Noname manuscript No.
(will be inserted by the editor)

A Unified Nurse Rostering Model Based on XHSTT (DRAFT)

**Jeffrey H. Kingston · Gerhard Post ·
Greet Vanden Berghe**

Received: date / Accepted: date

Abstract Nurse rostering represents an interesting timetabling problem which is recognized as 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 from several repositories have been converted to XESTT. Its form makes it likely that future instances could be accommodated easily. Basing the model on 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

The main obstacle to progress in research in automated timetable construction has always 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
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 [12,13], an XML-based high school timetabling model developed 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 many countries around the world; and there is a web server, HSEval [9], which evaluates solutions. There is also a freely available solver platform, KHE [10], which carries out many useful tasks, including reading and writing instances and solutions, and efficiently evaluating solutions incrementally as they evolve during solving. Owing in part to its use in the Third International Timetabling Competition [14], the first competition to feature high school timetabling, XHSTT has become widely accepted, to a point where it can be said to have revolutionized research into high school timetabling.

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, beginning with the one assembled by Curtois [4], 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 survey papers [5,20].

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 nurse rostering models to XESTT, and a comprehensive set of converted instances is available [11]. The HSEval web server [9] and the KHE solver platform [10] have been extended to accept these nurse rostering instances and solutions.

Current state: the converter is working on instances from INRC1, INRC2, and Curtois and Qu (2014), and HSEval and KHE have been updated. The rest is still to do

2 A brief introduction to XHSTT

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 *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.

Table 1 The 16 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

Whatever its type, each constraint has a *weight*, a non-negative integer. When the constraint is violated, its amount of violation (a positive integer) is multiplied by the weight to produce a *cost*. Each constraint may be marked *required*, in which case it is a *required* or *hard* constraint, and its cost contributes to a total called the *infeasibility value* in XHSTT, and the *hard cost* here. Otherwise the constraint is *non-required* or *soft*, and its cost contributes to a different total called the *objective value* in XHSTT and the *soft cost* here.

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 [9]; further details are given as needed throughout this paper.

3 XESTT: extending XHSTT to support nurse rostering

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

Since our extensions do not have the sanction of the high school timetabling research community, we first replace `HighSchoolTimetableArchive`, the XML tag which begins each XHSTT file, by `EmployeeScheduleArchive`. Except for this one change, every legal XHSTT file is also a legal XESTT file with the same meaning. The KHE solver and the HSEval web site accept either tag, but they only accept the other extensions described here when the tag is `EmployeeScheduleArchive`.

A limit busy times constraint takes a resource, a time group, an optional integer minimum limit, and an optional integer maximum limit. The constraint is violated when less than the minimum or more than the maximum number of the times of the time group are busy times for the resource (times when the resource attends an event). The amount by which the number of busy times falls short of the minimum or exceeds the maximum is multiplied by the user-supplied weight to produce the cost.

Many nurse rostering constraints can be expressed as limit busy times constraints. For example, to say that a nurse can take at most one shift on a given day, a limit busy times constraint would be added containing that nurse, a time group containing the times of the day, and maximum limit 1. This depends on the fact that each shift occupies one time, to be discussed in Section 4.1. The constraint is then replicated for every nurse on every day. (A single limit busy times constraint can accept many time groups and entire resource groups, saving much file space.)

Here are some other examples of the use of limit busy times constraints. If a nurse must work 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. This seems to be always wanted in high school timetabling, but in nurse rostering it turns out to be sometimes wanted and sometimes not. Making this choice optional is our first extension to XHSTT. The phrase ‘or else 0’ is used in this paper to say that zero is allowed, when the limits do not include it.

The cluster busy times constraint generalizes the limit busy times constraint by replacing each time with a time group. It contains a resource, a set of time groups, an optional integer minimum limit, and an optional 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. Constraints on numbers of busy days use cluster busy times constraints with one time group for each day concerned, whose times are the times of that day.

