SUBCONTRACTORS SCHEDULING ON RESIDENTIAL BUILDINGS CONSTRUCTION SITES

Thierry Benoist, Antoine Jeanjean, Guillaume Rochart
e-lab – Bouygues SA – 1 av. Eugène Freyssinet
F-78061 St Quentin en Yvelines, France
{*tbenoist, ajeanjean, grochart*}*@bouygues.com*

Hadrien Cambazard, Émilie Grellier, Narendra Jussien
École des Mines de Nantes – LINA CNRS 2729 – 4 rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3, France
{*narendra.jussien, emilie.grellier, hadrien.cambazard*}*@emn.fr*

Abstract

Erecting a residential construction is a complex project which implies many actors and strong deadline requirements. The aim of this paper is to present a case study about schedules of subcontractor's tasks on residential buildings. Directly after the end of structural works, start not only tasks like electricity, plumbing, cover, water circuit installation, but also paintings, furniture, wallpapers, etc. which are often executed by subcontractors. Solving this problem consists in finding for each elementary task its starting date, its ending date and its volume executed each day between these two dates. A constraint programming based solution is presented in this paper.

1. Introduction

Bouygues Habitat Résidentiel is a subsidiary of Bouygues Construction specialized in the construction of private residential buildings. Its methods department is focused on organization, scheduling and optimization. It must deal with teams and resources within the framework of a tight planning to set up: the edifice must be delivered on time. Erecting a residential construction is a complex project which implies many actors and offers multiple opportunities to implement scientific solutions of scheduling.

Bouygues' *Direction of new technologies*, e-lab, has been working for several years with the Methods department of Bouygues Habitat to develop decision-support solutions in building sites scheduling. The operational researcher must accompany decision makers in their response to complexity. The goal is to reduce global cost by optimizing the schedule. Engineers from methods department use these softwares to plan and follow the construction project at each stage. At the beginning of the project, they estimate the global charge in terms of working days per resource and build a Gantt chart. During the actual construction, they change the planning to fit with the real life situation.

The aim of this paper is to present a case study concerning schedules of subcontractor's tasks on residential buildings. Directly after the end of structural works, start not only tasks like electricity, plumbing, cover, water circuit installation, but also paintings, furniture, wallpapers... All of these tasks are included in the scope of the study at that time. All tasks executed outside the building are not taken into account (like earthworks, roof, sealing ...). Solving this problem consists in finding for each elementary task its starting date, its ending date and its volume executed each day between these two dates. The main objective is to minimize the capacity overload for each resource (represented by subcontractors). Subsidiary criteria are to minimize the number of breaks in their schedules and the global makespan (in the later the makespan will be considered as a hard constraint). A set of precedences, occupancy and rank constraints have to be respected.

We first define this Resource Constrained Project Scheduling Problem (RCPSP) in the next section. This description is enriched with a constraint formulation in Section 3. Section 4 explains how a pattern recognition could help saving some time during the assignment of tasks in our greedy algorithm. Section 5 presents some first results. Finally, we show in Section 6 that using a special feature of constraint programming, *explanations*, would help the decision makers to set the system.

2. Problem description

2.1 The scheduling of subconstractor Model

2.1.1 Variables description

Let us consider a building composed of M apartments, and involving K subcontractors. This construction has to be erected in less than T days. To each apartment is assigned a set of tasks to be performed (plumbing, electricity, painting, wallpapers, tiling and so on). Each task is given a unique index i and domains for associated start, end and duration variables are respectively (S_i^{min},S_i^{max}), (E_i^{min},E_i^{max})

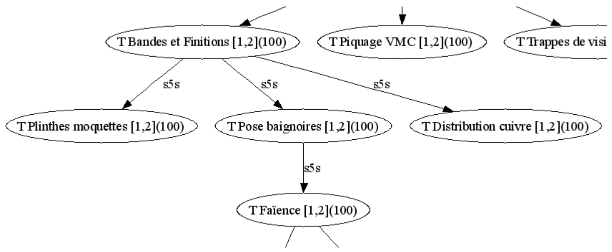
T Bandes et Finitions [1,2](100)
T Piquage VMC [1,2](100)
T Trappes de visi
s5s
s5s
s5s
T Plinthes moquettes [1,2](100)
T Pose baignoires [1,2](100)
T Distribution cuivre [1,2](100)
s5s
T Faïence [1,2](100)

Fig. 1 Precedence graph

and (D_i^{min},$D_i^2 max$). The total number of tasks on the site is N. The set of tasks for an apartment j (denoted by $A(j)$) is subject to generalized precedence constraints defined by a precedence graph $G(V,E)$ (see figure 1 for an example).

