# Staff Scheduling Problem by means of Optimization Algorithms

Kaninda Musumbu

# Staff Scheduling Problem by means of Optimization Algorithms

Kaninda Musumbu
LaBRI (UMR 5800 du CNRS),
Université Bordeaux 1, France
351, cours de la Libération, F-33.405 TALENCE Cedex
e-mail: musumbu@labri.fr

## Abstract

*Our application is placed in the domain of automatic generation of timetables. The generation of timetables is the assignment of employees to tasks in the goal of schedule during one fixed period, in general a week. This problem is by its nature a complex problem which requires the resolution of many constraints. Indeed, the SSP are usually addressed to the organization where a set of tasks must be achieved by a group of employees having their own qualifications, constraints and preferences. The organization imposes total regulations and tries to carry out such total objectives them such as the reduction of the total cost or the equitable division of the load of work between the employees. Research in this field gave place to several conference. The results resulting often are adapted to particular professional sectors. In this paper, we propose a general solution not depending on particular sector.*

**Key words :** artificial intelligence, constraints programming, CSP, scheduler, solver, timetable.

## 1 Introduction

Constraint programming is a powerful paradigm for solving combinatorial search problems that draws on a wide range of techniques from artificial intelligence, computer science, databases, programming languages, and operations research. Constraint programming is currently applied with success to many domains, such as scheduling, planning, vehicle routing, configuration, networks, and bioinformatics. Constraint-based systems are generally quite expressive and are thus good for modelling complex problems. The CSP paradigm consists of a set of variables, each of which have a discrete and finite set of possible values (their domain); and a set of constraints between the variables which specify which combinations of values are allowed and which are not. For this reason, this type of system is sometimes referred to as a finite domain constraint solver. A solution to a CSP is a set of variable-value assignments which satisfies all the constraints. Often, the computing power of the computers is not enough to examine all the possible combinations in an acceptable time, and it is necessary to introduce ”reasoning” and the ”heuristic ones” making it possible to reduce the combinative one and to guide research towards the good combinations. In this direction, the resolution practises of CSPs called upon techniques resulting traditionally from ”the artificial intelligence”.

The algorithms making it possible to solve of CSPs are called ”solvers” of constraints. Some of these solvers were integrated in programming languages, thus defining a new ”paradigm” of programming called ”programming by constraints”: to solve a CSP with a programming language by constraints, it is enough to specify the constraints, their resolution being dealt with automatically (without needing to program it) by the solveurs of constraints integrated into the language.

The aim of this paper is to discribe how we realise a generic application based on the constraint programming field which works in two passes: first solving the contraint and next assigning the attributes to value of variables. Information about dependancies of variables is of higher interest for applications such as our API. where the process of programming includes the generation of the conditions (forced)

Different research notes that the constraints can be divided into two group: on the one hand the strong constraints which must absolutely be respected and on the other hand weak constraints that we could try to respect as possible. If we hold only strong constraints, the problem consists with find one solution between others which satisfies the constraints. If we would like to take in account the weak constraints, we must then add an evaluation function measuring the degree of respect of the weak contraintes. It is a question of finding in the space of the solution respecting the strong constraints the solution which maximizes (or according to the case, which minimizes) this function.

The problem of satisfaction of constraints can be translate in the optmisation problem where the algorithm will look to initially violate the definite weak constraints like no priority (having a small weight) in the evaluation function before violating the priority weak constraints. Two steps are necessary:

- the determination of the solutions space respecting the strong constraints,
- then a research in this space of the solution minimizing or maximizing the evaluation function.

This problem is more similar with the otimization problem when the strong and weak constraints are both evaluate by this function. Then, the contraints satisfaction consists in only optimizing this function: We delete the stage of search for the solutions respecting the strong constraints, alone there remains the stage of minimization or maximization of the evaluation function.

# 2 Methods and Algorithms

## 2.1 Systematic search

From the theoretical point of view, to solve PSC is unimportant if one uses the systematic exploration of the solution space. From the practises view point where the effectiveness takes place. Even if the systematic methods of research (without additional improvement) appear very simple and not-effective, they are important because they form the base of the most advanced algorithms and more efficiant. Just to only quote some most known

- Generate-and-test which consists with a complete labelling of the variables randomizing produced and, consequently, if this labelling satisfies all the constraints then the solution is found if not another labelling is produced.
- Technical Backtracking this method consists in prolonging a partial solution which indicates the consistent values for certain variables towards a complete solution thanks several time choosing a value for an other variable consitante with the values in the current partial solution.

