A Constraint Based Generic Model for Representing Complete University Timetabling Data

Atish Chand, Dept of Mathematics and Computing Science,

University of The South Pacific

chand_at@usp.ac.fj

Abstract. Several proposals have been made for a standard timetabling data format to enable easier sharing of data and allow timetabling methods and solutions to be more easily evaluated and compared. All proposals state that due to the wide variety of data and constraints that differ from one timetabling instance to another, it is difficult to specify a standard timetabling data format that is general, complete and practicable. This paper contends that the variety of categories and the difference in constraints are superficial and that the constraints can be expressed as a smaller number of categories. A constrained based generic model is proposed that allows data and constraints to be easily converted into relational database tables and other formats making it practicable. Since the number of categories of constraints is reduced , it is easier to compare timetabling methods and algorithms since the similarity in constraints are more easily evident. The model is applied to two different timetabling environments specified by Burke and Goltz. The author hopes to build a database of instances of timetabling constraints specified in the proposed model and converters for translating data to and from the different current formats.

1 Introduction

Currently no universally accepted model exists for describing timetabling problems. Efforts have been made towards developing a standard information format for timetabling problems and although some data formats have been developed for representing different timetabling problems, they are usually incomplete in some aspect. This section examines the requirements of standard timetabling data format and several timetabling data formats are investigated for their strengths and limitations. Section 2 proposes a constraint based generic model of timetabling data while the last section the model is applied to represent the data and constraints of two timetabling environments.

Burke *et al* (1998) have stated three requirements of a standard format – generality, completeness and practicability. Generality requires that a standard format be able to express any reasonable instance of the timetabling problem. If a timetable instance is not able to be stated in the standard format then it would be unacceptable by users of that timetabling instance. Completeness requires that it must be possible to express an instance completely. This includes straightforward data expressing the resources and sessions, complex logical constraints and proposed solutions including hard and soft constraint importance. This paper elaborates on numerous examples of common timetabling constraints and explicitly shows how they can be modeled completely. This requirement allows the model to serve as benchmark for comparing different timetabling algorithms. The third requirement is that the translation of data between the existing format and the standard format should be practicable.

Reis *et al* (2000) have presented a very good review of work done on standardizing timetabling data. They have also proposed a verbose model that does not generalize the constraints but suggests a list of synonyms for various constraints and data, the intention being to help administrators specify constraints. In their verbose model they have suggested synonyms for key concept words.

The use of these synonyms makes the description of input data files more easily readable by human users. Also, users can build their own list of synonyms in a separate file and, with the help of a simple pre-processor, that file can be directly converted into their proposed specification language. In this author's proposed model the verbose functionality can be built into the user interface with the data model itself remaining simple.

Reis *et al* (2000) have not shown how all reasonable instances of timetabling problems can be specified, thus is of limited generality.
Johnson's (1993) database approach and the Slot Manager (Johnson D, 2000) are tools that assist administrators to 'book keep'the data. Timetabling resources are represented in an entity relationship model but it is up to the administrative user to interactively specify constraints. Reis *et al* (2000) , Cooper and Kingston (1995), Collingwood *et al,* (1996) and Burke *et al* (1998) have all described various constraints as being different. It will be shown in this paper that the constraints can be generalized into three main categories, thus simplifying the process of building a timetable and comparing solutions.

2. Generalization of Timetabling Constraints

Existing timetabling solutions typically use a database to store domain data but not constraints since the wide variety of constraints, expressed in varying formats and languages make it difficult to conceptually model the general university timetabling problem. A few timetabling techniques (Johnson, 1993, Weare 1995) studied modeled only the domains of the timetabling problem as an entity relationship model while constraints were represented in a separate model. The constraint models are varied and not amenable to comparison.
Several timetabling cases will be next investigated for how they have formulated their constraints, how they appear to be different, what the similarities are and how they can be generalized.

This paper uses the following keywords to describe the key timetabling objects and concepts.

Course:	A unit of study offered during a particular semester and year (or period of time).
Session:	A contiguous period of time allocated to teaching a course or conducting an academic activity . A course may have more than one session.
Day:	A day of the week, uniquely identified by a unique number representing each day of the week
Room:	A physical space in which a session is conducted. This may be a room , a playing field, swimming pool, a laboratory or any other physical space, uniquely identified by Room Code.
Time-slice:	A length of time, beginning at a time based on the 24 hour clock
Time- period:	A grouping of contiguous time-slices, for example, a group of particular time-slices may be called a day.
Time-slot:	A combination of a day and a time-slice
Timetable-allocation:	A session associated with a timetable-slot
Timetable:	A set of all timetable-allocations .
Constraint:	An expression that should be satisfied to some predetermined extend to successfully solve a timetabling problem.
Hard Constraint:	A constraint that must be satisfied necessarily to successfully solve a timetabling problem.
Soft Constraint:	A constraint that should preferably but not necessarily be satisfied to successfully solve a timetabling problem.
Resource:	A session, time-period or room .
Timetable:	A set of all timetable-allocations .

2.1 Typical Constraints

Burke *et al* (1994) delineates some of the most common types of timetabling constraints:

Resource Assignment

A resource may be assigned to a resource of a different type, or to a meeting. For example, a lecturer prefers to teach in a particular room (for whatever reason), or an exam should take place in a particular building.

