RAIRO-Oper. Res. 43 (2009) 375–386
DOI: 10.1051/ro/2009025

RAIRO Operations Research
www.rairo-ro.org

FRANCE TELECOM WORKFORCE SCHEDULING PROBLEM: A CHALLENGE

Sebastian Pokutta[1] and Gautier Stauffer[2]

Abstract. In this paper, we describe the methodology used to tackle France Telecom workforce scheduling problem (the subject of the Roadef Challenge 2007) and we report the results obtained on the different data sets provided for the competition. Since the problem at hand appears to be NP-hard and due to the high dimensions of the instance sets, we use a two-step heuristical approach. We first devise a problem-tailored heuristic that provides good feasible solutions and then we use a meta-heuristic scheme to improve the current results. The tailored heuristic makes use of sophisticated integer programming models and the corresponding sub-problems are solved using CPLEX while the meta-heuristic framework is a randomized local search algorithm. The approach herein described allowed us to rank 5th in this challenge.

Keywords. Workforce scheduling, schedule generation scheme, roadef challenge.

Mathematics Subject Classification. 90B35, 90B50, 90B70, 90C59.

1. Introduction

One of the keys to success in project management, from the organization of a dinner with class-mates to planning space expeditions, 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

Received June 8, 2007. Accepted June 18, 2009.

[1] Operations Research Center, MIT, 77 Massachusetts Avenue, Cambridge, MA 02139, USA; pokutta@mit.edu

[2] IBM Zurich Research Lab, Säumerstrasse 4, 8803 Rüschlikon, Switzerland; gst@zurich.ibm.com

Article published by EDP Sciences

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. These problems, often referred to as workforce scheduling problems, have received more and more attention in the last years. With the development of highly competitive markets, 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 customers. 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 [11].

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 $\mathcal{I}$ to be scheduled and a set of technicians $\mathcal{E}$ that should resolve the interventions. Each technician has N different skills $\{1, ..., N\}$ and L different levels of competence $\{1, ..., L\}$ for each skill. Moreover the days where employees are off, due to vacations or work conventions, are given. We denote by $D_e \subseteq \mathbb{Z}_+$ the set of days where employee e is available. Each day is subdivided into 120 time units. Each intervention $i \in \mathcal{I}$ is characterized by a duration $d_i \in \{1, ..., 120\}$ and a list of requirements *i.e.* for each skill $n \in \{1, ..., N\}$ and each level $l \in \{1, ..., L\}$, we are given the number of people $R(n, l)$ of level $\geq l$ needed for skill n (note that overqualified technicians can be used for simple tasks). Interventions are related by precedence constraints. For convenience, we define the *precedence graph* of a set of interventions $\mathcal{I}$ as the graph with vertex set $\mathcal{I}$ and arc set $A = \{(i, j) : i, j \in \mathcal{I} \text{ and } i \text{ is a predecessor of } j\}$. Each intervention i can be outsourced at a certain cost c_i but if intervention i is outsourced, all the successors of i have to be outsourced too *i.e.* for all j such that $(i, j) \in A$, j has to be outsourced. FT has a budget B for outsourcing the interventions *i.e.* a feasible set of outsourced interventions $\mathcal{O}$ must satisfy $\sum_{i \in \mathcal{O}} c_i \leq B$. Many interventions require several employees to be combined into *teams* to meet the skill requirements. Since the members of a team usually share a vehicle, they cannot be split during a day. Each intervention $i \in \mathcal{I}$ that is not outsourced has to be scheduled at time $t_i \in \{0, ..., 120 - d_i\}$ of day D_i. Technician cannot process multiple tasks at the same time.

The goal is to minimize the reaction time *i.e.* the schedule horizon but since the interventions have different priorities p 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 *i.e.* if t_p denotes the completion time of the last task of priority p, the objective is to minimize $28.t_1 + 14.t_2 + 4.t_3 + \max_{p=1,\ldots,4} t_p$. Note that France Telecom imposed a time limit of 1200 seconds for the computation.

We refer to the official subject description [4] and the FAQ of the Roadef Challenge 2007 for a more detailed/exact description and for the precise characteristics of the different instances of the challenge. In order for the reader to get a flavor of the dimension of the problem, let us mention that the typical instances have between 100 and 1000 jobs, between 20 and 150 technicians, between 15 and 120 requirements, and between 50 and 500 precedence constraints.

