Planning and Scheduling Teams of Skilled Workers

Laurent Perron, Paul Shaw,Didier Vidal
ILOG SA, 9 rue de Verdun, 92453 Gentilly Cedex, France

Abstract

Solving problems that mix planning and scheduling are often seen as a challenge. Discrete time-based scheduling, along with complex side constraints does not mix well with the more flexible nature of the planning model. This is demonstrated in our experiments when trying to solve a problem where we must assemble teams of skilled workers to perform jobs that require these skills, break these teams and then assemble new ones to perform more jobs. The mixing of the planning part (grouping workers into teams) and the scheduling part (creating a schedule for each worker), along with some difficult side constraints and a large problem size (800 workers, 2000 jobs over one month) combine to contribute to the challenge of finding good solutions for this problem.

Introduction

Planning and scheduling, the juxtaposition of the two names stems from the technical limitations of the engines used to solve them. On the one hand, we deal with the approximated nature of the long term planning; and we often use math programming to solve it. On the other hand, the discretization or bucketization of time, the low-level side constraints, the special cases and requests that have been approximated out in the planning phase, all ask for another kind of solver, often a constraint-based one.

In fact, we would like to get rid of the distinction and solve both problems at once. This is like solving the crew pairing and the crew scheduling problem at the same time in the airline industry, or the capacity planning and the detailed scheduling in the same model for the discrete manufacturing world.

However in doing so, we often face all kind of difficulties from fitting the model in memory to finding feasible solutions, even trivial ones as the solver has to deal with a heterogeneous model, in which seearch guidance information becomes lost or difficult to extract.

This article tells a version of the same story. The complex and heteronegeous nature of a timetabling problem forced us to look at a decomposition to get a grip on the problem itself. We tried different methods of avoiding a decomposition, from complex modeling to heuristics to reduce the problem size and complexity. All techniques were pitted and evaluated against a simple decomposition schema were the linear constraints were separated from the scheduling ones and given respectively to a MIP solver and a CP solver.

We began our work with an interesting timetabling problem with some twists: travel constraints, set covering constraints, knapsack constraints. We looked at it and came up with two alternative models for it. Both were evaluated against tiny, small, medium and large data sets and the results were extremely disappointing as one was able to treat only the tiny problems and the other was able to treat the tiny and the small ones. However, the goal was to solve the large instances. We were far from success at that time.

To deal with the size of the largest models, we tried two approaches. The first one was to give the packing and set covering part to ILOG CPLEX(CPLEX 2007) and the rest to ILOG CP Optimizer(CP-Optimizer 2007).

The second one was to do some something like simple column generation where part of the problems were precomputed (the packing + set covering part). The the master problem was not a linear one but a timetabling one and was solved with ILOG CP Optimizer.

The article is divided in four sections. The first one will present the problem and discuss its nature. It will also present the implementation details of the side constraints. The second section will present our initial failed experiments. The third section speaks about decomposition and model improvements. The last section presents experimental results on the final two approaches.

Presentation of the Problem

The skilled team problem can be described as follows. Given a set of skills like painting, plumbing, roofing, a set of workers with these skills and a set of jobs that requires these skills, the goal is to assign workers to jobs and to find a start date for each job such that they form an acceptable schedule for each worker. Meaning, all workers participating in the same job work on the same days. A worker can perform at most one job per day. And finally, if a worker has to go to a distant (far) job, he must stay at a hotel before and after this job. In addition, if a worker returns home because he has no job that day, then he cannot leave the same day. This is equivalent to saying that there cannot be exactly one free day between two jobs which are far from home.

This model is a closely related to the audit scheduling problem(Balachandran & Zoltners 1981; Chan & Dodin 1986; J.C. & Lofti 1990; Dodin 1991). Different methods have been proposed to solve it (Bajis & Elimam 1996; Dodin, Elimam, & Rolland 1996; Drexl, Frahm, & Salewski).