Time Assignment

A meeting or a resource may be assigned to a time. This constraint can be used to specify days on which a teacher is unavailable, or to pre-assign a time to a particular meeting.

Time Constraints between Meetings

Common examples of this class of constraint are that one particular lecture must take place before another one, or a set of exams must be sat simultaneously.

Meeting Spread

Meetings should be spread out in time. For example, no student should have more than one exam in any day and lectures of a course should be spread over different days.

Meeting Coherence

These constraints are designed to produce more organized and convenient timetables, and often run contrary to ``meeting spread'' constraints (above). For example, a lecturer prefers to have all his lectures in three days, giving him two lecture-free days.

Room Capacities

The number of students in a room may not exceed the room's capacity.

Continuity

A continuity constraint is any whose main purpose is to ensure that certain features of student timetables are constant or predictable. For example, lectures for the same course should be scheduled in the same room, or at the same time of day.

Goltz *et al* (1999) states the following constraints for the Medical Faculty at the Humboldt University timetabling problem:
C1. The teaching activities can begin every quarter of an hour and may run for different lengths of time.
C2. There are restrictions with respect to the starting time for each teaching activity.
C3. Two lectures about a same subject must not be scheduled for the same day.
C4. The lectures, seminars, and practicals of each group must not overlap.
C5. The number of seminars and practicals that a department (research group) can conduct at a certain time is limited.
C6. Some teaching activities are only held during certain weeks, where the following cases are possible: all weeks, A-weeks (weeks with odd numbers: 1,3,5, …), B-weeks (weeks with even numbers: 2,4,6,…), either A-weeks or B-weeks.
C7. There are lectures about the same subject carried out in parallel at the parts of the faculty. Thus a student can choose from among parallel lectures.
C8. Timetabling also involves allocating rooms to the individual lectures.
C9. Some lectures can only be held in certain specified rooms; in some cases only one room matches the requirements.
C10. The varying distances between the different locations of the teaching activities must be taken into account.
C11. The starting time preferences for teaching activities and the preferred rooms for lectures should also be taken into account if possible.

The constraint C 11 is a soft constraint. The other constraints are hard constraints and must be satisfied.

Burke *et al* (1994) and Goltz *et al* (1999) have specified different constraints different timetabling environments. A generic classification of constraints will now be defined, such that all of the above constraints, and others, can be modeled easily as a few classes of constraints.

There are many different objects that have to be timetabled – courses, lectures, exams etc. However an analytical modeling reveals that it is actually 'sessions' that are allocated a timeslot. According to Reis *et al* (2000) some of these words are synonymous, for example, a lecture is synonymous with exam. A lecture may be divided into smaller time periods – some lectures may have a half an hour session while others may have two half an hour sessions held one after the other. A course may have three one hour lectures with a requirement that 'a lecture (session) be timetabled at the same time and room as one of the lectures of another course because it is taught by the same lecturer and covers the same content'.

This paper takes the approach that students attend sessions of a course rather than courses. So if two courses have three sessions each with one common session, the approach is that that the first course has two sessions say S1 and S2, the second course has another two distinct sessions S3 and S4 while session S5 is common to both, with the number of students enrolled for S5 being the sum of students enrolled for S1 and S3. Hence forward, we will discuss timetabling of sessions, rather than courses or exams.

It is proposed that constraints are of three main types: Domain, Spread and Count constraints. Domain constraints are those that limit a session to certain values of time-slots and rooms. Spread constraints are those that specify how the timetable-allocations should be spread over time and rooms.

2.2 Domain Constraints.

Domain constraints are those that restrict a session to certain domain values. A session may be restricted to certain rooms, days, hour or combinations thereof. Constraints such as "The number of students in a session can not exceed the capacity of the room allocated to it", "Only computer lab classes can be held in the computer labs" and "Avoid scheduling examinations for video broadcast courses classes on Mondays" lie in the domain category. Domain constraints can be specified as

DomainConstraint((SessionList), (DomainList), Hardness-factor)

where *Domain* is replaced by the name of the actual domain such as day, hour, room etc. *SessionList* is the list of sessions that that must be allocated to a domain element that must be in the *DomainList*. *Hardness-factor* is a numeric value corresponding to the hardness of the constraint. A negative *Hardness-factor* indicates that that no session in *SessionList* must be allocated a value in *DomainList,* that is, satisfying the constraint should be avoided. Specific domain constraints are discussed next.

Room Constraints

These include constraints that a room should be able to accommodate the sessions they are allocated to and that certain rooms are reserved for special sessions. To a smaller universities and much larger extent at larger universities, sessions should also be allocated to rooms that are near to the students' main faculty.

Also certain sessions may need to be scheduled together in the same room or at other times a certain room may be allocated to more than one session at a time. Thus some other typical room constraints are:

- A session may need to be allocated to a room with special facilities.
- A session may need to be restricted to certain rooms for special reason, e.g. rooms close to the department.
- A room may be restricted to special sessions only, e.g. only Physics laboratory classes can be held in the Physics Laboratory.

All these constraints can be represented as

RoomConstraint(SessionList, RoomList, Hardness-factor)

where *SessionList* is a list of sessions that are constrained to be held in rooms in the *RoomList*.

