

A Constraint Programming Library Dedicated to Timetabling¹

Dr Alain DRESSE

DECIS sa-nv

av. Louis Bertrand 100 A 12

B-1030 Brussels

Belgium

email : adresse@ulb.ac.be

Abstract

Timetabling problems are typical examples of combinatorial problems. The actual timetabling problems encountered in the industry are further submitted to constraints which make them a domain of choice for the application of constraint programming.

We have used ILOG Solver to address the timetabling problems of the BBL (Banque Bruxelles Lambert). The requirements included both mandatory constraints (there must be between n_1 and n_2 people on week-day mornings), and preferences (as much as possible, respect the transitions night - evening - morning - morning or rest). The preferences were implemented in the choice criteria used to select the variables to instantiate. Propagation demons were used to maintain reversible variables indicating the “degree of satisfaction” of the preferences.

The modelisation of the BBL’s timetabling problem has been incorporated into a C++ library dedicated to timetabling. The major benefits of using ILOG Solver for this problem reside essentially in the ability to create new constrained variable types and constraints by derivation, and thus extend the ILOG Solver library, yielding specialized “add-ons”.

¹ This work is funded by the ‘Région de Bruxelles-Capitale’ through the medium of I.R.S.I.A.

Introduction

The Banque Bruxelles Lambert (BBL) needs to monitor its computer and telecommunications systems on a 24 hour a day, 365 days a year basis. The department responsible for this monitoring is split into six “rooms” of up to 40 people. Constraint programming enabled us to complete the daunting task of establishing a viable yearly timetable satisfying the stringent management and union requirements. In addition, the object oriented approach and the open design of ILOG Solver made it possible to extract from our application a specialized constraint programming library dedicated to timetabling.

We will start the description of our approach by the presentation of a simplified configuration for one of the rooms at BBL to illustrate the types of problems addressed by our library.

We then present the library itself. This section contains a description of the data structures defined in the library, as well as a list of the constraint templates it includes. These templates are illustrated by the requirements given in the BBL’s problem description. Also described are the hooks designed for easy integration of a graphical user interface and the resolution engine providing extensive control on the search process.

We will end our presentation by a brief survey of future enhancements charted for this project.

A sample timetabling problem at the BBL

The planning entity at the BBL is the “room”, regrouping up to 40 people. Workers in a room must be assigned to one of a predefined set of shifts every day. Work assignments in a typical three eight hour shift framework for a 24 hour a day activity are morning, evening and night. We will consider in this example that there are two types of morning shifts, denoted by 1a and 1b, one type of evening shift denoted by 2, and one type of night shift denoted by 3. There are also two inactive shift types. Workers are entitled to place their holidays (denoted by H in the sequel), which must be taken into account in the planning. In addition, the 8 hour daily assignments are longer than the conventional 6.9 hour daily work periods in the bank. The extra hours are compensated by assignments to the rest shifts, denoted by R.

Each room is partitioned into hierarchical levels, themselves partitioned into sublevels. For each one of these levels, minimum and maximum requirements are defined for each shift type. One could say, for instance, that there must be between 3 and 5 people from the support level assigned to the morning shift 1a on weekdays. Specific bounds are applied for week-ends, holidays or other specified dates.

The workers in a room are also partitioned into teams regrouping people of different hierarchical levels. Assignments should avoid breaking up teams as much as possible.

The assignments are done on a daily basis, but all assignments in the same week should be “coherent”. It is forbidden, for instance, to assign a worker to a morning shift on Monday, and to an evening shift on Tuesday. Assignments to both morning shift types in the same week are allowed, but should be avoided as much as possible. Similarly, the week-end shifts assigned to a worker should be compatible with his weekly assignments.

The required coherence of daily assignments in the same week naturally induces the notion of weekly assignments to either morning, evening, night, rest or holiday. Some interweek assignment successions are forbidden. Common sense forbids a weekly assignment to a night shift to precede a weekly assignment to a morning shift. Successive weekly assignments to night shifts are also forbidden. Sequences of the form night - evening - morning - rest or morning - rest are preferred.