Model Description

Given a set of Location $L = \{ l_1, \ldots, l_{\#l} \}$ along with a decision procedure **bool** *far*(l_i, l_j).
Given a set of Skills $S = \{ s_1, \ldots, s_{\#s} \}$.
Given a set of Jobs $J = \{ j_1, \ldots, j_{\#j} \}$.
where $j_i = < l, d, n, s \subseteq S, w >$ with l the index of the location of the job, d the number of days needed to perform the job, n the number of workers needed for the job, s the subset of S of skills required by the job and w the weight (importance) of the job.
Given a set of Workers $W = \{ w_1, \ldots, w_{\#w} \}$.
where $w_i = < l, s \subseteq S >$ where l is the index of the location of the home of the worker and s is the set of skills the worker is qualified for.
Given a number of work days nd.

Given the following variables:

bool $a_{x,y}$; $x \in [1..\#w], y \in [1..\#j]$	w_x performs j_y
bool b_y; $y \in [1..\#j]$	j_y is performed
int t_y in $[0..nd]$; $y \in [1..\#j]$	start time of j_y

The problem can be stated as:

maximize	$\sum_{y \in 1..\#j} j_y.w \times b_y$
subject to	
card:	$\forall_y \sum_x a_{x,y} = b_y \times j_y.n$
day worked:	$\forall_x \sum_y a_{x,y} \times j_y.d \leq nd$
skill covering:	$\forall_y \bigcup_x a_{x,y} \otimes w.s \supseteq j_y.s \otimes b_y$
unperformed:	$\forall_y t_y = 0 \Leftrightarrow b_y = false$
valid schedule:	At most one job at a time per worker
home:	If idle, a worker is at home
forbidden:	Far/home/far is forbidden

In the above model, $s \otimes b$ with s a set and b a boolean value is defined as $\emptyset$ if b is *false* and s if b is *true*.

Discussion

In this problem, we can distinguish between three sub-problems. The first one is a constrained variation on the *knapsack* problem where we want to pack jobs to workers and maximize the pack value. The second one is a *set covering* problem to determine valid combination of workers to assign to a particular job. The last one is derived from a classing *scheduling problem with alternative ressources* and some specific forbidden transitions between activities.

An important aspect of this problem is the size of it. The real life problem this model is derived from counts 800 workers, 2000 jobs, fifteen skills and the scheduler spans roughly twenty days.

Therefore, we have to be careful about model complexity. Let's imagine we maintain a precise agenda for each worker featuring the exact job he is performing each day. Then implementing the compatibility table that will link three consecutive days has a size of 2000 $\times$ 2000 $\times$ 800 = 3.2 billion cells in the dense graph of the relation!

Thus implementing the precise constraint cannot be done in a naive way. This will be the subject of the next section.

Implementation of the Home and Forbidden constraints as a Disjuction

As seen in the previous section, the tricky part in the implementation of the model is the definition of a valid schedule that will express correctly the forbidden sequence constraint.

We first tried to implement the complete schedule with just the start variables (t_y) of the jobs. In that case, we can add the following constraint to state the natural disjunction between jobs than can be performed or not:

disjunct$_1$:	$\forall_{x,y,y'} a_{x,y} \wedge a_{x,y'} \Rightarrow$ $(t_y >= t_{y'} + j'_y.d) \vee (t_y >= t_{y'} + j'_y.d)$

Of course, this formulation is quadratic, and thus does not scale well. Even if we filter out $a_{x,y}$ that we can safely set to false[1] there remains a huge number of constraints of this type.
In addition, this type of constaint (disjunction) is typically handled less efficiently than global or specialized constraints in typical CP solvers. Regardless of the quadratic complexity, we will try to improve the individual constraints themselves.

A better formulation would be to replace the implication by a term in the sum that would nullify the constraint if $a_{x,y} \wedge a_{x,y'}$ is false. This formulation is a bit better as it allows slightly more propagation.

disjunct$_2$:	$\forall_{x,y,y'} (t_y \geq t_{y'} + j'_y.d + \alpha_{x,y,y'})$ $\vee (t_y \geq t_{y'} + j'_y.d + \alpha x, y, y')$
interact:	$\alpha_{x,y,y'} = (a_{x,y} \times a_{x,y'} - 1) \times M$

where M is a big enough constant[2] and $\alpha_{x,y,y'}$ is a three dimensional array on intermediate expressions[3]. This kind of formulation is common in the math programming community.

