France Telecom Workforce Scheduling Problem: a challenge*

S. Pokutta[1] and G. Stauffer[2]

[1] Operations Research Center, MIT, 77 Massachusetts Avenue, Cambridge, MA 02139
pokutta@mit.edu
[2] Department of Mathematics, MIT, 77 Massachusetts Avenue, Cambridge, MA 02139
gstauffe@math.mit.edu

1 Introduction

Anybody who had to deal with the organization of any kind of project, from a dinner with classmates to sending humans on the moon knows that one of the keys to success is the right distribution of responsibilities among the members of the team. People are not interchangeable and some are clearly more talented than others when it comes to organizational tasks, marketing aspects or human relationships. Thus the central problem is how one should efficiently use the available resources (skills) to ensure the best possible outcome. This kind of problems, often referred as workforce scheduling problems, has received more and more attention in the last years. With the development of highly competitive market, nowadays it is not only a question of maximizing the profit of a company but in addition also to ensure its survival.

France Telecom (FT) is one of Europe's leading provider for telecommunication services. It employs a huge amount of technicians to maintain, repair and develop its infrastructure and to provide services to its customer. With the introduction of more and more new technologies, such as voice-over-IP or TV-on-demand and the liberalization of the french market, it is crucial for FT to efficiently manage its pool of technicians. The resulting dispatching problem (which in fact is a scheduling problem) is the basis for the Roadef Challenge 2007[3].

The aim of the challenge is to provide a solution to this problem that could serve two goals: at an operational level, to efficiently schedule the different interventions with the current pool of technicians available ; at a strategical level, to decide when the resources become critical and what kind of training or hiring could improve the flexibility and efficiency of the pool of technicians.

Problem description

In this section we want to give a brief, yet more detailed description of this workforce scheduling problem posed by France Telecom (FTWFSP). The problem is defined by a list of interventions to be scheduled and a set of technicians that should resolve the interventions. Each technician has different skills and different levels of competence for each skill. Moreover the days where employees are off, due to vacations or work conventions, are given. Each intervention is characterized by a list of requirements i.e. for each skill and each level, we are given the number of people needed (note that overqualified technicians can be used for simple tasks). Interventions are related by precedence constraints and some of the interventions can be outsourced at a certain cost (mainly the tasks without successor in the precedence graph; otherwise all the successors have to be outsourced too). Many interventions require several employees to be combined into *teams*. Since the member of a team usually share a vehicle, the cannot be split during a day.

The goal is to minimize the reaction time i.e. the schedule horizon but since the interventions have different priorities from 1 to 4, the formal objective is to minimize a weighted makespan of the tasks with priority 1,2,3 plus the total makespan. For a more detailed/exact description we refer to the official subject description and the FAQ of the Roadef Challenge 2007.

FTWFSP can be formulated as a Resource-Constrained Project-Scheduling Problem (RCPSP) (cf. [6]). RCPSP is well known to be NP-hard and even worse, it is NP-hard to approximate within a factor $n^{(1-\epsilon)}$ for any $\epsilon > 0$, where n is the number of interventions. There are very

* This work was supported by fellowships within the Postdoc-Programme of the German Academic Exchange Service (DAAD) and the Swiss National Foundation for Research (SNF)

[3] http://gilco.inpg.fr/ChallengeROADEF2007/

straightforward naïve integer programming formulation for FTWFSP. Nevertheless, as for most scheduling problem, the natural formulation leads to intractable problem for state-of-the-art solvers (given the dimensionality of the FT instances). Other exact approaches based on constrained programming for instance also suffer from the high dimension of the problem. We therefore decided to concentrate on heuristics in view of the mentioned bad approximation properties. Nevertheless, we also believe that it is very likely that some of the characteritics of the problem (e.g. limited number of intervention durations) could lead to interesting approximation results. We did not investigate this aspect into detail yet but we believe that some of our heuristics could be converted into approximation algorithms for large subclasses.