The XHSTT specification states that the case of zero busy time groups is not special: it is a violation if 0 falls outside the limits. Again, for nurse rostering, this special case needs to be optional, and this is our second extension.

To see the need for our third and last extension, consider a constraint which requires 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 Monday and Wednesday appear positively while Tuesday appears negatively.

Some patterns, including the one above, cannot be expressed in XHSTT, or can only be expressed in an artificial way (Section 4.4). 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. Some time groups are *positive time groups*, which means that they are considered to be active when they are busy. Others (and this is the extension) are *negative time groups*: 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.

Some other extensions exploit regularity in the structure of nurse rostering problems to reduce the verbosity of the XESTT file. These add nothing fundamental, and they are described elsewhere [11], so we omit the details; but, just briefly, one option allows some constraints to be repeated at intervals along the cycle (typically daily or weekly), while another allows their limits to be adjusted for each resource, to take account of *history*: what the resource did before the instance began, as in [19]. There is also a new *limit active intervals constraint* which is like the cluster busy times constraint except that its limits apply to the lengths of subsequences of consecutive active time groups, rather than to the total number of active time groups. This can be used to impose limits on the number of consecutive busy or free days (or weekends etc.) more concisely and naturally than the time windows described below.

4 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, but rather to examine common issues in preparation for the conversion of real instances in Section 5.

4.1 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 XESTT 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. 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.

The model does not say explicitly that times `1Mon1` and `1Mon2` overlap, but it is easy to prevent nurses from being assigned 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 implemented by a limit busy times constraint containing the times of the day and maximum limit 1.

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, 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 will be 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), 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.

4.2 Coverage constraints

Coverage constraints require various minimum and maximum numbers of nurses at each time, possibly broken down by skill type. 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, a soft assign resource constraint for the fourth, and (using an artificial but efficient construction) a prefer resources constraint with an empty set of preferred resources for the fifth. This works because the prefer resources constraint assigns a penalty when the resource request it applies to is assigned a resource outside the set of preferred resources held by the constraint. If that set of resources is empty, then every assignment will produce the penalty, and leaving the request unassigned will avoid it.

Constraints on the skills of nurses map to 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. (In XHSTT, each request has a label called its *role*, and constraints may be attached to an individual request by giving its role.) Interactions between the number of nurses and their skills (such as 'if only 3 nurses are present, then 2 must be senior') cannot be modelled with the existing constraints. They do not occur in the instances considered here.

4.3 Counter and sequencing constraints

Counter constraints place limits on the total amount of something that may appear in one nurse's timetable. For example, a nurse might be required to work at least 18 and at most 24 shifts. *Sequencing constraints* place restrictions on which shifts can follow which. For example, they might forbid three night shifts in a row, or forbid a night shift immediately before a free weekend.

We have already seen cases of counter constraints implemented by limit busy times constraints: limits on the number of shifts, on the number of shifts per week, and so on. We have also seen examples that require cluster busy times constraints: limits on the number of weekends worked, and so on.

Many sequencing constraints are convertible into sets of counter constraints, one for each time in the cycle where the sequence could begin. For example, a familiar sequencing constraint forbids 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 1. This is then replicated for every night shift with a following morning shift. This *time window implementation* of sequencing constraints is familiar from integer programming models of nurse rostering [16,15]. To forbid three free days in a row, define time groups for the days:

```
1Mon = {1Mon1, 1Mon2, 1Mon3}
1Tue = {1Tue1, 1Tue2, 1Tue3}
1Wed = {1Wed1, 1Wed2, 1Wed3}
1Thu = {1Thu1, 1Thu2, 1Thu3}
1Fri = {1Fri1, 1Fri2, 1Fri3}
```

and so on. Add cluster busy times constraints whose sets of time groups are

```
{1Mon, 1Tue, 1Wed}
{1Tue, 1Wed, 1Thu}
{1Wed, 1Thu, 1Fri}
```

and so on, each with minimum 1. Three free days in a row will violate one of these constraints, four in a row will violate two of them, and so on. However, the `nrconv` program described later uses the new limit active intervals constraint (Section 3) to express these kinds of constraints directly.

