RAIRO-Oper. Res. 43 (2009) 409–420
DOI: 10.1051/ro/2009026

RAIRO Operations Research
www.rairo-ro.org

INCORPORATING THE STRENGTH OF MIP MODELING IN SCHEDULE CONSTRUCTION

Cor A.J. Hurkens[1]

Abstract. Linear programming techniques can be used in constructing schedules but their application is not trivial. This in particular holds true if a trade-off has to be made between computation time and solution quality. However, it turns out that – when handled with care – mixed integer linear programs may provide effective tools. This is demonstrated in the successful approach to the benchmark constructed for the 2007 ROADEF computation challenge on scheduling problems furnished by France Telecom.

Keywords. Scheduling, integer programming, lower bounds, hybrid methods.

Résumé. L'application de techniques programmation linéaire pour la résolution de problème d'ordonnancement n'est pas trivial, particulièrement lorsque qu'un compromis entre qualité de la solution fournie et temps de calcul est recherché. Dans ce cas des heuristiques peuvent être couplées pour améliorer les performances des modèles de programmation linéaire. La combinaison de telles méthodes a montré son efficacité dans le cadre de la résolution du Challenge ROADEF 2007.

Mots Clés. Planification, programmation en nombres entiers, bornes inférieures, hybridisation.

Mathematics Subject Classification. 90C11, 90B35.

Received December 1, 2008. Accepted June 17, 2009.

[1] Eindhoven University of Technology, Department of Mathematics and Computer Science, P.O. Box 513, 5600 MB Eindhoven, The Netherlands; wscor@win.tue.nl

Article published by EDP Sciences

Introduction

In this paper we present an approach based on polyhedral techniques for solving a complex scheduling problem. The working example is the France Telecom technician assignment problem, posed as a challenge for the ROADEF 2007 competition.

Linear programming techniques are found in the literature concerning sequencing and scheduling but not in great abundance. For instance they show up for tackling relatively *simple* problems such as minimizing maximum lateness for a preemptive schedule of jobs with release times on unrelated machines [2]. Some production application can also be found [3], as well as computational studies [1]. Also linear programming models with a combinatorial structure such as flows or matchings can be fruitful in some scheduling contexts. It is very tempting to use complete *integer linear models* for scheduling problems in practical applications. However, it turns out that more than often, an integer linear programming model will at best describe formally the underlying problem correctly, while it does not provide any solution when passed onto a solver. We have taken this opportunity to test how the strength of MIP modeling can be applied – in practice – to real-life scheduling challenges. We developed a schedule constructing algorithm with a lot of subroutines that make use of a CPLEX-library for solving LP and ILP sub-problems.

The remainder of this paper discusses an analysis of the scheduling problem that pinpoints the various sections for which a polyhedral approach maybe useful, and proceeds by giving for the identified parts an appropriate MIP model.

1. Analyzing the France Telecom technician dispatching problem

1.1. Brief description

The scheduling problems in the 2007 ROADEF challenge were furnished by France Telecom and involved the daily formation of teams of skilled technicians and the assignment of tasks to teams over a certain scheduling period. The input for each problem is a list of tasks to be scheduled, and a list of technicians that – in conjunction with other technicians – may carry out those tasks. The output is a schedule which for each day describes a partition of the technicians into teams, an assignment of some tasks to these teams, and for each scheduled task a processing time interval. In the overall schedule, some tasks may not be scheduled at all, but then the total cost of these *abandoned* tasks may not exceed the predefined budget. If a task is being abandoned then all its successor tasks must be abandoned as well. The schedule has to obey certain restrictions regarding the expertise of the formed teams, and the order in which tasks are being carried out. Tasks have to be carried out without interruption, in the course of one day. A team can handle only one task at a time, and stays together for the duration of a day. A team can

TABLE 1. Characteristics of France Telecom problems.