2 Methodology

In this Section we want discuss our approach for solving the problem formulated by France Telecom. As already pointed out, the problem is closely related to the resource-constrained project-scheduling problem so that some insight gained from those problems may be applied to this workforce scheduling problem as well. For a more detailed discussion of resource-constrained project-scheduling problem and workforce scheduling problems see for example [6], [5], [7], [1] and [8].

Following the survey paper by Kolisch and Hartmann (cf. [6]), we use a meta-heuristic strategy that builds upon the concept of activity list representation and schedule generation schemes (SGS). More formally, each schedule is encoded as a ordered list of tasks (the activity list) and the schedule generation scheme is the decoder function which then transcripts this activity list into a feasible schedule. We use a sequential SGS to decrypt our activitiy list i.e. we traverse the activity list and greedily insert the tasks into the schedule as early as possible i.e. when all precedence constraints are satisfied and resources (technicians), that satisfy the task's requirements, are available. In order to incorporate the possibility of outsourcing interventions we apply a very simple heuristic right at the beginning and remove outsourced tasks from the list of interventions which have to be scheduled (see Section 2.3).

Checking that the predecessor of a task have already been scheduled is easy. Now in order to assign a new team to a task on day D, we solve a very simple set covering problem (nevertheless, as we see later, we have to solve a lot of them). Assume that we have N different skills and L different possible level per skill. Let E be the set of employees still available on day D. For each employee e we denote by $S_e \in \{0,1\}^{N\times L}$ the characteristic vector of the skills of employee e i.e. for every skill $n \in \{1,\dots,N\}$ and every level $l \in \{1,\dots,L\}$, $S_e(n,l) = 0$ if the level for skill n of employee e is less than l and $S_e(n,l) = 1$ otherwise. We also denote by R the characteristic vector of the intervention's requirements i.e. for all $n \in \{1,\dots,N\}$ and all $l \in \{1,\dots,L\}$, $R(n,l)$ is the number of people of level l required for skill n. We define for each employee e a boolean variable x_e to decide if employee e will be part of the team. Obviously, the goal is to minimize the number of people used for this intervention. Thus the problem can be formulated as:

$$\begin{aligned} \min & \textstyle\sum_{e\in E} x_e \\ & \textstyle\sum_{e\in E} S_e x_e \geq R \\ & x_e \in \{0,1\} \end{aligned}$$

In the final algorithm we use a slight variation of this problem where among all teams of minimum size, we take the one that is the least overqualified. By doing so we try to prevent the waste of skills. Note that we use previously defined teams of technicians when they satisfy the requirements of the new task and their daily capacity is not reached yet.

The meta-heuristic strategy we use is a randomized local search. In order to make this meta-heuristic work efficiently, we need to provide good initial solutions i.e. good initial activity lists. Since the characteristics of the Instance Sets itself are not *a priori* clear and often of a mixed nature, we calculate a set of different activity list (i.e. initial solutions) emphasizing on different characteristics of the Instance Sets. These lists are then passed to the local search. The current implementation uses 3 different initial candidates:

1. Easy ordering by priority.
2. First order by priority and then by the size of team needed.
3. Ordering derived from the critical paths.

This already provides a sophisticated algorithm to tackle the problem. Nevertheless, there is another key ingredient which improves the algorithm's performance: The pairing of interventions to so called intervention packs which we discuss in the following Section 2.1. The local search itself will be discussed in detail in Section 2.2 .

