Building and Optimizing Timetables for Airport Employees

P. Morignot, L. Somek, C. Miller[1], B. Gaudinat[2]
{ philippe.morignot, laurent.somek, bruno.gaudinat }@pactenovation.fr

PACTE NOVATION
2, Rue du Dr Lombard, 92441 Issy-les-Moulineaux Cedex, France
Phone : +33 1 45 29 06 06 Fax : +33 1 45 29 25 00 www.pactenovation.fr

Abstract

This paper presents the MAXIME system to build and optimize timetables for on-ground employees of Orly and Roissy airports. Tasks are composed of timely fixed tasks (e.g., embarkment, check in) and (eventually optional) tasks movable inside a time range but attached to a shift (e.g., lunch, dinner, medical break). Before being assigned to shifts, the tasks are reduced and moved in a deterministic way so as to reduce the peaks representing a large number of agents. The tasks are then assigned to agent shifts, in order to heuristically minimize the number of unassigned tasks. Six heuristics have been developed to speed up this constraint programming module. Then the number of optional movable tasks is increased, using a tabu algorithm. The global social equity of a planning is then maximized, by switching and moving (groups of) tasks among agent shifts, using a second tabu algorithm. Finally, additional tasks, resulting from unexpected flight events, are inserted into an existing planning by switching tasks, while minimizing the number of such switches, using an A* algorithm.
These five modules constitute the main components of the global activity planning system of these airports. MAXIME is functioning on a daily basis since June 2000.

1. Introduction

This paper presents a system to build and optimize timetables for on-ground employees of Orly and Roissy airports. Due to the high traffic increase in parisian airports (7% per year), the Aéroports de Paris (ADP) has made a large effort in promoting the automation of timetable building, with limited human intervention. Indeed, manually building a timetable took one day of a human expert supervisor, whereas experiments detailed in this paper show that it can be done in one minute through the use of a computer. The resulting global system, called MAXIME (Module d'Aide à l'eXploitation InforMatisée de l'Escale) is connected to ADP's existing databases, automatically computes timetables and displays an interactive Gantt chart for the expert supervisor to make final expert changes on the timetables.

[1] Has left PACTE NOVATION.

[2] This work has been supported by a contract with ADP through STERIA.

In this paper, among all the MAXIME components, we focus on the timetable building and optimization one. This timetabling component takes as input a fixed set of tasks and a fixed set of agents [Scha95] [Scha96]. The goal is to assign tasks to agents' shifts in order to minimize or maximize some quantities. Each task is fixed in time *(regular task)*, since the arrival and departure time of airplanes is known in advance. Some tasks are movable in time though *(movable tasks)*, but they are attached to particular shifts of agents (e.g., medical visit). These movable tasks are movable within a time range. Some of these movable tasks are even optional *(optional movable tasks)*, i.e., they may exist or not on particular shifts (e.g., lunch, dinner breaks). No tasks, movable or regular, may overlap.

In addition, there exists an inter-task time between two consecutive tasks. This time depends on the execution location of these two tasks: it represents the time needed for the agent executing these tasks to actually move from the location of the first task to the location of the second one. This can be represented in our framework by augmenting the end time of the first task by the duration of this inter-task time, and keeping the same non-overlapping criterion, hence turning it into a stronger version.

This assignment of tasks to shifts is achieved through several steps: first, tasks are moved in time and reduced in order to minimize the peaks of agent demand; second, an assignment of the tasks to agents' shifts is performed, in order to minimize the number of unassigned tasks; third, the number of movable tasks is maximized, while keeping the same number of assigned tasks; fourth, a global social equity is maximized, while keeping the same number of assigned tasks; fifth, new incoming tasks may be inserted into a timetable, while minimizing the number of changes made to the existing timetable.

The MAXIME system may be used in two modes:

- Planning mode: in this mode, tasks are assigned to shifts 6 months in advance. The timetables for this period of time are generated in a row through a computer's background task. This computation must be done in one night, which explains the maximum duration requirement of one minute for the generation of a single timetable.
- Day-to-day mode: in this mode, tasks may be inserted into an existing timetable, due to flight unexpected events. And the supervisor may make any change s/he wants on timetables (e.g., task overlapping) through an interactive Gantt chart.

The paper is organized as follows: the five chained modules above are successively described. Then we discuss our results and sum up our contribution.

2. Reduction and split of tasks

The goal of this module is to reduce the peak of agents demand: if all the tasks are summed (i.e., the workload curve), regardless of where they could be assigned to, peaks of agent's demand appear on specific time ranges, typically at 11am and 3pm. These peaks are counter-productive since agents should be hired by the supervisor to work only a few minutes per day. This module thus reduces these peaks through reduction and split of tasks, in order to reduce the agent's demand.

