A Unified Nurse Rostering Model Based on XHSTT

Jeffrey H. Kingston · Gerhard Post · Greet Vanden Berghe

Abstract Nurse rostering represents an interesting timetabling problem which is well known to be challenging from an optimisation perspective. Academic nurse rostering, however, lacks a generally accepted model, and therefore lacks a generally accepted format for sharing data. This paper presents XESTT, a unified model for expressing nurse rostering instances and solutions, based on the XHSTT high school timetabling model. Instances and solutions from several repositories have been converted to XESTT. Its form makes it likely that future instances could be accommodated easily. The use of XHSTT gives access to some useful existing software, including a web server for evaluating solutions, and a free, open-source solver platform.

Keywords Nurse rostering · Data models · XHSTT · XESTT

1 Introduction

A long-standing obstacle to progress in research in automated timetable construction has been the difficulty of sharing data, arising from the complexity of real-world instances of the problems. For many years, there was only one widely shared data set (the Toronto data set for university examination timetabling), and it omitted some real-world requirements, such as room requirements.

J. Kingston (`http://jeffreykingston.id.au`)
School of Information Technologies, The University of Sydney, Australia
E-mail: jeff@it.usyd.edu.au

G. Post
Houtsingel 5, 2719 EA Zoetermeer
E-mail: gerhard.post@ortec.com

G. Vanden Berghe
KU Leuven, Department of Computer Science, CODeS & iMinds - ITEC
Gebroeders De Smetstraat 1, 9000 Gent, Belgium
E-mail: greet.vandenberghe@cs.kuleuven.be

The most successful attempt so far to overcome this obstacle is XHSTT [16,17], an XML-based high school timetabling model created within the high school timetabling research community. XHSTT has many advantages: it is precisely specified; it can define real-world instances, and indeed it has been used to define many real-world instances from around the world; and it has a web server, HSEval [11], which evaluates solutions. A freely available solver platform, KHE [12], performs many useful tasks, including reading and writing instances and solutions, and evaluating solutions incrementally as they evolve during solving. Owing in part to its use in the Third International Timetabling Competition [18], the first competition to feature high school timetabling, XHSTT has become widely accepted, to a point where it can be said to have revolutionized high school timetabling research.

In nurse rostering, the current modelling situation is not as grim as it was in high school timetabling before XHSTT. The nurse rostering research community has produced several sets of real-world nurse rostering instances, and organized two competitions. But neither is the situation as good as it now is in high school timetabling. There is no widely accepted model, and no freely available solver platform. For more background, consult the surveys [7,24].

This paper leverages the high school timetabling work to define an extended version of XHSTT, called XESTT (ES stands for 'Employee Scheduling'), to model nurse rostering. Converters have been written from existing models to XESTT, and many converted instances and solutions are available [13]. The HSEval web server [11] and the KHE solver platform [12] have been extended to accept these nurse rostering instances and solutions.

2 Nurse rostering and employee scheduling

Nurse rostering is the problem of assigning shifts to the nurses of a hospital ward. Hospitals operate 24 hours a day, so there are usually at least three types of shifts: morning, afternoon, and night. Each shift demands a certain number of nurses, often with specific skills. There may be some flexibility in how many nures to assign, and the number typically changes from day to day.

Perhaps the most characteristic feature of the problem is the complex array of requirements that each nurse's timetable must satisfy. In addition to limits on total workload, rules such as 'a nurse must have a day off after a sequence of night shifts', 'a nurse may work at most four days in a row' and so on are very common, and highly variable, both between and within instances.

A nurse rostering timetable can only hope to fix a schedule for a week, or a few weeks, ahead. However, the rules just cited cannot be confined to within one planning period. That could lead to cases, for example, where a nurse works the last four days of one period and the first four days of the next, which is acceptable separately but unacceptable as a whole. Accordingly, real-world instances take account of *history*: what happened in preceding planning periods. XESTT supports history, as explained briefly in Section 6.4, and more fully in a companion paper [15].

Table 1 XHSTT's constraints, with informal definitions, grouped by what they apply to.

Event constraints	
Split Events constraint	Split event into limited number of sub-events
Distribute Split Events constraint	Split event into sub-events of limited durations
Assign Time constraint	Assign time to event
Prefer Times constraint	Assign time from given set
Spread Events constraint	Spread events evenly through the cycle
Link Events constraint	Assign same time to several events
Order Events constraint	Assign times in chronological order
Event resource constraints	
Assign Resource constraint	Assign resource to event resource
Prefer Resources constraint	Assign resource from given set
Avoid Split Assignments constraint	Assign same resource to several event resources
Resource constraints	
Avoid Clashes constraint	Avoid clashes involving resource
Avoid Unavailable Times constraint	Make resource free at given times
Limit Idle Times constraint	Limit resource's idle times
Cluster Busy Times constraint	Limit resource's busy days
Limit Busy Times constraint	Limit resource's busy times each day
Limit Workload constraint	Limit resource's total workload

Nurse rostering is a special case of the *employee scheduling* problem, which assigns shifts or other duties to employees in workplaces generally. Although the new format has been named in anticipation of these wider applications, so far it has been used only for nurse rostering. A substantial effort, going well beyond the scope of this paper, would be required to justify a claim that XESTT is a suitable model for general employee scheduling.

