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

A Unified Nurse Rostering Model Based on XHSTT

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

Received: date / Accepted: date

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. Instances and solutions from several repositories have been converted to XESTT. The new model is based on the XHSTT high school timetabling model, and this connection gives access to some useful software, including a web server for evaluating solutions, and a free, open source solve platform.

Keywords Nurse rostering · Data models · XHSTT · XESTT

1 Introduction

A long-standing obstacle to progress in automated timetabling has been a lack of data sharing, arising from the difficulty of modelling real-world instances of the problems. This paper addresses this issue in the area of nurse rostering.

It would be wrong to assert that modelling in nurse rostering is in crisis. On the contrary, the nurse rostering research community has produced several sets of real-world nurse rostering instances, and organized two competitions. But

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: g.f.post@utwente.nl

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 situation is not ideal either. There is no widely accepted model; indeed, each step forward seems to bring a new one. Some of these models are reviewed in Section 6 of this paper; for the full background, consult the surveys [8, 26].

This paper presents a new model that is flexible enough to express all, or nearly all, existing instances. It is based on the XHSTT high school timetabling format (Section 3), so it has been named XESTT, with 'ES' for 'Employee Scheduling' replacing 'HS' for 'High School'. A program called NRConv is available for converting instances and solutions from other models to XESTT, and many converted instances and solutions are available [13]. Two pieces of open source high school timetabling software, the HSEval web server [11] and the KHE solve platform [12], have been extended to accept the XESTT format.

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 nurses to assign, and the number typically changes from day to day.

A characteristic feature 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 at least two days off after a sequence of night shifts' and 'a nurse may work at most four days in a row' are very common.

A hospital ward is a dynamic environment, and one can only hope to fix a schedule for a short distance into the future, perhaps between a week and six weeks ahead. However, rules like those just cited cannot be confined to one planning period. That could lead to cases, for example, where a nurse works the last four days of one planning period and the first four days of the next, which is acceptable separately but unacceptable as a whole. So real-world instances include *history*: information about earlier planning periods [2, 3, 24]. XESTT supports history, but it is not discussed further here, since it is the subject of a companion paper [15].

Nurse rostering is one kind of *employee scheduling*: the assignment of duties in workplaces generally [26]. A precise definition is not attempted here. XESTT has been named in anticipation of these wider applications, but so far it has been used almost entirely for nurse rostering. Due to its many constraints, nurse rostering includes most shift rostering problems [7]. Still, a substantial effort, going well beyond the scope of this paper, would be needed to justify a claim that XESTT is a suitable model for employee scheduling.

3 A brief introduction to XHSTT

This section presents XHSTT [18], the XML-based high school timetabling format that is the basis for our nurse rostering model. A complete specification may be found online [11]; further details are given as needed later in this paper.

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

XHSTT is a complex format intended for precisely specifying real-world instances and solutions of high school timetabling problems in a form suitable for input to solvers and solution evaluators. It is not intended for specifying simplified instances, or for supporting management tools for end users.

XHSTT was used by the Third International Timetabling Competition [19], the only competition so far to tackle high school timetabling. Since then the competition instances and the `XHSTT-2014` data set derived from them have been the subject of continuous study [17].

An XHSTT instance contains four parts: the chronological sequence of *times* that may be assigned to events, called the *cycle*; 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.

Arbitrary sets of times, resources, and events may be defined, called *time groups*, *resource groups* and *event groups*. Each resource has one *resource type*, usually `Teacher`, `Room`, `Student`, or `Class`, 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 is added to a total called the *infeasibility value* or *hard cost*. Otherwise it is *non-required* or *soft*, and its cost is added 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 firstly hard cost and secondly soft cost. It may also split events of long duration into shorter events. And it may assign resources to unpreassigned event resources: often rooms, occasionally teachers, never (in practice) classes or students.

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.

There are practical reasons for basing XESTT on XHSTT: it is precisely specified; it handles real-world instances; and it comes with HSEval [11], a web server which evaluates solutions, and KHE [12], an open source solve platform. Beyond those reasons, XHSTT offers at least two valuable design features.