2.1.2 A constraint model

The objective function is:

$$\min \sum_{\substack{k<K \\ t<T}} d_{kt} \qquad (1)$$

In addition to precedence constraints, the following constraints are set:

$$\forall\, i < N \quad D_i^{min} \leq E_i - S_i \leq D_i^{max} \qquad (2)$$

$$\forall\, k < K, (i,j) \in R(k)^2, L_j < L_i - W_k,\; s_i > e_j + W_k \qquad (3)$$

$$\forall\, (X,Y,\delta) \in P \quad X \geq Y + \delta \qquad (4)$$

$$\forall\, i < N, t < T\; (t < s_i \vee t \geq e_i) \Rightarrow (c_{it} = 0 \wedge x_{it} = 0) \qquad (5)$$

$$\forall\, j < M, t < T \quad \sum_{i \in A(j)} F_i x_{it} \leq Z_j \qquad (6)$$

$$\forall\, i < N \quad \sum_{t<T} c_{it} = V_i \qquad (7)$$

$$\forall\, k < K, t < T \sum_{i \in R(k)} c_{it} \leq B_k + d_{kt} \qquad (8)$$

$$max_{i<N}(e_i) < makespan \qquad (9)$$

2.1.3 A constraint description

Generalized precedences correspond to *Task A must start/end after/before the start/end of task B plus δ days i.e.* $X \geq Y + \delta$ where X and Y stand for start or end variables of different tasks and δ is a (not necessarily positive) integer. They are defined via a set P of triples (X,Y,δ) where X and Y denotes starts or ends of two different tasks (see constraint 4). Moreover, the (weighted) number of simultaneous tasks processed in the apartment is limited: this limit (denoted Z_j) depends on the size of apartment j, and the occupancy coefficient (F_i) attached to each task is usually either 1 or Z_j (some tasks are incompatible with all others) (see constraint 6).

Each task uses a single resource k, consuming a total amount V_i of this resource. Depending on the resource, this value can be expressed in square meters (*30 m^2 of tiling*) or in abstract unit (*one shower*). This consumption can be dispatched freely among the processing days of the task with a maximum daily consumption bounded by some C_i^{max}. The capacity of resource k is denoted by B_k (constraint 8). This limit can be exceeded but the objective function for the problem is to minimize the sum of these excesses (see Figure 2).

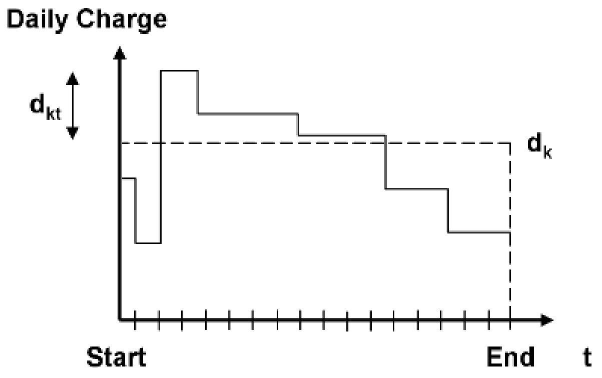

Fig. 2 Global daily charge of work for a resource k

Tasks consuming the same resource k (denoted by $R(k)$) are subject to so called rank constraints, stating that workers of the same resource cannot be dispatched in too distant areas in the construction site. More precisely each apartment j is given a rank L_j (usually equal to the storey of the apartment) and a maximum width W_k is given for each resource: tasks of a given rank r cannot start before all tasks (of the same resource k) of ranks smaller than $r - W_k$ are completed. Those constraints correspond to precedence constraints (see constraint 3).