3 A brief introduction to XHSTT

This section introduces XHSTT, the high school timetabling format that is the starting point for our nurse rostering model.

An XHSTT instance contains four parts: the *cycle*, which is the chronological sequence of *times* that may be assigned to events; a set of *resources*, which are entities that attend events; a set of *events*, which are meetings, each of fixed duration (number of times), and containing any number of named *event resources*, each specifying one resource that attends the event; and a set of *constraints*, which specify conditions that solutions should satisfy, and the penalty costs to impose when they don't.

Sets of times, resources, and events may be defined, called *time groups*, *resource groups* and *event groups*. Each resource has one *resource type*, usually `Teachers`, `Rooms`, `Students`, or `Classes`, saying what type of resource it is.

XHSTT currently offers 16 constraint types (Table 1), specifying preferred times for events, unavailable times for resources, and so on.

Whatever its type, each constraint has a non-negative integer *weight*. When the constraint is violated, its amount of violation (a positive integer), called the *deviation*, is multiplied by the weight to produce a *cost*. Alternatively, the

deviation may be squared before multiplying by the weight. Each constraint may be marked *required*, making it a *required* or *hard* constraint, whose cost contributes to a total called the *infeasibility value* or *hard cost*. Otherwise it is *non-required* or *soft*, and its cost contributes to another total called the *objective value* or *soft cost*.

In many applications, including nurse rostering, a solution with non-zero hard cost is considered to be *infeasible*, that is, useless. High school timetablers have traditionally accepted a few violations of constraints they call hard (such as teacher clashes), reasoning that a timetable with a few major problems is better than no timetable at all. XHSTT leaves such interpretation to the user.

A solver assigns starting times to events, except for *preassigned* events (events whose starting time is given by the instance), trying to minimize primarily hard cost and secondarily soft cost. It may also be required to split events of long duration into smaller events. And it may be required to assign resources to unpreassigned event resources: often rooms, occasionally teachers, never (in practice) classes or students. A precise specification may be found online [11]; further details are given as needed throughout this paper.

XHSTT also defines a solution format: just a list of assignments of times to events (and durations for split events), and resources to event resources.

The main disadvantages of XHSTT are its verbosity, arising from the use of XML, and its complexity, which can be daunting at first. However, the designers of XHSTT took a conservative approach, adding features only as required by actual instances, so the complexity is essentially unavoidable.

4 XESTT: extending XHSTT to support nurse rostering

This section describes XESTT, our extension of XHSTT for nurse rostering. It also details three constraints that are important in nurse rostering: the limit busy times, cluster busy times, and limit active intervals constraints.

Since our extensions do not have the sanction of the high school timetabling research community, we first replace `HighSchoolTimetableArchive`, the XML tag beginning each XHSTT file, by `EmployeeScheduleArchive`. Except for this one change, every legal XHSTT file is also legal XESTT with the same meaning. The KHE platform and the HSEval web site accept either tag, as well as the XESTT extensions when the tag is `EmployeeScheduleArchive`.

A limit busy times constraint contains a resource, a time group, an integer minimum limit, and an integer maximum limit. It is violated when less than the minimum or more than the maximum number of times of the time group are busy times for the resource (times when it attends events). The amount by which the number of busy times falls short of the minimum or exceeds the maximum is multiplied by the constraint's weight to produce the cost.

Many nurse rostering constraints can be expressed as limit busy times constraints. For example, to say that some nurse can take at most one shift on some day, add a limit busy times constraint containing the nurse, a time group containing the times of the day, and maximum limit 1. This assumes that each

shift occupies one time, to be discussed in Section 5.1. The constraint is then replicated for every nurse on every day. (One limit busy times constraint can accept many time groups and entire resource groups, saving much file space.)

Here are some more examples. If a nurse works between 18 and 24 shifts, the time group is the set of all times, with minimum limit 18 and maximum limit 24. If the limit is on the number of night shifts, the time group is the set of all night times. If the limit is on the number of shifts per week, there is one constraint per week, whose time group is the times of that week. And so on.

The XHSTT specification states that a limit busy times constraint is not violated when the number of busy times is 0, regardless of the limits. In nurse rostering, this is not always wanted. Making it optional is our first extension to XHSTT. The phrase 'or else 0' is used in this paper to say that zero is allowed even when the limits do not include it.

The XHSTT cluster busy times constraint is a kind of generalization of the limit busy times constraint, in which each time is replaced by a time group. It contains a resource, a set of time groups, an integer minimum limit, and an integer maximum limit. It is violated when less than the minimum or more than the maximum number of its time groups are busy for the resource. A time group is busy for a resource when it contains at least one busy time for the resource. Costs are calculated as for limit busy times constraints.