First, anything that could produce a cost is explicitly given as a constraint, with attributes saying whether it is hard or soft and what its weight is. This improves on some existing nurse rostering models, where some constraints have a hard-wired hardness or weight, and finding the weights can be a treasure hunt through the instance file and the accompanying documentation.

Second, XHSTT allows arbitrary sets of times to be defined, and used in constraints. This has major advantages, as explained in Section 5.5.

The main disadvantages of XHSTT are its verbosity, arising from the use of XML, and its complexity, which can be daunting. However, the designers of XHSTT took a conservative approach, adding features only as required by actual instances, so the complexity is essentially unavoidable. The reader can judge these issues by studying the examples in Section 5.

4 XESTT: extending XHSTT to support nurse rostering

This section describes XESTT, our extension of XHSTT, focusing on three constraints that are central to nurse rostering: the *limit busy times* constraint, the *cluster busy times* constraint, and the *limit active intervals* constraint. The need for each extension will be justified by examples, showing that XESTT is a parsimonious extension of XHSTT.

Since our extensions concern employee scheduling, our first change is to replace `HighSchoolTimetableArchive`, the XML tag beginning 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 platform and the HSEval web site accept either tag, as well as the XESTT extensions when the tag is `EmployeeScheduleArchive`.

4.1 The limit busy times constraint

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, a non-trivial point that will be discussed fully 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 usually wrong. For example, when the constraint defines a minimum workload limit, it is not acceptable for the workload to be 0. Making it optional is our first extension to XHSTT. The phrase 'or else 0' is used in this paper to say that 0 is allowed even when the limits do not include it.

Like many constraints, the limit busy times constraint evaluates an integer-valued function of the current solution and compares its value with minimum and maximum limits. The 'or else 0' feature disrupts this simple arrangement. Although its evaluation is trivial, it complicates analytical solvers (solvers that try to improve solutions by analysing their defects): cost can be reduced, not just by moving the function value closer to the limits, but also by reducing it to 0. It would be rash to introduce further extensions along the path marked out by 'or else 0', nor do any seem to be needed.

4.2 The cluster busy times constraint

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 0 busy time groups is not special: it is a violation if 0 falls outside the limits. Again, for nurse rostering, this needs to be optional, and this is our second extension. The leading example of the need for it is the 'complete weekends' constraint, which requires a nurse to either work all days of the weekend or else 0.

To see the need for our third extension, consider requiring a nurse's days off to occur in blocks of at least two days—prohibiting the pattern 'busy day, then day off, then busy day'. Take 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.

Some patterns, such as the one just given, cannot be expressed in XHSTT, or can only be expressed in an artificial way (Section 5.5). 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 is given a *polarity*: some are *positive*, which means 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.

Without polarity, the cluster busy times constraint is very simple: just a set of time groups with limits on how many may be busy. Polarity is less intuitive, and it is fair to ask whether there are better alternatives.

Negative time groups can often be replaced by positive ones. For example, requiring a resource to be free for at least two days each week is equivalent to requiring it to be busy for at most five. The example above, prohibiting an isolated free day, would in practice be handled by the limit active intervals constraint, introduced below. But there are examples where some extension seems unavoidable. A reviewer of this paper suggested one: ensuring that five consecutive busy days are followed by at least two free days. Also, it is trivial to find examples for the limit active intervals constraint: using it to restrict the number of *consecutive* free days, free weekends, and so on, cannot be done by limiting busy days; it demands something new. Negative time groups also make conversion of arbitrary unwanted pattern constraints easy (Section 5.5). So although we cannot prove that polarity is the best approach, we can say that something is needed, and that polarity is effective and parsimonious.

4.3 The limit active intervals constraint

The limit active intervals constraint is new in XESTT. Similarly to the cluster busy times constraint, it has 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. It is used to impose limits on the number of consecutive busy or free days, weekends, and so on. This could be done by using multiple cluster busy times constraints in a way that is 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.

