMPL, and a full code listing is available in appendix A on page 8. In short, the `proj2-k-msub.mod` file contains the model specification with all parameters, variables, objective functions and constraints of the model. The `proj2-k-msub.run` file loads the model and data and proceeds with solving the model using the algorithm described previously, followed by solving the feasibility problem (which is taken verbatim from the project description and solved without any decomposition).

2 Results

All results presented in this section were obtained using the data from Thörnblad (2011) given in `2010-11-17_15j_MTC6.dat` unless otherwise noted. Results for the Dantzig-Wolfe decomposition are divided into 1-subproblem and k-subproblem variants; although the 1-subproblem isn't described in this report it should be fairly simple to deduce given Lasdon (1970), and modifying the code of the k-subproblem variant to instead solve the 1-subproblem variant is fairly trivial.

2.1 Optimal solution

The three methods (two Dantzig-Wolfe decompositions and the regular problem with no decomposition) arrive at slightly different solutions. The 1-subproblem variant of the decomposed problem arrives at an objective value 3841.6 after 96 iterations in the second phase. Meanwhile, the k-subproblem variant arrives at the same objective value after only 19 iterations. In fact, the two variants arrive at the exact same solution, but the k-subproblem variant spends only 7.8 s in phase two while the 1-subproblem variant spends 27.5 s. Clearly, the k-subproblem variant is preferrable, which is why subsequent results are based on that model only. The undecomposed model arrives at the objective value 4040, with a slightly different solution compared to the Dantzig-Wolfe model. Tables 1 to 2 on the next page show the solutions given by the Dantzig-Wolfe method and the original model, respectively — the differing jobs have been marked in both tables.

2.2 Solution dependence on iteration count

In the second phase of the column generation algorithm, one can obtain upper and lower bounds on the optimal value. An obvious upper bound is the value of the master program in any given iteration, but obtaining a lower bound is less trivial. According to Lasdon (1970, p. 163), a lower bound is given by $z_B + \sum_k (z_k^0 - \pi_{0k})$, where z_B is the objective value of the master program in the current iteration, and $z_k^0 - \pi_{0k}$ is given by the solution to subproblem k.

Figure 1 on the following page shows the upper and lower bound with respect to CPU time spent in phase two, for the k-subproblem model. A similar graph with respect to iterations is not included, but one can note that each iteration in phase two takes approximately the same CPU time to calculate (roughly 0.4 s), so the graph will be very similar. As seen by the graph, a fairly tight bound is found halfway through the algorithm (at approximately 4 s), and the bound is very tight after 6 s.

Table 1: *Solution using Dantzig-Wolfe*

Job	MC1	MC2	MC3	MC4	MC5
1	0	1	0	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	0	1	0
5	0	0	1	0	0
6	0	0	0	1	0
7	1	0	0	0	0
8	0	0	1	0	0
9	0	1	0	0	0
10	1	0	0	0	0
11	1	0	0	0	0
12	0	0	0	0	1
13	0	0	1	0	0
14	0	0	1	0	0
15	1	0	0	0	0

Table 2: *Solution without decomposition*

	MC1	MC2	MC3	MC4	MC5
1	0	1	0	0	0
2	1	0	0	0	0
3	1	0	0	0	0
4	0	0	0	1	0
5	0	0	1	0	0
6	0	0	1	0	0
7	1	0	0	0	0
8	0	0	1	0	0
9	0	1	0	0	0
10	0	1	0	0	0
11	1	0	0	0	0
12	0	0	0	0	1
13	0	0	0	1	0
14	0	0	1	0	0
15	1	0	0	0	0

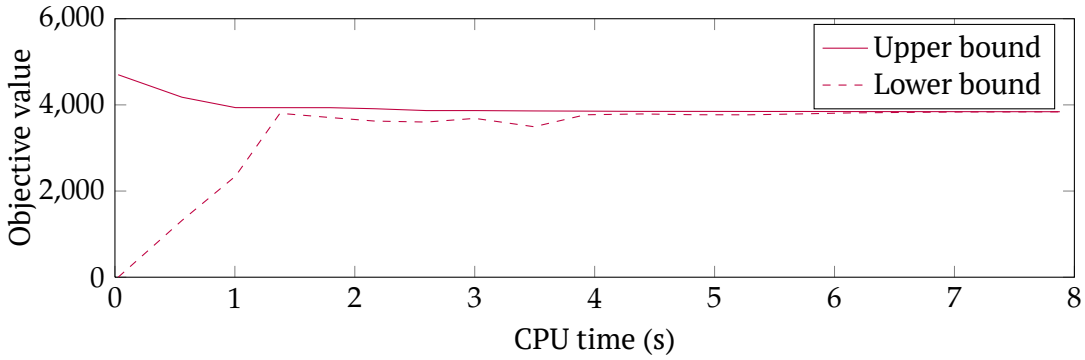

Figure 1: *Upper and lower bound of the Dantzig-Wolfe decomposition with respect to CPU time in phase two.*