For example, to limit a nurse to working on at most two weekends, define, for each weekend, one time group holding that weekend's times, and add a cluster busy times constraint with those time groups and maximum limit 2.

The XHSTT specification states that the case of zero busy time groups is not special: it is a violation if zero falls outside the limits. Again, for nurse rostering, this needs to be optional, and this is our second extension.

To see the need for our third extension, consider requiring a nurse's days off to occur in blocks of at least two days. This amounts to prohibiting the pattern 'busy day, then day off, then busy day'. Take three specific days, Monday, Tuesday, and Wednesday. Clearly, we need a constraint involving three time groups, for the times of Monday, Tuesday, and Wednesday. But it is not clear whether the constraint can be expressed using minimum and maximum limits on the number of busy days, because what counts about Tuesday is whether it is not busy, not whether it is busy.

Some patterns, such as the one just given, cannot be expressed in XHSTT, or can only be expressed in an artificial way (Section 5.3). Artificiality is a problem for solvers: it acts as a barrier to analysing a violation and finding ways to remove it. Accordingly, we introduce another extension to the cluster busy times constraint which allows these cases to be modelled naturally.

Instead of calling a time group *busy*, we call it *active*, and apply the minimum and maximum limits to the number of active time groups. Each time group has a *polarity*: some are *positive*, which means that they are considered to be active when they are busy, while others (and this is the extension) are *negative*: they are considered to be active when they are not busy.

The solution to the example above is now straightforward: positive Monday and Wednesday time groups, a negative Tuesday time group, and maximum

limit 2. If a maximum is exceeded, an analytical solver would aim to reduce the number of active time groups, by making positive time groups non-busy and negative time groups busy. Minimum limits need the opposite treatment.

The limit active intervals constraint (which is new to XESTT) has the same attributes as the cluster busy times constraint, notably maximum and minimum limits and a sequence of time groups with polarities. The meaning is the same except that the limits apply to the lengths of subsequences of consecutive active time groups, rather than to the total number of active time groups. This is used to impose limits on the number of consecutive busy or free days (or weekends etc.). This could be done using multiple cluster busy times constraints in a way familiar from integer programming formulations of nurse rostering, but the limit active intervals constraint is much more natural and concise, especially when expressing minimum limits.

There are several other extensions which we just mention here (see [11] for precise definitions). One allows a constraint to be repeated at intervals along the cycle (typically daily or weekly). Another allows a constraint's limits to be adjusted for each resource, to take account of *history*: what the resource did before the instance began [15,23]. The limit workload constraint may limit a resource's workload (measured in minutes, for example) over a subset of the cycle, not just over the whole cycle as in XHSTT. And there is a second new constraint, the *limit resources constraint*, described in Section 5.2.

5 Defining nurse rostering instances in XESTT

This section shows how nurse rostering instances may be defined in XESTT. It is not intended to be exhaustive; rather, it presents ideas common to many conversions. Issues raised by particular conversions are described in Section 6.

5.1 Times, resources, and events

This section gives the overall structure of nurse rostering instances, including times, resources, and events—everything, in fact, except constraints.

In high school timetabling, only the first week or fortnight is timetabled, because after that the timetable repeats. Each day of that week or fortnight typically contains between six and ten times, each representing one interval of time. The intervals are indivisible, non-overlapping, and follow chronologically after each other. The full sequence of times, over all days, is called the *cycle*.

A nurse rostering timetable does not repeat; nevertheless, we still refer to the full sequence of its times as the cycle. It consists of a sequence of days, possibly extending over several weeks. So far the two models are compatible, but when we consider the times of one day, they diverge. In nurse rostering the individual, indivisible pieces of work, called *shifts*, can occupy 8 hours or more of the day, and can overlap in time. For example, the morning shift might run from 8am to 5pm, along with an early shift running from 7am to 4pm.

One way to handle this, the way which follows the high school model most closely, is to divide the day into times by taking all the start times and end times of shifts, sorting them into chronological order, and defining one time for each non-empty interval between two start or end times. For the early and morning shifts just given, this would produce three times, representing the time intervals 7am-8am, 8am-4pm, and 4pm-5pm. The early shift would occupy the first two times, and the morning shift would occupy the last two.

While this can be done, it leads to accounting problems. Typically, nurse rostering constraints speak of numbers of shifts, while XHSTT constraints speak of numbers of times. It is much easier if each shift occupies one time.

Accordingly, a different model is used, in which each shift is mapped to its own time of day. In XESTT, the names given to times may be arbitrary. We use a simple convention for time names: each consists of a week number, then a short day name, then a number representing a shift (1 for early, 2 for morning, and so on). For example, the early shift on the Monday of the first week occurs at time `1Mon1`, the morning shift that day occurs at time `1Mon2`, and so on. All shifts have duration 1. Because of overlapping there is in reality no chronological order for the times of one day.

Incidentally, there are two concepts here: the *shift type*, which is a kind of shift such as 'the morning shift' or 'the night shift', and the *shift*, which is one piece of work, a shift type on a particular day. Our software separates these concepts, but here we follow common convention and use *shift* for both.