For example, consider a constraint requiring busy days to come in sequences of at least 3 consecutive days. This can be expressed very naturally using a single limit active intervals constraint containing one time group for each day and minimum limit 3. To do it using cluster busy times constraints takes one constraint prohibiting the pattern ‘free day, then busy day, then free day’ for each day that the pattern could begin, and one prohibiting the pattern ‘free day, then busy day, then busy day, then free day’ for each day that that pattern could begin. These short sequences of time groups are called *time windows.* Conventional integer programming models also use time windows [20, 22]. They are clumsy: a day not near the start or end of the cycle appears once in each position of these constraints (seven times in our example); it appears only once in the limit active intervals constraint. The clumsiness increases when these constraints need to be adjusted to take account of history (Section 2).

Most constraints are very easy to evaluate incrementally during solving. When the timetable of a resource changes, the affected times report themselves to the constraints concerned, which update themselves in a straightforward manner and report any change in cost to the solution, taking a small constant amount of time per change. The limit active intervals constraint is the sole exception. Active intervals must be split and merged as time groups become active and inactive. (The coding, too, is longer: in KHE, 2851 lines, compared with 1513 lines for the cluster busy times constraint.) Still, the implementation is much faster than updating the multiple constraints needed by time windows.

4.4 Other extensions

There are other extensions which we just mention here ([11] has the details). One allows a constraint to be repeated at intervals along the cycle, typically daily or weekly. Another allows a constraint to be adjusted for each resource, to take account of history (Section 2). 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 5.4.

5 Defining nurse rostering instances in XESTT

This section shows how XESTT is used in practice. It follows the four-part division of XESTT instances into times, resources, events, and constraints, further dividing constraints into *cover constraints*, specifying the number and skills of the nurses staffing each shift, and *resource constraints*, specifying rules that individual nurses' timetables should obey. The focus is on general issues. For specific issues raised by conversions from other models, see Section 6.

As a running example we use `COI-GPost`, the XESTT version of instance GPost from the Curtois original instances (Section 6.2). Here it is in outline:

```
<Instance Id="COI-GPost">
  <MetaData>
    <Name>COI-GPost</Name>
    <Date>No date</Date>
    <Country>No country</Country>
    <Description>No description</Description>
  </MetaData>
  <Times> ... </Times>
  <Resources> ... </Resources>
  <Events> ... </Events>
  <Constraints> ... </Constraints>
</Instance>
```

Here '`...`' stands for missing material that will be presented below.

5.1 Times

In high school timetabling, only the first week or fortnight is timetabled, since 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, overlapping an early shift running from 7am to 4pm.

Real time comes in intervals that need to be discretized for solving. What one time really represents, then, is an interval of time (in university course timetabling, a set of intervals) that may be assigned to an event, making the event's resources busy for that interval. So it makes sense in nurse rostering to have one time for each interval that could be assigned to a shift. It does not matter that these intervals are long, or that they overlap.

In XESTT, the names of times are arbitrary. NRConv, our converter, uses a simple convention for them: 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 Week 1 occurs at time `1Mon1`, the morning shift that day occurs at time `1Mon2`, and so on. Time groups are used to define useful sets of times: the weekend times, the `1Mon` times, and so on.

Here is the times part of instance `COI-GPost`. There are two shifts per day, so the times are named `1Mon1`, `1Mon2`, `1Tue1`, `1Tue2`, and so on. Time groups are declared at the start, then each time says which time groups it belongs to. `Week` and `Day` are synonyms for `TimeGroup`. Each named entity has an `Id` attribute for reference within the instance, and a `Name` attribute for displays.

```
<Times>
  <TimeGroups>
    <Week Id="1Mon-Sun"><Name>1Mon-Sun</Name></Week>
    <Week Id="2Mon-Sun"><Name>2Mon-Sun</Name></Week>
    <Week Id="3Mon-Sun"><Name>3Mon-Sun</Name></Week>
    <Week Id="4Mon-Sun"><Name>4Mon-Sun</Name></Week>
    <Day Id="1Mon"><Name>1Mon</Name></Day>
    ...
    <Day Id="4Sun"><Name>4Sun</Name></Day>
  </TimeGroups>
  <Time Id="1Mon1">
    <Name>1Mon1</Name>
    <Week Reference="1Mon-Sun"/>
    <Day Reference="1Mon"/>
  </Time>
  ...
  <Time Id="4Sun2">
    <Name>4Sun2</Name>
    <Week Reference="4Mon-Sun"/>
    <Day Reference="4Sun"/>
  </Time>
</Times>
```

There are 41 other time groups, for weekends etc., that have been omitted.