Note that the volume of the workload to be performed actually changes due to this reduction module, but the changes are made in an expert way, which ensures that the services offered to the flight company are not degraded.

Tasks may be changed in the following ways:

1. **Task reduction**. A task may be reduced starting from its beginning within a fixed percentage of its length.
2. **Task partial split/move**. The beginning of a task may be removed from a task and added at the beginning of another task that offers services to the same plane. The percentage of the task duration which can be moved is limited.

The algorithm then scans the workload curve in ascending order of time. Each time a peak is detected, the two operations above happen on the tasks of the peak until either all the tasks have been changed, or the maximum peak reduction surface has been reached. A peak is characterized by patterns which change in time. A pattern defining a peak is characterized by:

- The minimum height of the peak;
- The maximum width of the peak;
- The left free zone, i.e., a zone without sub-peaks at the left of the candidate peak;
- The right free zone;
- The reduction percentage.

The effect of this algorithm on the workload curve is to reduce the peaks by (1) lowering them and (2) pushing them forward in time. The second point has the effect of augmenting the "valley" level directly to the right of a peak.

This deterministic algorithm has no time limit. In practice, it runs very quickly even on large sets of tasks.

3. Assignment of tasks to agents

In this module, a fixed set of tasks (regular and movable) and a fixed set of agents are available. Constraint programming is used to assign tasks to agents' shifts [ILOG 98a]. Such a model is composed of variables, constraints definitions and heuristics used during the search; we successively present each of them.

An integer variable is attached to a task and represents the shift's number to which this task is assigned. The domain of each variable has an extra value, which semantics is "unassigned" --- this feature makes this model close to the implementation of a dynamic CSP.

The constraints are:

1. **Integrity**: every task must be totally included into its shift.
2. **Adequation**: check that a task can actually be placed on a shift, due to shift's offerings (i.e., the agent's capabilities) and task's requirements (i.e., the capabilities needed to perform this task).
3. **Non overlapping**: Each time two tasks can overlap, their associated variables must be different.
4. **Movable tasks always present**: Each time a variable is bound (i.e., each time a task is assigned to a shift), check that there is enough space left for the movable tasks of this shift to be placed.
5. **Minimum number of optional movable tasks**: The number of optional movable tasks has a minimum value.

The first and second constraints are unary ones that essentially remove values from the domain of each variable, by preventing a task to be placed on a shift. The third constraint is a binary constraint between variables based on the "!=" primitive.

The fourth constraint cannot be encoded using usual operators among variables: it is encoded as a depth-first search algorithm that scans the space of possible places for the movable tasks, for the considered shift. It is unusual to use a search algorithm as a constraint, but in practice this search algorithm terminates very quickly. Also, this fourth constraint must know all the tasks that are present on a shift: this could be done by scanning all the variables and keeping those related to the current shift. We have chosen to maintain a dual model: an integer set variable is associated to each shift, the integers representing the task numbers that are present onto the shift. A constraint maintains the consistency between the task model and the shift dual model. This way, limited computation is dedicated to the search of all the variables relevant to a shift.

The fifth constraint will force optional tasks to be assigned, which may prevent tasks to be assigned. This is recognized here as a constraint, i.e., something stronger than the quantity to maximize (see next section for a discussion of this point). In practice, this constraint is implemented using additional binary variables, one per optional movable task. When one of these variables equals 1, the corresponding lunch or dinner break is present on the corresponding shift. Then, the constraint is simply an inequality over a sum of these variables.

Note that symmetries are not broken with this model (and the dual model). This trick usually is a powerful way of reducing the search space. A way to perform this would be to replace the "!=" constraint (third constraint) by a "<" constraint. An extra work is required to handle the case where a variable is set to the extra-unassigned value. This could be encoded using a user-defined constraint.

Several heuristics are available to the user to scan the search space:

1. **Leftmost**: Tasks are placed preferably on the left of the shift.
2. **Most filled**: Tasks are placed on the shift with the largest number of already placed tasks with the largest lengths (productivity).
3. **Least filled**: The opposite of the previous heuristics.
4. **Leftmost + most filled**: combination of the two heuristics above.

5. **Regret**: This heuristics computes for each task the number of tasks that could have taken its place. This heuristics then minimizes the number of potential tasks for a task placed on a shift.
6. **Priority**: This heuristics aims at placing tasks on most critical shifts. For this purpose, a priority is computed for each shift. This priority reflects the instantaneous distance between the workload curve and the shift curve. The smaller the distance, the higher the priority. The tasks are then placed by decreasing order of priority.