Examples of the above room constraints are:

The constraint that Physics sessions (PH101T1, PH101T2,) need to be allocated to a lab room R1with special facilities is specified as

RoomConstraint((PH101T1, PH101T2), (R1), 100)

Another constraint may be that session CS121L1, CS121L2, and CS121L3 need to be held in a room with enough capacity. Let's say rooms R10, R11, and R12 have enough capacity, than the constraint is specified as:

RoomConstraint((CS121L1, CS121L2, CS121L3), (R10, R11, R12), (100))

In some situations two or more sessions may be held in the same room. In that case, *RoomConstraint* is dynamically changed by allocating attributes *Maximum Capacity* and *Current Capacity* to room. If a room has a maximum capacity of 200 and is allocated a session of size 50 for a particular time, then current capacity is updated to 150 for that particular time. Relational database systems typically have the capability to automatically update current capacity via a trigger whenever a session is allocated to a particular room. Non-relational database timetabling systems could use a similar mechanism to update room constraints. If only one session can be held at a time in particular room then current capacity is update to 0 for that particular time-slot.

Time Constraints

Time domain constraints in university timetabling involve division of the year into smaller semesters or trimesters which are further divided into weeks. Typically, the five week-days are used for teaching activities. Each day is further divided into hours

or parts of an hour. The various ways in which time is grouped results in various time constraints which appear to be difficult to standardize.

The standard time constraint proposed in this paper is

PeriodConstraint(SessionList, PeriodList, Hardness-factor)

where *Period* is the time period being constrained, *SessionList* is the list of sessions that are being constrained to the *Periods* in the *PeriodList.*

Time periods are typically divided into days and hours of the day. A session may be restricted to certain days, certain hours and/or combinations of certain days and hours. We will now examine how the generic time domain constraint format can be used to specify the day, hour and combination of day and hour constraints.

Day Constraints

These constraints are those that restrict the variables to certain values in the day domain. For example, all sessions must be held on a day from Monday to Friday, but not during weekends.

The constraint

DayConstraint(SessionList, DayList, Hardness-factor)

specifies that all the sessions in the *SessionList* must be timetabled on a day that is in the *DayList*.

Days can be conveniently represented with numbers that indicate their positional relationship with each other. The following table shows the day numbering scheme used hereinafter.

Table .1 Day numbering

Day	**Number**
Monday	1
Tuesday	2
Wednesday	3
Thursday	4
Friday	5
Saturday	6
Sunday	7

Thus the constraint

DayConstraint((CS100, CS102, CS122). (1,2,3,4,5), 100)

means that the sessions CS100, CS102 and CS122 must be timetabled on a weekday.

Hour Constraints

These constraints are those that restrict the variables to certain values in the hour domain. For example all, sessions must be timetabled on the hour, from 8 am to 7 p.m., but not before 8 a.m. and not after 7 p.m.
The constraint

HourConstraint(SessionList, HourList, Hardness-factor)

specifies that the sessions in the session list must be timetabled at an hour in the hour list.

The above example can be specified as:

HourConstriant(CS100, CS102, CS122), (8,9,10,11,12,13,14,15,16,17,18,19), 100)

Certain sessions (say CS121L1 and CS121L2) may need to be held at the same hour each day. If it is a specific hour e.g. 10, then

HourConstraint(CS121L1, CS121L2), (10), 100)

suffices but if the constraint is that the sessions can be at any hour but it should be same for all (referred to by Burke (1999) as continuity constraints), then this constraint is a spread constraint that is discussed later on in this paper.

Day Hour Constraints

These constraints are those that restrict the variable to certain combinations of day and hour values. A combination of a day and hour is referred to as a time slot. For example, certain sessions may need to be allocated time slots only between Monday 10 a.m. and Monday 2 p.m..

The constraint

TimeConstraint(SessionList, TimeList, Hardness-factor)

specifies that the sessions in the session list must be allocated a time in the time list. *TimeList* is a cross product of *DayList* and *HourList*. Day and Hour can be integrated into a single number using various functions such as

$$T(day, hour) = day * 100 + hour$$

where day is any whole number between 1 and 7 and hour is any integer between 0 and 24.

For example Monday 10 a.m. is allocated 110, Monday 11 a.m. is allocated 111 and so forth. Thus the constraint that the sessions CS111 and CS101 should be timetabled at any hour from 10 am to 2 pm on Monday, can be specified as

TimeConstraint ((CS111, CS101), (110, 111, 112, 113, 114), 100)

Since time-slot is a combination of day and hour, time constraints can be specified as combinations of day and hour constraints, if each hour is identified by a unique key and each hour of the day is identified by a unique key derived from the hour's key as discussed previously

For example the constraint

TimeConstraint ((CS111, CS101), (110, 111, 112, 113, 114), 100)

has the same meaning as

DayConstraint ((CS111, CS101), (1), 100) and
HourConstraint ((CS111, CS101), (10, 11, 12, 13, 14), 100)

The Day constraint specifies that the two sessions should be allocated the day 1, while the *Hour* constraint specifies that the two sessions should be allocated an hour between 10 and 14 inclusive. Both, Day and Hour constraints need to be satisfied simultaneously. However the advantage of using *TimeConstraint* is that it is more intuitive and since only one constraint is used, searching for and evaluating a solution is expected to be more efficient.