This model does not say that times `1Mon1` and `1Mon2` overlap, but nurses are easily prevented from taking shifts at both times, using a limit busy times constraint for these two times with maximum limit 1. The usual case of a nurse taking at most one shift per day is expressed by a limit busy times constraint containing the times of the first day and maximum limit 1, repeated each day.

The limit busy times and cluster busy times constraints are only concerned with whether a resource is busy at each time or not. If the resource has a clash at some time, in XHSTT that still only counts as one busy time. So all resources need avoid clashes constraints to forbid clashes at each time, even if a constraint is present to forbid more than one shift per day.

Time groups are used to define useful sets of times: the weekend times, the `1Mon` times, and so on. Any number of time groups may be defined, and a time may lie in any number of time groups.

High school timetabling usually has several types of resources: teachers, rooms, students, and so on. In nurse rostering, there are only nurses. So there is just one resource type, `Nurses`, and one resource for each nurse.

Resource groups are used to model nurses' skills. For example, a resource group containing the senior nurses, and another containing all nurses, may be needed. Any number of resource groups may be defined, and a resource may lie in any number of resource groups.

An event represents one piece of work done together by some resources. It contains a starting time, a duration (the number of consecutive times the event is to run), an optional workload (measured in arbitrary units, for example minutes), and any number of resources, which are required to attend the event

for its full duration. The starting time and resources may be preassigned, or left to the solver to assign; the duration and workload are fixed constants.

In our model of nurse rostering, the work is represented by one event of duration 1 for each time, preassigned that time. For example, if 3 nurses are required during time `1Mon1`, the event for that time is defined by

```
Event E-1Mon1
  Duration 1
  StartingTime 1Mon1
  3 Nurses
End Event
```

or rather, by some equivalent but verbose XML.

An event only specifies the type of each resource it requests (always `Nurses` here). Constraints are used to request particular skills, as described below.

When all events have preassigned times and duration 1, as here, the seven event constraints shown in Table 1 offer nothing useful and are not used.

5.2 Cover constraints

Cover constraints limit the number of nurses (or nurses with certain skills) assigned to each shift. This section explains how they are expressed in XESTT.

Many cover constraints can be expressed as XESTT assign resource and prefer resources constraints. For example, suppose 4 nurses are wanted at time `1Tue3`, but 3 or 5 are allowed, with a penalty. Then event `E-1Tue3` would request 5 nurses, with hard assign resource constraints for the first 3 requests (defining a hard penalty that applies when any are left unassigned), a soft assign resource constraint for the fourth, and a prefer resources constraint with the empty set of preferred resources for the fifth. This last constraint assigns a penalty when the resource request it applies to is assigned a resource outside the set of preferred resources. If that set is empty, then every assignment produces the penalty, and leaving the request unassigned avoids it.

Constraints on nurses' skills map to ordinary prefer resources constraints. For example, if one of the 4 nurses must be a senior nurse, a prefer resources constraint with resource group `SeniorNurses` is applied to the first request.

Sometimes there is no definite upper limit to the number of nurses required for a shift, merely a minimum and perhaps a preferred number. This can be handled by placing sufficiently many event resources into each event. Strictly speaking, this alters the instance by imposing an unbreakable upper limit on the number of resources that may be assigned, but if that limit is generous (say, twice the preferred number) the inexactness is not significant.

A cover constraint may apply to a particular interval of time during the day. For example, it might require at least one nurse with a particular specialty to be on duty during the interval. If several shifts are running then, such a constraint cannot be converted into constraints on individual event resources. So XESTT introduces the *limit resources constraint*, which places upper and

lower limits on the number of resources, or on the number of resources from a given resource group, that may be assigned to an arbitrary set of event resources. This models these kinds of constraints directly.

5.3 Resource constraints

Resource constraints constrain resources' timetables. They may be roughly classified into counter constraints and sequence constraints.

Counter constraints limit the number of times that something may appear in a nurse's timetable. For example, a nurse might be required to work at least 18 and at most 24 shifts. These are easily implemented in XESTT, in simple cases by limit busy times constraints (limits on the number of shifts, on the number of shifts per week, and so on), and in other cases by cluster busy times constraints (limits on the number of weekends worked, and so on).

Sequence constraints limit the number of times that something may appear *consecutively* in a nurse's timetable. For example, they might forbid three busy weekends in a row, or a night shift just before a free weekend. They can be implemented by sets of counter constraints, one for each point where the sequence could begin. Consider forbidding a morning shift following a night shift. To forbid the combination of night shift `1Mon3` with morning shift `1Tue1`, a limit busy times constraint is added for each nurse, with time group

`{1Mon3, 2Tue1}`

and maximum limit 1, and replicated for every night shift with a following morning shift. This *time window implementation* of sequence constraints is familiar from integer programming models of nurse rostering [19,20].

When the same time groups appear in adjacent windows, our conversions generate one limit active intervals constraint (Section 4) instead of many limit or cluster busy times constraints. This is likely to be significantly more efficient when solving. Forbidding three busy weekends in a row can be done this way, for example, but not forbidding a morning shift following a night shift.