2. Methodology

In this section we want to discuss our approach for solving the problem formulated by France Telecom. It is easy to see that FTWFSP can be formulated as a Resource-Constrained Project-Scheduling Problem (RCPSP) (*cf.* [8]). Unfortunately, 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. One can easily (and naïvely) formulate the FTWFSP as an integer program. Nevertheless, as for most scheduling problems, the natural formulations lead to intractable problems for state-of-the-art solvers (given the dimensionality of the FT instances). Other exact approaches based on constrained programming for instance suffer from the high dimension of the problem as well. We therefore decided to concentrate on heuristics in view of the mentioned bad approximation properties. Regardless of this we do believe that some of the characteristics of the actual instances could lead to interesting approximation results for large subclasses of FTWFSP (*e.g.* limited number of intervention durations). We did not investigate this aspect in detail yet but we believe that some of our heuristics might be good candidates for approximation algorithms in this respect. For a detailed discussion of Resource-Constrained Project-Scheduling Problems and Workforce Scheduling Problems we refer the reader to [1,7,8,10,12].

Kolisch and Hartmann [8] have investigated different heuristics for Resource-Constrained Project-Scheduling Problem. Following their findings we decided to apply a meta-heuristic strategy (described in Sect. 2.2) that builds upon the concept of *activity list representation* and *schedule generation schemes* (described in Sect. 2.1). The activities we consider are daily tasks that we generate upfront by aggregation of interventions to so called *intervention packs* (see Sect. 2.3). 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 Sect. 2.4).

2.1. Activity list representation and schedule generation scheme (SGS)

Each schedule is encoded as an ordered list of tasks (the activity list) and the schedule generation scheme is the decoder function which then transcribes 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 transcribe the activity list into a valid schedule we have to satisfy certain constraints for every insertion. These constraints and their satisfaction (checks) are explained in the following. Checking that the predecessor of a task have already been scheduled is easy. If teams have already been defined, we check if one satisfies the requirements of the new task and its daily capacity is not reached yet. Otherwise in order to assign a new team to a task on day $D \in \mathbb{Z}_+$, we solve a very simple set covering problem (nevertheless we have to solve a lot of them). Let E be the set of employees still available on day D *i.e.* $E = \{e \in \mathcal{E} : D \in D_e\}$. 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 $n \in \{1, \ldots, N\}$ and every level $l \in \{1, \ldots, L\}$, $S_e(n,l) = 0$ if the level for skill n of employee e is less than l and $S_e(n,l) = 1$ otherwise. By $R \in \{0,1\}^{N \times L}$ we denote the characteristic matrix of the intervention's requirements i.e. for all $n \in \{1, \ldots, N\}$ and all $l \in \{1, \ldots, L\}$, $R(n,l)$ is the number of people of level $\geq l$ required for skill n. Now, for each employee e we define a boolean variable x_e to decide if employee e will be part of the team. A first intuitive approach is to minimize the number of people used for this intervention. This problem can be formulated as:

$$\begin{array}{ll} \min & \sum_{e \in E} x_e \\ & \sum_{e \in E} S_e(n,l)\ x_e \geq R(n,l), \forall n \in \{1, ..., N\}, \forall l \in \{1, ..., L\} \\ & x_e \in \{0,1\}, \forall e \in E. \end{array}$$

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. For this purpose we slightly change the formulation above by introducing slack variables for each constraints and adding the sum of the slacks to the objective function with a small cost guaranteeing that we first minimize the number of people. By doing so we try to prevent the 'waste of skills'.

2.2. Local Search on the activity list

The meta-heuristic strategy we use is a randomized local search. This strategy works on the activity list rather than the actual schedule. 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 lists (*i.e.* initial solutions) emphasizing on different characteristics of the Instance Sets. These lists are then passed to the local search. The current implementation creates 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 in the precedence graph *i.e.* order by earliest possible time when no resource constraints are taken into account.

In general, local search techniques suffer from a lot of drawbacks. They neither provide optimality criterions nor (lower) bounds for example. Nevertheless they have proven to be a very effective for improving given (good) starting solutions. Clearly, since we only search locally around a given solution it may be very well that we run into a local optimum which is far away from the global optimum. Especially, when no quality measures like lower bounds (*e.g.* from relaxations) are known it is almost impossible to decide if a resulting solution is a *good* solution. Yet worse, for the considered problem the search space is enormous which makes even an exhaustive local search around a given feasible solution impossible. Therefore the problem characteristics had to be carefully considered while designing the local search.