Inst	Jobs	Techs	Reqs	Prec	LB	Cost	$C^1_{\max}$	$C^2_{\max}$	$C^3_{\max}$	$C^4_{\max}$
A1	5	5	6	0	2265	2340	60	15	90	-
A2	5	5	6	2	2055	4755	135	-	195	-
A3	20	7	6	0	11310	11880	300	120	360	-
A4	20	7	12	7	10629	13620	210	360	540	-
A5	50	10	6	13	26910	29355	855	240	300	-
A6	50	10	20	11	17625	20280	525	120	780	-
A7	100	20	20	31	28442	32520	600	780	960	-
A8	100	20	20	21	16191	18960	480	180	600	-
A9	100	20	20	22	25553	28320	720	240	960	-
A10	100	15	20	31	36399	40650	1020	435	1200	-
B1	200	20	16	47	32085	35460	420	1005	1755	2610
B2	300	30	15	143	14296	18300	450	165	615	930
B3	400	40	16	57	14610	16965	195	480	870	1305
B4	400	30	120	112	16635	27015	645	240	1005	1575
B5	500	50	28	427	45060	94200	1620	2310	3060	4260
B6	500	30	24	457	24180	30510	750	285	1035	1380
B7	500	100	50	387	25290	33060	720	480	1080	1860
B8	800	150	40	440	31920	32160	480	840	1230	2040
B9	120	60	25	55	25681	28080	720	360	480	960
B10	120	40	25	55	32790	35040	960	360	480	1200
X1	600	60	60	195	87990	151980	2160	4500	5580	6180
X2	800	100	36	536	6075	9090	210	105	330	420
X3	300	50	60	224	37950	50400	720	1440	1920	2400
X4	800	70	105	321	61130	65640	1650	480	2460	2880
X5	600	60	60	201	86145	147000	1980	5340	2760	5760
X6	200	20	36	128	6180	10440	240	90	480	540
X7	300	50	60	235	26220	33120	510	720	1680	2040
X8	100	30	105	40	19050	23580	540	240	990	1140
X9	500	50	60	184	86520	136020	1980	3960	4920	5460
X10	500	40	60	184	97950	131700	1920	3780	4920	5340

handle a task only then, when the cumulative skills of the technicians in the team dominate the skills required by the task. Tasks can be handled not earlier than at the latest completion time of all its predecessors.

The quality of a schedule is determined by the weighted sum of completion times of the various priority classes. Each task belongs to some priority class, where higher priority classes evidently get a higher weight. A priority class is considered completed once all of its unabandoned tasks have been completed.

The principal characteristics and results for the thirty Challenge problems are given in Table 1. In the course of the challenge, three benchmarks were provided. The first one, A, was a set of relatively small problems. In this set no tasks could

be abandoned. Set B contains real-life problems, or at least problems of real-life size. These two sets were used to fine-tune the algorithms developed for the challenge. Finally, there was a set X of problems that would be used for the final ranking. They were not disclosed until *after* all algorithms had been submitted and tested. Their complexity was to be comparable to those in set B.

In Table 1 we give for each of the test instances the number of jobs, the number of technicians, the number of skill requirements, and the number of immediate precedences between jobs. Moreover we present a lower bound on the objective function, based on a relaxation described in a later section. In the right half of the table we give the values of the best solution found. The value of each schedule is a certain weighted combination of 3 or 4 priority-makespans. Here $C^p_{\max}$ denotes the latest completion time of a job in the schedule belonging to priority class p. Most of these were obtained by running the generic implementation of our algorithm for a period of 40 min or less. For the X-instances a time limit of 20 min was used.

It can be seen from the table that the instances are varying a lot. The A-instances are relatively small and do not allow to reject (or outsource) jobs. The B-instances allow for outsourcing jobs (under restrictions) and some of them have a lot of precedences between jobs. Furthermore, in some B-instances the number of requirements may be rather high.

1.2. Notation and key observations

In the remainder tasks will be indexed by j. A task j has a duration of $d(j)$ time units where a working day consists of 120 time units. A task must be processed without interruption by a team of technicians, within a working day. Task j has a priority $P(j) \in \{1,2,3,4\}$, where a low number denotes a high priority. In case task j has to be finished before task k can be started we denote this by $j \to k$, and we say that j is a *predecessor* of k and k is a *successor* of j. In a schedule let $\mathrm{CT}(j)$ denote the completion time of task j in time units. Then, for $p \in \{1,2,3,4\}$, the *priority makespan* of priority p, $C_{\max}(p)$ or $C^p_{\max}$ is defined as the maximum completion time $\mathrm{CT}(j)$ over all unabandoned jobs with $P(j) = p$. Each priority is weighted by a number $W(p)$. In all instances we used the weights $W(1) = 28, W(2) = 14, W(3) = 4$, and $W(4) = 0$. Moreover the overall makespan denoted by $C^0_{\max}$ or $C_{\max}(0)$ and defined by $C^0_{\max} = \max_{p\in\{1,2,3,4\}} C^p_{\max}$ was weighted by a factor $W(0) = 1$. Hence, the value of a schedule is given by $\sum_{p\in\{0,1,2,3,4\}} W(p)C^p_{\max}$.

