Local search for the ROADEF 2007 challenge

I. Dotu[1], A. del Val[1], et P. van Hnetenryck[2]

[1] Departamento de Ingenierï£¡ Informï£¡ica, Escuela Politï£¡nica Superior, Universidad Autï£¡oma de Madrid, 28049 Madrid
{ivan.dotu,alvaro.delval}@uam.es
[2] Department of Computer Science, Brown University, Box 1910 Providence, RI 02912, USA
pvh@cs.brown.edu

1 Introduction

The ROADEF 2007 challenge involves the scheduling of a set of technicians, of varying technical ability in various domains of expertise, to handle a set of "interventions". A number of constraints are placed on these interventions, such as precedence relations, priorities, and in particular technical requirements of each task. The goal is to optimize the schedule according to a given cost function based on the times of completion of the tasks of each priority level. The challenge is described in detail in `http ://gilco.inpg.fr/ChallengeROADEF2007/en/sujet/sujet2.pdf`, so we won't go further into details. Perhaps what makes it more distinctive when compared to other traditional scheduling tasks such as job shop scheduling is the complex and discrete nature of the resources required to accomplish the tasks. Namely, each task requires as a resource a *team* of technicians with the appropriate technical skills. Such teams are not given ; instead, they must be assembled by the solver according to the given constraints. Once formed, a team is fixed for the whole day, so it is important that they can be used for various tasks, as most of them take less than a day. The right choice of teams appears crucial to obtaining more optimal schedules.

Our approach is based on randomized local search, and is rather simple conceptually, though the implementation quickly becomes complex. After an initialization phase that provides an initial solution that is already of some quality, guided by several heuristic criteria, a number of local search movements are performed, trying to improve the solution. Both initialization and local movements are repeated as long as time permits to include randomized elements, specially in the choice of teams. We describe these steps in more detail in the following sections. As with other local search procedures, our method is not complete and is not guaranteed to find an optimal solution. However, it obtains what appear to be quite good solutions in a very short time (though for the challenge we just let the program run for as long as the timeout allows).

2 The program

The main flow of the program is, as said, very simple. Initialization begins by selecting an order in which the task are going to be assigned. We describe later the three heuristic methods we use to generate different orders, and how precedences and priorities are handled in this phase. Once an order is selected, each intervention is assigned in order in the first available spot (or "gap") in the schedule. The notion of an available "gap" (for a given intervention) depends on the availability of technicians. These may be available as part of an already formed team, that has been already assigned to some other intervention but has part of the day without assigned work ; or they can be picked from those technicians that have not been assigned to any team in the given day. Note also that many different tasks may be performed in parallel (by different teams), so though interventions are assigned in the order computed in the first step, this order does not generally match the temporal order in which tasks are performed in the resulting schedule.

There are a number of aspects to be considered in choosing a team for a task, that are described later. But from the point of view of the main control flow of the program, what matters most is that we randomize the choice of available technicians. This means that different iterations of the program make teams up in different ways.

Thus we have in fact three different ways of initializing the order in which the program assigns the tasks, and for each of them, many different ways to choose the teams by introducing this random element. Each combination yields an initial schedule, and the program simply keeps generating

such initial schedules, and then trying to optimize them by applying local search movements, as described below, until timeout.

3 Initialization

Selecting the order in which tasks are to be assigned is done separately for tasks with no predecessors or succesors and for tasks in the precedence graph (usually a tree). The later are sorted topologically, so that predecessors are assigned before successors. The former are sorted according to one of three criteria :

- sort by priorities only
- sort by priorities plus task duration (longest tasks first)
- sort by priorities plus number of required technicians (larger sets first)

Since no criteria dominates in terms of the quality of solutions provided, we try them all in each iteration.

Both orderings are then merged. The idea is that tasks in the precedence tree should be assigned earlier (thus being also scheduled earlier, typically), so that they cannot push back their succesors too much in time. On the other hand, we shouldn't schedule a low priority task very early as it may push back a higher priority task to later days by using up the available technicians. So the merge is done in a stratified way, i.e. all higher priority tasks can be placed before lower priority tasks.