The time declarations do not say which time intervals the times represent. This omission is compensated for in various ways. Events have a *workload* attribute that can be used to store a duration in minutes, if needed to support constraints that limit workload in minutes. If two shifts overlap, a limit busy times constraint can prevent a nurse from taking both. The usual case of a nurse taking at most one shift per day calls for one limit busy times constraint each day, containing the times of the day and maximum limit 1.

A few existing models include time intervals, but our judgment is against that. It only makes sense if they are used, presumably by constraints like those of the UniTime university course timetabling model [21,25]. That would be a major extension, going well beyond the needs of the instances known to us.

5.2 Resources

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, `Nurse`, and one resource for each nurse.

Resource groups are used to model nurses' skills. An example would be a resource group containing the senior nurses. They are also used to model nurses' contracts. Examples here are resource groups containing the full-time nurses, and containing the part-time nurses. Any number of resource groups may be defined, and a resource may lie in any number of resource groups.

Resource groups save space when defining constraints. For example, if the full-time nurses have a maximum workload limit of 20, only one constraint, referencing the resource group of full-time nurses, is needed to say so.

Here is the resources part of instance `COI-GPost`:

```
<Resources>
  <ResourceTypes>
    <ResourceType Id="Nurse">
      <Name>Nurse</Name>
    </ResourceType>
  </ResourceTypes>
  <ResourceGroups>
    <ResourceGroup Id="Contract-FullTime">
      <Name>Contract-FullTime</Name>
      <ResourceType Reference="Nurse"/>
    </ResourceGroup>
    <ResourceGroup Id="Contract-PartTime">
      <Name>Contract-PartTime</Name>
      <ResourceType Reference="Nurse"/>
    </ResourceGroup>
  </ResourceGroups>
  <Resource Id="A">
    <Name>A</Name>
    <ResourceType Reference="Nurse"/>
    <ResourceGroups>
      <ResourceGroup Reference="Contract-FullTime"/>
    </ResourceGroups>
  </Resource>
  ...
  <Resource Id="H">
    <Name>H</Name>
    <ResourceType Reference="Nurse"/>
    <ResourceGroups>
      <ResourceGroup Reference="Contract-PartTime"/>
    </ResourceGroups>
  </Resource>
</Resources>
```

As for time groups, resource groups are declared at the start, and each resource says which resource groups it belongs to. Two resource groups, `RG:All` defining the set of all nurses, and `RG:Empty` defining the empty set of nurses, have been omitted, as have 6 of the 8 resources. Resource groups `Contract-FullTime` and `Contract-PartTime` are used in constraints that apply to full-time and part-time nurses; `RG:All` is used in constraints that apply to all nurses.

5.3 Events

An event represents one piece of work done together by some resources. It contains a starting time, a duration (the number of 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, there is one event of duration 1 for each time, preassigned that time, representing the shift that runs at that time. The term 'duration' is carried across from high school timetabling and is not really appropriate for nurse rostering. Following the discussion in Section 5.1, the meaning is merely that a single time, representing one time interval, is to be assigned. The fact that this time is preassigned expresses a basic truth about nurse rostering: the times of the shifts are fixed in advance, unlike lessons in high school timetabling, whose times are chosen by a solver.

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.

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

Event groups may be defined, like time groups and resource groups. Event groups are used in nurse rostering for defining the sets of events that cover constraints apply to.

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.

An alternative design would allow more than one resource to be assigned to an event resource, with cover constraints constraining their number. This would change the model substantially, but it might well be worthwhile for reasons other than eliminating an unimportant inexactness. When many nurses are requested it would save a significant amount of file space. And without it, solvers have to analyse constraints in order to determine whether event resources are equivalent. With it, equivalence is evident.