As already mentioned, the designed local search algorithm performs a search on the activity lists. Those lists are then transcribed into feasible schedules as described in Section 2.1. We start from a given activity list and traverse its *neighbors*. Two activity lists are considered to be *neighbors* if one can be obtained from the other by a transposition *i.e.* we swap two interventions (or the corresponding intervention packs, see Sect. 2.3). 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. Note that even using the activity list given by the optimal solution would not guarantee that we can re-generate this optimal schedule by our algorithm. This would be true only if we can guarantee to assign optimally the technicians to the different tasks but the procedure we defined in Section 2 is a heuristic. In a classical manner we try now to iteratively improve the activity list. 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 a huge number 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 similar to Polya's urns [9] that 'attract' new balls with probability proportional to their filling with balls; sometimes also referred as 'more gains more'. More precisely, we select randomly a neighborhood type over the 4 possible ones (initially with equal probability). Each time a neighborhood types improves the solution we slightly increase its probability of being chosen while slightly decreasing the probability of choosing the three other ones. Thus when a neighborhood type performs better, it tends to be chosen more often. The quality of every generated candidate is checked by actually transcribing the activity list into the corresponding schedule using the SGS. This is one of the most time consuming tasks in our algorithm. Having a quick estimator to reject bad solutions right away might improve the performance significantly.

2.3. 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 members works while the others play belote... 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 measures which emphasize on different aspects. 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 precisely define the similarity measure $sim(i,j)$ between two different tasks i and j as the overhead in the intervention pack. Note that in the final version of the code, the overhead used was the average number of extra persons over the period in consideration *i.e.* if task i requires 2 persons and has duration 30; task j requires 3 persons and has duration 90 and combining the 2 tasks requires 4 persons, $\text{sim}(i,j) := \frac{30.(4-2)+90.(4-3)}{120} = \frac{5}{4}$. 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 the durations of the interventions are most of the time multiples of 15 *i.e.* 15, 30, ..., 120. We use this property of the instance sets and we round up the intervention durations to the closest multiple of 15. Even further we exploit this structure in approximating the packing by iteratively pairing the tasks by similarity. More precisely, we apply a pairing algorithm to the initial list of intervention, we remove the daily tasks (*i.e.* of length 120) thereby created and repeat this procedure twice with the newly created tasks. The reason for iterating this procedure three times is that it is very likely that after each iteration we get rid of the tasks of smallest duration and thus, after the first round mainly tasks with duration ≥ 30 are likely to be left, after the second mainly tasks of length ≥ 60 and finally mainly daily tasks at the end of the procedure.

The pairing problem can be formulated as a follows. Let $1, ..., n$ be the different tasks under consideration. We can define a complete bipartite graph G with vertex set $V(G) = V_l \cup V_r$ with $V_l = \{v_1, ..., v_n\}$ and $V_r = \{v'_1, ..., v'_n\}$. A pairing of the tasks will correspond to a perfect matching in G with the additional constraint that if (v_i, v'_j) is an edge in the matching with $i \neq j$, also (v_j, v'_i) has to be selected. Observe that having edge (v_i, v'_i) in the perfect matching is interpreted as task i not being matched to any other task. We aim to find the best possible pairing with respect to minimizing the total overhead, or equivalently maximizing the utilization. Thus we need to define a cost function c on the edges of G which resembles this objective. For all $i \neq j \in \{1, ..., n\}$, we put $c(v_i, v'_j) = \text{sim}(i,j)$ and for all $i = 1, ..., n$, we set $c(v_i, v'_i) = p(i)$. The term $p(i)$ is a penalty function chosen to be big enough in order to ensure that i is paired with itself (*i.e.* not paired with any another task) only when necessary. In the algorithm we used the average overhead over the full day period as the penalty *i.e.* if task i requires 3 persons and has duration 90, $p_i = \frac{0.90+3.30}{120} = \frac{3}{4}$. The problem can be formulated as the integer program:

$$
\begin{array}{rrcl}
\min & \sum_{u \in V_l, v \in V_r} c(u,v)\ x(u,v) & & \\
& \sum_{v \in V_r} x(u,v) & = & 1, \forall u \in V_l \\
& \sum_{u \in V_l} x(u,v) & = & 1, \forall v \in V_r \\
& x(v_i, v'_j) - x(v_j, v'_i) & = & 0, \forall i \neq j \\
& x(u,v) & \in & \{0,1\}, \forall u \in V_l, v \in V_r.
\end{array}
$$