4.4 Pattern constraints

Patterns are used in several formulations of nurse rostering to express the idea that some sequences of shifts—a morning shift on the day after a night shift, for example—should be forbidden, or penalized. Pattern constraints are sequencing constraints, and several of the examples given earlier can be expressed with patterns. Our aim in this section is to model arbitrary pattern constraints. But first, we have to define what we mean by a pattern.

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 time 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 at its time, which can be useful.

Patterns can be used by constraints in several ways. RosterBooster [4] has a constraint which places lower and upper limits on the number of points where the members of a set of patterns may match. The Second International Nurse Rostering Competition [1] forbids certain patterns. They are all sequences of two shifts (or days off) which may start on any day except the last.

We will now model *unwanted patterns* (patterns which are to be forbidden or penalized) of any length, applied at any subset of the days of the cycle. We will not try to model limits on the number of matches. They add another layer of complexity which can only be handled by XESTT in simple cases.

Arbitrary subsets of the days of the cycle are easy: just use time windows. Consider `Sat:[123][123]`. Suppose the cycle has four Saturdays, but that the last, `4Sat`, is the last day of the cycle, so a match cannot start there. Break the pattern into three patterns, `1Sat:[123][123]`, `2Sat:[123][123]`, and `3Sat:[123][123]`, and model them separately.

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.

A general method of making arbitrary patterns unwanted like this does not seem to exist in unextended XHSTT. We omit the details, but patterns containing one or more `[0]` terms and two or more other terms seem to be unrepresentable. If there is a way, it must be very artificial.

5 Converting existing instances to XESTT

Instances in several formats have been converted to XESTT using a program called `nrconv`. This program and its results are available open-source on this project's web site [11]. All conversions are entirely by `nrconv`; there are no hand adjustments. This section details these conversions.

The formats are handled in full generality: `nrconv` accepts any start and end dates in any years, any number of shifts per day, weekends made of any non-empty sequence of consecutive days, and so on. For brevity this generality is not always expressed below.

The `nrconv` program has two layers. The lower layer, called the *platform*, defines an *intermediate model* which supports concepts commonly found in nurse rostering models: shift types, days, workers, history, and so on. The higher layer contains the converters, each of which builds an instance in the intermediate model (usually an easy task 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 makes it easy to add new converters.

5.1 The First International Nurse Rostering Competition

The XML versions of the instances of the First International Nurse Rostering Competition [6, 7] have been converted. Following XHSTT convention, rather than one file per instance, the instances from each set (the long instances, for example) are placed into one XESTT archive file.

The competition evaluator and XESTT agree that the cost of a violation is the amount by which some value exceeds its maximum limit or falls short of its minimum limit, times the weight. Usually, then, XESTT will naturally find the right cost; but in a few cases, discussed below, care is needed.

For this first conversion we give the complete list of constraints, with brief descriptions, and say how each is converted.

Coverage constraints appear as numbers of nurses wanted for each shift, for each skill. These are handled as explained earlier (Section 4.2).

`SingleAssignmentPerDay` limits each nurse to at most one shift per day. It is converted to, for each day, one hard 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 earlier.

`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 that matches $l + 1$ consecutive busy days, and may be converted accordingly. Fortuitously, a block of $l+1$ busy days will match one pattern, a block of $l+2$ busy days will match two, and so on, ensuring that cost equals amount of violation times weight. However, `nrconv` uses the new limit active intervals constraint (Section 3) to model the constraint directly.

`MinConsecutiveWorkingDays` requires busy days to occur in blocks of at least a given length l. It may be implemented using unwanted patterns. If $l = 1$, it cannot be violated. If $l = 2$, `[0][123][0]` is unwanted. If $l = 3$, `[0][123][0]` and `[0][123][123][0]` are unwanted. And so on. Patterns with $l - k$ non-zero terms should have their weight multiplied by k, to ensure that cost equals amount of violation times weight. No two of these patterns can match the same busy day, so there can be no double counting. However, `nrconv` uses the new limit active intervals constraint (Section 3) to model the constraint directly, and indeed the complexity and artificiality of these unwanted patterns is the main reason why the limit active intervals constraint was introduced.