Moreover, it is not always to resolve a *TimeConstraint* into two other constraints. For example the constraint that the sessions CS111 and CS101 should be timetabled at any hour from 10 am to 2 pm on Monday or 11 am to 3 pm Tuesday may be stated as

TimeConstraint ((CS111, CS101), (110, 111, 112, 113, 114, 211, 212, 213, 214, 215), 100)

But this constraint does not have the same as

DayConstraint ((CS111, CS101), (1), 100)
HourConstraint ((CS111, CS101), (10, 11, 12, 13, 14), 100) and
DayConstraint ((CS111, CS101), (2), 100)

HourConstraint ((CS111, CS101), (11, 12, 13, 14, 15), 100)

Because the above specification allows combinations of Monday 3 pm and Tuesday 10 am which are not allowed.

2.3 Spread Constraints.

Spread constraints are those that dictate how the sessions should be spread over time relative to each other. Spread constraints also apply to other resources such as rooms as shown below. The various categories of spread constraints are discussed next and then it is shown how these constraints can be specified with the proposed generic spread constraint

DomainSpread(SessionList, SpreadList , Order, Hardness-factor)

where the *domain* is one of the timetabling resources or cross product of the resources with each element of the domain being denoted by a numerical value, *SessionList* is the list of sessions that need to be spread over the domain, *SpreadList* is a list of numbers that the sessions in *SessionList* should be spread by, that is the difference between the domain element allocated to a session in *SessionList* and the domain element allocated to any session appearing later in the list, should be equal to a number in the *SpreadList*. A session may need to be timetabled before or after another, for example a theory session must be held before the practical session. The value of *Order* indicates if the order of the allocated sessions is important. A value of 1 means that the order should be maintained while a value of 0 means that the order is not important, that is, if *Order* is 1 and *SessionList* is (S1, S2, S3 and S4) then S4 must be scheduled after S3 which must be scheduled after S2 which must be scheduled after S1.

For example the constraint *DomainSpread((S1, S2, S3, S4, S5), (1,2,3),1,100)* specifies that if session *Sn* is allocated a domain element *Dn,* and session *Sm* is allocated a domain element *Dm* then for $n<m$, *Dm-Dn* equals *1, 2 or 3* , where *n* and *m* indicate the position of the session in the session list.

No-Clash Constraints

A no-clash constraint specifies that a particular session *a* should not clash with another given session *b*, that is, *a* should not be timetabled at the same time as the given session *b*. Some situations which create no-clash constraints are: when a session should not be scheduled at the same time as it is a co-requisite or the two sessions are taught by the same lecturer, or two sessions have a number of common students enrolled in both. Typically sessions of the same course should clash. For example the sessions CS121L1, AF101L1, and MA131L1 should not be timetabled at the same time as they have common students. These constraints can be specified as

TimeSpread((CS121L1, AF101L1, MA131L1) ,(0), 0, -100)

which means that the given sessions must not be (as indicated by the *Hardness-factor* of –100) allocated time-slots such that the difference between any pair of time-slots is zero.

Concurrent Constraints

A concurrent constraint, the very opposite of a no clash constraint is one that specifies that sessions are required to be scheduled at the same time. For example. a 3rd year session may need to be conducted at the same time as a 4th year session where the 3rd year session is part of the 4th year session. Perhaps, the examination sessions of CS391 and CS491 need to be held at the same time. These constraint can be specified as

TimeSpread((CS391, CS491, (0), 0, 100).

Note that if the two examinations were not to be timetabled at the same time, the constraint would be specified as

TimeSpread((CS391, CS491),(0), -100)

which is exactly the same as the previous constraint except that the *Hardness-factor* is minus 100. Thus the proposed domain spread constraint mathematically captures the meaning of the no clash and concurrent constraints and their difference.

Contiguity Constraints:

Sometimes certain sessions may need to be timetabled contiguously, that is one after the other. Such constraints have also been termed as ordering constraints . For example a tutorial session may need to be held immediately after a lecture session. Also some lectures may be longer than one session. In such cases the two or more sessions should be timetabled contiguously.

If the sessions PH101P1, PH101P2, PH101P3 form one continuous laboratory activity then the constraint can be specified as

TimeSpread((PH101P1, PH101P2), (1), 1, 100)
and *TimeSpread((PH101P2, PH101P3), (1), 1, 100)*

that specify that the three PH101 sessions must be held contiguously, one after the other – PH101P2 should be allocated a time-slot immediately after PH101P1 and PH101P3 should be held immediately after PH101P2.

An important requirement in university timetabling is that certain sessions should be spread over the timetabling period. For example the lectures of a course should be

timetabled on different days of the week. In another case, the sessions may need to be distributed over the hours of a particular day.