To summarize, the following integer variables are used to defined the problem:

- $\forall \quad i < N : s_i$ and e_i are the start and end variables of task i such that $S_i^{min} \leq s_i \leq S_i^{max}$ and $E_i^{min} \leq e_i \leq E_i^{max}$.
- $\forall \quad k < K,\, t < T : d_{kt}$ is the overload for resource k on day t with $d_{kt} \geq 0$.
- $\forall \quad k < K,\, i \in R(k),\, t < T : c_{it}$ is the consumption of resource k by task i on day t so that $0 \leq c_{it} \leq C_i^{max}$ (c_{it} does not depend on k as each task is associated to a single resource), and $x_{it} \in \{0,1\}$ is equal to 1 if task i is active on day t (0 otherwise).

2.2 Task Volume management

Each resource works on zone for executing a specific task corresponding to a quantity of work in adequacy with the power of the associated resource. The originality of the model is to consider here task as a volume instead of a length *i.e.* The limitation on the resource is not anymore *we dispose of 3 painters* but *we are able to make a maximum of 90 m^2 a day*. For a given task, it is then necessary to determine not only its starting date and its completion date but also the distribution of its usage over the selected days:

For example, let us consider the following Problem 1 with 3 tasks:

- T_1: starting day 2, ending day 2, surface = 30 m^2

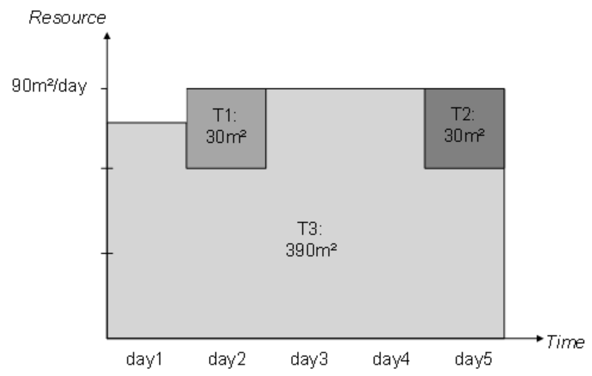

Fig. 3 A solution for the introduction problem ($75m^2$+$60m^2$+$90m^2$+$90m^2$+$60m^2$).

- T_2: starting day 5, ending day 5, surface = 30 m^2
- T_3: starting day 1, ending day 5, surface = 375 m^2

In this instance, the resource is painting and the daily maximum is 90 m^2 per day. A solution for this problem consists in distributing surfaces of T_3 over its 5 days of execution (see Figure 3). This peculiar way of considering the workload attached to each task allows reasoning at the scale of days while avoiding rounding problems. Indeed, specifying starting times in hours for instance would be costly and meaningless since delays in the precedence graph usually represent drying duration (and count the required number of nights between two tasks). On the contrary if all tasks taking 1.5 days for one worker were considered as utilizing this worker for 2 full days, resource consumption would be overestimated.

The resource constraint would not be expressed any more as human beings but in m^2. The min/max durations imposed on the resources have also an impact on daily volume: indeed, the smaller the duration, the larger the average daily resource consumption.

3. Constraints formulation

The model presented in section 2 is solved with the `choco` constraint solver by constraint programming implemented with Choco[1]. To respect precedences and rank constraints, a PERT (Malcolm et al. 1959) is set up using inequality constraints. For each resource, a flow constraint ensures capacities of the subcontractor. Finally, to be sure that occupancy will be respected on each apartment, a cumulative constraint is implemented.

3.1 Precedence constraints

Some precedence constraints are imposed on the model: within an apartment, some tasks must comply with precedence rules (for example, it could be more judicious to install the bath-tub before posing earthenware). The graph in Figure 1 represents a portion of the graph we can set using precedence constraint provided by methods office. This constraint is modeled using a set of simple inequalities between starting / ending dates.

[1]choco-solver.net

In addition, by assigning a rank to each stage, an approximate order of realization can be specified by the user. For each subcontractor, no task from the stages of rank N can start before all the tasks of the stage of $N-x$ rank are not completed (x would be the number of ranks which one can open at the same time).