Here is the events part of instance `COI-GPost`, showing the event group declarations at the start, and the first event:

```
<Events>
  <EventGroups>
    <EventGroup Id="EG:All"><Name>EG:All</Name></EventGroup>
    <EventGroup Id="EG:X"><Name>EG:X</Name></EventGroup>
    <EventGroup Id="EG:A=s1000">
      <Name>EG:A=s1000</Name>
    </EventGroup>
    <EventGroup Id="EG:U=s1000">
      <Name>EG:U=s1000</Name>
    </EventGroup>
  </EventGroups>
  <Event Id="1Mon:D">
    <Name>1Mon:D</Name>
    <Duration>1</Duration>
    <Time Reference="1Mon1"/>
    <Resources>
      <R>A=s1000:1</R>
      <R>A=s1000:2</R>
      <R>A=s1000:3</R>
      <R>U=s1000:1</R>
      <R>U=s1000:2</R>
      <R>U=s1000:3</R>
      <R>U=s1000:4</R>
    </Resources>
    <EventGroups>
      <EventGroup Reference="EG:All"/>
      <EventGroup Reference="EG:A=s1000"/>
      <EventGroup Reference="EG:U=s1000"/>
    </EventGroups>
  </Event>
  ...
</Events>
```

The event groups other than `EG:All` are sets of events that contain event resources subject to certain cover constraints. Their names are arbitrary as usual; these ones have been chosen by NRConv to reflect the cover constraints. For example, `EG:A=s1000` is the set of all events that contain at least one event resource requesting a nurse such that a soft cost of 1000 is incurred if the request is not met.

The event shown, `1Mon:D`, represents the day shift on the first day of the cycle. As explained above, it has duration 1 and is preassigned its own time, `1Mon1`. Each `<R>` entry within the `<Resources>` part is one event resource—a request for one resource. The name of each event resource (‘`A=s1000:1`’ and so on) has been chosen by NRConv in a similar manner to the names of the event groups. Cover constraints given later in the instance (and appearing as examples below) say that this event wants 3 nurses, but will accept up to 7.

5.4 Cover constraints

Cover constraints limit the number of nurses (or nurses with certain skills) assigned to each shift. Many cover constraints can be expressed as XESTT assign resource and prefer resources constraints. Instance `COI-GPost` has 7 cover constraints; here are two of them:

```
<AssignResourceConstraint Id="ARC:A=s1000:1">
  <Name>Assign resource</Name>
  <Required>false</Required>
  <Weight>1000</Weight>
  <CostFunction>Linear</CostFunction>
  <AppliesTo>
    <EventGroups>
      <EventGroup Reference="EG:A=s1000"/>
    </EventGroups>
  </AppliesTo>
  <Role>A=s1000:1</Role>
</AssignResourceConstraint>

<PreferResourcesConstraint Id="PRC:U=s1000:1">
  <Name>Do not assign resource</Name>
  <Required>false</Required>
  <Weight>1000</Weight>
  <CostFunction>Linear</CostFunction>
  <AppliesTo>
    <EventGroups>
      <EventGroup Reference="EG:U=s1000"/>
    </EventGroups>
  </AppliesTo>
  <ResourceGroups>
    <ResourceGroup Reference="RG:Empty"/>
  </ResourceGroups>
  <Role>U=s1000:1</Role>
</PreferResourcesConstraint>
```

The assign resource constraint says that event resources named `A=s1000:1` within the events of event group `EG:A=s1000` should be assigned a resource, or else a soft cost of 1000 applies. The prefer resources constraint says that event resources named `U=s1000:1` within the events of event group `EG:U=s1000` should be assigned resources from resource group `RG:Empty` or nothing, or else a soft cost of 1000 applies. Since `RG:Empty` contains no resources, this amounts to a preference for leaving the event resource unassigned.

Constraints on nurses' skills map to prefer resources constraints with non-empty resource groups. If a senior nurse is wanted, say, a prefer resources constraint with resource group `SeniorNurses` is applied to the event resource.

Some models offer cover constraints that apply to particular intervals of time. For example, such a constraint might require at least one nurse with a particular skill to be on duty during the afternoons. If several shifts are running then, the constraint cannot be expressed by constraints on individual event resources, since the nurse in question could be assigned to any of those shifts. So XESTT introduces the *limit resources constraint*, which places maximum and minimum 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.