Using this type of formulations, we can add a constraint a simple constraint that is stronger than the *forbidden* constraint stating that if two jobs are far from the same worker and can interact, then they cannot be one day apart.

[1]Because $w_x.s \cap j_y.s = \emptyset$.

[2]greater than nd for instance.

[3]That are lazily generated, in order not to hit the dreaded $\#j \times \#j \times \#w$ complexity.

$$\begin{array}{ll} \textit{forbidden}_1: & \forall_{x,y,y'|\text{far}_{w_x.l,j_y.l} \wedge \text{far}_{w_x.l,j_{y'}.l}} \\ & (t_y \neq t_{y'} + j'_y.d + \alpha_{x,y,y'} + 1) \\ & \wedge(t_y \neq t_{y'} + j'_y.d + \alpha_{x,y,y'} + 1) \end{array}$$

This constraint is actually cutting valid solution as it would have been possible to have a one day job in between two far jobs. We will evaluate them in the experimentation section.

Maintaining the Precise Agenda of Workers

Another possible implementation is to introduce variables that will record the precise agenda of workers.

$$\textbf{int } g_{x,d} \text{ in } [0..\#j];\ x \in [1..\#w],\ d \in [1..nd]$$

The variable $g_{x,d}$ represent the job performed by the worker at the date d. A value of zero indicates that the worker is idle.

To help implement the *forbidden* and *home* constraints, we will introduce three sets of auxiliary variables:

$$\begin{array}{l} \textbf{bool } h_{x,d};\ x \in [1..\#w],\ d \in [1..nd] \\ \textbf{bool } f_{x,d};\ x \in [1..\#w],\ d \in [1..nd] \\ \textbf{int } \textit{worked}_x,\ x \in [0..nd] \end{array}$$

where $h_{x,d}$ is **true** when the worker x is idle on day d, **false** otherwise; and $f_{x,d}$ is **true** when the worker x is working far from home on day d and **false** otherwise. The variable *worked* computes the total number of days worked per workers.

When this is done, we can pose constraints that will set the g and f variables when a job is assigned to a worker.

$$\begin{array}{ll} \textit{agenda}: & \forall_{x,d},\ a_{x,y} \Rightarrow \bigvee_{\delta \in [0..j_y.d-1]} g_{x,t_y+\delta} = y \\ \textit{far}_1: & \forall_{x,d},\ a_{x,y} \Rightarrow \\ & \bigvee_{\delta \in [0..j_y.d-1]} f_{x,t_y+\delta} = \text{far}(w_x.l, j_y.l) \end{array}$$

Computing the h variables is a bit more complex. As the constraints that maintain the agendas are implications between the a variables and the g variables, deciding if a worker is idle is a bit tricky if not all teams have been built and all start times assigned.

To compute the h variables, we count the number of days worked and we know that for any worker, the number of days worked + the number of days idle is always equal to nd. Thus we can write the following constraints:

$$\begin{array}{ll} \textit{worked}: & \forall_x\ \textit{worked}_x = \sum_y a_{x,y} w_y.d \\ \textit{idle}_1: & \forall_{x,d},\ h_{x,d} \Leftrightarrow g_{x,d} = 0 \\ \textit{full schedule}: & \forall_x,\ \textit{worked}_x + \sum_d h_{x,d} = nd \end{array}$$

With all the extra variables and constraints, we can now state the *forbidden* constraint:

$$\begin{array}{ll} \textit{forbidden}_2: & \forall_{x,d \in [1..nd-2]} \\ & f_{x,d} + h_{x,d+1} + f_{x,d+2} \leq 2 \end{array}$$

This constraint, as opposed to the *forbidden*$_1$ constraint, the implementation of *forbidden*$_2$ is exact. It does not rule out valid solutions. Unfortunately, it propagates very late as only when the schedule for a worker finished is this constraint fired – because only at that time are the h variables completely defined.