2.1 Pairing and packing of interventions to larger blocks

In order to generate good schedules it seems natural that teams do not waste their time within their daily operations i.e. it would make no sense if in the morning a team of, say four people is fully involved in an intervention while in the afternoon only one of the member works while the other play poker... Therefore we considered the idea of initially aggregating the tasks by similarity. This has two positive impacts. First, when critical resources are involved, we then tend to use them more efficiently. Moreover, it removes a lot of symmetries and narrows down the solution space which then makes the problem more tractable for the local search. In order to do so, we need a good measure of similarity. It is clear that there are different measure which emphasize on different aspects of similarity. After investigating on the different measures we decided to use the following one. A pack of tasks is considered to be *similar* when the number of people required to process the whole *intervention pack* is not too different from the minimal number of technicians needed for the simplest one, i.e the overhead is small. We would thus ideally like to aggregate interventions to intervention packs by minimizing the total overhead. Unfortunately, the different algorithms we had in mind to solve this problem were impracticable due to the size of the problem and it is not clear if there are efficient algorithms. Nevertheless, we observed that most of the time the duration of the tasks are multiples of 15 (i.e. $15, 30, \ldots, 120$). We decided to exploit this property and to approximate the packing by iteratively pairing the tasks by similarity. The pairing problem can be formulated as a matching problem and thus can be solved in polynomial time. We formulate this matching problem as an integer program and we use Cplex [4] to solve it.

Despite from this, the actual problem which has to be solved here is much more complicated. The precedence constraints have to be satisfied and naïvly packing the interventions to intervention packs can easily lead to cycles in the precedence graph which make the problem infeasible. We tackle this with a branch-and-cut approach by dynamically checking whether the current best solution contains cycles and if so, we add the following cut: If variables $x_1, \ldots, x_k$ are responsible for creating a cycle, we simply add the inequality $\sum_{i=1}^{k} x_i \leq k-1$. This branch-and-cut approach is implemented using Cplex and the concept of lazy constraints. Moreover, due to the dimensionality and in order to satisfy the time limit imposed by the challenge, we decided to implement an approximate version of above pairing problem.

A packing that does not contain any cycles may nevertheless be problematic: It may contain a very long precedence path which bounds the makespan from below, e.g. if the longest path is of length 13 (with respect to a daily measure) then we need at least 13 days to process all interventions. For an example of a highly complicated precedence graph arising from a naïve packing see Figure 1. This precedence graph is arising from instance set B6.

These complicated precedence graphs impose two major problems. First, as already mentioned, it imposes an artificial lower bound for the makespan and hence can result in bad schedules. On the other hand, it is easy to imagine that such a complicated graph can also have severe impact on the performance of the local search. Indeed, very specific transformations are needed to actually change the schedule by changing the activity list and thus many changes to the activity lists may remain without any effect as they violate a precedence constraint and hence are cancelled by the SGS later. So there is a certain trade-off between packing tasks together and not creating a too complicated precedence graph. To get hold of this problem we adaptively pack the interventions together as long as the precedence graph does not get too complicated.

2.2 Local Search on the activity list

Local search techniques suffer from a lot of drawbacks although they have proven to be a very effective for improving given starting solutions. First of all, since we only search locally around a given solution it may be very well that we run into a local optimium which is far away from the global optimum. Especially, when no quality measures like lower bounds (e.g. from relaxations)