## 2.2 Concistancy technique

Another approach of the solution is based on the removal of the contradictory values of the domains of variable until we obtain the solution. These methods are called consistency technique based on the theories of graph and represented as a graph of constraint (network) where the nodes correspond to the variables and edges are labelled by constraints

## 2.3 Stochastic and heuristic algorithms

During the last years, the local strategies of research became popular. These algorithms gradually change soft values with all the variables. They use a distribution or hill climbing “methaphore” to move towards increasingly complete solutions To avoid wedging itself at least local they are equipped with various heuristic randomizing research. Their stochastic nature aims generaly the guarantee of the “ perfection’ ’ provided by the systematic mthodes of research. Hill climbing is probably the most famous algorithm of research room. It starts by labelling variables by random and, with each stages, it changes a value of a certain variable in such a way that the labelling resulting satisfies more constraints. If a strict local minimum is then reached the algorithm starts again in another state produced by randomizing research, the algorithm stops as soon as a total minimum is found, for example, all the constraints are satisfied or a certain resource is exhausted.

## 2.4 Genetic Algorithms

The genetic approach is inspired by the concept of evolution observed in the real life. The idea is to create a population of individuals, each individual representing a possible timetable. Starting with a first population, one simulates the normal evolution towards better candidates with the optimal solution of timetable. The principal claim is that the more suitable individu develop and are reinforce along generations, because the best individuals are most likely to survive for longer periods. Thus, it results the possibility of obtaining best the solution with the problem “ timetabling’ ’. represent by the most adequate individu.

However, all the attempts providing a rigorous proof to show that indeed the genetic algorithms converge towards an almost optimal solution failed all. It is difficult to provide such evidence because of nature of the problem. On the other hand the justifications are all intuitive and mainly based on the experimentation.

# 3 Implementation of the solution

The important point of this project was the possibility of generating automatically a timetable. This problem being NP-Complete, it was necessary to find a means of solving it in the fastest possible way. Moreover, at the time of the phase of research preceding the development, it appeared that very little of solution, even commercial, had such a solver. For example the software used by the University Bordeaux 1 for the management of the timetables makes only the checking of constraints and not the timetables generation.

### 3.1 Choice of Solver

Once the choice of the programming by constraint was carried out, it was necessary to find a solver because there was no question of coding to us even a traditional solver starting from rather complex algorithms. Among the possible solution found on Internet, `Gnu-Prolog, SICStus Prolog, CHIP`, we focalize ourselves on two solutions seeming better to agree

- **CHOCO** library java dedicated to the resolution of constraints, CHOCO seems a possible solution because it has many advantages: facilitated integration to code java making the interface between the data base and the solver, many constraints of higher order. Disadvantage, after having tested this solution, it proved that the solver has bad performance contradictory with documentation let it predict. Indeed, of many modules being failing, it was not possible to optimize the resolution, which makes this solution completely unusable (obselete).

- **Mozart** It is a platform of development implementing the OZ language. OZ allows almost all the types of programming: declaratory, functional, object, etc and by constraints. This platform thus has a library of resolution which is relatively powerful and reliable: Mozart is developped partly by SICStus Prolog. Moreover Mozart is portable on all architectures more common to date.

### 3.2 Rapid presentation of the problem

Schematically, for a timetable, one needs teachers, courses, students, buildings (rooms). The problem is to place all the lesson in the week. However, each teaching gathers a teacher and students who must not have several simultaneous lesson. The rooms of course are limited, then it is necessary that the obtained result corresponds to a realizable solution. With final, each teaching, will need a certain type of room and each type of room will be available in a limited number. The objectif one is thus that at every moment, the number of teaching simultaneous occupying a certain type of room does not exceed the quantity available of this resource.

### 3.3 Data structure

**course** Mozart has a record data type, making it possible to store information in a structured way. A teaching can thus be represented thanks to such object. The component fields will be:

- `name` this one is single and is recovered in the base. This does not intervene in calculation but makes it possible to identify each course/lecture in order to have an interpretable result

- `start` its hour of beginning. It is this value which will be calculated for each teaching

- `duration` number of course/lecture (example 36h).