Unwanted pattern constraints are sequence constraints, used to express the idea that some sequences of shifts—a morning shift on the day after a night shift, for example—are undesirable. Some of the examples given earlier can be expressed with patterns. Our aim in the remainder of this section is to model arbitrary unwanted pattern constraints. But first, some definitions.

Suppose there are three shifts per day: morning, day, and night, denoted `1`, `2`, and `3`. The presentation generalizes trivially to any number of shifts.

If a nurse takes at most one shift per day, the choices on each day are one of its shifts or nothing. Let `0` denote the absence of a shift (a free day). Using regular expression notation, an arbitrary subset of these choices is represented by enclosing them in brackets. For example, `[02]` means 'shift 2 or nothing.' A *pattern* is a sequence of these *terms*, representing the choices on consecutive days. For example, `[3][1]` means 'an early shift following a night shift'.

A pattern *matches* a nurse's timetable at any day where the nurse has a sequence of shifts or days off, each of which lies in the corresponding term of

the pattern. For example, if a nurse's timetable contains a `3` on day `1Wed` and a `1` on day `1Thu`, then pattern `[3][12]` matches that timetable at `1Wed`.

A pattern can also specify that a match is only allowed to start on certain days. For example, the pattern `Sat:[123][123]` matches a nurse's timetable at every point where the nurse works for both days on a weekend.

The empty term `[]` is not forbidden, but it never matches, which makes it useless in practice, as it turns out. On the other hand, `[0123]` says that we don't care what happens on its day, which could be useful in unusual cases.

Constraints may use patterns in several ways. Here we model *unwanted pattern constraints*: constraints containing patterns, of any length and starting at any subset of the days of the cycle, which are to be forbidden or penalized.

Arbitrary subsets of the days of the cycle are easy: just use time windows. In each time window, an arbitrary pattern can be modelled by a cluster busy times constraint with one time group for each term of the pattern. If the term does not contain `0`, the time group is a positive time group containing the times of the term. If the term contains `0`, the time group is a negative time group containing the complement of the non-zero times of the term. There is a maximum limit, equal to the number of terms minus 1.

For example, to make pattern `1Mon:[123][0][123]` unwanted, add a cluster busy times constraint with time groups

```
{1Mon1, 1Mon2, 1Mon3}
{1Tue1, 1Tue2, 1Tue3}*
{1Wed1, 1Wed2, 1Wed3}
```

and maximum limit 2. Here `*` marks negative time groups.

It seems impossible to model arbitrary unwanted patterns in unextended XHSTT. Patterns with one or more `[0]` terms and two or more other terms are the problem; we omit the details. If there is a way, it must be very artificial.

6 Converting existing instances and solutions to XESTT

We have converted instances and solutions in several formats to XESTT using a program called NRConv. NRConv and its results are available open-source at [12] and [13]. All conversions are carried out entirely by NRConv; there are no hand adjustments. This section describes the key issues that arose during each conversion. More details appear in the NRConv documentation [14].

NRConv has two layers. The lower layer defines a familiar *intermediate model* of nurse rostering: shift types, days, shifts, workers, history, and so on. The higher layer contains converters, each of which builds an instance in the intermediate model (easily done owing to the familiarity of that model), and then, in one function call, converts the intermediate instance into an XESTT instance and writes it. This simplifies the addition of new converters.

XESTT offers something not found in the nurse rostering world: the *archive*, a file holding any number of instances plus any number of sets of solutions to those instances. The converted instances and solutions are grouped into

archives. Each archive has a name, and each instance's name is adjusted to begin with its archive's name. Some long instance names have been shortened.

6.1 The BCV instances

The BCV instances are the ones with names starting with `BCV`, available, with documentation, from [6]. Their format is the oldest mentioned here, dating to about 2000, but it is also the most elaborate. The format of the First International Nurse Rostering Competition is a simplified version of it.

The BCV instances have not been converted, because the time it would take to implement their many detailed requirements is more than the authors can give. A careful study of their documentation suggests that the only problem is the `Partnerships` feature, which requires some pairs of resources to attend the same events (or to not attend the same events, but that variant is convertible, using a limit resources constraint). It seems, then, that all 19 BCV instances except the two that use this feature could be converted. Even they could be converted if an 'or else 0' option was added to the limit resources constraint.

6.2 The Curtois original instances

Curtois pioneered the assembly of instances from around the world, and their expression in a common format. Published at [6] under the heading 'Original instances', there are 28 of these instances, with 66 solutions. Following [2], we omit instance `HED01b` (it is very similar to instance `HED01`) and its solution, so we convert 27 instances and 65 solutions, to archive `COI`. For every solution, the cost reported by HSEval agrees with the cost reported by Curtois.

Four instances use the `TimePeriod` feature, which gives cover requirements for periods of the day rather than for each shift. Several other instances have complex overlapping cover requirements. Accordingly, NRConv produces a generous number of extra event resources, and where necessary it constrains them using the new limit resources constraint instead of assign resources and prefer resources constraints, as described in Section 5.2.