Solving the Complete Problem

In this section, we investigate the effect of data size on the feasibility of the previous approaches and the different consumptions in term of memory and time.

Test Sets

To evaluate the different consumptions for the model, we have generated 4 tests sets of different size:

Tiny: 20 workers and 60 jobs

Small: 40 workers and 200 jobs

Medium: 100 workers and 500 jobs

Large: 800 workers and 2000 jobs. This is the size of the real world problem this model is inspired from.

All these test sets have 15 skills, 20 days. The *far* predicate is implemented in the following way. All workers homes and all jobs locations are placed randomly on a 10×10 grid. Then we use a cutoff distance (6) and a manhattan distance.

Thus, one job y and one worker x 's home are far from each other if and only if

$$abs(j_y.posX - w_x.posX) + abs(j_y.posY - w_x.posY) > 6$$

We will use these data sets to test ideas. As the large size is very challenging to solve, we cannot hope to test new ideas easily. Thus the need for smaller test sets to evaluate ideas before the polishing needed to solve the large instance.

Experimental Context

Due to various external constraints, the model has to be coded in ILOG OPL 5.2(OPL 2007) and the search part has to be very simple.

The goal here is find how we can solve a large and complex problem without writing complex search procedures or custom constraints.

Hitting the Size Limit

We evaluate our two implementations and the different test sets. For all experiments, we present the number of constraints in our engine used to solve it, the number of variables in the model, the memory used, the number of possible assignments – that is the number of pairs of compatible worker - job, and the number of possible interactions between two jobs, that is the number of times two jobs may share a worker. This will for instance count the number of disjunctions in the disjunctive model.

We begin with the disjunctive model on the tiny samples as any other size of sample will not fit into 1.5 GB memory. We tried with and without a simple shaving schema (as exposed in the next section).

We first report the disjunctive model on the tiny sample with and without shaving.

	no shaving	shaving
# constraints	2208	2045
memory (MB)	37	35
# variables	1340	1340
# assignments	488	293
# interactions	3600	961

This implementation would not even create the model for other sizes of test sets (small, medium and large).

We move on to the agenda based implementation on the tiny test sets.

	no shaving	shaving
# constraints	10073	7192
memory (MB)	17.6	12
# variables	2180	2180
# assignments	488	293
# interactions	3600	961

And the on the agenda based implementation of the small test sets.

	no shaving	shaving
# constraints	62764	49805
memory (MB)	212	166
# variables	10120	10120
# assignments	3721	2857
# interactions	40000	20736

and finally on the medium test sets.

	no shaving	shaving
# constraints	too large	too large
memory (MB)	too large	too large
# variables	too large	too large
# assignments	22792	20481
# interactions	250000	190969

The large test set is not reachable with this model. For the medium test sets, only the shaving part is performed. The engine would not create the model and post constraints.

Discussion

The two model tested in this section performs very badly. We can analyse why.

On the **disjunctive** model, the problem comes from the implementation of the *forbidden* constraint. While the rest of the model is very light, this constraint set is not. In fact, if p_1 is the probability of a worker to be able to perform a job, then this worker may perform $\#j \times p_1$ jobs. If p_2 is the probability of a job to be far from home of a worker, then the number of forbidden constraints for a worker is $(\#j \times p_1 \times p_2)^2$.

Thus we have a total number of constraints in term of $\#j^2 \times \#w$. This is catastrophic.

If we look at the **agenda** based model, what is costly in the model is the *agenda* constraint itself. In the constraint, we have an element constraint:

$$\forall_{x,d},\ a_{x,y} \Rightarrow \bigvee_{\delta \in [0..j_y.d-1]} g_{x,t_y+\delta} = y$$

The $g_{s,t_y+\delta}$ part. This one is expensive because we have $\#w \times \#j \times nd \times$ average duration of these constraints. This means 160000 * 2 = 320000 constraints if the average duration of a job is 2. This is not as bad as before but still it will not even reach the medium instances (2,000,000 of this constraints).

Improving the Model

As we have seen before, solving the large model directly is not tractable. First we have improved the timetabling model and second we have investigated two possible ways of containing the complexity of the model.

