First International Nurse Rostering Competition 2010

Stefaan Haspeslagh[1], Patrick De Causmaecker[1],
Martin Stølevik[2], Andrea Schaerf[3]
1. CODeS, Department of Computer Science, KULeuven Campus Kortrijk
Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
{stefaan.haspeslagh,patrick.decausmaecker}@kuleuven-kortrijk.be
2. SINTEF ICT, Department of Applied Mathematics
P.O. Box 124, Blindern, NO-0314 Oslo, Norway
martin.stolevik@sintef.no
3. DIEGM, University of Udine
via delle Scienze 206, 33100, Udine, Italy
schaerf@uniud.it

May 5, 2010

Introduction

In hospitals much effort is spent producing rosters which are workable and of a high quality for their nurses. Though the Nurse Rostering Problem is known to be a difficult combinatorial optimisation problem of practical relevance, it has received ample attention mainly in recent years.

Building on the success of the two timetabling competitions, ITC2002 and ITC2007 [2], the First International Nurse Rostering Competition (INRC2010) aims to further develop interest in the general area of rostering and timetabling while providing researchers with models of the problems faced which incorporate an increased number of real world constraints.

A first important goal of INRC2010 is to generate new approaches to the associated problems by attracting users from all areas of research. As with many cases in the past, significant advancements have been made in research areas by attracting multi-disciplinary approaches and comparing them on a common ground.

The second important goal is to close the gap which currently exists between research and practice within this important area of operational research. Although for the sake of the competitive element, we do not include all aspects of the 'real world' problem, we do build on the recent developments to introduce significantly more depth and complexity.

A third goal of INRC2010 is to further stimulate debate within the widening rostering and timetabling research community.

The competition is composed of three tracks; called, after the Olympic disciplines, *1. Sprint*, *2. Middle Distance*, and *3. Long Distance*. The tracks differ

from each other based on the maximum running times and on the size of the proposed instances, whereas the problem formulation considered is the same throughout the competition. These tracks represent distinct solution settings in practice. Track 1 (Sprint) requires a solution in a few seconds, and it is meant for interactive use. Track 2 (Middle Distance) requires the solution in a few minutes and simulates the practical situation in which the problem has to be solved a few times in a solving session. Track 3 (Long Distance) grants the solver a few hours of running time and is related to overnight solving. The algorithm features are often tuned to the available running time so that the three tracks represent different challenges to the participants. For each track there are three sets of instances. The Early instances are released immediately after the competition launch. The Late instances will be made available two weeks prior to the end of the Competition on 15 May 2010. A number of instances will be kept aside in order to test the best performing algorithms. These are the Hidden datasets and will be released to the community at a later stage once the competition ends.

This document provides information needed to participate on the INRC2010. In the first section we list the competition rules, which have been elaborated starting from the rules of ITC2007. A second section describes the nurse rostering problem of the competition. Section 3 gives a description of the input and output formats of the problem instances of the competition and their solutions. Section 4 elaborates on the participant ordering and in Section 5 we give information about the machine benchmarking tool used in the competition. The appendices provide example of input and output files.

1 Competition Rules

The competition has a set of rules that the participants have to obey. The rules are the following:

Rule 1: This competition seeks to encourage research into automated nurse rostering methods, and to offer prices to the most successful methods in particular tracks. It is the spirit of these rules that is important, not the letter. With any set of rules for any competition it is possible to work within the letter of the rules but outside the spirit. The organisers ask that you: "Please don't do this". It's not fair, it's not good science, and it will result in disqualification.

Rule 2: The organisers reserve the right to disqualify any participant from the competition at any time if the participant is determined by the organisers to have worked outside the spirit of the competition rules. The organisers' decision is final in any matter. Decisions will be made democratically by the organisers.

Rule 3: The organisers reserve the right to change the rules at any time and without notice. Any change of rules will be accompanied by a general email to all participants.

Rule 4: The competition has an opening day and a deadline when all submissions must be uploaded. These deadlines are absolute and no extensions will be

given under any circumstances because to do so would be unfair to other participants.

Rule 5: Participants have to implement an algorithm to tackle the problem on a single processor machine; they can use any programming language. The use of third-party software is allowed under the following restrictions:

- it is free software
- it's behaviour is (reasonably) documented
- it runs under a commonly-used operating system (Unix/Linux or Windows)

Rule 6: The goal is to produce rosters in which all hard constraints are satisfied (i.e. feasible rosters) and to minimise the number of broken soft constraints.