Experiments show that the heuristics that minimizes the number of unassigned tasks is the second one.

The cost function, which has to be minimized over the solutions found, includes the number of unassigned tasks (first priority) and the number of empty shifts (second priority). This is the regular mode. In a second mode, this module may violate the inter-task time constraint, but must minimize the number of such violated constraints. This is encoded by not posting these constraints but using the implicit boolean value of these constraints and putting the sum of these values in the cost function to be minimized during search. The cost function is then a combination of the initial number of unassigned tasks and the number of violated inter-task time constraints, the weights still highly favoring the number of unassigned tasks.

4. Movable tasks increase

The goal of the previous module was to minimize the number of unassigned tasks. This means putting as many tasks as possible on agents' shifts, which in turn minimizes the number of optional movable tasks (i.e., lunch and diner breaks). The goal of the present module is to maximize this number of optional movable tasks, while keeping the same number of fixed tasks assigned. This maximization operation is inconsistent with the goal of the previous module, which explains why it has been decoupled from it by putting it as a separate minimization objective in a different module. Note that the previous module has a constraint on the minimum number of such optional movable tasks. When considering the previous and current modules together, two strategies are clearly possible:

1. Setting the minimum number of optional movable tasks in the previous module to a high value and do not expect much from the current module.
2. Setting the minimum number of optional movable tasks in the previous module to a small value and expect the current module to highly augment this number.

The first strategy imposes a minimum number of optional movable tasks, while the second one relies on the optimization performed by the current module. The main difference between the two strategies is that the first one will lead to lots of unassigned tasks, while the second one (more conservative) tries to assign as many tasks as possible and then optimize the number of optional movable tasks while keeping constant the number of unassigned tasks. There is a trade-off to be made between the number of assigned tasks that the user wants and the number of assigned optional movable tasks. Both parameters are left to the user and further study is necessary to properly rate this trade-off.

This module is performed using local optimization, since a first solution to the problem already exists: a tabu search [GloLag97] is used with an escape mechanism (reactive tabu search [BatTec94]). A tabu search essentially looks iteratively for the best neighbor, provided that it has not already been encountered. A tabu search is defined by a state, a neighborhood generator and a cost function [GloLag97]. We successively define these elements here:

- A **state** is the set of shifts with their tasks included, i.e., the whole timetable. Tasks involve regular fixed tasks and optional movable tasks.
- **A neighborhood generator**: a state is a neighbor of an initial state iff it can be deduced from the initial state by moving (regular) tasks or switching tasks or groups of tasks between two shifts (see figure 1). Optional movable tasks which are already assigned to a shift may be unassigned due to task exchange, or place for new movable tasks may appear.
- The **cost function** is minus the number of optional movable tasks that can be placed in a timetable. The minus sign is necessary since a cost function is minimized.

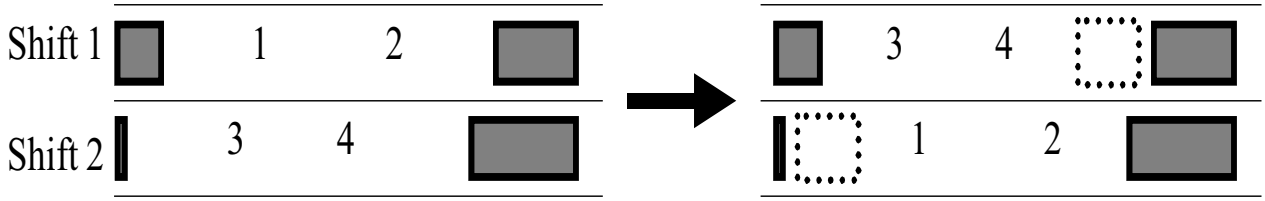

Figure 1: Switching tasks groups 1,2 and 3,4 leaves room for two lunch/dinner breaks.

The tabu list has an arbitrary length of 200. In practice, it behaves as an infinite length list, since all the states are stored, because the response time of one minute limits the number of searched states to approx. 15. This small number of searched states is due to the high number of neighbors to a state, approx. 17,000, which takes time to be generated within a minute minus the time taken by the second module.

The escape mechanism tries to avoid local minima, a serious pitfall of tabu search, by performing a sequence of random moves when a local minimum is detected [BatTec94]. A local minimum is said to be detected when the search reaches the same state twice. A sequence of 20 random moves (arbitrary number) is launched, and the regular search starts again from that new state.