It seems best not to penalize blocks of busy times adjacent to either end of the cycle, since they might extend into an adjacent instance. However, nothing is said about that in the specification, so, taking the $l = 2$ case as an example, we also make `[123][0]` at the start and `[0][123]` at the end unwanted.

`MaxConsecutiveFreeDays` requires free days to occur in blocks of at most a given length l. It is handled like `MaxConsecutiveWorkingDays`, swapping ‘busy’ and ‘non-busy’.

`MinConsecutiveFreeDays` requires free days to occur in blocks of at least a given length l. It is handled like `MinConsecutiveWorkingDays`, swapping ‘busy’ and ‘non-busy’. This produces the unwanted pattern `[123][0][123]`, which is a problem for unextended XHSTT (Section 4.4).

`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 resources 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. We have not implemented this refinement. It can be done using unwanted patterns.

`IdenticalShiftTypesDuringWeekend` requires a nurse to work the same shift on both days of the weekend, if at all. It is converted into, for each weekend W, for each shift s, 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 which 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. These are converted as described in Section 4.4.

Each day-on request becomes 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.

Report on how this conversion turned out in practice, including comparing its solution evaluations with official evaluations, still to do. Use solutions from http://www.goal.ufop.br/nrp/.

5.2 The Second International Nurse Rostering Competition

The Second International Nurse Rostering Competition [1,2] has a similar model to the first, so we will discuss only the two significant differences here.

The first difference concerns coverage constraints. Minimum and optimum numbers of nurses are given for each skill on each day, but no maximum. Extra nurses above the optimum are not penalized directly. We handle this by adding extra nurse requests, above the optimum, omitting the usual assign resource constraints that penalize non-assignment. The total number of requests for each skill on each day is twice the optimum number. This is arbitrary, but it could easily be changed.

The second difference 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* is an instance covering several weeks. The idea is that a solver solves a global instance 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 instance. The last weekly instance includes global constraints, such as limits on total workload, which all weekly instances know about and should help to enforce.

XESTT has no notion of a sequence of instances connected by history, ruling out true conversion of global instances. Instead, `nrconv` produces an XESTT representation of one weekly instance, based on files giving the general scenario, the week in question, and history. History can be handled by adjusting the limits of the constraints affected on a per-resource basis, using one constraint per resource; but `nrconv` takes advantage of the XESTT history option (Section 3) to generate a single constraint with a history adjustment for each resource, which is much less verbose.

The competition includes two constraints affected by all weeks, although only enforced during the last: limits on the total number of shifts worked, and the total number of weekends worked. Weekly instances need constraints that encourage weekly solutions to help to satisfy these constraints. It is easy to see which kinds of constraints are needed, but not so easy to select limits and weights for them. `Nrconv` makes a reasonable choice: the limits increase linearly each week, reaching the true limits in the last week. History adjustments are made for both constraints. The weights are just the given weights. It would not be difficult to change `nrconv` to induce it to make other choices.

We have not tried to produce weekly instances whose solutions make things easier for following instances in other respects [8].

Report on how this conversion turned out in practice, including comparing its solution evaluations with official evaluations, still to do. Use solutions from http://www.goal.ufop.br/nrp/.

5.3 RosterBooster

The current RosterBooster model [4] is descended from the model that was adopted by the First International Nurse Rostering Competition, so it has much that is familiar. It also has some very general constraints, including some that contain Boolean conditions. These constraints cannot in general be expressed in XESTT, but nor is it clear that they have sufficiently widespread support in the nurse rostering research community to deserve inclusion.