Technicians are indexed by t and days are indexed by $d \in \{0,1,\ldots\}$. Day d is a time interval starting at time $120d$ and ending at $120(d+1)$. A technician t may or may not be available on day d. By Amd_d we denote the number of technicians available on day d. By Cmd_d we denote the cumulative number of man-days up to and not including day d. These numbers will be useful when computing bounds on priority makespan values.

Teams will be composed for each day in the schedule, where only technicians that are available can be assigned. On each day a technician is assigned to at most

one team. Teams will be indexed by τ. A team can carry out a task only if it contains enough competent technicians.

In the France Telecom problems, competence was described in terms of *skills*, *domains* and *levels*. Each instance was characterized by D skill domains, and within each domain, we consider L levels of expertise. For each technician t we are given his competence level $\mathrm{Comp}(t,s) \in \{0,1,\ldots,L\}$ in skill domain s. Here a level 0 denotes no competence at all, and a level L refers to a real expert. For the processing of a task j the necessary team competence is described by a number $\mathrm{Req}(j,s,\ell)$ denoting the minimum number of technicians in the team that have a competence level at least ℓ in skill domain s. Note that this definition implies a cumulative notion, hence we have $\mathrm{Req}(j,s,\ell) \geq \mathrm{Req}(j,s,\ell{+}1)$. In the following linear programming models it is much easier to account for competence requirements by using a slightly different but equivalent way of bookkeeping competence levels and requirements. We have $D*L$ pairs, and we use an index $i \in \{1,\ldots,D*L\}$ to refer to these pairs. For each *pair* $i=(s,\ell)$ we denote by a 0/1 value S_i^t whether or not a technician t contributes to skill domain s at level ℓ or not. Here a value $S_i^t = 1$ denotes that t has skill level ℓ or higher, in domain s. For $i=(s,\ell)$ we use $R_i^j = \mathrm{Req}(j,s,\ell)$ to denote the required level of competence of a team handling j. So team τ can handle task j if and only if $\sum_{t\in\tau} S_i^t \geq R_i^j$, for all i. In the remainder one can view i as a skill that a technician has or has not. The numbers of required skills as mentioned in Table 1 are the products DL.

As already indicated, it is not always the case that an optimal solution is reached by processing all tasks in order of priority. In the following, both in the constructing heuristics, as well as in the determination of lower bounds on the schedule cost, we have to prefix the order by which priority classes will be completed. As priority class 4 does not have a positive weight, it suffices to distinguish between solutions in which priority classes are completed in order $\pi_1,\pi_2,\pi_3,\pi_4,\pi_5$ where (π_1,π_2,π_3) is some permutation of $1,2,3$, and where $\pi_4 = 4$ and $\pi_5 = 0$. For a given ordering we say that task j *contributes* to the makespan of π_m, if it belongs to priority class $\pi_1,\ldots,\pi_{m-1}$ or π_m, or if it has a successor task that contributes to π_m. We should only count its contribution if the task is not abandoned.

2. Solution strategy

To have some idea how the solution will look like, and to have an a posteriori solution quality guarantee, we first compute a lower bound on the value of the schedule cost. The lower bound is based on a relaxation of the problem by allowing preemption, and by estimating the minimum number of technicians needed for a job. Next we construct a solution, partially based on the relaxed schedule. Since we have four priority classes, three of which having a non-zero weight, we consider schedules with a fixed order for the makespans $C^1_{\max}$, $C^2_{\max}$, and $C^3_{\max}$. Note that it may be profitable to process first all jobs of say priority 2, and then the jobs of priority 1. This is particularly true if there are only few jobs of priority class 2. The table with solutions indeed shows very good solutions with makespans in

other orders than 1, 2, 3, 4. We therefore consider all 6 permutations of priority classes 1, 2, 3.