The XESTT cover constraints are effective, but they are not a significant advance over previous models. There is some new generality in allowing the limit resources constraint to apply to an arbitrary set of event resources, but that generality is not much used. Both the assign resource constraint and the prefer resources constraint are special cases of the limit resources constraint.

5.5 Resource constraints

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

Counter constraints limit the number of something that may appear in a nurse's timetable. They are implemented by cluster busy times constraints, or in simple cases by limit busy times constraints. Here is an example from `COI-GPost` that requires part-time nurses to work 2 or 3 days of Week 1:

```
<ClusterBusyTimesConstraint Id="Constraint:6">
  <Name>Shifts per week</Name>
  <Required>false</Required>
  <Weight>1</Weight>
  <CostFunction>Quadratic</CostFunction>
  <AppliesTo>
    <ResourceGroups>
      <ResourceGroup Reference="Contract-PartTime"/>
    </ResourceGroups>
  </AppliesTo>
  <TimeGroups>
    <TimeGroup Reference="1Mon"/>
    <TimeGroup Reference="1Tue"/>
    <TimeGroup Reference="1Wed"/>
    <TimeGroup Reference="1Thu"/>
    <TimeGroup Reference="1Fri"/>
    <TimeGroup Reference="1Sat"/>
    <TimeGroup Reference="1Sun"/>
  </TimeGroups>
  <Minimum>2</Minimum>
  <Maximum>3</Maximum>
</ClusterBusyTimesConstraint>
```

There is one positive time group for each day of that week.

A basic requirement of virtually all nurse rostering instances is that a nurse should work at most one shift per day. This is a counter constraint, easily implemented as mentioned earlier by one limit busy times constraint for each day. Its times are the times of its day, and it has 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, 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. This is clumsy from the perspective of nurse rostering; it is essentially a penalty paid for preserving compatibility with XHSTT.

Sequence constraints limit the number of *consecutive* appearances of things in a nurse's timetable. They can be implemented by sets of counter constraints, one for each time window (Section 4) where the sequence could begin. This example prohibits a day shift following a night shift:

```
<LimitBusyTimesConstraint Id="Constraint:4">
  <Name>Unwanted pattern</Name>
  <Required>false</Required>
  <Weight>1000</Weight>
  <CostFunction>Linear</CostFunction>
  <AppliesTo>
    <ResourceGroups>
      <ResourceGroup Reference="RG:All"/>
    </ResourceGroups>
  </AppliesTo>
  <AppliesToTimeGroup Reference="1-3Mon-Sun1:4Mon-Sat1"/>
  <TimeGroups>
    <TimeGroup Reference="1Mon2:1Tue1"/>
  </TimeGroups>
  <Minimum>0</Minimum>
  <Maximum>1</Maximum>
</LimitBusyTimesConstraint>
```

It limits to 1 the number of busy times in {`1Mon2, 1Tue1`}, the times of time group `1Mon2:1Tue1`, defined in the `COI-GPost` times part. It is repeated for each day by the `<AppliesToTimeGroup>` attribute, whose details we omit.

When the time groups in adjacent windows are compatible, in a sense that we will not pause to define, a limit active intervals constraint may replace the sets of limit busy times or cluster busy times constraints. Forbidding three consecutive busy weekends can be done this way, for example, but forbidding a morning shift following a night shift cannot. As explained in Section 4, limit active intervals constraints are likely to be significantly more efficient when solving. NRConv makes these replacements.

Here is an example from the `COI-GPost` instance which requires sequences of busy weekends to have length at most 2:

```
<LimitActiveIntervalsConstraint Id="Constraint:17">
  <Name>At most 2 consecutive busy weekends (recoded)</Name>
  <Required>false</Required>
  <Weight>1000</Weight>
  <CostFunction>Linear</CostFunction>
  <AppliesTo>
    <ResourceGroups>
      <ResourceGroup Reference="RG:All"/>
    </ResourceGroups>
  </AppliesTo>
  <TimeGroups>
    <TimeGroup Reference="1Sat-Sun"/>
    <TimeGroup Reference="2Sat-Sun"/>
    <TimeGroup Reference="3Sat-Sun"/>
    <TimeGroup Reference="4Sat-Sun"/>
  </TimeGroups>
  <Minimum>0</Minimum>
  <Maximum>2</Maximum>
</LimitActiveIntervalsConstraint>
```

For each weekend it contains one time group holding that weekend's times.