Workers assigned to night shifts are entitled to substantial bonuses. It is thus important to spread out the assignments to night shifts between workers of the same hierarchical level. This is partially guaranteed by setting minimum and maximum bounds on the number of night shifts assigned to each worker. The planning should in addition reduce the gaps between counts of night assignments to different workers as much as possible. This requirement indicates the need to have both a constraint enforcing the bounds, and search heuristics minimizing the gaps.

It is also important to spread out night shifts in time for each individual worker. This translates into the requirement that each worker be assigned to one or two night shifts for every sequence of five consecutive weeks, in order to reduce variations in the monthly salaries.

Newcomers to the department must be patronized for some of the delicate shifts : for the first three months in the department, any newcomer must be assisted by a more experienced worker for all shift types during week-ends and holidays, and for the night shifts during the week.

The constraints cited in this description are only a sampling of the constraints involved in the project selected to illustrate the options taken in the design of our library. The complete description of the problem solved for the BBL goes beyond the scope of this presentation.

The Library

Constrained variables

The objective of our library is to allow the resolution of timetabling problems similar to the above, i.e. consisting in assignments of shift types to workers at every point in time.

Definition : Let W be a set of workers, and let S be a set of shift types that can be assigned to the workers of W . A valid timetable is the result of the assignment of a shift of S to every agent of W at every point in time.

We will consider time as cut into periods of equal length, so that specifying the assignment of a worker at each point in time results in specifying the assignment of the worker for each period. The meaning of an assignment varies with the actual problem being solved : assigning a worker to the shift “morning” in the example above would mean that the worker is actually working on some task from 4 to 12, whereas an assignment to the task “painting” in another application may be used to specify that a given worker is dedicated to a painting task for the complete period. The actual meaning of an assignment is however irrelevant for building the timetable, as long as two assignments are not allowed to coexist in the same period. A precise definition of the period in the constraint programming framework follows.

Definition : A period is a constrained variable representing the assignment of a worker at some point in time.

The domain of a period is a set of shift types. The two basic domain modifying operations on a period are instantiating the period to a given shift type, or removing a shift type from the period’s domain.

Different time scales can be used in the same problem giving rise to different periods for the same worker at the same point in time. The description above, for instance, uses both daily and weekly time scales. The discretization of time can therefore not be determined globally. Instead, we will introduce the notion of an agenda combining a worker with a time scale.

Definition : An agenda is an abstract data type encapsulating the assignments of a given worker for a given time scale. It can be seen as an array of periods. The agenda gives access to each individual period it contains. In addition, it allows the user to post constraints on the complete agenda. Constraints posted on agendas react to the events triggered by any period in the agenda.

Agendas with identical time scales can be regrouped into entities describing all assignments of a set of workers.

Definition : A category of workers of time scale s is a set of workers sharing the time scale s .

To a category of workers of time scale s can be assigned a category agenda regrouping the agendas of time scale s .

Constraints can be posted on category agendas to react to the events pertinent to any period in the category agenda.

Resolution scheme

The resolution scheme we have adopted for our library is based on a standard iterative resolution scheme in constraint programming consisting in the following. One repeatedly selects the pair (constrained variable - value in its domain) according to various problem specific criteria, and then instantiates the constrained variable to the value, propagating all consequences as implemented by the constraints. If this assignment is determined incoherent by propagation of the constraints, then the value chosen is removed from the domain of the variable, the consequences of that removal being propagated by the constraints, and the process is pursued with the new configuration.

We will come back later in this paper to the process of selecting a variable - value pair when discussing the interaction between the user and the search process. In the next sections, we will instead concentrate on the expression of the constraints defining the problem.

Constraints

The constraints in the library have been selected from those developed for the BBL when considered general enough to be of application to other timetabling problems. They can be arranged as pertinent to an agenda, to a category agenda, or to a set of agendas of possibly different time scales. Note that we have no predefined constraints acting directly on periods, as we have not felt the need for such constraints in our applications. They can nevertheless be defined by any user of the library : the definition and implementation of new constraints requires no inside knowledge of the library, as usual with ILOG Solver.

In addition to the constraints actually describing the problem to solve, our library implements redundant constraints ensuring the coherence of the data with respect to several constraints taken simultaneously. Those redundant constraints, which we call metaconstraints in the sequel, have proven essential to an efficient resolution of timetabling problems. We have gone into great length not to restrict access to data used in predefined constraints that could be pertinent to user defined redundant constraints. Some examples of metaconstraints will be considered in the next section.