5. Global social equity

Global social equity means that there should be no agent who could complain because his/her shift is clearly worse than the others' are. The goal of this module is to make homogeneous the quality of shifts. This quality of a shift is defined by several quality concepts:

1. **Complexity**: each task has a binary flag indicating whether a task is considered to be complex or not, at execution time. The complexity of a shift is the number of complex tasks of the shift, divided by the number of tasks of the shift.

2. **Diversity**: each task has a type (e.g., embarkment, check in). The diversity of a shift is the number of task type change divided by the number of tasks minus 1 (i.e., the number of task change).
3. **Productivity**: the productivity of a shift is characterized by the sum of the durations of the tasks of the shift, divided by the duration of the shift.
4. **Mobility**: each task has a location where the task is to be performed. The mobility of a shift is characterized by the number of change of location during the execution of the shift.

As before, this module is performed by a tabu search [GloLag97], since a first solution already exists, with an escape mechanism [BatTec94].

- **State** and **neighborhood generation** are similar to the previous module's. Optional movable tasks which are already assigned to a shift are attached to this shift, despite task change --- which may prevent some neighbors to be examined. This strong hypothesis is made in order to prevent the number of optional movable tasks (computed in the previous module) to change during this module.
- **Cost function**: a linear combination of the standard deviation of the quality concepts, except for the production quality, which is characterized by minus the mean over the set of shifts. Weights associated with this combination are defined by the user, who may prefer to emphasize some concepts and lower others. The default values of these weights have been set by testing the module in real cases. The importance of this weighting phase comes from the fact that the quality concepts are interleaved, i.e., are not independent.

The tabu list has an arbitrary length of 200, as before. In practice, it behaves as an infinite length list, since all the states are stored, because the response time of one minute limits the number of searched states to approx. 15. This small number of searched states is due to the high number of neighbors to a state, approx. 17,000, which takes time to be generated within a minute minus the time taken by the second module.

6. Tasks insertion

In this module, a timetable is already built and a new task has to be inserted into it with a time limit, with the constraint that the number of changes made to the existing timetable must be the minimum. The existence of this module is due to the fact that flights may be unexpected, e.g., due to bad weather conditions, they may come to an airport and disturb the timetable previously built carefully with the modules above. These flights have to be serviced too, as regular flights are.

The constraint of minimum number of changes led to choose the A* search algorithm to implement this module [Nil80]. An A* algorithm basically is a best-first search in the space of paths from an initial state to an (expected) solution state. Such an algorithm is defined by the following elements:

- **State**: a state is an entire timetable, with the optional movable tasks attached top the shifts, plus one task, the current task to insert into the timetable. The initial state is composed of the initial timetable plus the initial task to insert.
- **Successor states**: as before, a state is a successor of an initial state iff it can be deduced from the initial state by moving (regular) tasks or switching tasks or groups of tasks between two shifts. Optional movable tasks which are already assigned to a shift may be unassigned due to task exchange, but the optional movable task which has been destroyed on a shift due to tasks exchange must reappear on another shift of the same move, for the number of optional movable tasks to stay constant during this module.
- **Cost function from the initial state to the current state**: This cost function is the number of states encountered on the path from the initial state to the current state.
- **Cost function from the current state to an expected solution state**: This cost function equals zero. This assumption turns the A* algorithm into a breadth-first search with a heuristic for scanning the next level of successor states in some specific order, hence degrades the A* search. But this assumption ensures that the minimum number of states will be used for reaching a solution, since the search progresses by level of states of increasing distance to the initial state. This was the original goal to meet.

Actually, when a task is moved into a shift, several tasks, and not only one, may have to jump out of the shift. Thus several tasks have to be handled at once in the state. This is carried out by implementing a state as a timetable and a FIFO stack of tasks, instead of a unique task. The stack contains all the tasks that have to be inserted for the solution to be found. When a task from this stack has been inserted, it pops up and the next task of the stack has to be inserted in the partial solution state. This mechanism is a way to implement a conjunctive solution, i.e., solving one branch and then solving the other while starting with the solution of the first branch. It can be considered as a partial sort of the nodes of a tree.

This mechanism makes it easier to actually solve the initial problem: inserting several tasks at once. These tasks are put into the stack of tasks of the initial state, and the search starts as described.

One major drawback of this breadth-first search is that it is computationally explosive, hence can quickly reach the time limit imposed on this module. This is a direct consequence of the constraint of minimizing the number of changes performed on the initial timetable. But at least this protects the user from missing too early solutions, which is what the user actually wanted: if this algorithm fails in inserting a task, then the user is sure that there is no solution within depth 5, let us say, thus it may either restart the search while augmenting the time limit (which will lead to the same drawback), or use expert knowledge on tasks to manually modify some tasks to insert the initial task. This knowledge uses high expertise on the domain of airport services and could not be implemented into a computer system (it was not asked to). This shows the limits of a computer system as an aid to a human expert.