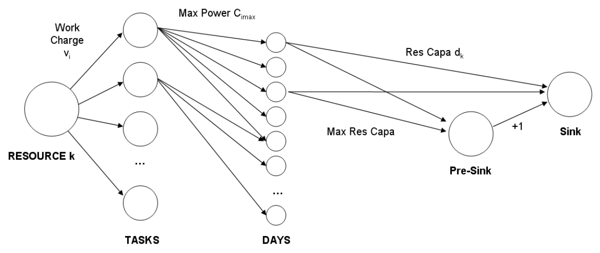

Fig. 4 The flow

3.2 Flow constraints

For each resource of the problem (most of our instances concern studies with approximately 50 different resources), we can set up a flow constraint (Gaudin et al. 2004) regarding the ressource consumption. We know the volume required by each task of the resource. All of these tasks have a certain domain, meaning a time window where it can be executed. Therefore, the amount of flow going through an edge (i,j) between a task node i and a day node j corresponds to a c_{ij} variable (the volume of task i completed on day j). The total flow on days should not exceed the capacity of the resource B_k. If it does, we configure a pre-sink that will accept the excess of flow coming from these days. The aim of the flow constraint is to measure the overload of the ressource capacity. This overload has to be minimized.

Such a flow helps to reduce the time windows of tasks: the flow between a task and the days it is not realized will be null (ensured by constraint 5). It is really important to decrease the idle time of the subcontractor on the building site. If one of them has important variation in the volume of work each day, it will have difficulties to decide how many teams to assign to this site and its efficiency could be affected. It is necessary to avoid inactivity days for each subcontractor. This constraint optimizes the total work of this resource for each day.

3.3 The cumulative constraint

In addition to resource constraints, limited space in each zone implies that only few subcontractors can work in the same place at the same time. In order to model this constraint, a space capacity is associated to each zone and a space occupancy is associated to each subcontractor (the F_i constants in the model of section 2).

With this model, a `cumulative` constraint (Beldiceanu and Carlsson 2002) is used in order to ensure that at each instant, tasks affectation respect this rule.

4. Resolution

Section 3 introduced a constraint model including flow, precedences, ranks and occupancy constraints. The default complete tree search resolution provided by constraint solvers could not scale up for our problem. It is indeed impossible to apply the previous model on a problem of 2500 tasks scattered on fifty different areas. Moreover we were not interested by an optimal solution but a good solution within a 30 seconds time limit. The constraint model is however very valuable for its propagation abilities. We overcome this difficulty by solving the problem with a heuristic approach implemented in two steps while keeping advantage of the strong propagation mechanisms of the flow, cumulative, and pert-related constraints:

- First of all, identical patterns of tasks on the different apartments are identified and the corresponding subproblems are solved to optimality (section 4.1 explains how these patterns are used to decrease computational time).
- Optimal solutions of previous sub-problems are used to extract a set of precedence constraints which can replace the cumulative constraint. The cumulative is removed from the complete formulation (to scale up) and the model is then solved by a greedy approach (with propagation) and a pragmatic management of contradiction (section 4.2).

4.1 Pattern search

Using pattern recognition helps the solver to save time during propagation. A pattern is a sequence of tasks. The similarity of two patterns is defined according few criteria. First of all, both patterns of elementary tasks have to be defined on zones with the same occupancy capacity. Secondly, two corresponding tasks on two similar patterns have to be similar. How the similarity is defined between two tasks? The minimum and the maximum amounts of daily work have to be the same. Futhermore, theirs parents and children have to be similar.

In the exemple presented on figure 5, Pattern 1 and Pattern 2 will be similar if they represent a sequence of tasks executed on two apartment of the same capacity and each couple of tasks are similar. For instance, task 4 and task 10 are similar, if they respect the work extrema constraint, if Task 3 and 10 are similar and if couples of tasks 5-11 and 6-12 are similar.

When this pattern recognition has been processed, we are ready to set up a smaller model on a specific zone. We select a zone, a pattern of tasks and we solve as many prob-

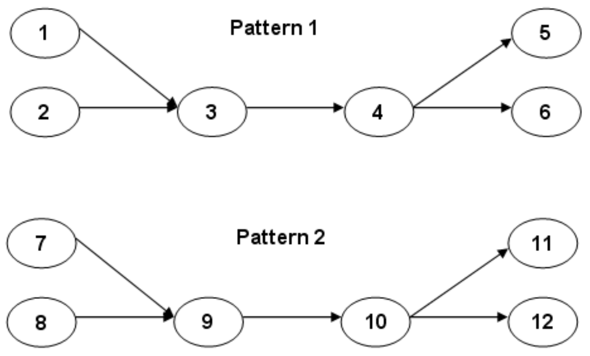

Fig. 5 Pattern 1 and 2 will be similar if each task are similar and if they are defined on zones with the same capacity

lems as number of zones. The strategy consists in adding a cumulative constraint to the set of constraints already effective. It ensures the respect of occupancy on each apartment. These smaller problems are much smaller than the precedent global one: 50 tasks on average to schedule on one zone. Then, solving them to optimality is possible.