Constraints posted on agendas

The most basic constraint one can post on an agenda is one limiting the set of admissible shift types for a given period, or set of periods, resulting into a simple restriction of the domain of all concerned periods. Such a constraint could be used for instance to specify that workers from the foreman level should not be assigned to night shifts on week-ends.

More elaborate are the forbidden transitions. This constraint forbids certain sequences of shift assignments. It is used, for instance, to specify that night shifts cannot precede morning shifts at the BBL.

The constraint enforcing agenda counters is used to specify upper and lower bounds on the number of assignments to each shift type for a given agent. In the BBL example, it would be used to ensure that every agent has between 55 and 65 daily night shift assignments each year, ensuring that the difference between night shift assignments between any two agents is less than 10 days.

The constraint we have implemented does not allow the definition of a counter for a single shift type : posting this constraint will define counters for all admissible shift types in the agenda. We enforce this restriction because we have experienced a noticeable enhancement of resolution efficiency when posting a redundant constraint expressing that the sum of all agenda counters is equal to the number of periods in the agenda. A planned extension of this constraint will allow the specification of sets of shift types which should be counted together. Hence, if the only meaningful bounds pertain to night shifts in the BBL problem described above, for instance, then the user could regroup counters to mornings and evenings into a single counter.

In addition to the specifications of the upper and lower bounds enforced on the counters, the library gives access to the actual number of assignments to each shift type in the agenda, as well as the total number of periods that still have the given shift type in their domain. These pieces of data are used for search heuristics, for the implementation of metaconstraints, and for the display during the search process.

A specialized version of the agenda counters is the period set counter, pertaining to a subset of the periods in the agenda. These periods need not be consecutive, so that one could use this constraint to limit the number of night shifts assigned to Sundays and holidays for a given worker.

It is often necessary to define several period set counters with identical bounds on the same agenda. Most common is the adjacent fork counter configuration, partitioning the agenda into a finite number of subsets of consecutive periods, with identical bounds for each one of the subsets. Such a constraint could express the requirement that there must be at least one weekly assignment to night shifts every month.

This adjacent fork configuration is implemented as a separate constraint for two reasons. Firstly, its implementation reduces the memory consumption by avoiding unnecessary duplication of data structures : the bounds are specified once for all period sets. Secondly, the specific implementation allows the incorporation of a redundant constraint relating all the period set counters.

Adjacent forks are not sufficient if one wants to spread out the assignments to a given shift type in an agenda. Asking one night shift assignment per month does not prevent a configuration with one assignment at the beginning of a month, and another at the end of the next month, thus placing about 6 consecutive weeks with no night assignments. Another constraint filling this gap is the gliding fork. A gliding fork sets bounds on the number of assignments to given shift types for all fixed length sequences of consecutive periods. Using a gliding fork, one can express that there must be at least one night assignment every five consecutive weeks, without reference to month boundaries.

The gliding fork constraint ends the list of constraints bearing on isolated agendas in our library. As stated above, this by no means restricts the use of our library to problems that can be expressed using these constraints, as the library user may define other constraints specific to the problem under study. We now address the constraints expressing conditions on category agendas.

Constraints posted on category agendas

All the constraints defined above can be posted on a category agenda, in which case they apply to all agendas in the category.

However, posting them on the category has several benefits. Of course, there is the benefit of reduced memory consumption due to data sharing. But in addition, several metaconstraints can be defined on agenda constraints posted on a category as a whole.

Other constraints implemented in the library have a meaning only in the framework of a category agenda. The first one of these constraints we address is the patronizing constraint, expressing that some worker cannot be assigned to a given shift type unless supervised by another agent in a given category. The worker being patronized need not belong to the patronizing category, but he must have an agenda with the same time scale as the patronizing category.

Generalizing the patronizing constraint is the forced teamwork constraint, expressing that all workers in a given subset of a category must work together.

At the opposite of the above is the agent incompatibility constraint, expressing that several workers of a given category should never be assigned simultaneously to the same shift type.

Probably the most important category constraint, the category counters constraint specifies upper and lower bounds on the number of assignment to each admissible shift type for each period in the category. This constraint describes the staff requirements for each shift type for each category of workers, and is essential in all timetabling problems we have met. It is used at the BBL to specify that there must be between 5 and 7 controllers every morning.