Two instances have *conditional constraints*, which require one pattern to match if another does, and all have *pattern constraints*, which limit the number of occurrences of the elements of a set of patterns (a more complex constraint than simply making the patterns unwanted, as in Section 5.3). These cannot be converted in general, but those in the instances can be and have been [14].

In [2], the pattern constraint is said to be widely useful, which makes it a kind of competitor for the cluster busy times constraint. A detailed comparison is beyond our scope, but the pattern constraint's orientation towards days can be awkward for other time periods. Limiting the number of busy weekends, for example, takes $2^n - 1$ patterns, where n is the number of days in one weekend.

6.3 The First International Nurse Rostering Competition

The instances of the First International Nurse Rostering Competition [8,9] have been converted into two XESTT archives, `INRC1-Long-And-Medium` and `INRC1-Sprint`. These hold the 15 long, 15 medium, and 30 sprint instances. Also included are the 60 solutions published by the GOAL group [20,21]. The competition's model is widely known, and it makes a good mean between the unrealistically simple and the tediously complex. Accordingly its conversion is described in detail in this section, as an example.

Cover constraints appear as numbers of nurses wanted for each shift, for each skill. These are easily handled, as in Section 5.2.

The `SingleAssignmentPerDay` constraint limits each nurse to at most one shift per day. It is converted to, for each day, one limit busy times constraint with maximum limit 1 whose time group contains the times of that day. Avoid clashes constraints are also needed, as explained in Section 5.1.

`MaxNumAssignments` places a maximum limit on the number of shifts that a nurse can work over the cycle. It is converted to one limit busy times constraint with the given maximum limit whose times are the full set of times of the cycle.

`MinNumAssignments` is like `MaxNumAssignments`, but with a minimum limit.

`MaxConsecutiveWorkingDays` requires busy days to occur in blocks of at most a given length l. It is just an unwanted pattern matching $l+1$ consecutive busy days, and may be converted accordingly. However, NRConv uses the new limit active intervals constraint (Section 4) to model the constraint directly.

`MinConsecutiveWorkingDays` requires busy days to occur in blocks of at least a given length l. Again, NRConv models the constraint directly with a limit active intervals constraint. There is a time windows implementation, but it is especially complex and artificial here, and extending it to support history (Section 6.4) would be very awkward. Avoiding all this complexity is a strong argument in favour of introducing the limit active intervals constraint.

`MaxConsecutiveFreeDays` requires free days to occur in blocks of length at most l. It is `MaxConsecutiveWorkingDays` with 'busy' and 'non-busy' swapped, which is done by making all the time groups negative.

`MinConsecutiveFreeDays` requires free days to occur in blocks of length at least l. It is `MinConsecutiveWorkingDays` with 'busy' and 'non-busy' swapped.

`MaxConsecutiveWorkingWeekends` and `MinConsecutiveWorkingWeekends` place limits on the number of consecutive busy weekends (weekends when a nurse works at least one shift). These are like `MaxConsecutiveWorkingDays` and `MinConsecutiveWorkingDays`, except that instead of time groups holding the times of days they use time groups holding the times of weekends.

`MaxWorkingWeekendsInFourWeeks` places a maximum limit on the number of weekends worked in any four-week period. For each set of four consecutive weekends, make a cluster busy times constraint, with time groups containing the times of those weekends, and the given maximum limit.

`AlternativeSkillCategory` defines the penalty to apply when a nurse is assigned to a shift without having the required skill. This allows each nurse to have a different penalty, which is a problem for XESTT, since its prefer re-

sources constraint (the obvious target when converting) associates the penalty with the shift, giving all unqualified nurses the same penalty.

This problem is solved as follows. For each skill s_i and each distinct non-zero weight w_j for `AlternativeSkillCategory`, let $S(s_i, w_j)$ be the set of all nurses n such that the assignment of n to a place requiring skill s_i should attract penalty w_j. Let N be the set of all nurses. For each place requiring skill s_i, define one prefer resources constraint for each non-zero weight w_j such that $S(s_i, w_j)$ is non-empty, with weight w_j and set of nurses $N - S(s_i, w_j)$. In practice this produces just one or two prefer resources constraints per skill.

`CompleteWeekends` requires a nurse to work both weekend days or neither. It is converted into, for each weekend W, one cluster busy times constraint with one time group for each day of W, and minimum limit 2 or else 0.

When weekends have three or more days, it is possible to work on the first and last days and be free in between them. The competition assigns a higher cost for such cases than for other cases of incomplete weekends. Although some of the competition instances do contain three-day weekends, we have not implemented this refinement. It can be done using unwanted patterns.

`IdenticalShiftTypesDuringWeekend` requires a nurse to either work the same shift on both days of the weekend, or to be free both days. Violations of `CompleteWeekends` also violate this constraint. It is converted, for each weekend w and shift s, to one limit busy times constraint whose time group contains the times of s on the days of w, with minimum limit 2 or else 0.