Table 3: *Tardiness h_j for the original and modified objective functions*

Job	$A_j = B_j = 1$	$A_j = 0$	$B_j = 0$
1	0.0	0.0	0.0
2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0
5	1486.6	1486.6	1486.6
6	0.0	0.0	0.0
7	1.6	1.6	2.6
8	1319.6	1319.6	1318.6
9	0.0	0.0	0.0
10	0.0	0.0	0.0
11	0.0	0.0	0.0
12	672.2	672.2	668.2
13	0.0	0.0	0.0
14	161.2	175.2	162.2
15	0.0	0.0	0.0
Total	3641.2	3655.2	3638.2

Table 4: *Completion times s_j for the original and modified objective functions*

Job	$A_j = B_j = 1$	$A_j = 0$	$B_j = 0$
1	25.6	15.6	112.6
2	3.6	25.6	74.6
3	5.6	33.6	41.6
4	10.0	5.0	151.0
5	1.6	1.6	1.6
6	34.0	10.0	136.0
7	1.6	1.6	2.6
8	2.6	2.6	1.6
9	15.6	3.6	67.6
10	33.6	5.6	6.6
11	13.6	13.6	37.6
12	7.2	7.2	3.2
13	27.0	24.0	151.0
14	5.2	19.2	6.2
15	23.6	23.6	19.6
Total	200.4	192.4	813.4

2.3 Changing the objective function

The objective function of the original model seeks to minimize both tardiness and completion time. This is not necessarily the most relevant objective, and trying other objective functions may yield better results.

Two objectives that spring to mind easily are the minimization of only tardiness or completion times, which is easily implemented by setting $A_j = 0$ or $B_j = 0$ in the original objective function, respectively. As seen in tables 5 to 6 on the next page, we obtain slightly different solutions compared to the original model (table 1 on the preceding page). We note that when minimizing the tardiness, resource MC5 is unused.

Perhaps more relevant statistics for these two objective functions are the tardiness and completion times of each job. Tables 3 to 4 on the current page show this data for the original model as well as the tardiness and completion time objectives. We note that an interesting consequence of minimizing the tardiness (for a rather small gain of 3 h) has a large 613 h penalty on the completion times, while minimizing the completion times (gaining 8 h) only penalizes the tardiness by 14 h.

Table 5: *Solution with* $A_j = 0$

Job	MC1	MC2	MC3	MC4	MC5
1	0	1	0	0	0
2	0	1	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	1	0	0
6	0	0	0	1	0
7	1	0	0	0	0
8	1	0	0	0	0
9	1	0	0	0	0
10	0	1	0	0	0
11	1	0	0	0	0
12	0	0	1	0	0
13	0	0	0	1	0
14	1	0	0	0	0
15	0	1	0	0	0

Table 6: *Solution with* $B_j = 0$

	MC1	MC2	MC3	MC4	MC5
1	0	1	0	0	0
2	0	1	0	0	0
3	1	0	0	0	0
4	0	0	1	0	0
5	0	0	1	0	0
6	0	0	0	1	0
7	1	0	0	0	0
8	0	0	1	0	0
9	1	0	0	0	0
10	0	1	0	0	0
11	1	0	0	0	0
12	0	0	0	0	1
13	0	0	0	1	0
14	0	0	1	0	0
15	1	0	0	0	0

3 Conclusions

The Dantzig-Wolfe decomposition is clearly a good method of solving the given problem. It is fairly efficient, and arrives at a better optimal solution than simply letting CPLEX decide what to do. Additionally one can see that for this problem, decomposing the model into k subproblems is more efficient than using a single subproblem.

Modifying the objective function can be an important tool in optimizing the solution for different situations. For instance, while this report only investigated the two extremes of $A_j = 0$ and $B_j = 0$ for all j, one can imagine that setting $A_j = 0$ for some j and $B_j = 0$ for others will yield solutions that will prioritize the early completion of some jobs while penalizing the tardiness of others.