Constraints posted on agendas of possibly different time scales

Every constraint we have encountered involving agendas of possibly different time scales could be expressed as an assignment compatibility constraint. This constraint is a generic constraint which can be tailored to meet requirements of the type « weekdays can be assigned to mornings of type a or b only if weeks are assigned to mornings », or « a Saturday can be assigned to evening if and only if the adjacent Friday is assigned to

evening » or « a Saturday can be assigned to a non rest shift if and only if the Monday in the same week is assigned to the rest shift ».

Our generic assignment compatibility constraint can be expressed as follows.

Let A1 and A2 be two agendas. Suppose there is a relationship between the time scales of the two agendas such that

- every period of A1 is related to at least one period of A2
- the relationship is reversible, in the sense that every period of A2 is related to at least one period of A1, and if a period p1 of A1 is related to a period p2 of A2 (plus possibly other periods of A2), then the inverse relationship should relate p2 to p1 (plus possibly other periods of A1).

Let S1 be a set of shift types admitted in A1, and S2 a set of shift types admitted in A2.

The compatibility constraint states that every period p1 in A1 assigned to some shift type of S1 implies that its related periods in A2 are assigned to a shift type in S2.

Metaconstraints

The metaconstraints, which maintain the coherence of the data with respect to several constraints taken simultaneously, are not necessary for a proper description of the problem to solve, as they are redundant. Their incorporation in the library only enhances the efficiency of the resolution by eliminating incompatible assignments early in the search process. We will not describe here all metaconstraints which have been found useful to avoid heavy technical descriptions, but will present two of them in order to illustrate their usefulness.

The first metaconstraint we consider relates category counter constraints for categories which form a partition of some other category. Suppose for instance that the staff of one of the rooms at the BBL is split into 10 controllers, and 15 support agents. Suppose that there can be maximum 5 controllers but that there is a minimum requirement of 12 workers every morning. If it has been specified that the room is partitioned into controllers and support agents, then it can be deduced that there must be minimum 7 support agents every morning. The category partition constraint expresses that some category is partitioned into specified subcategories by relating the category counter constraints.

The other metaconstraint we consider also pertains to counter constraints, relating the agenda counter and category counter constraints inside a category. It is obvious that the total count of morning shifts should be the same whether one adds up the number of mornings assigned to each agenda, as counted by the agenda counters, or whether one adds up the number of mornings assigned for each period, as maintained by the period counters. Incorporating a constraint implementing this equivalence efficiently propagates modifications of the domains of any counter involved in the metaconstraint to all the other counters.

Preferences

Besides the strong constraints which must be satisfied if the timetable is to meet the requirements, timetabling problems often include « soft requirements » which turn out to be some measurement of the quality of a valid timetable. The soft requirements we have met at the BBL that could be used in other timetabling contexts are included in the library as preferences.

The preferences are implemented as constraints that never fail, but that update reversible variables indicating the degree of satisfaction of soft requirements, providing a measure of the quality of the timetable under construction. This measure is then used when choosing variable - value pairs in the resolution engine.

The first such preferences we address are the preferred transitions. These are used for instance to indicate that it is more appropriate for an evening to be followed by a morning rather than a night shift.

An extension of the preferred transition is the natural rhythm indicating a sequence of assignments to be abided to as much as possible. Our implementation of the natural rhythm comprises a sequence of sets of shifts, allowing the user to specify for instance that the ideal cycle consists in a night, followed by an evening, followed by a morning, followed by either a morning or a rest.

We have seen in the description of the sample timetabling problem that it was important to spread out the assignments to some specific shift types among workers of the same hierarchical level. We have also seen that maximum gaps between the number of such assignments could be set by specifying a minimum and maximum number of assignments per agenda, but this is often insufficient. We have implemented a preference to reduce the gaps between assignments to different workers in order to reduce unfair distributions.

The last preference included in our library is a teamwork enhancement preference. Workers in a department are often regrouped into teams consisting in workers of different hierarchical levels. It is best to keep the teams together as much as possible. It is however often impossible to strictly enforce this as the capacity requirements dictated by the work load differ with the hierarchical levels. Our teamwork preference hence tends to avoid splitting the established teams without forbidding violations to the teamwork.