The next two constraints concern night shifts (shifts that span midnight). Night shifts are denoted `3` here, and non-night shifts are denoted `12`, although NRConv allows an arbitrary subset of the shifts to be night shifts.

`NoNightShiftBeforeFreeWeekend` requires a nurse to not work the night shift directly before a free weekend. It is equivalent to the unwanted pattern `Fri:[3][0][0]` and is converted accordingly.

`TwoFreeDaysAfterNightShifts` requires that on the two days after a night shift, a nurse should either have the day off or else work another night shift. Our solution makes patterns `[3][12][12]`, `[3][12][03]`, and `[3][0][12]` unwanted. On the second-last day of the cycle, only `[3][12]` is unwanted.

A violation on both days should cost more than a violation on one, so `[3][12][12]` should get double weight. However, the competition evaluator does not do this, so we assign the same weight to all patterns. We can then merge the first two, producing unwanted patterns `[3][12]` and `[3][0][12]`.

`UnwantedPatterns` defines unwanted patterns, converted as in Section 5.3.

A day-on request asks for work on a given day, and is modelled by a limit busy times constraint whose time group holds the day's times, with minimum limit 1. Shift-on requests are similar, with singleton time groups. Day-off and shift-off requests become avoid unavailable times constraints.

HSEval's evaluations of the 60 solutions published by the GOAL group agree with GOAL's. However, at first there were 7 discrepancies, which led the GOAL group to discover that they had posted some wrong solutions. Fixing this left 1 discrepancy, which proved to be an NRConv error, since corrected. This highlights the importance of cross checking solutions, and the advantages

of a web server like HSEval that can supply fully detailed evaluations, suitable for hand checking, to anyone with a web browser, at any time.

6.4 The Second International Nurse Rostering Competition

The Second International Nurse Rostering Competition [3, 4] has much that is familiar. We will discuss only the one significant difference here.

This is that the competition reflects the way nurse rosters are often made in reality: week by week, not all at once. A *weekly instance* is an instance covering one week; a *global instance* covers several weeks. A global instance is solved by solving a sequence of weekly instances for consecutive weeks. Each weekly instance is hidden from the solver until it has solved the previous weekly instances, and includes *history*: information about the previous solutions, such as the total workload so far of each nurse.

No file format can model all this: it involves a whole process of making instances available only after solutions to previous instances are received, with history added based on those solutions. Global instances could be generated, or sequences of weekly instances stored together in one archive (without history), but solvers would have to be trusted to solve each week without looking ahead.

NRConv produces an XESTT representation of one weekly instance, based on files in INRC1 format giving the general scenario, the week in question, and history. History can be handled by adjusting the limits of the constraints affected, using one constraint per resource. NRConv uses XESTT's history features (Section 4 and [15]) to generate a single constraint with a history adjustment for each resource, which is clearer and less verbose.

We have chosen not to add archives of particular weekly instances to our web site. Weekly instances after the first depend on history derived from the solutions for previous weeks, so cannot be produced in advance. And all weekly instances need *heuristic constraints* [15] which encourage solutions to help satisfy the two global constraints, on the total number of shifts worked and the total number of weekends worked; but opinions may differ about how to do this. So altogether it seems best for users to generate their own instances.

6.5 The 2014 Curtois and Qu instances

In 2014, Curtois and Qu produced a new set of 24 instances [5], and these, with solutions posted by Curtois at [6], have been converted into archive `CQ14`. Again, much is familiar, and we discuss only the significant differences here.

Minimum limits on consecutive busy or free days do not apply to sequences that include the first or last day. This is modelled using XESTT's history mechanism in a somewhat artificial manner.

The solutions to these instances available at [6] are not the full set reported in [5]. Accordingly we requested and received a larger set from Curtois.

As in the Curtois original instances, more nurses may be assigned to a shift than the specified optimum, and NRConv creates extra event resources to allow

for this. A complication is that some of the solutions received from Curtois, especially for the larger instances, overload shifts this way to an unreasonable degree. The worst cases occur in `Instance22.Solution.516686.roster` and `Instance22.Solution.516686_1.roster`, which assign 25 nurses to shift `d1` on day `140`, when the instance specifies an optimum cover of 1.

We have chosen to generate shifts with maximum cover $2c + 5$, where c is the optimum cover, making solutions that overload shifts beyond that point invalid. Of the 372 solutions received from Curtois, 42 were rejected for this reason. The rest were classified using metadata in the solution files into four solution groups, one for each algorithm in Table 2 of [5]. The best solution for each instance in each group was included in archive `CQ14` (67 solutions in total). HSEval's and Curtois' evaluations of these solutions agree, although HSEval's table differs from Table 2 of [5], owing to the different solutions used.

7 Conclusion

This paper has proposed a unified model for nurse rostering instances and solutions, using an extension of the XHSTT high school timetabling model called XESTT. XESTT offers several advantages which, we hope, are enough to justify its use: flexibility, consistency of design, and software support.