There are different ways to reduce the size of the problem.

- The first one is *exact* and and is based on an real computation of feasible combination of workers to perform a job. With this information, we can rule out workers that never appear in any feasible combination[4].
- The second one is *heuristic*. We need a way to reduce the number of possibilities. We will implement two methods, one based on a limitation of the previous exact method and the second on a hybrid decomposition of the problem using a simplex to solve the assignment part.

Maintaining Active Jobs

The previous implementations of the scheduling were not satisfying. We worked on another one that would count active jobs on a given day. For this model, we reused the same *f*, *h* and *g* variables from the previous models:

int $g_{x,d}$ in $[0..\#j]$; $x \in [1..\#w]$, $d \in [1..nd]$
bool $h_{x,d}$; $x \in [1..\#w]$, $d \in [1..nd]$
bool $f_{x,d}$; $x \in [1..\#w]$, $d \in [1..nd]$

and we introduce a new kind of variables *e* to decide if a job y s active at a given date d.

bool $e_{y,d}$; $y \in [1..\#j]$, $d \in [1..nd]$

We can now post constraint that will maintain these *e* variables:

effective: $\forall_{y,d} e_{y,d} = (d - j_y.d + 1 \leq t_y \leq d)$

Which basically says that the time interval representing the job y is spanning over the day d.

We can now implement the *idle*, *far*, *valid schedule* and *forbidden* constraints.

valid schedule$_1$:	$\forall_{x,d} \sum_y e_{y,d} \times a_{x,y} \leq 1$
idle$_2$:	$\forall_{x,d} \sum_y e_{y,d} \times a_{x,y} + h_{x,d} = 1$
far$_2$:	$\forall_{x,d} \sum_{y \mid far(x,y)} e_{y,d} \times a_{x,y} = f_{x,d}$
forbidden$_3$:	$\forall_{x,d \in [1..nd-2]}$
	$f_{x,d} + h_{x,d+1} + f_{x,d+2} \leq 2$

[4]This will force the corresponding $a_{x,y}$ variable to 0

The *valid schedule* is a simple constraint. It states that at most one job is active for any given day and any given worker.

The *idle* is also simple as it states that a worker is either performing a job or idle. It is interesting to see that the h variables are in fact the slack variables of the *valid schedule* constraints. In that case, the *idle* constraints subsumes the *valid schedule* constraint and the latter can be removed.

In the same spirit, the *far* constraint just checks if there is one far job active for a given worker and a given day.

The *forbidden* constraint is the same as the previous one.

This model is much better than the previous one in our case as the complexity depends on the number of time points, which is low in our case. Thus the discrete time approach is much lighter in memory than the disjunctive one.

Shaving Combinations of Workers

The scope of the *skill covering* constraint is limited to one assignemnt at a time. We have added another constraint that rules out workers that have no skills needed by the job:

$$\textit{exclusion}: \quad \forall_{x,y} \;\; j_y.s \cap w_x.s = \emptyset \Rightarrow a_{x,y} = 0$$

With this method, we can create a sub-model that will compute feasible solutions of the *skill covering*, *card* and *exclusion constraints.*

Now, we can embed this algorithm inside a script that will loop over feasible solutions and record workers selected by the sub-algorithm.

We can now experiment with this shaving module.

	tiny	**small**	**medium**	**large**
# jobs	60	200	500	2000
# workers	20	40	100	800
# possible	488	3721	22792	762596
# removed	305	1513	6334	–
# removed jobs	32	56	63	–
run time (s)	0.2	2.8	58	–

where *possible* counts the number of possibles pairs (workers, jobs) as given by the *exclusion* constraint and *removed* gives the number of such pairs the shaving procedure has removed.

A '–' indicates that the computation excedeed a 20 minutes time limit.

While promising, this technique is not useful in practise because of the runtime for the large instances – the one we want to solve. We must find a solution for this runtime problem.

What we can do is limit the maximum number of explored solutions for one job. If we hit the solution limit, we use the possible assignments as given by the *exclusion* constraint This approach will just sacrifice quality of shaving w.r.t. time.