The requirement that sessions are spread over the week is handled by the constraint

DaySpread(SessionList, DaySpreadList, Order, Hardness-factor)

that specifies that the sessions in SessionList must be timetabled such that the difference between the timetabled days of any pair of sessions in SessionList, is equal to a number in DaySpreadList which is a list of numbers. For example

DaySpread((S1, S2, S3, S4), (1,2,3,4) ,0, 100)

specifies that the sessions S1, S2, S3 and S4 must be timetabled such that they are separated by 1, 2, 3 or 4 days, that is, they can not be timetabled on the same day if DayConstraint has specified that only days 1 to 5 can be used for timetabling. Alternatively, the constraint

DaySpread((S1, S2, S3, S4),(0), 0,-100)

specifies the same constraint by stating that the difference in the timetabled day should not be zero.

Just as sessions may need to be spread over certain days, they may also need to be spread over certain hours. The constraint

HourSpread(SessionList, SpreadList, Order, Hardness-factor)

specifies that the sessions in *SessionList* must be timetabled such that the difference between the timetabled days of any pair of sessions in *SessionList* is equal to a number in *SpreadList* which is a list of numbers. For example, perhaps a part-time lecturer desires that all of the sessions CS111L1, CS111L2, CS111L3 need to be timetabled at the same hour of the day lectures either at 12 noon or at 17 hour (5 pm) The constraint

HourSpread((CS111L1, CS111L2, CS111L3), (0), 100)

specifies that the three sessions must be timetabled such that the difference between their allocated hours is zero, that is, they are timetabled at the same hour. Together with the above constraint, the following constraint

HourConstraint((CS111L1, CS111L2, CS111L3), (12, 17), 100)

specify the constraint that the sessions CS111L1, CS111L2, CS111L3 are all scheduled at 12 noon a.m. or all at 5 p.m.

Room Spread Constraints.

In *TimeSpread* constraints the spread list is a list of numbers that is the difference between the time periods allocated to sessions. A point in time is separated by another point in time by a certain number of time intervals. Similarly, a room, which is an enclosure of space, is separated from another room by certain intervals of space. Rooms can be approached as being placed on a dimensional plane. The basic unit of room space can be chosen depending on the physical construction of the room and the distance between the rooms. Generally, a basic unit of one meter should suffice.

To specify RoomSpread constraints, it is important to correctly model the location of rooms.

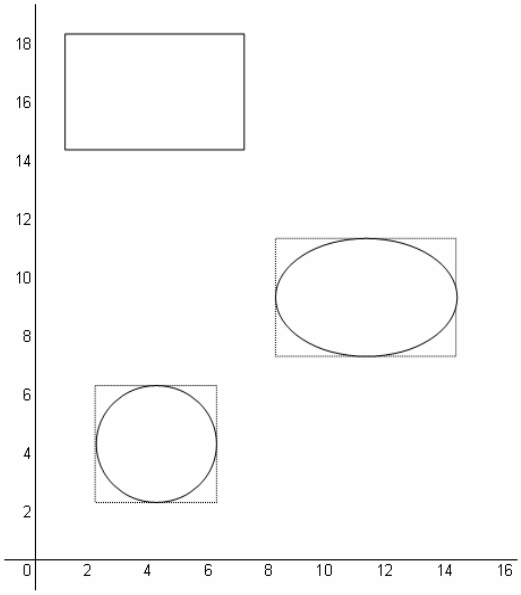

Fig. 1. The ground plan of rooms can be equated to rectangles.

Most rooms are rectangular in shape, or sometimes circular or ellipsoid or some other shape (*Figure 2.1)*. In all practical university timetabling environment the room can be equated to a rectangular block of space as the free space around the circle and the ellipse (the area between the solid and the dashed lines) are not used by other rooms can be treated as part of the room. Room locations can be specified by the positions of the centre of the rooms and the distance between the rooms calculated accordingly.

The *RoomSpread* constraint restricts the spatial spread of rooms that can be allocated to a set of sessions and is generally a soft one. Sometimes a lecturer may prefer to use only one room. It is much easier to remember that all your lectures are in one room. The constraint

RoomSpread(SessionList, SpreadList, Order ,Hardness-factor)

specifies that the distance between different rooms that are allocated to the sessions in session list must be limited to a number in *SpreadList.* For example

RoomSpread((CS121L1, CS121L2, CS121L3 CS121L4),(1,2),0, 100)

specifies that the sessions CS121L1, CS121L2, CS121L3and CS121L4 must be allocated to rooms that area not more than two units apart.

The constraint that certain sessions (say S1 and S2) must be held in the same room is specified

RoomSpread((S1, S2), (0), 0, 100)

which specifies that the distance between the rooms allocated to S1 and s2 be zero, that is, they should be the same room.

Room-TimeSpread, Room-DaySpread, Room-HourSpreadConstraints

Just as the *TimeConstraint* is an expression over the combined day and hour that can be allocated to a session, expressions over combinations of room, time, day and hour may also be stated. Of these only *RoomTimeSpread* is of common interest while others are rare in the timetabling problem. None the less the specification described below for *RoomTimeSpread* applies equally to all other combinations.

Usually only one session may be held in a room at a time. If sessions S1, S2, S3 and S4 are the four sessions that have to be timetabled and we do not wish to allocate them to the same timetable slot (same room at the same time) this requirement is fulfilled by the RoomTimeSpread constraint

RoomTimeSpread(SessionList, RoomTimeSpreadList ,Order, Hardness-factor)

which specifies that the number of different timetable slots that are allocated to the sessions in session list must be (or not be) limited to a number in *RoomTimeSpreadList*. Thus the above constraint can be specified as

RoomTimeSpreadConstraint((S1, S2, S3, S4), (0), 0, -100)

which means that sessions S1, S2, S3, S4 must not have a room time spread of zero – that is must not be timetabled in the same room at the same time.