Using the upper and lower bound, which is easily obtained for each iteration, one could compute a duality gap which could be used to abort the algorithm when 'good enough' solutions are found. In the case investigated in this report (*i.e.*, $j = 15$), this is not necessary since the time required to find an optimal solution is fairly low, but for larger problems (say, the $j = 70$ dataset) this may be very useful.

References

Lasdon, Leon S. (1970). *Optimization Theory for Large Systems*. Mineola, New York: Dover Publications.

Thörnblad, Karin (2011). 'On the Optimization of Schedules of a Multitask Production Cell'. Licentiate thesis. Göteborg: Department of Mathematical Sciences, Chalmers University of Technology.

A AMPL implementation

A.1 Main file (`proj2-k-msub.run`)

The preamble of the code simply loads the model and data, sets a few useful options and declares a couple of parameters that are irrelevant to the model but used by the algorithm.

```
# TMA521 - Large scale optimization
# Spring 2013
# Project 2
# Simon Sigurdhsson

option solver cplexamp;
option cplex_options
  'clocktype=1'
  'timing=1';
model proj2-k-msub.mod;
data MTC6_v2_proj_course/2010-11-17_15j_MTC6.dat;

param upper; param lower; param sumfij default 0;
param pIIcpu default 0;
param epsilon := 0.000000001;
param nIter default 0; param pIIiter default 0;
param phase symbolic, default "I";
param s_disc {JOBS}; param h_disc {JOBS};
```

After that, all problems are declared. Only one master program is declared (it will be modified later when moving from phase one to phase two). Since the subproblems are harder to change, they are defined for both phases separately.

```
problem Master_Program: A_MP, MP_multi, MP_convex, alpha, slack;
    option presolve 0;

problem SubproblemI {k in K_mach_RESOURCES}: A_SUB[k],
  {j in JOBS} SUB_limit[k,j],
  {u in T_ALL_INTERVALS} SUB_occupy[k,u],
  {j in JOBS, u in 0..(max(r_disc[j],a_disc[k])-1)} SUB_delay[k,j,u], x;
    option presolve 0;
```

```
28
29 problem SubproblemII {k in K_mach_RESOURCES}: SUB[k],
30   {j in JOBS} SUB_limit[k,j],
31   {u in T_ALL_INTERVALS} SUB_occupy[k,u],
32   {j in JOBS, u in 0..(max(r_disc[j],a_disc[k])-1)} SUB_delay[k,j,u], x;
33     option presolve 0;
```

A.1.1 First and second phase

The first (finding a basic feasible solution) and second (finding the optimal solution) phase are handled in the same loop by simply changing the master program and solving different subproblems after the initial basic feasible solution has been found.

```
36 repeat {
37     let nIter := nIter + 1;
38     printf "\nPHASE %s -- ITERATION %d\n\n", phase, nIter;
39
40     for {k in K_mach_RESOURCES} {
41         printf "\nRESOURCE %s\n\n", k;
```

Here, the subproblems are solved. If the objective of the subproblems (defined as $z_k^0 - \pi_{0k}$) is non-negative, nothing is done. If the objective is positive, the solution is added to the list of extreme points and its cost is saved.

```
43         if phase = "I" then {
44             solve SubproblemI[k];
45             if A_SUB[k] < -epsilon then {
46                 let nPROP[k] := nPROP[k] + 1;
47                 let {j in JOBS, u in 0..T_HORIZON}
48                   propx[j,u,k,nPROP[k]] := x[j,k,u];
49                 let c_prop[k,nPROP[k]] :=
50                   sum {u in 0..T_HORIZON, j in JOBS} c[j,u]*x[j,k,u];
51             }
52         } else {
53             solve SubproblemII[k];
54             if SUB[k] < -epsilon then {
55                 let nPROP[k] := nPROP[k] + 1;
56                 let {j in JOBS, u in 0..T_HORIZON}
57                   propx[j,u,k,nPROP[k]] := x[j,k,u];
58                 let c_prop[k,nPROP[k]] :=
59                   sum {u in 0..T_HORIZON, j in JOBS} c[j,u]*x[j,k,u];
60             }
61         };
62     };
```