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 fit this description. We will now model arbitrary unwanted pattern constraints.

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.

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 (within the day) of the non-zero times of the term. There is a maximum limit, equal to the number of terms minus 1.

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.

We conclude this section with a brief evaluation of XESTT's handling of resource constraints. It is in this area, more than anywhere else, that XESTT advances over other models. Following XHSTT, XESTT allows arbitrary sets of times to be defined, and used in constraints. This is the key to unifying the many resource constraints found in nurse rostering: constraints on night shifts, on weekends, and so on. As we have just seen, because the constraints can reference different time groups, just three types of constraints implement all these constraints. (In fact, just two would do, because a limit busy times constraint can be replaced by a cluster busy times constraint with one time group for each time.) So XESTT has a fixed list of types of constraints, whereas in other models the list is long and in principle open-ended.

Having a fixed list of types of constraints matters for several reasons. A comparison of other models will reveal that one reason why a unified nurse rostering model has not emerged already is because of the variations between the resource constraints they offer. Also, the fixed list obviously reduces the implementation burden. Furthermore, it allows the history issue (Section 2) to receive a *complete* analysis [15], something that is not possible when the list of types of constraints is open-ended.

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 every significant issue that arose during each conversion. The NRConv documentation [14] has more details.

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 inherits from XHSTT a minor but useful feature 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 the name of each instance 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. 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 is convertible, using a limit resources constraint). So all 19 BCV instances except the two that use this feature are convertible. 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 nurse rostering instances from around the world, and their expression in a common format. Published at [6] under the heading ‘Original instances’, there are 28 instances, with 66 solutions. As in [1], 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 limit resources constraint instead of assign resource and prefer resources constraints, as described in Section 5.4.

In most instances known to us, a day off is represented by not assigning any shift on that day. One of the Curtois original instances, `COI-QMC-1`, contains a special shift type, such that assigning a shift of that type represents a day off. This allows negative time groups to be avoided in many cases, possibly all; but it is not common in our experience, and it can feel artificial. In any case XESTT accommodates it, as our conversion of `COI-QMC-1` shows.

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 5.5). These cannot be converted in general, but those in the instances can be and have been [14].

The pattern constraint is the only resource constraint we know of (outside XESTT) claimed to be general-purpose [1]. We discuss this claim now.

Clearly, the pattern constraint is more general than resource constraints often are: it can prohibit a day shift following a night shift, more than three night shifts in a row, and so on. On the other hand (using XESTT terminology) the time groups of pattern constraints are limited to subsets of the times of one day, which is awkward for other time groups. Limiting the total number of busy weekends, for example, takes $2^n - 1$ patterns, where n is the number of days in one weekend. Limiting the number of consecutive busy weekends is simple and natural with the limit active intervals constraint (Section 5.5), but it is impossible, practically speaking, with the pattern constraint.

6.3 The First International Nurse Rostering Competition

The instances of the First International Nurse Rostering Competition [9,10] 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 [22,23]. The competition's model is widely known, and it makes a good mean between the unrealistically simple and the tediously complex, so its conversion is described fully in this section, as a case study. It proceeds very smoothly and naturally.

Cover constraints appear as numbers of nurses wanted for each shift, for each skill. These are easily handled, as in Section 5.4.

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 5.5.

`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. It is converted to a single limit active intervals constraint.

`MinConsecutiveWorkingDays` is like `MaxConsecutiveWorkingDays`, but with a minimum limit.

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

`MinConsecutiveFreeDays` is like `MaxConsecutiveFreeDays`, but with a minimum limit.

`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. 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 just before a free weekend. In the notation introduced in Section 5.5 it is equivalent to 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 5.5.

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. At first there were 7 discrepancies (6 GOAL mistakes and 1 NRConv mistake), but these have all been corrected.

6.4 The Second International Nurse Rostering Competition

The Second International Nurse Rostering Competition [2,3] 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 (Section 2).

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 the competition format giving the general scenario, the week in question, and history. History is handled using XESTT's history features. These are not described here, since they are the subject of a companion paper [15].