The *RoomTime* constraint could be expressed as two separate constraints – *Room-Spread* and *TimeSpread*. The constraint

RoomTimeSpreadConstraint((S1, S2, S3, S4), (0),0, -100)

can be specified as

RoomSpreadConstraint((S1, S2, S3, S4), (0),0, -100) and
TimeSpreadConstraint((S1, S2, S3, S4), (0), 0, -100).

Recall that *TimeSpreadConstraint* can be also be specified as two separate constraints – *DaySpreadConstraint* and *HourSpreadConstraint.*

Thus *RoomTimeSpreadConstraint* may be specified as three separate constraints – *DaySpreadConstraint, HourSpreadConstraint* and *RoomSpreadConstraint.* Future research will examine and analyse the impacts that the choice of using combined or separate constraints have on building and evaluating timetables. It is not possible to always split up constraints into more basic ones. For example if a session is restrained to either Monday 9 am of Tuesday 3 pm, the constraints

DayConstraint((S1), (1,2), 100) and
HourConstraint((S1), (9,15), 100)

do not specify the requirements since the above constraints accept Monday 3 pm and Tuesday 9 am as feasible. Only

DayHourConstraint((S1), (0109, 0215), 100)

correctly specifies that only Monday 9 am and Tuesday 3 pm are acceptable.

2.3 Count Constraints.

A generic constraint

CountResource1Resource2(ResourceList1, ResourceList2, CountExpression, Hard-ness-factor)

specifies that the resources in *ResourceList1* should be allocated resources in *Resour-ceList2* such that *CountExpression,* which is a numerical expression of the number of resources in *ResourceList2* assigned to the resources in *ResourceList2*.

A lecturer may wish to have his or her sessions (say S1, S2, S3 and S4) timetabled on two week days only. This constraint is specified as

CountSessionDay((S1, S2, S3, S4), (1, 2, 3, 4, 5), (2),100).

2.4 Mathematical and Verbose Expressions of Session and Spread Lists.

In the examples above, the values of the session and spread lists have been stated explicitly. It will be tedious for a user to enter some constraints that have very large number of values. For example if a timetabling instance has 1000 sessions then it require all to explicitly all these session that have to be timetabled on a weekday:

DayConstraint((S1, S2, S3, ….), (1, 2, 3, 4, 5), 100).

However, it proposed that a user interface be developed (see further work) whereby expressions can be entered that generate the required session and spread lists. Wild-cards and regular expressions could be used to specify general cases. Thus the above constraint could be specified as

DayConstraint((S.), (<=5), 100).*

Or if only days 2 to 5 were to be used :

DayConstraint((S.), (>=2 and <=5), 100).*

3. Application of the Constraint Based Generic Model.

In this section, it is shown how the generic model can be applied to Burke's (1994) and Holtz' (1999) constraints.

3.1 Representation of Burke's Constraints

- **Resource Assignment**
  The typical resource in university timetabling is the room. If a lecturer prefers a particular room, say RX, for his course which has four sessions S1, S2, S3 and S4 then this constraint is specified as

  RoomConstraint((S1, S2, S3, S4), (RX), 100)

- **Time Assignment**
  This is a domain constraint. A session may be restricted to certain days, hours or Time Slots. For example, sessions S1 and S2 are restricted to hours 5 pm, 6 pm and 7 pm because the sessions are meant for employed people who are working during the day. Using the twenty four hour notation this constraint is specified as

HourConstraint((S1, S2), (17, 18, 19), 100)

where *100* is the upper limit of the Hardness-factor scale thus making the constraint a hard one.

- **Time Constraints between Meetings**
  Common examples of this class of constraint are that one particular lecture must take place before another one, or a set of exams must be sat simultaneously. This is a spread constraint. If sessions S1 must take place before S2 then the constraint is specified as

  TimeSpread((S1, S2), (1), 1, 100)

  which means that the difference between the time allocations for S1 and S2 must be 1, S2 must follow S1 because the value of *Order* is 1. In the case of two exams S1 and S2 that must be sat simultaneously, the constraint is

  TimeSpread((S1, S2), (0), 0, 100)

  which means that the difference between the time allocations of S1 and S2 must be 0, that is the sessions are timetabled at the same time. The value of Order does not matter.

- **Meeting Spread**
  Meetings should be spread out in time. For example, no student should have more than one exam in any day and lectures of a course should be spread over different days. This again is a spread constraint. If a student is enrolled for sessions S1, S2, and S3 of a course or sitting three different exams S1, S2 and S3 then the constraint

  DaySpread((S1, S2, S3), (0), 0, –100)

  specifies that the difference in the allocated day between any pair of sessions should not be 0, that is the three sessions should be scheduled on separated days.