Now, we check the termination criteria. If all objectives are non-negative, we either have an infeasible problem (phase one) or an optimal solution (phase two), and terminate the loop. In phase two we also save the sum of the objective functions in order

to calculate the lower bound later.

```
64      if phase = "I" then {
65          if min {k in K_mach_RESOURCES} A_SUB[k] >= -epsilon then {
66              printf "\n*** NO FEASIBLE SOLUTION ***\n";
67              break;
68          }
69      } else {
70          if min {k in K_mach_RESOURCES} SUB[k] >= -epsilon then {
71              printf "\n*** OPTIMAL SOLUTION ***\n";
72              break;
73          }
74          let sumfij := sum {k in K_mach_RESOURCES} SUB[k];
75      };
```

Next, the master program is solved.

```
77      solve Master_Program;
78      printf "\n";
79      display alpha;
80      #display MP_multi.slack;
```

If the algorithm is in phase one, the slack variable is checked. If it is zero (or rather, very small), it enters phase two by modifying the master program.

```
82      if phase = "I" then {
83          display slack;
84          #display MP_multi.dual;
85
86          if slack <= epsilon then {
87              printf "\nSETTING UP FOR PHASE II\n\n";
88              let phase := "II";
89              let pIIiter := nIter;
90              let pIIcpu := _total_solve_time;
91              printf "BOUNDS iter cpu lower upper\n";
92
93              problem Master_Program;
94                  drop A_MP; restore MP; fix slack;
95
96              solve Master_Program;
97              printf "\n";
98              display alpha; display MP_multi.dual; display MP_multi.slack;
99          };
100     };
```

Finally, π_{0k} and π are set, and (if in phase two) the upper and lower bound is printed along with iteration count and CPU time consumed.

```
102     let {j in JOBS} pi[j] := MP_multi[j].dual;
103     let {k in K_mach_RESOURCES} piz[k] := MP_convex[k].dual;
104
105     if phase = "I" then { } else {
```

```
106         let upper := MP;
107         let lower := MP + sumfij;
108         printf "BOUNDS %d %f %f %f", nIter-pIIiter,
109           _total_solve_time-pIIcpu, lower, upper;
110     };
111 };
```

A.1.2 Third phase

The third phase simply solves the modified problem described in the report.

```
114 printf "\nPHASE III\n\n";
115 problem Master_ProgramIII: MPIII, MP_multi, SUB_limit, SUB_occupy,
116   SUB_delay, MPIII_match, x;
117 let {j in JOBS}
118   opt[j] := sum {u in T_ALL_INTERVALS, k in K_mach_RESOURCES, i in 1..nPROP[k]}
119     propx[j,u,k,i] * alpha[k,i];
120 solve MPIII;
```

It also transforms the solution x_{jku} to obtain the useful variables s_j, h_j and t_j.

```
124 let {j in JOBS, q in JOBS, k in K_RESOURCES} y_disc_solution[j,q,k] := 0;
125 for {j in JOBS, q in JOBS,k in K_mach_RESOURCES} {
126     if (0 < sum{u in T_ALL_INTERVALS} u*x[j,k,u]
127           < sum{u in T_ALL_INTERVALS} u*x[q,k,u]) then
128         let y_disc_solution[j,q,k] := 1 };
129 let {j in JOBS, k in K_RESOURCES} z_disc_solution[j,k] := 0;
130 let {j in JOBS, k in K_mach_RESOURCES} z_disc_solution[j,k] :=
131   sum{u in T_ALL_INTERVALS} x[j,k,u].val;
132 let {j in JOBS} t_disc_solution[j] :=
133   T_length_interval*(sum{k in K_mach_RESOURCES, u in T_ALL_INTERVALS} u*x[j,k,u].val);
134 printf "\n*** SOLUTION ***\n";
135 display z_disc_solution;
136 #display t_disc_solution;
137 let {j in JOBS} s_disc[j] := sum {u in T_ALL_INTERVALS, k in K_mach_RESOURCES}
138   (u+p_postmach[j])*x[j,k,u];
139 let {j in JOBS} h_disc[j] := sum {u in T_ALL_INTERVALS, k in K_mach_RESOURCES}
140   max(0,u+p_postmach[j]-d_disc[j])*x[j,k,u];
```

A.1.3 Feasibility problem

The very last thing done is to solve the feasibility problem, which is very straightforward.

```
144 printf "\n*** FEASIBILITY PROBLEM ***\n";
145 problem Feasibility: F, F_scheduled, F_flex, F_order1, F_order2,
146   F_earliest, F_sorder, F_interop, F_release, F_avail, F_completion,
147   F_tardiness, F_fsol1, F_fsol2, t, z, y, s, h;
148 solve Feasibility;
149 display z;
```

```
#display y;
display t;
#display s;
#display h;
```