Let's see the effect of shaving when we experiment with this solution limit. We look at the number of removed assignments and run time when constraint the number of solutions explored. This is guide us in the time/quality balance.

First with the tiny test sets

solution limit	**#possible**	**# removed**	**run time**
5	488	290	0.2
10	488	293	0.2
50	488	305	0.2
100	488	305	0.2
200	488	305	0.2
500	488	305	0.2
1000	488	305	0.2

Then with the small test sets:

solution limit	**#possible**	**# removed**	**run time**
5	3721	1153	1.0
10	3721	1265	1.2
50	3721	1454	2.0
100	3721	1485	2.4
200	3721	1502	2.7
500	3721	1513	2.8
1000	3721	1513	2.8

then with the medium test sets:

solution limit	**#possible**	**# removed**	**run time**
5	22792	4535	6.4
10	22792	5043	7.9
50	22792	6241	18.7
100	22792	6301	28.2
200	22792	6313	39.9
500	22792	6325	53.7
1000	22792	6334	57.3

and finally with the large test sets:

solution limit	**#possible**	**# removed**	**run time**
5	762596	85734	258
10	762596	87225	320
50	762596	140997	793
100	762596	–	–
200	762596	–	–
500	762596	–	–
1000	762596	–	–

The idea to limit the loop is useful in practice and allow a correct shaving and a robust one in term of runtime if we restrict ourselves to small limits (less than 50).

Furthermore, the sheer numbers displayed illustrates the complexity of the problems. In the large instances, 762596 possible assignments is simply to big.

Limit Combinations of Workers

The idea is to change the behavior of the shaving procedure when the solution limit is crossed. In that case, instead of recording the possible assignments, we record the assignments found in the previous solution.

Thus we limit the possible combination and remove feasible solutions from the model. On the other hand, we will get a much smaller problem. In that sense, it is interesting to look at small values for the loop limit.

First with the tiny test sets

solution limit	#possible	# removed	run time
3	488	366	0.2
6	488	343	0.2
10	488	322	0.2
30	488	305	0.2
60	488	305	0.2
100	488	305	0.2
300	488	305	0.2

Then with the small test sets:

solution limit	#possible	# removed	run time
3	3721	3007	0.9
6	3721	2725	1.0
10	3721	2443	1.2
30	3721	1835	1.7
60	3721	1623	2.1
100	3721	1538	2.4
300	3721	1513	2.6

then with the medium test sets:

solution limit	#possible	# removed	run time
3	22792	20653	5.5
6	22792	19579	6.4
10	22792	18402	7.4
30	22792	14232	12.6
60	22792	11019	19.3
100	22792	9250	26.8
300	22792	6967	45.9

and finally with the large test sets:

solution limit	#possible	# removed	run time
3	762596	753728	211
6	762596	748467	255
10	762596	741720	309
30	762596	715522	560
60	762596	686295	907
100	762596	649563	1341
300	762596	522302	3540

This technique shows good results in reducing the total size of the model. We will evaluate these techniques in the results section.

Hybrid Implementation

The idea here is to use ILOG CPLEX to solve the packing + set covering problem. More specfically the *card*, *day worked*, *skill covering* and *exclusion* constraints. The following unique assignment is then given to the schedule.

This method can be seen as an optimized shaving version. The good effect is that it simplifies a lot the scheduling module.

Here are the remaining constraints (we note $\beta_{x,y}$ if the assignment (worker x on job y) is selected by the planning. This is now a data and not a variable anymore):

unperformed:	$\forall_y t_y = 0 \Leftrightarrow b_y = false$
effective:	$\forall_{y,d} e_{y,d} = (d - j_y.d + 1 \leq t_y \leq d)$
idle$_3$:	$\forall_{x,d} (\sum_{y\|\beta_{x,y}} e_{y,d} \times b_y) + h_{x,d} = 1$
far$_3$:	$\forall_{x,d} \sum_{y\|far(x,y) \wedge \beta_{x,y}} e_{y,d} \times b_y = f_{x,d}$
forbidden$_4$:	$\forall_{x,d\in[1..nd-2]}$
	$f_{x,d} + h_{x,d+1} + f_{x,d+2} \leq 2$

We can make an important remark. By combining *unperformed* and *effective*, we can notice that a job is active only if its start time is greater than 0.