- `professor` the teacher to whom is allocated this course

- `size` the type of room in which the course must proceed

Of all these data, it will have there only the hour of beginning which will be calculated and allocated to each course if a solution is found. We ,in addition, decided to manage the temporary values by crenel of ten minutes.. This allows more performance to the solver.

**Teacher** The teachers are represented only sucinctement. Indeed, only the useful information to their subject are their identifier and their unavailabilities.

**Week** The week is divided into crenels of 10 minute. The daytime starts at 8 a.m. and finishes at 7 p.m.. Monday will thus proceed over crenel 1 to 66, Tuesday 67 to 133, etc. The feries days are made inalienable, before all traitments..

### 3.4 Principle of solver operation

Hours of beginning of courses are not fixed but represented by their domains of values. Any value in the domain being potentially correct, The constraints solver tries to solve them in order to arrive at a solution. At the beginning, these domains cover all the week, each course can start at any time. Then, they will be tiny room following various types of constraints: those which are basic, taking care so that courses :

- don't overlap various days, that

- the breaktimes are respected,

Then some more complex which check that any course overlap, and the quota of rooms are respected, etc

### 3.5 Application of the basic constraints

After dialogue, it was agreed various pauses: Thursday afternoon is left free, the crenels of 12h30 $\rightarrow$ 14h are them too. For these two cases, we dispose of a function `DayBreak` its operation is relatively simple: it takes

a crenel as well as a list of course. For each one of these courses, it withdraws domain, hour of beginning, the crenels incompatible with this constraint, i.e. for a course having for one duration of 3h, it is necessary to except eliminate from the domain the pause in it even as well as the two crenels preceding at the beginning of morning or from afternoon.

### 3.6 Application of the no "coarse" constraints

This constraints relate to the courses which do not overlap because, either they relate to interdependent groups (group/sub-group), or they concern the same teacher. In these two cases, in more of avoiding to overlap they must be separated by ten minutes. With this intention, we have the functions `NoOverlapLectures` and `NoOverlapSession`. The first of these two functions takes in parameter the list of courses which do not overlap (incompatibles), especially useful for the same affected professor for two or several courses. The second is rather used for the courses assigned to the same group of students. These two functions guarantee the coherence of the obtained result. For the respect of the number of rooms available, one uses another function `AtMostLecture` intervenes once the courses are placed (once that a possible solution was found on the levels of the teachers and the students. The function takes in parameter a list of course needing a specific room as well as the available number of this this one. Thus the treatment amounts counting, for each crenel, the number of courses provided in the list overlapping. This number should not exceed the second parameter. This very powerful constraint, from a complexity point of view in time, it is the weakness of our solution it would require to be improved (worked over again)! Once all the constraints posed, a solution should be found not violating any of those. With this intention, it is enough to launch the solver included in the platform of Mozart development.

## 4 The Java Module

The solver computes and stores in a data structure the part of the inforamation which is relevant for a given set of constraints. (a skeleton of timetable). To complete the informations we use the Java module which maps with all crenels of solution the name of course, teacher, group and room. This "two passes" approach is simpler and more efficient than the usual one. The interest of such a module is to work on an image of the data base. Indeed, this one is made up starting from objects representing each one of tuple, organized by tables in forms of collections. The handling of data is some thus facilitated. This information must be formatted in files intended to be exploited by the solver. Once that this one to find a solution, it generates an output file of which the data are treated by the Java module. Some aspect such as the personal attribution of the rooms to the various courses , are not manage by the solver and must thus be dealt with by java module. Once exhautives, the data is then inserted in the data base. It can then be extracted by Web interface which will exploit them, to post a timetable. A sending an e-mail makes it possible to hold the teaching supervisor to inform of the computation result. The different functionalities offered by the Java modules are articulated mainly with the turn two packages: `tables` which contains the classes allowing the handling of the data and `solver` gathering all the tools necessary to the data processing in entry and exit of the solver.

### 4.1 Gestion of constraints of materials

#### 4.1.1 Small recall

Each room has to the more five materiels(retroprojector, videoprojector, television, etc). For a teaching, a professor can need a specific material present only in certain rooms. He must inform it at the level of the Web interface and the application will have to then manage this new constraint the best possible one. The solver guarantees that a teaching has a room. But it does not allow it in a personal way. Consequently, it cannot take into account the constraints of materials of each teaching. This task is left to the Java module implemented in the classes `tables.Room, tables.Course, tables.Solver.output` cfr figure 2.