RosterBooster's `EmployeePairings` constraint, saying that some resources should or should not work together, is more in XESTT's style. A recent audit of nurse rostering models [18] mentions it, and it also occurs in high school timetabling: some teachers have preferred 'home rooms'. XESTT does not have this constraint, but there is a commitment to add constraints when instances that require them become available, so it could be added in the future.

The authors of RosterBooster have recently published a new set of instances using a simpler model [3,4], and these (the 24 plain text instances referenced by the documentation) have been converted. Some of the instances are very large. The only new feature is that shifts have durations in minutes and there are maximum and minimum limits on the total number of minutes worked. Fortunately, XESTT can handle this: each event can be given a workload in arbitrary units, and limit workload constraints implement the limits.

This conversion, and a report on how it turned out in practice, still to do.

5.4 Instances supplied by Pieter Smet

Instances from [17,18] still to investigate and (hopefully) convert.

6 Conclusion

This paper has proposed a unified model for expressing nurse rostering instances, using an extension of the XHSTT high school timetabling model. The model, XESTT, offers several advantages which, we hope, are enough to justify its use. We conclude this paper by listing these advantages.

XESTT is comprehensive. Instances from several leading formats have been converted, without simplification, to XESTT, and they and the conversion program are available online [11].

XESTT offers preassignments. These open the way to a more general treatment of history than in other models (Section 5.2).

XESTT is consistent. There are striking inconsistencies in how constraints are expressed in other formats. Some weights appear in instance files, some only in the accompanying documentation; some constraints are listed in contracts, others outside them; and so on. In XESTT, every constraint is listed in the instance file in a uniform manner, including its hardness and weight.

XESTT is flexible. This largely arises from its ability to define arbitrary time groups for use in constraints. Constraints that seem unrelated in other models are revealed by XESTT to be variants of one idea—the cluster busy times constraint, or, in simple cases, the limit busy times constraint. The successful conversion of the First International Nurse Rostering Competition format (Section 5.1) demonstrates this flexibility.

XESTT is supported. There is a web site, HSEval [9], which offers objective evaluation of solutions, as well as other services, including timetable display and precise documentation. There is also a free, open-source solver platform, KHE [10], which reads and writes instances and solutions, and incrementally evaluates solutions as a solve proceeds.

References

1. 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). URL http://arxiv.org/abs/1501.04177
2. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stephaan Haspeslagh, and Andrea Schaerf, Second international nurse rostering competition (INRC-II) web site, URL http://mobiz.vives.be/inrc2/.
3. Tim Curtois and Rong Qu, Computational results on new staff scheduling benchmark instances (2014)
4. Tim Curtois, Employee Shift Scheduling Benchmark Data Sets, URL http://www.cs.nott.ac.uk/ psztc/NRP/ (2016)
5. A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier, Staff scheduling and rostering: a review of applications, methods, and models, European Journal of Operations Research, 153 (1), 3–27
6. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stlevik, and Andrea Schaerf, First international nurse rostering competition website, URL: http://www.kuleuven-kortrijk.be/nrpcompetition (2010)
7. 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)
8. 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, Italy, August 2016), 245–262 (2016), 499–501
9. Jeffrey H. Kingston, The HSEval High School Timetable Evaluator, URL `http://www.it.usyd.edu.au/~jeff/hseval.cgi` (2010)
10. Jeffrey H. Kingston, KHE web site, `http://www.it.usyd.edu.au/~jeff/khe` (2014)
11. Jeffrey H. Kingston, Home Page of XESTT, `http://www.it.usyd.edu.au/~jeff/xestt` (2016)
12. Gerhard Post, XHSTT web site, `http://www.utwente.nl/ctit/hstt/` (2011)
13. 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)
14. 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)
15. 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, Italy, August 2016), 245–262 (2016)
16. 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)
17. https://people.cs.kuleuven.be/pieter.smet/nurserostering.html
18. 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)
19. Pieter Smet, Fabio Salassa, and Greet Vanden Berghe, Local and global constraint consistency in personnel rostering, International Transactions in Operational Research (2016)
20. 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).