2.1. Preprocessing

One of the ingredients in the heuristics developed is the estimation of the size of teams handling some task j. For this we use the vector of required skills, and compare it to the skills provided by the technicians. The skill requirement of a task j is expressed by a vector $(R_i^j)_{i\in S}$ where $R_i^j \in \mathbb{Z}_+$ denotes the required number of technicians in the team having skill i. Obviously, the value $\max_i R_i^j$ is a lower bound on the size of a team handling task j. However, it may very well be the case that no expert has both skill x and skill y. If a task requires a team with one expert for x and one expert for y, such a team then consists of at least two experts, even if we have $\max_i R_i^j = 1$. To find the exact minimum of technicians needed to handle task j we solve for each task j a Mixed Integer Programming problem. Here, technician t offers skills $\underline{S}^t \in \{0,1\}^n$, and task j requires skills $\underline{R}^j \in \{0,1,2,\ldots\}^n$.

Minimum number of technicians in team doing job j:

$$\mu(j) = \begin{array}{lllll} \min & \sum_t x_{tj} & & & \\ \text{s.t.} & & & & \\ & \sum_t \underline{S}_i^t x_{tj} & \geq & \underline{R}_i^j & \forall i = 1,\ldots,n \\ & x_{tj} & \in & \{0,1\} & \forall t. \end{array}$$

Evidently, $\mu(j) \geq \max_i \underline{R}_i^j$.

2.2. Lower bound

Consider a permutation π_1, π_2, π_3 of $1, 2, 3$, and define $\pi_4 = 4, \pi_5 = 0$. For a given order of the makespans, $C_{\max}(\pi_1) \leq C_{\max}(\pi_2) \leq C_{\max}(\pi_3) \leq C_{\max}(4) \leq C_{\max}(0)$, and a given set of jobs to schedule, a lower bound on $C_{\max}(\pi_m)$ is found by calculating the total amount of man-days (technician-units) MD needed to process all jobs that should be finished by time $C_{\max}^{\pi_m}$, and computing the first time T at which this number of man-days has been made available. If at each day μ technicians are available $T = \frac{MD}{\mu}$ days. If the number of available technicians varies per day (which is the case here) a simple adjustment is needed. See figure 1 for the more general case.

To estimate the number of technicians needed for a task j we use the lower bound $\mu(j)$ obtained in Section 2.1. By multiplication with the duration, $d(j)$, and division by 120, it is converted in total required man-days. Lower bounds on the makespans are computed simultaneously. In case it is considered to abandon jobs it is really fruitful to formulate the lower bound problem as an integer linear programming problem. The following Mixed Integer Program models the general minimum cost of a preemptive schedule, providing enough man-days to carry

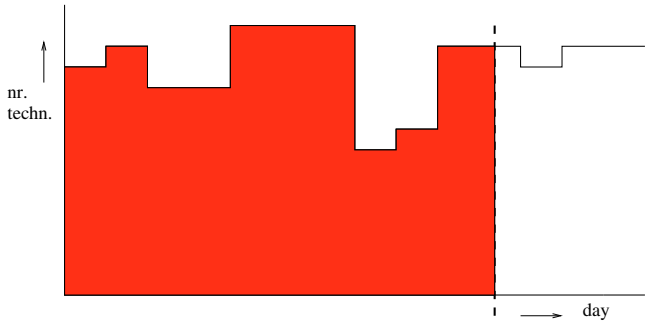

FIGURE 1. Profile of man-days offered.

out the selected jobs. The variables $C(\pi_m)$ denotes the priority class makespan, expressed in time-units. Considering the order of priority classes, let a_{mj} denote the amount of man-days that task j contributes to makespan $C(\pi_m)$. Let $\text{Rmd}_m = \sum_j a_{mj}$ denote the total man-days of tasks j contributing to $C(\pi_m)$. The variable X_j indicates whether a task j is abandoned ($X_j = 1$) or not ($X_j = 0$). The cost of abandoning task j is given by Abc_j, whereas Budget is an upper bound on the cost spent on abandoning tasks. For a proper lower bound on the schedule cost, we must take coefficients $F_j = 0$. This coefficient is only introduced to allow for variations on preemptive schedules, as it turns out that there can be difficult tasks that one may want to abandon with high preference.