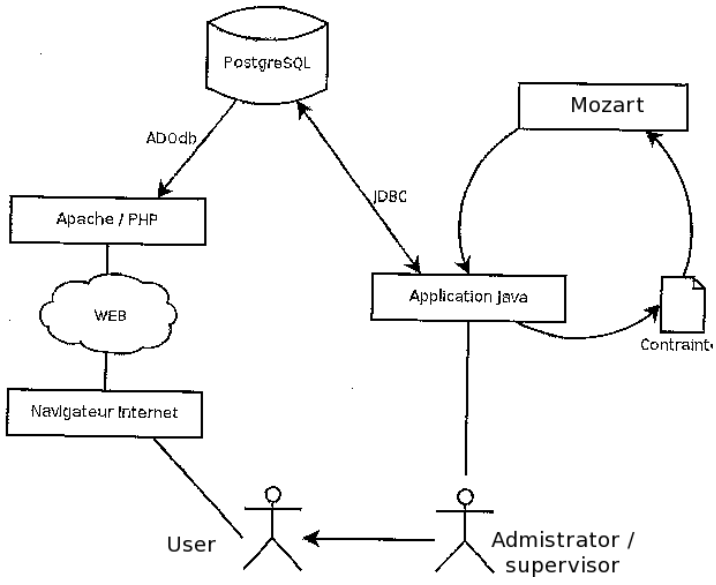


**Figure 1. API architecture**

#### 4.1.2 Principle of the algorithm governing the attribution of rooms

The objectif of this algorithm is to assign to each teaching a room by respecting the best possible constraints of materials. First of all, we classify the course/lesson by type of room in decreasing order of a number of necessary materials . Then, for a given course, we associate with each material present an element coming from a super-increasing sequence. We recall to reader that a sequence $(a_n)$ is call a super-increasing sequence, if it satisfies the following property:

$$\forall j : \Sigma_{j=0}^{i-1} \ a_j < a_i$$

It is easy to show that $a_n = 2^n$ is a super-increasing sequence and this is the smallest possible super-increasing sequence. Let $N$ = a number of materials available of a certain type, we associates to the first material $2^{N-1}$, to the second (if there exists) $2^{N-2}$, so on to each material available in the data base or related to an integer of the super-increasing. type. Then, we add an integer associated with the materials with $C$. This sum will be noted $SC$. We make in the same way for each room available compatible to the type of the course $R_i$ , we will note result $SR_i$. At this stage, course $C$ and all the free rooms $R_i$ are respectively associated an integer $SC$ and $SR_i$, it remains to find the room which would more correspond (on the level material) to course. For that we apply the following algorithm.

```
choiceRoom(integer SC,ListRoom Ri):Room;
result <- createliste();
integer rm;

For i from 1 to Nb-Free-Room do
    if (SC == SRi)
    //The room has exactly the material
   //  needed
        return Ri;
    fi;
od;

For i from 1 to Nb-Free-Room do
    rm <- returnMoney(SC,Ri);
    result.put(Si, rm);
od;

//output is a room associated with the
//smalest integer
return min(resulta;
end;

returnMoney(Integer SC,Room Ri):integer;
integer weigth, res <- SC;
list Super-list<-Ri.weigthingMaterial();
Sort-Decreasing(Super-list);
while(!(empty(Super-list)) && res!=0) do
     weight <- Next(Super-list);
     if (SC <= res)
         res <- res - weigth;
     fi
od;
return res;
end;
```