Which information can be extracted from these resolutions? Scheduling a pattern of tasks on a zone by focusing the attention only on the occupancy generates some new precedence constraints. In the global problem, we add the the corresponding set of constraints for each similar pattern. Propagation will make new domain cuts: this first step in our greedy algorithm is an important diminution of computational time.

4.2 Pragmatic filtering

As explained in section 2.1.1, the operational context led to a model where the capacity of a resource can be exceeded. Precedence constraints are strict on a construction site (shortening delays between tasks may result in quality or safety defects). On the contrary the workforce can be adjusted through the use of temp workers or extra working hours. Such adjustments are costly but often feasible. From the customer point of view a solution with overloaded resources on a few days is more informative than a *No solution* pop-up. On the contrary suggesting a solution with wallpaper tasks preceding plaster application would make no sense. The main advantage of our model from an operational point of view is that as soon as the set of preferences is consistent (including pattern preferences defined in the previous section), a solution can easily be computed. For instance earliest scheduling may exceed resources capacity on many days but it is still a (costly) feasible solution. This specificity allows designing a robust greedy algorithm whose objective is to find a solution minimizing resource consumption excesses.

However this choice degrades the efficiency of a constraint programming approach, since resources limitations are not strict, hence cannot propagate. Even after a first solution has been found, propagating the maximum excess remains inefficient because it induces no limitation on any specific day.

A first idea would consist in solving the problem with increasing resource capacities until a solution is found. However in addition to requiring several solving procedure (hence increasing total solving time), this approach would yield poor results since failures of the first step would give no hint about which resource capacity should be increased, on which days, and by which amount. Finally we would probably need to uniformly increase resource capacities, missing our objective of highlighting *difficult* days.

In order to overcome this intrinsic limitation of our model we introduce the notion pragmatic filtering. The principle of this heuristic approach is to consider our relaxation of resources capacity constraints as a mere error recovering strategy. The problem that we actually want to solve is the one with resources limitations enabled, and allowing resources overload is merely a way to make sure that a (possibly degraded) solution will be returned even when this original problem is too difficult to be solved in 30 seconds (if not impossible). A pragmatic filtering algorithm is a classical filtering algorithm for a variable or a set of variables implemented as follows.

```
let $D_1, \ldots, D_n :=$ filteredDomains($V_1, \ldots, V_n$)
if ($\exists i, D_i = \emptyset$) forgetIt()
else for $i$ in ($1..n$) domain($V_i$) $:= D_i$
```

In our case we implemented a function `filteredDomains`(S_i) computing the earliest start (S_i^{min}) that would be computed for task i if its resource was actually limited. This filtering is classically based on the current consumption profile for this resource and on the height of task i. If it does not empty the domain of S_i we are sure that *committing* these changes cannot make the problem infeasible, since restricting a single domain in a consistent PERT cannot make the PERT infeasible. On the contrary if it empties the domain of S_i it means that no start date for task i can avoid generating resource overload: instead of just ignoring the filtering, we set S_i to the value generating the smallest amount of resource overload (`forgetIt`()). We also implemented the symmetrical function refining the latest end of task i (E_i).

Contrary to real filtering the result may depend on ordering issues. Consider, for example, tasks i and j (both with duration = 1, consumption = 10) using the same resource R, with remaining capacity only on two days: 10 on day t_1 and 5 on day t_2. The first filtered task will see its domain restricted to day t_1 (successful pragmatic filtering) and the second one will be assigned to day t_2 (minimum overload day).

The advantage of this approach is that it performs efficient filtering as if resource limitation were enabled while avoiding generating contradictions for the few days on which the constraint cannot be satisfied.