- **Meeting Coherence**
  These constraints are designed to produce more organised and convenient timetables, and often run contrary to ``meeting spread" constraints (above). For example, a lecturer prefers to have all his lectures in three days, giving him two lecture-free days. If a lecturer teaches sessions S1, S2, S3, S4 and S5 then the count constraint

  CountSessionDay((S1, S2, S3, S4, S5), (1, 2, 3, 4, 5), 3 , 100)

  states that the required sessions must be allocated one of the five days and the total number of days allocated must be 3.

- **Room Capacities**

  The case of assigning rooms to sessions such that the room is able to accommodate the number of students in that session is specified as

  RoomConstraint((S1,S2,S3,S4), (R1, R4, R5), 100)

  where *R1, R4* and *R5* are the rooms with sufficient capacity.

- **Continuity**
  A continuity constraint is any whose main purpose is to ensure that certain features of student timetables are constant or predictable. For example, lectures for the same course should be scheduled in the same room, or at the same time of day." The constraint that lectures should be held in the same room is specified as

  CountSessionRoom((S1, S2), (R1, R2, R3), 0, 100)

  Which specifies that the distances between the rooms scheduled for the sessions S1 and S2 should be 0, that is the they should be the same room.

  The constraint

  HourSpread((S1, S2.), (0),0, 100)

  specifies that the difference in the hours allocated to the sessions should be 0, that is, they are timetabled at the same hour.

Thus the proposed model is able to completely describe the constraints stated by Burke (1994).

3.2 Representation of Goltz' Constraints.

Goltz *et al* (1999) state very general constraints for the Medical Faculty at the Humboldt University timetabling problem. These constraints are examined next and it is shown how they are specified in the proposed model.

C1 The teaching activities can begin every quarter of an hour and may run for different lengths of time.

This is in fact multiple constraints, the first being that activities can begin every quarter of an hour. The basic time period being used in this case is the quarter hour. To accommodate the second constraint that teaching activities may vary in length, a teaching activity may be divided into smaller sessions each one quarter of an hour in length. If a teaching activity requires *n* (e.g. 3) quarters of an hour, the activity can be divided into *n* (3) sessions each quarter of an hour long, with the constraint that the *n* (3) sessions should be held contiguously in the same room. In the proposed model, the above constraint C1 resolves into a day quarter room spread con-

straint that the sessions should be spread by a quarter hour each on the same day in the same room..

Recall that the generic domain spread constrain

DomainSpread(SessionList, SpreadList, Order, Hardness-factor)

allows one to replace *Domain* with specific domain types or cross product of domain types. The constraint:

DayQuarterRoom(SessionList, SpreadList, Order, Hardness-factor)

Thus if a teaching activity is divided into three sessions each quarter of an hour , the constraints

DayQuarterRoomSpread((S1, S2),(1),1, 100)
DayQuarterRoomSpread((S2, S3),(1),1, 100)

specify that the sessions S2 must be allocated a time period that is 1 time period away from the previous session S1 (*Order* is 1 hence the order of the sessions must be maintained) thus ensuring that S2 is timetabled immediately after S2 on the same day, and room and S3 is similarly timetabled immediately after S2.

C 2: There are restrictions with respect to the starting time for each teaching activity.

This is a domain constraint. For example, if a teaching activity's starting time is restricted to the following quarters of an hour, 1, 3, 5, the restriction can specified by the constraint

QuarterConstraint((S1, S2), (1,3,5), 100)

C 3 Two lectures about a same subject must not be scheduled for the same day.

This is a day spread constraint. If sessions S1 and S2 are lectures of the same subject then the constraint

DaySpread((S1, S2),(0), 0, -100))

specifies that the sessions must not be on the same day.

C 4 : The lectures, seminars, and practicals of each group must not overlap.

This is a time spread constraint. The required sessions S1, S2, .etc., should not be timetabled at the same time, as specified by the constraint

DayQuarterSpread((S1, S2, …), (0), 0, -100)

C 5 : The number of seminars and practicals that a department (research group) can conduct at a certain time is limited.

This statement is very generalized and meaning of this constraint is not very clear. Obviously all resources are limited in timetabling problems. The reason why sessions are limited is not stated. Is it because of limited resources, or staff? In that case domain constraints would specify the limitation. Other wise if the limit is a certain maximum number of resources then a *CountResource* constraint would represent the requirement.

C 6 : Some teaching activities are only held during certain weeks, where the following cases are possible: all weeks, A weeks (weeks with odd numbers: 1,3,5, …), B weeks (weeks with even numbers: 2,4,6, …), either A weeks or B weeks.

This is a domain constraint. If sessions S1 and S2 can be held any week then the constraint

WeekConstraint((S1,S2), (1,2,3,4,5,6), 1)

specifies that the sessions can be held any week, given that the timetabling weeks are 1 through 6.

If sessions S3 and S4 can be held on odd numbered weeks then the constraint

WeekConstraint((S3,S4), (1,3,5), 1)

specifies that the sessions can be held on an odd numbered week. Similarly if sessions S5 and S6 can be held on even numbered weeks then

the constraint

WeekConstraint((S1,S2), (2,4,6), 1)

specifies that the sessions can be held on even numbered weeks

C 7 : There are lectures about the same subject carried out in parallel at the parts of the faculty. Thus a student can choose from among parallel lectures.