A.2 Model (`proj2-k-msub.mod`)

The model file describes the models, variables, parameters, constraints, sets and objective functions used in solving the problem.

```
# TMA521 - Large scale optimization
# Spring 2013
# Project 2
# Simon Sigurdhsson
```

It begins by defining a couple of sets. Most of these are pesent in Thörnblad's (2011) model as well (in fact, so are most of the parameters, too).

```
param T_HORIZON;
set T_ALL_INTERVALS := 0..T_HORIZON;
param maxjobs;
set JOBS := 1..maxjobs;
set K_RESOURCES ordered;
set K_mach_RESOURCES ordered;
param n {JOBS};
set ACTIVE_I {j in JOBS} := setof {i in 1..n[j]}(i);
```

Next, a couple of parameters are defined (again, from Thörnblad (2011)).

```
param T_length_interval;
param lambda_mach {JOBS,K_mach_RESOURCES};
param r_disc {JOBS};
param a_disc {K_mach_RESOURCES};
param proc_time_disc {JOBS};
param p_postmach {JOBS};
param d_disc {JOBS};

param c {j in JOBS, u in T_ALL_INTERVALS} =
  ((u+p_postmach[j]) + max(0,u+p_postmach[j]-d_disc[j]));
param y_disc_solution{JOBS, JOBS, K_RESOURCES};
param z_disc_solution{JOBS, K_RESOURCES};
param t_disc_solution{JOBS};

param nPROP {K_mach_RESOURCES} default 0;
param propx {JOBS,T_ALL_INTERVALS,k in K_mach_RESOURCES,1..nPROP[k]} >= 0;
param c_prop {k in K_mach_RESOURCES,1..nPROP[k]};
param pi {JOBS} default 0;
param piz {K_mach_RESOURCES} default 1;
```

Finally, a couple of parameters that aren't used by this model, but present in the data.

```
set I_OP; param M; param w; set Q_PREC;
param q_follow{Q_PREC}; param v_jq{Q_PREC}; param v_disc_jq_ext{Q_PREC};
param v_mach_jq{Q_PREC}; param proc_time_mach{JOBS};
param proc_time{I_OP,JOBS}; param p_j_o_postmach_disc{JOBS};
param a{K_RESOURCES}; param resource_weight{K_RESOURCES}; param r_mach{JOBS};
param r{JOBS}; param d{JOBS}; param lambda{I_OP,JOBS,K_RESOURCES};
```

The variables of the model are declared.

```
var slack >= 0;
var alpha {k in K_mach_RESOURCES, 1..nPROP[k]} >= 0;
var x {JOBS,K_mach_RESOURCES,T_ALL_INTERVALS} binary;
```

A.2.1 Master program

The objective functions of the master program. One for phase two (`MP`) and one for phase one (`A_MP`).

```
minimize MP:
    sum {k in K_mach_RESOURCES, i in 1..nPROP[k]} (c_prop[k,i]*alpha[k,i]);
minimize A_MP:
    slack;
```

The constraints of the master program.

```
subject to MP_multi {j in JOBS}:
    sum {u in T_ALL_INTERVALS, k in K_mach_RESOURCES, i in 1..nPROP[k]}
      propx[j,u,k,i]*alpha[k,i] + slack = 1;
subject to MP_convex {k in K_mach_RESOURCES}:
    sum {i in 1..nPROP[k]} alpha[k,i] = 1;
```

A.2.2 Subproblems

The objective functions of the master program — k of them for phase two (`SUB`) and equally many for phase one (`A_SUB`).

```
minimize A_SUB {k in K_mach_RESOURCES}:
    sum {j in JOBS, u in 0..T_HORIZON}
      (0 - pi[j]*x[j,k,u])
    - piz[k];

minimize SUB {k in K_mach_RESOURCES}:
    sum {j in JOBS, u in 0..T_HORIZON}
      ((c[j,u] - pi[j])*x[j,k,u])
    - piz[k];
```

Constraints of the subproblems.

```
subject to SUB_limit  {k in K_mach_RESOURCES, j in JOBS}:
    sum {u in T_ALL_INTERVALS} x[j,k,u] <= lambda_mach[j,k];
subject to SUB_occupy {k in K_mach_RESOURCES, u in T_ALL_INTERVALS}:
```

```
    sum {j in JOBS, nu in max(u-proc_time_disc[j]+1,0)..u} x[j,k,nu] <= 1;
subject to SUB_delay  {k in K_mach_RESOURCES, j in JOBS,
  u in 0..(max(r_disc[j],a_disc[k])-1)}:
    x[j,k,u] = 0;
```