5. Results

The algorithm has already been used to solve a real life problem. One of the instance was a residential building of 65 apartments. On each of them, 38 different elementary tasks were scheduled. These 2470 tasks had to be organized in a schedule of 130 days. We found 3 different patterns of 38 tasks. The software provides a solution of this problem in less than one minute. The user is able to establish a Gantt chart for the whole planning horizon, with a starting/ending date per task and the volume executed each day. We will see in section 6 that explanations is also an important part of the results that the user is expecting.

6. Explanations to help decision makers

We focused in this article on a way to find solutions to the described problem: planning subcontractor tasks. However, final users do not only need to find these solutions. Designing an interactive application is mandatory since the user does not know how the solving is working and cannot accept an application that only answers no if no solution is found or which cannot incrementally modify the description on order to catch problem evolutions.

6.1 Explanations

An important point is the need of explanations. Indeed, even when the problem is static, the user (the building site coordinator) needs to know why there is no solution or why the solution is not the expected one (for instance to prove to his hierarchy that he really need more time than the expected one).

This is another feature which can benefit from explanation-based constraint programming (as its name suggests) (Jussien 2003). Explanations (search and filtering-related information) stored on the fly (while solving the problem) can give some valuable information to the user to know why the achieved state is not valid and so why in the given time, the algorithm did not find valid solutions.

6.2 Interactive solving

Another need is interactivity. Indeed, planning must be easily updatable, since a subcontractor can be late for a given task, bad weather can disturb the planning and so on.

However whenever the problem changes, the user do not want to search a completely new solution: planning modifications can be expensive in terms of organization, team managements and so on. This implies that the problem needs to be resolved dynamically: when a constraint changes, only dependant instantiation should be modified.

As shown recently, explanation-based constraint programming (Elkhyari et al. 2004) is a good candidate for such a task. The idea is to maintain search-related information in order to be able to quickly determine what can be kept as is when handling a modification of the problem. This technique will be applied in our context.

6.3 Explanations for the PERT constraint

Our constraint solver, `choco`, provides some explanation capabilities thanks to the `palm` extension. All the constraints used in our model do possess an explained version. However, in order to improve the interest of the explanations provided to the user, we plan to develop a specific global constraint for handling the precedence graph of the problem. Indeed, it is important to encapsulated as much as possible global information into constraints in order to provide valuable explanations (Gaudin et al. 2004).

We are currently developing such a PERT constraint based on incremental versions of the well-known pert resolution algorithms (Beldiceanu et al. 2005).

7. Conclusion

We presented in this paper a real-life problem encountered in residence buildings construction sites. The problem of scheduling subcontractors is a difficult yet interesting task. We sketched out in this paper our solution that meets the tight time limits set by the final user for solving the problem.

Has often in designing decision-support systems, meeting the original requirements of the final user is not sufficient. Providing new and efficient optimization tools introduces new needs: interactive solving, explanations, etc. We are therefore currently working on these additions to the original problem.

References

N. Beldiceanu and M. Carlsson. A new multi-resource *cumulatives* constraint with negative heights. In P. Van Hentenryck, editor, *Principles and Practice of Constraint Programming (CP'2002)*, volume 2470 of *LNCS*, pages 63–79. Springer-Verlag, 2002. Preprint available as SICS Tech Report T2001-11.

Nicolas Beldiceanu, Pierre Flener, and Xavier Lorca. Combining precedences, incomparableness and tree partitioning constraints. Technical report, EMN, 2005.

Abdallah Elkhyari, Christelle Guéret, and Narendra Jussien. Constraint programming for dynamic scheduling problems. In Hiroshi Kise, editor, *ISS'04 International Scheduling Symposium*, pages 84–89, Awaji, Hyogo, Japan, May 2004. Japan Society of Mechanical Engineers.

Étienne Gaudin, Narendra Jussien, and Guillaume Rochart. Implementing explained global constraints. In *CP04 Workshop on Constraint Propagation and Implementation (CPAI'04)*, pages 61–76, Toronto, Canada, September 2004.

Narendra Jussien. The versatility of using explanations within constraint programming. Research Report 03-04-INFO, École des Mines de Nantes, Nantes, France, 2003.

D. G. Malcolm, J. H. Roseboom, C. E. Clark, and W. Fazar. Application of a technique for research and development program evaluation. *Operations Research*, 7(5):646–669, 1959.