Minimize $\sum_m W(\pi_m) C_{\max}(\pi_m) + \sum_j F_j X_j$
subject to:

$$
\begin{array}{rcll}
C_{\max}(\pi_m) & \leq & C_{\max}(\pi_{m+1}) & \forall m < 5 \\
C_{\max}(\pi_m) & \geq & 120 * d * Z_{md} + 120 Y_{md} & \forall m, d \\
\sum_d Z_{md} & = & 1 & \forall m \\
Y_{md} & \leq & Z_{md} & \forall m, d \\
\sum_d (\text{Cmd}_d Z_{md} + \text{Amd}_d Y_{md}) & \geq & \text{Rmd}_m - \sum_j a_{mj} X_j & \forall m \\
Z_{md} & \in & \{0, 1\} & \\
Y_{md} & \geq & 0 & \\
\sum \text{Abc}_j X_j & \leq & \text{Budget} & \\
X_k - X_j & \geq & 0 & \forall j \rightarrow k \\
X_j & \in & \{0, 1\}. &
\end{array}
$$

In this formulation variable Z_{md} indicates whether or not the makespan of priority class π_m is realized on day d, and variable Y_{md} says which fraction of day d passes before the makespan is realized. The parameter Cmd_d denotes the cumulative man-days offered upto day d, whereas Amd_d gives the man-days offered on day d. The total of required man-days for priority class π_m is given by parameter Rmd_m and the term $-\sum_j a_{mj} X_j$ compensates for the not needed man-days for

all jobs that have been abandoned. The latter constraints stipulate that the total abandonment cost should not exceed the available Budget, and that a task can be abandoned only if all its successors are being abandoned as well.

Computation of this lower bound has several results. First it gives a proper lower bound on the schedule cost. Second it provides a selection of tasks to be abandoned. It therefore serves as a starting point in constructing good schedules. After computing the lower bound, we only consider schedules for the tasks that have not been abandoned in the lower bound computation.

Note that the lower bounds depend on the prefixed order of priority class completions. Therefore only the *lowest* of the six lower bounds (corresponding with 6 permutations) forms a true lower bound on the scheduling cost.

Further note that if we have found a schedule of cost C^*, and if permutation $(\hat{\pi}_1, \hat{\pi}_2, \hat{\pi}_3)$ yields a lower bound $\hat{L}$ such that $\hat{L} \geq C^*$, then we do not have to consider schedules with priority makespans in this order because cheaper solutions with this makespan order provably do not exist. In this way the lower bound helps us in not wasting our precious computation time in unprofitable directions.

It turns out that in some cases the computed lower bound is much smaller than the best solution found. This is usually caused by the fact that the real bottleneck is formed by a subset of expert technicians. It is possible to obtain stronger lower bounds than the one described above, by going through the same analysis, with the exception that the minimum number of required technicians (as provided by the first procedure) and the number of available technicians per day are restricted to those technicians that contribute positively to some *specific subset of skills*. It is not easy to detect which set of skills will lead to a better lower bound. This approach might obtain better bounds for instances $B5$, $X1$ and $X5$, for instance. In instance $B5$ the lower bound could be raised to nearly 79000, by considering the demand for technicians that had a non-zero expertise in domain number 7. There are only a few technicians that give a contribution and there were relatively many tasks needing expertise in this particular domain. Taking this into account requires computation of alternative $\mu'(j), \mathrm{Amd}'(d), \mathrm{Cmd}'(d), \mathrm{Rmd}'(m)$ and a'_{mj} values, restricting everything to technicians skilled in domain 7.

2.3. Constructing solutions

In short, the construction algorithm is based on a strategy of building a solution from scratch, starting at day 0, assigning a number of jobs *in parallel* to teams of technicians. These teams are being built on the fly. For each day, the technicians that are clustered to a team stay together. The basic ingredient of the algorithm is a model of matching a number of available jobs (those for which the predecessors have been scheduled) to a number of teams. here, teams may also refer to a group consisting of a single technician. Initially, at the beginning of each day, each team consists of one available technician. The set of jobs is either the complete set of available jobs, or the set of available jobs with a certain priority class. For the larger instances the set of candidate jobs had to be reduced to a manageable size.

This was done by sorting the candidate jobs according to a certain *SortingRule*, and then splitting the sorted sequences into almost equal size parts of approximately 50 jobs. The default *SortingRule* was based on task-difficulty. One task can be considered more difficult than another if it requires more expertise, if it requires more technicians, if it is part of a longer chain of tasks etcetera. Different ways of quantifying these features lead to different *SortingRules*.