This is a spread constraint that requires sessions to be held at the same time. If sessions S1 and S2 are held in parallel then the constraint

DayQuarterSpread((S1, S2), (0), 0, 100)

must have a spread of 0, that is timetabled at the same time.

C 8 : Timetabling also involves allocating rooms to the individual lectures.

This involves room domain and room spread constraints. If session S1 can be held only in room R5 then the constraint

RoomConstraint((S1), (R1), 100)

specifies that only room R1 can be allocated to session S1

C 9 : Some lectures can only be held in certain specified rooms; in some cases only one room matches the requirements.

This is a room domain constraint same as *C8.*

C 10 : The varying distances between the different locations of the teaching activities must be taken into account.

If the rooms R1, R6 and R7 are geographically suitable (situated within 100 length units)for sessions S1 and S2 then the CountResource constraint

CountSessionRoomt((S1, S2),(<=100), 100)

those rooms should be used.

C 11 : The starting time preferences for teaching activities and the preferred rooms for lectures should also be taken into account if possible.

This is a soft time and room domain constraints. If Rooms R1 and R2 are preferred for sessions S1 and S2 then the soft

RoomConstraint((S1, S2), (R1,R2), X)

where *X* is less then 100 .

Conclusion.

Timetabling constraints vary from one problem to another. These various timetabling constraints can be specified using only three generic constraints: Domain, Spread.and CountResource.

Domain Constraints

The generic domain constraint is

DomainConstraint((SessionList), (DomainList), Hardness-factor).

Domain constraints restrict the feasible domain for a session. Specific to USP timetabling , the domain constraints are –

- *Day((SessionList)t, (DayList), Hardness-factor)*
- *Hour((SessionList), (HourList), Hardness-factor)*
- *DayHour((SessionList), (TimeList), Hardness-factor)*
- *Room((SessionList), (RoomList), Hardness-factor)*

Spread Constraints

The generic spread constraint is

DomainSpread((SessionList), (DomainSpreadList), Order, Hardness-factor).

Spread constraints govern the how the feasible domain of a session is spread. The spread constraint of interest are those that involve time.

Specific to the USP timetabling environment, the spread constraints are

DaySpread((SessionList), (DaySpreadList), Order, Hardness-factor)

- *HourSpread((SessionList), (HourSpreadList), Order, Hardness-factor)*
- *DayHourSpread((SessionList), (TimeSpreadList), Order, Hardness-factor)*
- *RoomDayHourSpread((SessionList), (RoomDifferenceList), Order, Hardness-factor)*

Count Resource Constraints

A generic constraint

CountResource1Resource2(ResourceList1, ResourceList2, CountExpression, Hardness-factor)

specifies that the resources in *ResourceList1* should be allocated resources in *ResourceList2* such that *CountExpression,* which is a numerical expression of the number of resources in *ResourceList2* assigned to the resources in *ResourceList2*.

This constraint can be used to specify all limitations on the number of resources scheduled.

It may be argued that some meaning is lost by generalizing the constraints, for example *DayHourSpread((S1, S2), (0), 0, -N)* (the sessions must be scheduled at the same time) may mean that the two sessions have students in common or are taught by the same lecturer. However it should be noted that standardizing involves eliciting the general essence of a problem devoid of local causes or idiosyncrasies. The proposed model can be extended to include more verbose constraints, however at the core of timetabling solving method, they would resolve into the proposed constraints.

It has been shown that the various timetabling constraints can be specified using the proposed generic model and is applicable to both course and exam timetabling. The proposed format can be easily applied in languages such as Lisp, Prolog and constraint logic languages, as well as relational databases. The model was succesfully implemented for the University of South Pacific's examinationing timetabling problem for the last few years using Ms Access and Visual Basic.

The author hopes to build a database of instances of timetabling constraints specified in the proposed model and converters for translating data to and from the different current formats.

Work is in progress to develop a model that non-computer savvy timetablers can use to convert constraints stated in everyday verbose language into the proposed model.

BIBLIOGRAPHY.

1 **Burke E, Elliman D Weare R, 1994,** *A Genetic Algorithm for University Timetabling,* Proceedings of the AISB workshop on Evolutionary Computing, University of Leeds.

2 **Burke, E, Kingston J, Pepper P, 1998,** *A standard data format for timetabling for timetabling instances*. Practice and Theory of Automated Timetabling II Springer-Verlag LNCS

3 **Collingwood E, Ross P, Corne D, 1996,** *A Guide to GATT* University of Edinburgh.

4 **Cooper T, Kingston J, 1995,** *The Complexity of Timetable Construction Problems,* Proceedings of the First International Conference on the Practice and Theory of Automated Timetabling.

5 **Goltz H, Matzke D, 1999** University Timetabling Using Constraint Logic Programming *Practical Aspects of Declarative Languages, LNCS 1551, SpringerVerlag.*

6 **Johnson D, 1993**. *A Database Approach to Course Timetabling* Journal Operational Research Society, Vol 44.

7 **Johnson D, 2000**, *Slot Manager: a microcomputer-based decision support system for university timetabling*, Decision Support Systems Vol 27.

.

8 **Reis L, Oliveira E, 2000** *A Language for Specifying Complete Timetabling Problems* Proceedings of PATAT2000 International Conference on the Practice and Theory of Automated Timetabling.