A.2.3 Phase three problem

The phase three problem only introduces one new constraint (and an associated parameter) that matches the solution to the optimal value of the column generation method.

```
param opt{JOBS} >= 0;
minimize MPIII:
    sum {j in JOBS, k in K_mach_RESOURCES, u in T_ALL_INTERVALS}
      c[j,u]*x[j,k,u];
subject to MPIII_match {j in JOBS}:
    sum {k in K_mach_RESOURCES, u in T_ALL_INTERVALS} x[j,k,u] = opt[j];
```

A.2.4 Feasibility problem

The feasibility problem is taken verbatim from the project description. It's not very interesting.

```
var t {I_OP,JOBS} >= 0;
var z {I_OP,JOBS,K_RESOURCES} binary;
var y {I_OP,JOBS,I_OP,JOBS,K_RESOURCES} binary;
var s {JOBS} >= 0;
var h {JOBS} >= 0;
minimize F:
    sum {j in JOBS} (s[j]-0.001*t[1,j]+h[j] +
      sum {i in ACTIVE_I[j], k in K_RESOURCES} resource_weight[k]*z[i,j,k]);
subject to F_scheduled {j in JOBS,i in ACTIVE_I[j]}:
    sum {k in K_RESOURCES} z[i,j,k] = 1;
subject to F_flex {j in JOBS,i in ACTIVE_I[j], k in K_RESOURCES}:
    z[i,j,k] <= lambda[i,j,k];
subject to F_order1 {j in JOBS, q in JOBS, i in ACTIVE_I[j],
  p in ACTIVE_I[q], k in K_RESOURCES}:
    y[i,j,p,q,k] + y[p,q,i,j,k] <= if (i=p and j=q) then 2 else z[i,j,k];
subject to F_order2 {j in JOBS, q in JOBS, i in ACTIVE_I[j],
  p in ACTIVE_I[q], k in K_RESOURCES}:
    y[i,j,p,q,k] + y[p,q,i,j,k] + 1 >= if (i=p and j=q)  then 0
      else (z[i,j,k] + z[p,q,k]);
subject to F_earliest {j in JOBS, q in JOBS, i in ACTIVE_I[j],
  p in ACTIVE_I[q], k in K_RESOURCES}:
    t[i,j] + proc_time[i,j] - M*(1-y[i,j,p,q,k]) <=
      if (i=p and j=q) then M else t[p,q];
subject to F_sorder {j in JOBS, i in 1..(n[j]-1)}:
    t[i,j] + proc_time[i,j] + w <= t[i+1,j];
subject to F_interop {j_prec in Q_PREC}:
```

```
123     s[j_prec] + v_jq[j_prec] <= t[1,q_follow[j_prec]];
124 subject to F_release {j in JOBS}:
125     t[1,j] >= r[j];
126 subject to F_avail {j in JOBS, i in ACTIVE_I[j], k in K_RESOURCES}:
127     t[i,j] >= a[k]*z[i,j,k];
128 subject to F_completion {j in JOBS}:
129     s[j] = t[n[j],j] + proc_time[n[j],j];
130 subject to F_tardiness {j in JOBS}:
131     h[j] >= s[j]-d[j];
132 subject to F_fsol1 {j in JOBS, q in JOBS, k in K_RESOURCES}:
133     y[2,j,2,q,k] = y_disc_solution[j,q,k];
134 subject to F_fsol2 {j in JOBS, k in K_RESOURCES}:
135     z[2,j,k] = z_disc_solution[j,k];
```

A.3 Changed objectives

The changed objective functions discussed in the report are implemented as copies of the model file with some lines changed. Below, the lines changed are listed as the new value of that line.

A.3.1 First case ($A_j = 0$)

The first case simply entails a change of the constants c_{ju}.

```
25 param c {j in JOBS, u in T_ALL_INTERVALS} =
26   (max(0,u+p_postmach[j]-d_disc[j]));
```

A.3.2 Second case ($B_j = 0$)

Like the first case, the second case only changes the constants c_{ju}.

```
25 param c {j in JOBS, u in T_ALL_INTERVALS} =
26   ((u+p_postmach[j]));
```