7. Discussion

7.1. A second model for task assignment

Other models could be used to assign tasks to shifts. For example, a model using mixed integer linear programming follows: for each task *i*, let us define *j* variables $X_{i,j}$, one per shift, that belong to [0,1]. $X_{i,j} = 1$ iff the task *i* is assigned to shift *j*, and 0 otherwise. The following equations hold:

Integrity and adequation: $Xi,j = 0$ for every task *i* that cannot be assigned to shift *j*.
Non overlapping (1) : For every couple of tasks $X_{i,j}$ and $X_{k,l}$ that can overlap, $X_{i,j} + X_{k,l} <= 1$.

For each movable task *i* of a shift, let us define a decision variable $Y_{i,j}$ that belongs to [0,1]. $Y_{i,j} = 1$ iff movable task *i* can start at time point *j* in its range in this shift. This assumes that the time range of the movable task *i* has been discretized by some time step in this shift.

Non overlapping (2) : For every shift *j* and for every pair of tasks $X_{i,j}$ and movable task $Y_{k,l}$ that can overlap, $X_{i,j} + Y_{k,l} <= 1$.

Movable tasks: for every shift, for every of its movable task *k*, $\sum_l Y_{k,l} = 1$

Due to the use of several variables per task, the following equation holds too:
For every task *i*, $\sum_j X_{i,j} \leq 1$

This model may be used, in conjunction with a branch and bound heuristic, to extract an integer solution from the floating-point relaxed solution.

Early experiments with this model show that the computation time is much too high, even with small time steps for each set of $Y_{k,l}$ variable. For example, it takes 2.5 hours to obtain the relaxed solution for 50 tasks using CPLEX [ILOG 98c] on a PC machines with 128 Mo RAM. Most of this time is spent swapping data on disks.

So this model, although technically doable, has been abandoned in favor of the constraint programming model: both methods are led by heuristics, after all.

7.2. A third model for task assignment

A second model to assign tasks to shifts can even be built with a scheduling package [ILOG 98b]. This package bases its reasoning on movable tasks. But (1) there are at most 5 movable tasks per shift, and (2) heuristics cannot be changed dynamically. So this second model has been abandoned too in favor of the constraint programming model.

8. Conclusion

The whole 5 models have been implemented on PC machines using C++ and ILOG Solver [Ilog 98a], and has been ported on a Bull's RS6000 machine for performance increase. On that machine the second module (task assignment) takes 20 seconds for 600 tasks and 150 shifts. Most of this time is spent initializing data structures and only a few seconds are spent actually computing the solution. The other modules may be interrupted on demand.

Some experiments have been made towards relaxing hypothesis: for example, what happens if tasks actually may overlap? This is possible since an agent in two close locations may work partially on two tasks at the same time and actually complete them on time. Experiments show that in that case there is much less unassigned tasks, as intuition dictates.

Other possible developments involve having more than two lunch breaks, i.e., some arbitrary number of breaks (not necessarily devoted to eating). Again, this is easily doable in the current implementation. Still other possible developments involve adding a constraint related to the French law regarding 35 hours per week: no agent should work more than 6 hours long without a minimum 10-minute break. This can be easily implemented with a user-defined C++ constraint in the task assignment module, and with a close function in the other modules.

Finally, MAXIME has passed the test phase and is fully operational on a daily basis for the supervisors in Orly and Roissy airports since June 2000.

9. References

[BatTec94] R. Battiti, G. Tecchiolli. The Reactive Tabu Search. ORSA Journal on Computing. Vol. 6, n. 2 (1994), p. 126-140.
[GloLag97] F. Glover, M. Laguna. Tabu Search. Kluwer, Boston, 1997.
[Nil80] Nils Nilsson. Artificial Intelligence. Elsevier, North Holland, 1980.
[ILOG 98a] ILOG S.A. Solver Reference Manual, version 4.3. Gentilly, France, June 1998.
[ILOG 98b] ILOG S.A. Scheduler Reference Manual, version 4.3. Gentilly, France, June 1998.
[ILOG 98c] ILOG S.A. Planner Reference Manual, version 3.0. Gentilly, France, September 1998.
[Scha95] A. Schaerf. A survey of automated timetabling. Tech. Rep. CS-R9567. CWI, C.S. Dept., Amsterdam, NL, 1995.
[Scha96] A. Schaerf. Tabu Search Techniques for large high-school timetabling problems. Tech. Rep. CS-R9611, CWI, C.S. Dept., Amsterdam, NL, 1996.