Once the ordering is built, the program attempts to assign the tasks in order as described above. We already mentioned the aleatorization in the choice of technicians while we build a team (pick available technician which provides one of the skills required, until all the skills are fulfilled). Other aspects that are considered are : the possibility of reusing a team that is already formed for the day; and the possibility of reducing the size of a team once is formed if we discover that some of the technicians added later to the team make the skills of a previously added technician redundant (which opens the possibility of removing the technician from the team, thus making him available for other teams and tasks). These checks for redundancy in the teams are also performed during the local search phase.

4 Superflous Technicians

Since we assign technicians to teams incrementally in order to fit the requirements of a given intervention, some of the first technicians assigned can be superflous. This means that we can remove them from the team without affecting the correctness of the schedule.

After the initial schedule is created we check for superflous technicians. For each day we create a new team with all superflous technicians from other teams. This obviously considering the availability of each technician.

After each local move (described below) we apply a reduced check for superflous technicians. The procedure is similar to the one previously explained but in this case we only consider the teams related to the move (Team where the intervention was before the move and team where the intervention will be after).

5 Local search movements

Once an initial schedule is built, we attempt to improve it by doing local changes, until no further improvement is possible (and a new iteration begins). We distinguish between intraday changes and changes that can move tasks from one day to another. The reason for this distinction is that teams must be fixed for the whoile day, as said. The intraday changes that can be made are the following :

- remove redundant technicians, and try to use them for additional tasks within the day
- swap taks between teams
- swap order of tasks assigned to a team

A more general kind of movement is based on the notion of "gaps". A gap is just an idle period for a given set of technicians, that may be filled by some task. We also consider "semigaps", relative to a given task. By this we mean periods of work that are filled by tasks of lower priority than the task that we are trying to reassign. Swaping both tasks can in this case be beneficial, specially as the cost function penalizes heavily the ending time for tasks of higher priority. Our approach is to proceed in trying to make these changes in order of decreasing priority, from the latest scheduled task at each priority. Thus, we start from the latest task of priority 1 trying to find gaps or semigaps earlier in the schedule, continue with other tasks of priority 1, etc. After each move we check if some of the technicians related to the intervention(s) in the move are now superflous.

Each of these movements may create other gaps, for example by liberating technicians, or simply in the place where the moved task was before the movement. However, the implementation was complicated by factors like the need to coalesce contiguous gaps that may appear due to various movements or to create new gaps when one is only partially filled by a movement, the need to reevaluate the relevance of technicians after each movement, and other decisions as to when to reuse a gap. etc. Thus, while we believe that our approach is a promising way to get reasonably good solutions fast, it is certainly work still in progress.

6 Future Work

Future work is closely related to the decisions mentioned in the last paragraph in the previous section. Only a reduced number of "move-to-gap" situations have been considered, but there are several others such as : swap interventions of different duration if there are free gaps that can supply the extra time needed, move interventions closer so there are no gaps between two interventions after an in between intervention has been move, etc.

It would be also interesting to study ways to move interventions that violate constraints if there is a repairing strategy to apply afterwards. Note that we only allow feasible moves in our approach.

7 Results

The following table shows the results in cost that we obtained by running our program for 20 minutes. With the exception of data1 in data set A (and this can be fixed, but we couldn't do it in time for the deadline), all our results improve on the results provided by the organizers of the challenge. We remark also that we can obtain at least some improvement over the provided results quite early in the run, though allowing more time does provide more opportunity for optimization.

Tab. 1. Experimental results (timeout 20 minutes)

Data set A	Best cost found	Data set B	Best cost found
data1	2550	data1	58440
data2	4755	data2	25920
data3	15540	data3	29115
data4	14040	data4	41475
data5	34740	data5	135120
data6	25815	data6	41970
data7	36480	data7	48780
data8	23850	data8	47160
data9	33960	data9	40320
data10	46080	data10	53880