Besides the specified preferences, we are currently at work implemented soft versions of all the constraints of the previous section, enabling the relaxation of these constraints without neglecting them in the search process.

User Interaction

It seems to us that the end user of any timetabling application (which we will denote by the timetabler in the sequel to avoid confusion with the library user) should be allowed to interact with the construction of the timetable as some information will necessarily require « human involvement », be it in the choice of constraints to relax when the problem is impossible or too hard, in « hand made » assignments indicating last minute changes in the data, or simply using human intuition to drive the search for a valid timetable.

In order to allow such a human involvement in the resolution process we needed two ingredients :

- a graphical interface allowing an ergonomical presentation of the partial results during the search process and an easy modification of the data structures,
- hooks in the resolution engine enabling the user to pause the search, modify the configuration and continue the search.

Graphical interface

We have developed a graphical interface for the BBL problem in ILOG Views, but do not intend, for the time being, to make that interface part of the library. We have however included in the library the necessary hooks to allow the display of meaningful information during the search.

A specialized display constraint triggered at every modifications of the domains of periods in the agendas can be posted either on agendas or on agenda categories. Demons of this constraint can also be triggered by modification in the upper and lower bounds of agenda counter or category counter constraints. This constraint can then maintain the display of information available to the timetabler.

Resolution engine

The resolution engine described at the beginning of this paper has been enhanced to allow user interaction with the search process. At each step in the iteration, after instantiation of the selected variable, our resolution engine calls a user defined function that can pause the search, make modifications to the data structures, and continue, stop or restart the search.

The process of selecting a pair (variable - value) to instantiate is also enhanced. We define an object — which we call an action — describing the next domain modification to carry out. This action is selected by a user defined function. There is no restriction to the nature of that action, but there is no guarantee that the search will terminate if the action does not systematically reduce the domains of the variables. The user defined function specifying the action can either call predefined heuristics selecting most appropriate actions based on the constraints and preferences from the library, or redefine entirely the action selection process, possibly incorporating requirements from the human timetabler.

Further Developments

We have now described our library as extracted from our solution to the BBL problem. Several enhancements are planned in order to make it as generic as possible.

New constraints from other timetabling contexts

The first enhancement of our library involves of course the incorporation of constraints dictated by other timetabling contexts. The problems currently under study are nurse scheduling and bus driver timetabling. These are not abstract toy problems taken from the literature, but come from specifications of real life industrial problems issued from partnerships of DECIS.

Violating the constraints

Most constraints specified by our industrial partners were sometimes violated in the hand made plannings, indicating that the timetabler may accept controlled deviations from the initial specifications if the latter are too hard to meet. This is often the case in the summer when many workers are unavailable. We have decided to include in our library the possibility to relax constraints.

This also brings us to help the timetabler to choose the appropriate constraints to relax, identifying the constraints that may be responsible for the complexity of the problem.

Also planned is a module for modifying a given problem in order to reduce the number of violations, or increase the satisfaction of preferences, using operations research techniques such as simulated annealing or tabu search.

Graphical interface building blocks

Also important is the display of timetabling information, the interface for specifying the constraints, their persistence and the interface guiding the interaction between the timetabler and the problem solving. We have developed these for the BBL problem, and intend to extract the generic graphical objects for inclusion in an extension to the ILOG Views library specialized in timetable display.

Increased efficiency

Last but not least, we are at work reducing the memory consumption of our constraints, metaconstraints, preferences and resolution engine, and enhancing the performance of the timetabling process.

Conclusion

We have successfully solved a non trivial timetabling problem for the BBL (Banque Bruxelles Lambert) using constraint programming techniques. Our work in that field has given us the experience necessary to design and implement an efficient library dedicated to timetabling problems on top of ILOG Solver.

This library includes all constraints we have felt useful to describe generic timetabling problems, and all redundant constraints that, in our experience, extensively prune the search tree. It also includes a flexible resolution engine giving the end user extensive control on the search process. Hooks are provided for the development of a graphical interface giving the end user the information necessary for taking the right decisions.

Planned future developments include new constraints, control of relaxation of the constraints and graphical interface building blocs in ILOG Views.

The library will be available in beta version by the end of this year.