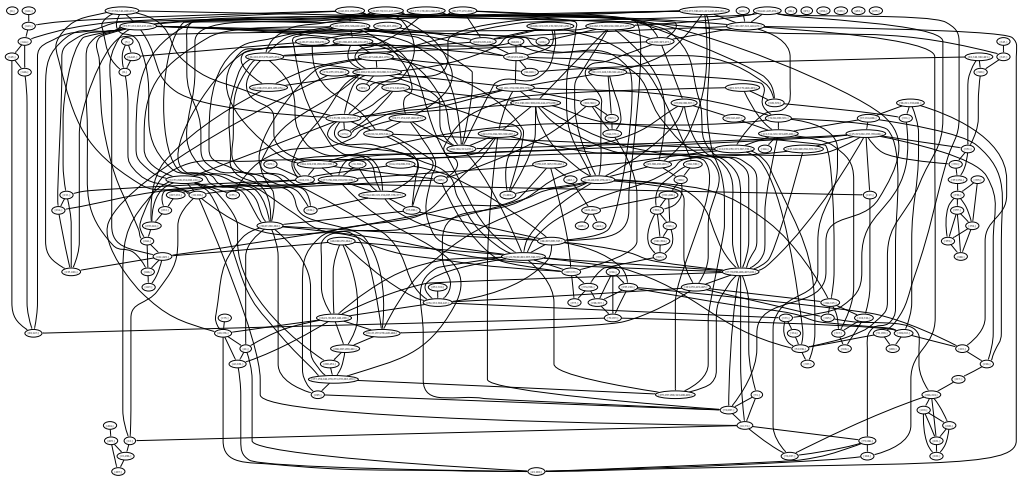

Fig. 1. The precedence graph of Instance Set B6 for a naïve pairing

are known it is almost impossible to decide if a given solution is a *good* solution. Moreover, for the considered problem the search space is enormous which makes even an exhaustive local search around a given feasible solution impossible.

The designed local search algorithm performs a search on the activity lists. Those lists are then transcripted into feasible schedules as described in Section 2. We start from a given activity list and traverse its *neighbors*. Two activity lists are neighbors if one can be obtained from the other by a transposition i.e. we swap two interventions (or the corresponding intervention packs). This transformation is motivated by the assumption that a (very) good solution can be obtained by using the right activity list and the fact that every bijection (i.e. permutation) of this list can be represented by a product of transpositions (it is not clear that there is a permutation that produce an optimal solution due to the teaming procedure see Section 2). Then in a classical manner we try to iteratively improve the activity list. We want to point out that this iterative procedure may not generate an optimal solution as it may be very well that the optimal solution may not be reached by moving in the search space in a way that the objective function value is non-increasing. Nevertheless, if the initial solution is already of a good quality this approach has proven to be effective.

In order to design an effective local search algorithm for this workforce scheduling problem, we had to deal with different problems. One problem is to choose the right amount of neighbors (i.e. candidates close to a given feasible activity list) and the depth of the local search. Since the time is limited, testing too many candidates may lead to too easy permutations. This then results in only minor improvements of the activity list. On the contrary, if we check too few neighbors it is possible that we create very complicated permutations (i.e. a product of transpositions with a lot of transpositions) with only little impact on the actual activity list. Again an improvement is rather unlikely. Our empirical experiments in this respect have shown that the right trade-off between depth and breadth is essential for the performance of the local search.

Another point is that it does not make sense to try an arbitrary transposition as it is clear that a lot of those transpositions are unlikely to generate improvements. For example it only makes sense in very seldom cases to exchange an intervention of priority 1 with an intervention of priority 4. Therefore we put effort in understanding the transformations of the list which are more likely to improve the quality. The following transformations have proven (by statistical analysis) to be quite effective. Moreover, we add some random transpositions so that we use the following *neighborhood (transformation) types* on the activity list:

1. Exchanging interventions of the same priority.
2. Exchanging interventions of priority difference at most 1.
3. Exchanging interventions that are not too far away from each other in the list.
4. Exchanging 2 randomly chosen interventions.

Since the structure of the instances is quite diverse it is likely that different configurations of neighborhoods and neighborhood sizes perform differently on the different instances. This assumption is supported by empirical tests on the instance sets. Therefore we decided to implement a local search algorithm which dynamically adapts its behavior to the actual instance set with respect to the neighborhood size and we generate candidates according to the distribution of the improving neighborhood types. The quality of every generated candidate is checked by actually transcripting the activity list into the corresponding schedule using the SGS.

2.3 Outsourcing interventions

The current implementation handles the possibility of outsourcing tasks to external contractors only in a very rudimentary way and there is still room for further improvements. In fact right at the beginning we calculate a certain cost-measure, which relates the amount of resources (technicians) a job occupies, the job's priority and the costs of outsourcing the intervention. Afterwards the jobs of highest cost-measure are outsourced which is done by removing them from the list of interventions we have to schedule.