Thus we can rewrite the *effective* constraint this way:

effective$_1$:	$\forall_{y,d}$
	$e_{y,d} = (\max(1, d - j_y.d + 1) \leq t_y \leq d)$

With this new formulation, a job is effective only if it is performed. This has an impact on other constraints as the multiplication by b_y is not needed any more.

Thus we have the simplified and last model:

unperformed:	$\forall_y t_y = 0 \Leftrightarrow b_y = false$
effective$_1$:	$\forall_{y,d}$
	$e_{y,d} = (\max(1, d - j_y.d + 1) \leq t_y \leq d)$
idle$_4$:	$\forall_{x,d} (\sum_{y\|\beta_{x,y}} e_{y,d})) + h_{x,d} = 1$
far$_4$:	$\forall_{x,d} \sum_{y\|far(x,y) \wedge \beta_{x,y}} e_{y,d} = f_{x,d}$
forbidden$_4$:	$\forall_{x,d\in[1..nd-2]}$
	$f_{x,d} + h_{x,d+1} + f_{x,d+2} \leq 2$

This model is much smaller that the full scheduling model developped in the previous sections. This will be visible in the memory consumption of the different tests.

Experimental Results

It is time now to evaluate these two new models on the different test sets.

All experiments are made with ILOG OPL 5.2. They are made on a Intel quad 2.67 GHz Xeon with 4 GB of memory running Fedora 7 (64 bit).

Results with Limited Combinations of Workers

We give the results with the full model and the model limited with a solution limit of six and three. The time limit is 2s per job, thus 120, 400, 1000 and 4000s.

Here is the tiny test set with a solution limit of 3 and 6:

tiny test set	Full	Limited 6	Limited 3
Memory (MB):	7.1	6.4	6.2
Best Solution found:	48	48	48
Time (s):	11	30	30
Planning Solution:	48	48	48

and for the small test set:

small test set	Full	Limited 6	Limited 3
Memory (MB):	52	32	28.8
Best Solution found:	111	142	145
Time (s):	350	347	348
Planning Solution:	200	200	200

and the medium data set:

medium test set	**Full**	**Limited 6**	**Limited 3**
Memory (MB):	350	108	98
Best Solution found:	**0**	159	190
Time (s):	308	910	953
Planning Solution:	510	510	510

As we have seen before, the full model is not able to create the problem for the large instances. The limited model is able to create the problem but does not find a solution when any job is assigned in less than 1h. We could have improved the search heuristics to solve it, but we had decided early in the project that we would use the default search of ILOG CP Optimizer(Refalo 2004) with a minimum effort.

Results with Hybrid Model

With the hybrid instances, we get much better results:

Here is the tiny test set:

tiny test set	**Hybrid Model**
Memory (MB):	2.3
Best Solution found:	48
Time (s):	0
Planning Solution:	48

and for the small test set:

small test set	**Hybrid Model**
Memory (MB):	6
Best Solution found:	196
Time (s):	358
Planning Solution:	200

and the medium data set:

medium test set	**Hybrid Model**
Memory (MB):	14
Best Solution found:	510
Time (s):	81
Planning Solution:	510

and the large data set:

large test set	**Hybrid Model**
Memory (MB):	102
Best Solution found:	1759
Time (s):	3240
Planning Solution:	3327

Discussion on the Results

Without limiting the complexity of the problem, we simply cannot solve the problem.

Furthermore, with a naive heuristics to reduce its size as implemented by the shaving part, we still do not get good results. We get the optimal solutions on the tiny samples, but even the full model finds them. We get good solutions on the small instances, the smaller the limit, the better they are. Thus, the more we reduce the problem, the lowest the optimal value but the better the best solution found.

On medium instance, we find poor solutions, far from the optimal ones.