We also realized that this problem could be formulated as a classical perfect matching problem in the graph $G' = (V, E)$ with vertex set $V = V_l \cup V_r$ with again $V_l = \{v_1, ..., v_n\}$ and $V_r = \{v'_1, ..., v'_n\}$. In contrast to the formulation from above, $G'[V_l]$ and $G'[V_r]$ are complete graphs now and $\{(u, v) \in E : u \in V_l, v \in V_r) = \{(v_i, v'_i), i = 1, ..., n\}$. We can define a cost function c on the edges of G' as $c(v_i, v_j) = c(v'_i, v'_j) = \frac{sim(i,j)}{2}$ for all $i \neq j \in \{1, ..., n\}$ and $c(v_i, v'_i) = p(i)$ for all $i = 1, ..., n$. It is easy to see that we can assume without loss of generality that a minimum cost perfect matching M in G' will have the property that if $(v_i, v_j) \in M$, then $(v'_i, v'_j) \in M$. Thus a minimum cost perfect matching can directly be interpreted as a pairing. It follows that this simple pairing problem can be solved in polynomial time.

Despite of this, the actual problem which has to be solved here is more complicated and we thus keep the integer formulation presented above (for its nice flow-like structure) to which we will add additional constraints. Indeed, not all pairings are feasible since the given precedence constraints have to be satisfied. Thus 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 precedence cycles or not. In order to detect cycles, we simply create a new precedence graph with the current pairing as the set of interventions (equivalently, we "shrink" the different pairs of interventions to *pseudo-vertices* in the original precedence graph) and we use an adaptation of a deep first search algorithm to detect cycles. If the pseudo-vertices $1, ..., k$ form a cycle, and pseudo-vertex $l \in \{1, ..., k\}$ corresponds to a pair of intervention $\{i_l, j_l\}$, we simply add the inequality $\sum_{l=1}^{k} x(v_{i_l}, v'_{j_l}) \leq k-1$ (those simple inequalities are closely related to the so-called *cover inequalities* for the knapsack problem). 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 (1200 s), we decided to go a step further and implement an approximate algorithm for the above packing problem by subdividing the set of interventions into smaller pools and using the pairing algorithm on those pools separately. Although we restrict the possible pairing capabilities which, on the one hand, may affect optimality, we guarantee on the other hand that the pairing problems involved are practically tractable. The maximum size of the pairing problem which is considered to be tractable was estimated by statistical analysis and obviously depends on the performance of the machine.

Unfortunately, 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. The displayed precedence graph here is arising from instance set B6 and a naïve pairing of interventions.

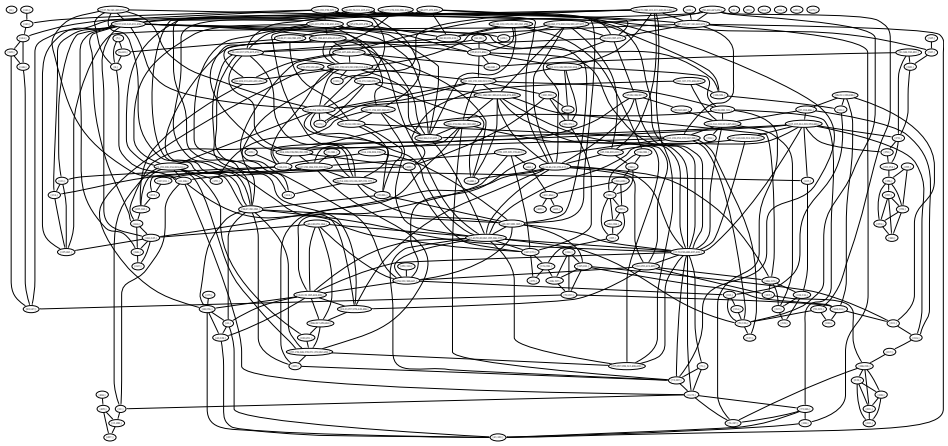

FIGURE 1. The precedence graph of Instance Set B6 for a naïve packing.