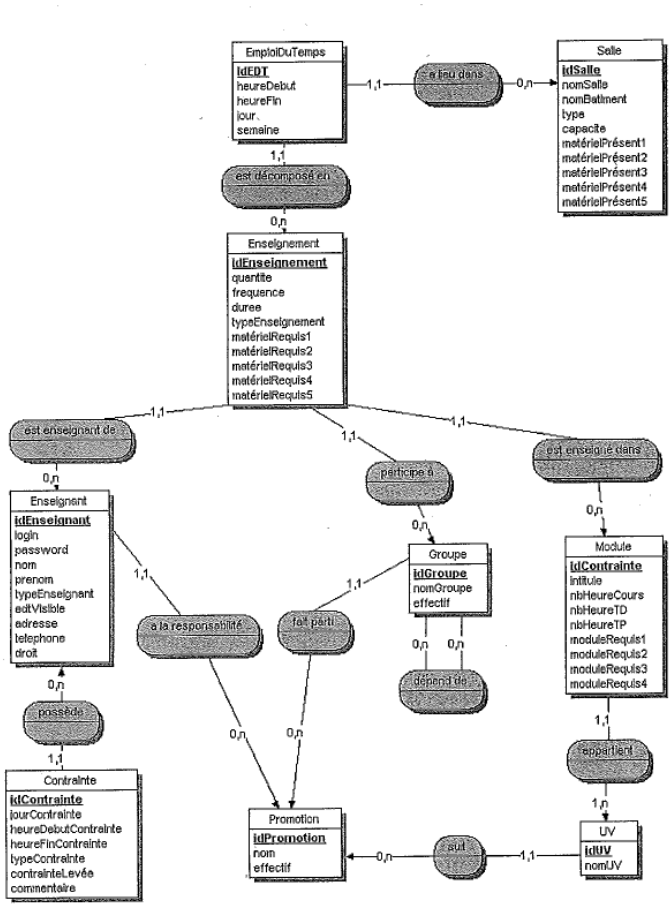


**Figure 2. Final MCD**

### 4.2 Gestion of remedial class

Our application makes it possible to a teacher to catch up with a cancelled or missed course. For that we develop an algorithm which proposes choices of crenel for a remedial class. The algorithm creates a table representing the future days, then, we remove all the crenels corresponding to the constraints of the teacher. Then we remove the crenels where the teacher and/or the studants are incompatible by paying attention to the dependences of group and subgroup. The second part of the algorithm scours the remaining crenels in the table and keeps only those which are

| Table | Nb of lines | Size of lines | Size of table |
|---|---|---|---|
| Room | 450 | 180 | 81000 |
| Course | 900 | 207 | 186300 |
| Constrainte | 4500 | 122 | 549000 |
| UV | 600 | 78 | 46800 |
| Module | 2900 | 160 | 464000 |
| Group | 1000 | 146 | 146000 |
| Teacher | 1000 | 87 | 870000 |
| SST | 320000 | 58 | 185560000 |
| | | T. bytes | 20413100 |
| | | Total Ko | 20413.18 |
| | | Total Mo | 19.193 |

**Table 1. Computation volumetry**

compatible with the duration of the course. It only remains to find a room. It should be noted that the algorithm seeks in priority the room in which this course should have taken place.

### 4.3 Volumetry

For sample taken into account University Bordeaux 1, we roughly have 5 constraints per teacher in the worst case, roughly 5 modules by UV, 10 courses by group and 32 weeks of course per annum. The occupied disk space will be unimportant on the other hand it is difficult to envisage the response time because depend on the machine, the algorithm and the heuristic ones. We made tests for two promotions of a computer department time was rather short.

## 5 Conclusion

The automatic realization of a SST with multitudes of constraints is a very difficult problem NP-Complet. But thanks to Mozart and with the algorithms which we added, we are successful propose a suitable API of which reaction time depending on the heuristic ones. From an accessible interface since any navigator on internet, all necessary data can be entry to design a SST. One can, moreover, to generate the individual SST for a teacher, a student or a room with a safety i.e. it is necessary to be identified to have access to this service. and that. We do not claim completely to replace the man by the machine, but we let us propose to him a substantial help with a task not always easy to realize. Its generics reside in the fact that it does not apply solely to the field taken in example. Developped with free tools, it makes it possible to save licence (free charge).

## References


[1] **André Arnold, Gérald Point, Alain Griffault Antoine Rauzy**: The AltarRica Formalisme for describing concurrent systems. Nov. 1999, Fundamental Informatica Volume 40 issue 2-3, Publisher: IOS Press,

[2] **Aarts, E. and Korst, J.** Simulated Annealing and Boltzmann Machines a stochastic approach to combinatorial optimization and neural computing. Wiley. 1989.

[3] **S. Benhamide and M. Schoemaner** An adaptive algorithm for constrained optimization problem. In Schoenauer et al edior Proceeding of the 6th Conference on Parallel Problem Solving from Nature p. 529-539. springer Verlaag, LNCS 1912 ( 2000)

[4] **Mena, M.B. , Batouche M.** An effecive heuristic search algorithm for maximum satisfiability problem. Applied Intelligence 24, (3), 2006

[5] **Christian Schulte**, Programming Constraint Services, Springer-Verlag, LNCS volume 2302, 2002.