Four formats have been converted to XESTT, and 111 instances and 192 solutions are available at [13]. As with any new software, correctness cannot be guaranteed, but the work has been done with care, and the agreement on solution cost in every case suggests that remaining bugs are few.

Some of the converted instances, notably the Curtois original instances, are not easy to interpret in their native format. XESTT may be the best hope for ensuring that they continue to receive attention into the future.

More conversions may be undertaken in future, depending on the level of interest. The XESTT format itself is not likely to change very much. XHSTT has been stable now for several years. One desirable extension would optionally allow an event resource to be assigned any number of resources, not just 0 or 1 as at present. That would also be useful for assigning students to lectures, if the XHSTT/XESTT family is ever applied to university course timetabling.

Acknowledgement. The authors thank Túlio Toffolo for help with the GOAL solutions to the INRC1 instances, and Tim Curtois for help with his solutions to the CQ14 instances.

References

1. Edmund K. Burke, Tim Curtois, Rong Qu, and Greet Vanden-Berghe, Problem Model for Nurse Rostering Benchmark Instances, `http://www.cs.nott.ac.uk/~psztc/NRP/`
2. Edmund K. Burke and Tim Curtois, New approaches to nurse rostering benchmark instances, European Journal of Operational Research 237, 71–81 (2014)
3. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stephaan Haspeslagh, and Andrea Schaerf, Second international nurse rostering competition (INRC-II), problem description and rules. oRR abs/1501.04177 (2015). `http://arxiv.org/abs/1501.04177`

4. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stephaan Haspeslagh, and Andrea Schaerf, Second international nurse rostering competition (INRC-II) web site, http://mobiz.vives.be/inrc2/
5. Tim Curtois and Rong Qu, Computational results on new staff scheduling benchmark instances http://www.cs.nott.ac.uk/~psztc/NRP/ (2014)
6. Tim Curtois, Employee Shift Scheduling Benchmark Data Sets, http://www.cs.nott.ac.uk/~psztc/NRP/ (2016)
7. A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, Staff scheduling and rostering: a review of applications, methods, and models, European Journal of Operational Research, 153 (1), 3–27
8. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stlevik, and Andrea Schaerf, First international nurse rostering competition website, http://www.kuleuven-kortrijk.be/nrpcompetition (2010)
9. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stlevik, and Andrea Schaerf, The first international nurse rostering competition 2010, Annals of Operations Research, 218, 221–236 (2014)
10. Hujin Jin, Gerhard Post, and Egbert van der Veen, ORTECs contribution to the Second International Nurse Rostering Competition, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, August 2016), 245–262 (2016), 499–501
11. Jeffrey H. Kingston, The HSEval High School Timetable Evaluator, http://jeffreykingston.id.au/cgi-bin/hseval.cgi (2010)
12. Jeffrey H. Kingston, KHE web site, http://jeffreykingston.id.au/khe (2014)
13. Jeffrey H. Kingston, XESTT web site, http://jeffreykingston.id.au/xestt (2018)
14. Jeffrey H. Kingston, NRConv: A Nurse Rostering Converter, obtainable from http://jeffreykingston.id.au/khe (2018)
15. Jeffrey H. Kingston, Modelling history in nurse rostering, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)
16. Gerhard Post, XHSTT web site, http://www.utwente.nl/ctit/hstt/ (2011)
17. Samad Ahmadi, Sophia Daskalaki, Jeffrey H. Kingston, Jari Kyngäs, Cimmo Nurmi, Gerhard Post, David Ranson, and Henri Ruizenaar, An XML format for benchmarks in high school timetabling, Annals of Operations Research, 194, 385–397 (2012)
18. Gerhard Post, Luca Di Gaspero, Jeffrey H. Kingston, Barry McCollum, and Andrea Schaerf, The Third International Timetabling Competition, PATAT 2012 (Ninth international conference on the Practice and Theory of Automated Timetabling, Son, Norway, August 2012), 479–484 (2012)
19. Florian Mischek and Nysret Musliu, Integer programming and heuristic approaches for a multi-stage nurse rostering problem, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, August 2016), 245–262 (2016)
20. Haroldo G. Santos, Túlio A. M. Toffolo, Rafael A. M. Gomes, and Sabir Ribas, Integer programming techniques for the nurse rostering problem, Annals of Operations Research 239, 225–251 (2016)
21. Haroldo G. Santos, Túlio A. M. Toffolo, Rafael A. M. Gomes, and Sabir Ribas, http://www.goal.ufop.br/nrp/
22. Pieter Smet, Burak Bilgin, Patrick De Causmaecker, and Greet Vanden Berghe, Modelling and evaluation issues in nurse rostering, Annals of Operations Research, 218(1), 303–326 (2014)
23. Pieter Smet, Fabio Salassa, and Greet Vanden Berghe, Local and global constraint consistency in personnel rostering, International Transactions in Operational Research (2016)
24. J. Van den Bergh, J. Belien, P. De Bruecker, E. Demeulemeester and L. De Boeck, Personnel scheduling: a literature review, European Journal of Operational Research, 226(3), 367–385, (2013).