As one might guess, these complicated precedence graphs do impose two major problems. First, as already mentioned, it may impose an artificial lower bound on the makespan which in turn 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 on the activity list are needed to actually change the schedule and thus many changes to the activity lists may remain useless as they violate a precedence constraint and hence are cancelled by the SGS later (see Sect. 2.2). 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 evaluate, at each round of the iterative pairing (remember we have three rounds of pairing), the quality of the current packing by using the SGS procedure. We keep the best round of pairing obtained as the starting point for our local search strategy.

2.4. 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 with the highest costs are outsourced which is done by removing them from the list of interventions we have to schedule. This approach is inspired by the fact that the optimal solution to the linear relaxation of a knapsack problem is obtained by greedily selecting objects with maximum ratio $\frac{\text{profit}}{\text{cost}}$ (note that a budget constraint is a knapsack constraint). In the algorithm, we rank the interventions that are

candidates for outsourcing with respect to their priority and then by the ratio $\frac{\text{number of resources needed}}{\text{cost of intervention}}$. For the sake of simplicity, we only considered intervention without successors as candidate for outsourcing.

It is clear that we could have improved this simple heuristic by outsourcing the tasks on the fly but our time in view of the challenge deadline was limited so that we could not implement such a solution.

3. Computational results

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.

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 Section 3.1 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 is not multi-processor aware). The results presented in Section 3.2 were computed on the server provided by FT whose specifications are quite similar see FAQ of the Roadef 2007 Challenge for details.

All the experiments were run with a time limit of 1200 seconds. For details about the different instance sets, the reader is referred to the Roadef Challenge 2007 webpage.

3.1. Qualification phase

In the following we want to present our computational results for the instance sets A and B provided by FT for the qualification phase. We also report the reference results originally released by FT. 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. 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.

3.2. Final results

In Table 2, we report the final results officially announced by the head of the Roadef Challenge 2007 for Instance Set X. We compare it to the overall best solution obtained by the different teams and the solution obtained by the winner of the challenge: Cor Hurkens.

TABLE 1. Results for Instance Set A, Instance Set B and reference results.

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

	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

TABLE 2. Final results for Instance Set X.

IX	Our algorithm	Hurkens	Overall best
1	197550	151140	151140
2	15780	9120	7260
3	59160	50400	50040
4	75720	65400	65400
5	194700	147000	147000
6	13080	10320	9480
7	54120	33240	33240
8	33120	23640	23640
9	171480	134760	134760
10	173760	137040	137040

Those results allowed us to rank 5th in the Roadef Challenge 2007. Although we could have tried to work further on this problem to improve our approach and the resulting performance we decided not to follow up since we believe the framework proposed by the winner of the challenge [5] is better suited.

Acknowledgements. 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) while the two authors were post-doctoral fellows at the Massachusetts Institute of Technology, Cambridge, US.

References

[1] K.R. Baker, Workforce allocation in cyclical scheduling problems: a survey. *Oper. Res. Q.* **27** (1976) 155–167.
[2] Boost-Library Team, The Boost C++ libraries. http://www.boost.org/ (2006).
[3] J. Bedaux, C++ Mersenne Twister pseudo-random number generator, http://www.bedaux.net/mtrand/ (2006).
[4] P.-F. Dutot, A. Laugier and A.-M. Bustos, Technicians and interventions scheduling for telecommunications. http://www.g-scop.inpg.fr/ChallengeROADEF2007/en/sujet/sujet2.pdf (2006).
[5] C. Hurkens, Incorporating the strength of MIP modeling in Schedule construction. http://www.g-scop.inpg.fr/ChallengeROADEF2007/TEAMS/roadef28/abstract_roadef28.pdf (2007).
[6] ILOG Inc., CPLEX Solver. http://www.ilog.com (2006).
[7] R. Kolisch and S. Hartmann, Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **127** (2000) 394–407.
[8] R. Kolisch and S. Hartmann, Experimental investigations of heuristics for resource-constrained project scheduling: an update. *Eur. J. Oper. Res.* **174** (2006) 23–37.
[9] H. Mahmoud, *Pólya Urn Models*. Taylor & Francis Group LLC – CRC Press (2008).
[10] D. Merkle, M. Middendorf and H. Schmeck, Ant colony optimization for resource-constrained project-scheduling, *IEEE Trans. Evol. Comput.* **6** (2002) 333–346.
[11] http://www.g-scop.inpg.fr/ChallengeROADEF2007/
[12] E. Tsang and C. Voudouris, *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, UK (1995).