[6] **Peter Van Roy**, Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004, Charleroi, Belgium, October 7-8, 2004, Revised Selected and Invited Papers, Springer, LNCS, volume 3389, 2005

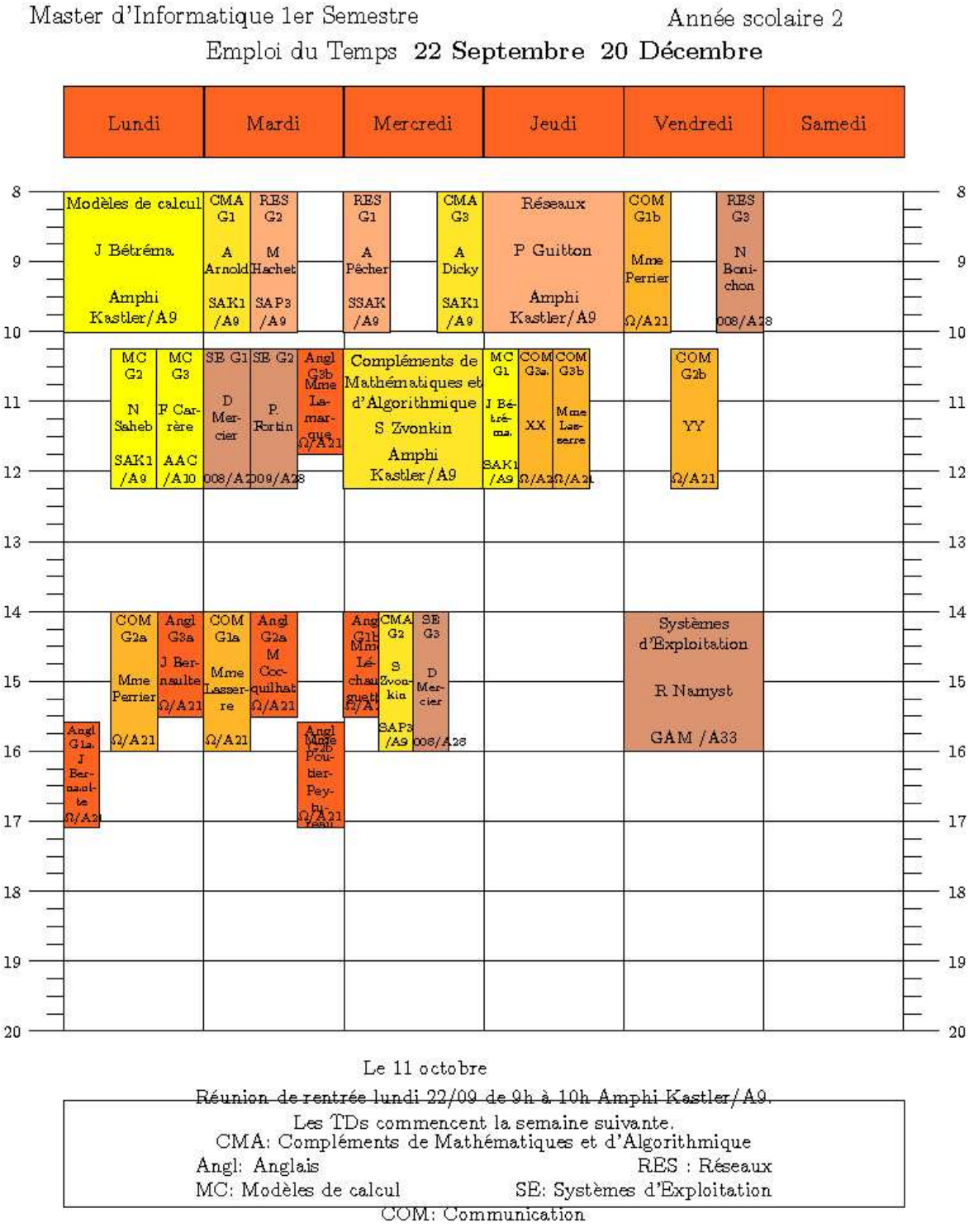


**Figure 3. SST generate by our application**