3 Computational results and general remarks

In the following we want to present our computational results for the instance sets A and B provided by FT. Since the performance of our algorithm is (for obvious reasons) subject to the actual characteristics of the computer it is run on, we want to clarify that the results presented in the tables below were obtained on a machine with four Intel(R) Xeon(TM) 3.00GHz processor, 4 gb of shared RAM memory and 512 kb of cache memory (note that the machine was not dedicated and our algorithm uses only one CPU). The specifications of this machine are quite similar to the server provided by FT, see FAQ of the Roadef 2007 Challenge. Due to the non-deterministic nature of our algorithm the results may slightly vary. To pay attention to this fact, we include the best and the worst result we obtained over 10 runs for every instance sets. All the tests were run with a time limit of 1200 seconds. Moreover, we include a third column that gives the corresponding deviation factor. The mean variation for the full Instance Set B is about 2.3% which is acceptable in this context. The results for Instance Set A and Instance Set B can be found in Table 1.

IA	**best**	**worst**	△%
1	2340	2340	1.0000
2	4755	4755	1.0000
3	11880	11880	1.0000
4	14760	14760	1.0000
5	33480	34740	1.0367
6	22380	22575	1.0087
7	33360	33360	1.0000
8	21180	22320	1.0538
9	30000	30000	1.0000
10	42740	42740	1.0000
∑	216875	219470	1.0100

IB	**best**	**worst**	△%
1	44025	44160	1.0031
2	21240	21240	1.0000
3	20280	21135	1.0422
4	31815	34155	1.0736
5	122760	124320	1.0127
6	37965	38800	1.0220
7	38820	40680	1.0479
8	34440	35520	1.0314
9	33360	33360	1.0000
10	44640	44640	1.0000
∑	429345	438010	1.0233

Table 1. Results for Instance Set A and Instance Set B

For the development of the graph-theoretical functions/framework in our algorithm we used the very powerful Boost C++ framework [2]. The random numbers for the local search were generated using a C++ implementation [3] of the famous Mersenne-Twister-Generator which generates high quality random numbers. Both libraries are freely available.

Last but not least we want to point out that we tried to design an algorithm that does not only emphasize on generating good results within the provided time limit of 1200 seconds but also provides a decent performance within short time. In fact the current implementation allows to obtain quite reasonable results already within 30 seconds (see Table 2).

	Reference Results	
	Instance A	**Instance B**
1	2490	69960
2	4755	34065
3	15840	34095
4	14880	50340
5	41220	150360
6	30090	47595
7	38580	56940
8	26820	51720
9	35600	44640
10	51720	61560

	Results after 30 secs	
	Instance A	**Instance B**
1	4260	46710
2	6240	23100
3	15540	24015
4	14760	46020
5	37740	130080
6	27495	41445
7	36720	41100
8	24480	36840
9	33780	35640
10	49440	46920

Table 2. Reference results provided by FT for Instance Set A and Instance Set B and results for a time limit of 30 secs

References

1. Baker, K.R.: *Workforce Allocation in Cyclical Scheduling Problems: A Survey*, Opl Res. Q. **27** (1976) 155–167.
2. Boost-Library Team: *The Boost C++ libraries*, http://www.boost.org/ (2006).
3. Bedaux, J.: *C++ Mersenne Twister pseudo-random number generator*, http://www.bedaux.net/mtrand/ (2006).
4. ILOG, Inc.: *CPLEX Solver*, http://www.ilog.com (2006).
5. Kolisch, R.; Hartmann, S.: *Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem*, Euro. Journal of Operations Research **127** (2000) 394–407.
6. Kolisch, R.; Hartmann, S.: *Experimental Investigations of Heuristics for Resource-Constrained Project Scheduling: An Update*, Euro. Journal of Operations Research **174**(2006) 23–37.
7. Merkle, D.; Middendorf, M.; Schmeck H.: *Ant colony optimization for resource-constrained project-scheduling*, IEEE Transactions on Evolutionary Computation **6** (2002) 333–346.
8. Tsang, E.; Voudouris, C.: *Fast Local Search and Guided Local Search and Their Application to British Telecom's Workforce Scheduling Problem*, Technical Report CSM-246, Department of Computer Science, University of Essex (1995).