Thus it is the complexity of the problem that forbids the a good search strategy. The search is lost and the number of constraints is so huge that we just do not search enough. We tried more aggressive search strategies, ones that would try to perform all jobs instead of one that would first try not to perform any job and then perform more and more of them (branch up instead of branch down on the b_y variables). But this one is not robust enough and while very good solutions are found on the small and tiny test sets, we do not find any solution for the medium and large instances.

Finally, the hybrid solution is by far the best and most robust approach. It consumes less memory, finds good solution. Still, on the large instances, there is room for improvement as we are quite far (1759 vs 3327).

Conclusion

The story repeats itself. We tried to get rid of the distinction between planning and scheduling on this timetabling problem and we failed.

The combinatorial explosion of the search space and of the number of constraints are the main limiting factors. As a result, the problem cannot be solved by one engine in a single run. Decomposition have to be used.

Furthermore, we are a bit disappointed by the results of the model with limited combination of workers. This is particularly visible on the medium data set where the obtained results (around 150) are very far from the planning solution (510).

If we look at the bright side, the hybrid solution is very small and elegant. It finds optimal solutions quickly for all instances except the large ones. And for the large instances, it finds good solutions and we are confident we will find a way to solve the problem effectively with a little bit of tweaking.

Finally, in order to sparkle discussion and comparison with other methods, we have decided to make the instances public. They can be obtained upon request from the author. Please note that we are working on a more complex version of the problem where some days are unavailable for workers. This will be the subject of future work.

References

Bajis, D., and Elimam, A. 1996. Audit scheduling with overlapping activities and sequence dependent setup costs. The A. Gary Anderson Graduate School of Management 96-09, The A. Gary Anderson Graduate School of Management. University of California Riverside. available at http://ideas.repec.org/p/fth/caland/96-09.html.

Balachandran, B., and Zoltners, A. 1981. An interactive audit–staff scheduling decision support system. *The Accounting Review* 56:801–812.

Chan, K., and Dodin, B. 1986. A decision support system for audit–staff scheduling with precedence constraints and due dates. *The Accounting Review* 61:726–733.

CP-Optimizer. 2007. *ILOG CP Optimizer 1.0 User's Manual and Reference Manual*. ILOG, S.A.

CPLEX. 2007. *ILOG CPLEX 10.2 User's Manual and Reference Manual*. ILOG, S.A.

Dodin, B.; Elimam, A.; and Rolland, E. 1996. Tabu search in audit scheduling. The A. Gary Anderson Graduate School of Management 96-25, The A. Gary Anderson Graduate School of Management. University of California Riverside. available at http://ideas.repec.org/p/fth/caland/96-25.html.

Dodin, B., C. K. 1991. Application of production scheduling methods to external and internal audit scheduling. *Journal of Operational Research* 52:267–279.

Drexl, A.; Frahm, J.; and Salewski, F. Audit-staff scheduling by column generation.

J.C., R. H., and Lofti, V. 1990. A multiperiod audit staff planning model using multiple objectives: Development and evaluation. *Decision Sciences* 21:154–170.

OPL. 2007. *ILOG OPL 5.2 User's Manual and Reference Manual*. ILOG, S.A.

Refalo, P. 2004. Impact based strategies for constraint programming. In *Proceedings of CP 2004*.

Acknowledgements

I would like to thank Alex Fleisher, Frank Wagner, Philippe Refalo, Olivier Lhomme and Frédéric Delhoume for their contribution to this work.

Annex

Here is the tuple definition in ILOG OPL 5.2(OPL 2007)

```
tuple Assignment {
    int duration;
    int required;
    int weight;
    int posX;
    int posY;
    int skills[allSkills];
}

tuple Worker {
    int homeX;
    int homeY;
    int qualifications[allSkills];
}
```

and here is what a data test looks like

```
nbWorkers = 20;
nbJobs = 60;
nbSkills = 15;
nbDays = 20;
assignments = [
<3 4 1 2 1 [1, 0, 1, 0, 0, 1, 0,
1, 0, 1, 0, 0, 0, 0, 1] >
...
];
workers = [
<7 3 [0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 1] >
...
];
```