The ILP formulation has the following form. Here team τ's skill $\underline{S}^\tau$ is sum of technician skills over all technicians assigned to this team; its load $\lambda(\tau)$ is current work load, that is the total duration of all tasks already assigned to the team; the team processes its assigned tasks in the order in which they have been assigned, without any idle time. Task j cannot start earlier than $e(j)$. This earliest starting time (within the day) is based on the completion time of predecessors of task j that have already been assigned to a team in the current day. More than one team can be assigned to a task, but we do not allow two teams that have tasks with potential successors to be combined. By this we are sure that precedence relations are not violated. For this purpose, let $\hat{T}$ denote the set of teams that process at least one task j with a successor.

Maximize $\sum_\tau \sum_j C_{\tau j} X_{\tau j} + \sum_j G_j Y_j$
subject to:

$$\begin{array}{rcll}
\sum_j X_{\tau j} & \leq & 1 & \forall \tau \\
\sum_\tau \underline{S}_i^\tau X_{\tau j} & \geq & \underline{R}_i^j Y_j & \forall i = 1, \ldots, n \\
\sum_\tau \lambda(\tau) X_{\tau j} & \leq & (120 - d(j)) Y_j & \forall j \\
\sum_\tau \lambda(\tau) X_{\tau j} & \geq & e(j) Y_j & \forall j \\
X_{\tau j} & \leq & Y_j & \forall \tau, j \\
\sum_{\tau \in \hat{T}} X_{\tau j} & \leq & Y_j & \forall j \\
X_{\tau j} & \in & \{0, 1\} & \forall \tau, j \\
Y_j & \in & \{0, 1\} & \forall j.
\end{array}$$

In the resulting matching problem a number of teams can be assigned to a job only if the combined expertise of the teams meets the requirements of the job. A variable $X_{\tau j}$ equals 1 if team τ is assigned to task j and zero otherwise. The expertise of a team can be written as the sum of the expertises of the technicians that are in the team. A team is assigned to at most one job (in each matching iteration) and a job may or may not be selected. The variable Y_j is 1 only if task j is selected. It can only be selected, if it is simultaneously assigned a set of teams the union of which is competent with respect to the requirements of task j. The additional restrictions enforce that a selected task is processed immediately after a team has finished its already assigned work, without the introduction of idle time, and without breaking any precedence constraint. The assignment and selection problem bears much resemblance with capacitated facility location problems.

The objective is to select a 'nicest' combination of jobs. For measuring the quality of an assignment we consider several *MatchingObjectives* defined by setting coefficients $C_{\tau j}$ and G_j to appropriate values. These values are based on job length, job load, job difficulty or job priority or a combination of these. A default setting

is choosing $C_{\tau j} = 0$, and $G_j = d(j) * \mu(j)$. This reflects the desire to schedule as much work as early as possible. The problem is formulated as a linear integer programming problem and passed to the solver (CPLEX).

A further characteristic that influences the outcome is found in the way the ILP-model of the matching problem is formulated and passed to the solver. One reason is that the solver is given only a limited amount of time. It may therefore have to break off its branch-and-bound search prematurely. A second reason is that there may be more than one optimal solution. In case the coefficients $C_{\tau j} = C_\tau$ are task-independent, the model has a sparse formulation which tends to solve more quickly, and a dense formulation which seems to have a tendency of producing slightly better solutions (in terms of complete day schedules and use of scarce resources). By introduction of a slack variable σ_τ, the first inequality of the model can be converted into an equation: $\sum_j X_{\tau j} + \sigma_\tau = 1 \ \forall \tau$, and then the variables $X_{\tau j}$ may be (partially) deleted from the objective function. In the algorithm we may use different values for the *SparsityFactor* (between 0.0 and 1.0). This refers to the fraction of inequalities for which the above conversion is implemented.

Obviously, for the instances in which part of the jobs can be abandoned, it makes quite a difference for the resulting schedule value, which set of jobs is abandoned. We consider different objectives for selecting jobs to be abandoned. Restrictions are that the cost of abandoning jobs must be within the budget, and secondly that if a job is being abandoned, then also its successors should be abandoned. The selection of abandoned jobs further influences the lower bound on the schedule cost (for the remaining jobs). Therefore we consider several *LowerBoundObjectives*. They reflect the bonus obtained by skipping certain jobs. Again this depends on length, load, priority, difficulty of jobs, or of combinations of these.