We have chosen not to build archives of weekly instances. Weekly instances after the first depend on history derived from solutions of previous weeks, so cannot be made in advance. And weekly instances need *heuristic constraints*: constraints that encourage the solutions of weekly instances to work towards satisfying the two global constraints, on the total number of shifts worked and the total number of weekends worked; but opinions may differ about what heuristic constraints to add. For more on heuristic constraints, see [15].

Two recent papers test some 4-week and 8-week instances derived from the competition [4,16]. These are global instances, unaffected by the issues just described. They have been converted into XESTT archives `INRC2-4` and `INRC2-8`, together with solutions received from an author of one of the papers.

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 features 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 introduced a unified nurse rostering data model: XESTT, an extension of the XHSTT high school timetabling model. XESTT has several advantages which, we hope, are enough to justify its use: flexibility, consistency of design, and efficient open source software support. While we cannot assert that our design is the best possible, we can say that it is clearly and precisely specified, that it expresses existing instances exactly and naturally, and that it is parsimonious, in that very little was added to XHSTT to get to XESTT.

Four formats have been converted to XESTT, and 111 instances and 192 solutions are available at [13]. The work has been done carefully, and the total agreement on solution cost suggests that remaining bugs are few. More conversions may be undertaken in future, depending on the level of interest.

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.

Acknowledgements. The authors thank Túlio Toffolo for help with the GOAL solutions to the INRC1 instances, Andrea Schaerf and Jeremy Omer for help with their solutions to the INRC2 instances, Tim Curtois for help with his solutions to the CQ14 instances, and the anonymous reviewers.

References

1. Edmund K. Burke and Tim Curtois, New approaches to nurse rostering benchmark instances, European Journal of Operational Research 237, 71–81 (2014)
2. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stefaan 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`
3. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stefaan Haspeslagh, and Andrea Schaerf, Second international nurse rostering competition (INRC-II) web site, `http://mobiz.vives.be/inrc2/`
4. Sara Ceschia and Andrea Schaerf, Solving the INRC-II nurse rostering problem by simulated annealing based on large neighborhoods, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 331–338

5. Tim Curtois and Rong Qu, Computational results on new staff scheduling benchmark instances `http://www.cs.nott.ac.uk/~pszte/NRP/` (2014)
6. Tim Curtois, Employee Shift Scheduling Benchmark Data Sets, `http://www.cs.nott.ac.uk/~psztc/NRP/` (2016)
7. Patrick De Causmaecker and Greet Vanden Berghe, A categorisation of nurse rostering problems, Journal of Scheduling 14, 3–16 (2011)
8. 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
9. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stølevik, and Andrea Schaerf, First international nurse rostering competition website, `http://www.kuleuven-kortrijk.be/nrpcompetition` (2010)
10. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stølevik, and Andrea Schaerf, The first international nurse rostering competition 2010, Annals of Operations Research, 218, 221–236 (2014)
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), 97–111. Also Annals of Operations Research, DOI 10.1007/s10479-019-03288-x
16. A. Legrain, J. Omer, and S. Rosat, A rotation-based branch-and-price approach for the nurse scheduling problem (2017). Working paper, available at `https://hal.archives-ouvertes.fr/hal-01545421`
17. Gerhard Post, XHSTT web site, `http://www.utwente.nl/ctit/hstt/` (2011)
18. 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)
19. 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)
20. 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)
21. Tomás Müller, Hana Rudová, and Zuzana Müllerová, University course timetabling and International Timetabling Competition 2019, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 5–31
22. 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)
23. Haroldo G. Santos, Túlio A. M. Toffolo, Rafael A. M. Gomes, and Sabir Ribas, `http://www.goal.ufop.br/nrp/`
24. Pieter Smet, Fabio Salassa, and Greet Vanden Berghe, Local and global constraint consistency in personnel rostering, International Transactions in Operational Research 24, 1099-1117 (2017)
25. The Comprehensive University Timetabling System, `www.unitime.org`, especially `www.unitime.org/uct_dataformat_v24.php` (2018)
26. J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester and L. De Boeck, Personnel scheduling: a literature review, European Journal of Operational Research, 226(3), 367–385, (2013)