Finally, one can try to obtain a solution as close as possible to the computed lower bound. To this purpose it may be necessary to temporarily 'upgrade' some jobs: if $J_1 \to \ldots \to J_k$ is a chain of jobs all of priority 4, and the total length is higher than the difference of the relaxed makespans $C^4_{\max} - C^1_{\max}$, we should complete job J_1 earlier than $C^1_{\max}$. We may therefore change its priority class to 1, before we call the schedule construction algorithm. We consider two levels of *UpdatingPriorities* (yes/no).

2.4. Creating multiple solutions

In principle there are two ways to generate many solutions greedily according to the above scheme. One is to introduce randomness in the ordering and randomness in the matching objective coefficients. We have chosen not to do this but only vary the *SortingRules*, the *LowerBoundObjectives*, the *MatchingObjectives*, the *UpdatingPriorities* over a limited set of parameter settings. By doing this we limit ourselves to the creation of at most $6 \cdot 4 \cdot 3 \cdot 3 \cdot 2 = 432$ schedules. The factor 6 of course results from the different permutations of the priority class makespan orderings. Also one could consider adding a post-processing stage to polish the

obtained schedules. Most of the solutions found in the earliest stages already show good quality.

3. Reflection and prospects

The use of linear programming techniques in the area of schedule construction poses the potential user for several decision questions where effectivity and computation time have to traded off. In particular the fact that the algorithm is allowed a limited amount of time can cause difficulties.

3.1. Time management

In the area of *local search* which is a more standard approach to complex scheduling problems, one usually defines a construction heuristic to come up with a first schedule, and next a search strategy is employed to search the solution space as long as time permits. A variant is one in which the construction heuristic is adapted to the observed solutions, and is repeated again and again.

For integer linear programming models it is very much harder to predict the duration of certain calculations. Branch-and-bound solution methods may end fairly quickly, but may also get stuck in a deep search tree and never end.

For the problems at hand we found the first two problems *preprocessing* and *lower bounds* to behave rather well and we could solve those to optimality without having to set time limits. The *team-job* assignment problem however was much harder to handle. The problems in set A could be handled in their complete formulation – as far as size was concerned. The running times were rather unpredictable. Problems in set B were already so big that finding a first solution could already take much time, and waiting for an optimal answer was out of the question. This problem was resolved by ordering tasks from highly to less important, selecting subsets of 50–80 consecutive tasks, and trying to solve the matching problem with this limited set of candidate tasks. Furthermore it was necessary to set rather small time limits (5–10 s) on the MIP solver, for each matching problem.

For the future we may want to design algorithms capable of rescheduling or of on-line scheduling, based on this MIP approach.

3.2. Lower bound quality

As mentioned before, in some cases the lower bound deviates a lot from the best solution found. In some cases this is caused by a critical set of scarce expertise skills, or by experts only available a part of the time. There is some room for improvement.

3.3. Flat objective

The objective by which schedule quality is measured is rather flat. There is not yet a good understanding of how to prioritize tasks so as to have low priority class

makespans. Of course this difficulty holds also for standard local search methods. It would also be interesting to see how this LP approach will perform when other quality measures are being used, for instance weighted sum of completion times, over all jobs.

3.4. Constraints

The complex restriction of having enough qualified technicians in a team that handles a task is very well suited for our Mixed Integer Linear program approach. We may further study more complex restrictions, such as providing for *stable* assignments. We will further collaborate with France Telecom to investigate many other features of maintenance schedules. For now the conclusion is that MIP methods seem to promise a lot of possibilities.

References

[1] D. Applegate and W. Cook, A computational study of the job-shop scheduling problem. *ORSA J. Comput.* **3** (1991) 149–156.

[2] E.L. Lawler and J. Labetoulle, On preemptive scheduling of unrelated parallel processors by linear programming. *J. Assoc. Comput. Mach.* **25** (1978) 612–619.

[3] Y. Pochet and Laurence A. Wolsey, *Production planning by mixed integer programming.* Springer Series in Operations Research and Financial Engineering. Springer, New York (2006) xxiv+499 p.