Rule 7: Instances of different size and type will appear on the web site from the opening day. Two weeks before the deadline more instances will be placed on the web. A third set of datasets will be used to internally rank the top participants. The datasets are therefore classified as Early Instances, Late Instances and Hidden Instances. Participants should refer to the information associated with each track for further specifics on datasets. The Hidden Instances will be released after the competition is closed.

Rule 8: Participants have to benchmark their machine with the program provided in order to know how much time they have available to run their program on their machines.

Rule 9: The algorithms should take as input a problem file in the format described, and produce as output a solution in the described format. It should do so within the allowed CPU time. Obviously the algorithm should not take account of additional hard coded knowledge about the instance (e.g. introducing instance specific heuristics). The same version of the algorithm must be used for all instances. That is, the algorithm should not "know" which instance it is solving - while your particular algorithm might analyse the problem instance and set parameters accordingly, it should not "recognise" the particular instance. The programmer should not set different parameters for different instances although of the program is doing this automatically then this is acceptable.

Rule 10: The algorithm can be either deterministic or stochastic. In both cases, participants must be prepared to show that these results are repeatable in the given computer time. In particular, the participants that use a stochastic algorithm should code their program in such a way that the exact run that produced each solution submitted can be repeated (by recording the random seed, etc..). They can try several runs to produce each submitted solution (each with the allowed computer time), but they must be able to repeat the run for any solution submitted.

Rule 11: Participants should submit for each Early and Late Instance the best score found by their algorithm in the specified computer time, by uploading it onto the web site.

Rule 12: Participants should also submit a concise and clear description of their algorithm, so that in principle others can implement it. A template will be made available one month before the end date for this purpose. This is a fundamental part of a participants' submission.

Rule 13: For each track, a set of 5 Finalists will be chosen after the competition deadline. Ordering of participants will be based on the scores provided on the Early and Late Instances. The actual list will be based on the ranks of solvers on each single instance. The mean average of the ranks will produce the final place list. More details on how the orderings will be established can be found in Section 4.

Rule 14: The finalists, will be asked to provide the executable that will be run and tested by the organisers. The finalists' solver will be rerun by the organisers on all instances (including the Hidden ones). It is the responsibility of the participant to ensure all information is provided to enable the organisers to recreate the solution.

The solver submitted by the finalist should require as command-line arguments, input and output file names and, for stochastic solvers only, the random seed. For example (stochastic solver):

```
> my_solver.exe sprint1.txt my_solution.txt 1542955064
```

If appropriate information is not received or indeed the submitted solutions cannot be recreated, another finalist will be chosen from the original participants.

Rule 15: Finalists' eventual place listings will be based on the ranks on each single instance for a set of trials on all instances (including the Hidden ones). As with Rule 10, an explanation of the procedures to be used can be found in Section 4.

Rule 16: In some circumstances, finalists may be required to show source code to the organisers. This is simply to check that they have stuck to the rules and will be treated in the strictest confidence.

Rule 17: Entries from participating organising partners will not be permitted. However, results from participants who choose to work on the problems will be presented for comparison.

2 The Nurse Rostering Problem

The nurse rostering problem involves assigning shifts to nurses taking several constraints into account. As usual, we consider two levels of constraints:

- **hard constraints**: constraints that must be satisfied
- **soft constraints**: the sets of constraints that should be to satisfied but for which we expect that it will not be possible to satisfy them all

For example, the demand, i.e. the number of shifts to be covered per day, is a hard constraint. Personal preferences of nurses, work regulations, legal issues, ... provide the soft constraints.

A feasible solution is one in which all hard constraints are satisfied. The quality of the solution is measured in terms of soft constraint violations.

First a more detailed description of the problem is given. Second, we elaborate on the hard and soft constraints and the evaluation of the solution.

2.1 Problem Description

The problem consist of the following:

- a roster is made for a number of days for one ward in a hospital
- shift types: a shift type represents a time frame for which a nurse with a certain skill is required. E.g. between 08h30 and 16h30 a head nurse needs to be present. A night shift type is a shift type that includes midnight.
- for each day and each shift type, the number of required nurses is provided
- the set of contracts representing the work regulations of the nurses. Each nurse works according to exactly one contract. A contract provides the following information:

  maximum number of assignments :
  the maximum number of shifts that can be assigned to the nurse

  minimum number of assignments :
  the minimum number of shift that must be assigned to the nurse

  maximum number of consecutive working days :
  the maximum number of consecutive days on which a shift can be assigned to a nurse

  minimum number of consecutive working days :
  the minimum number of consecutive days on which a shift must be assigned to a nurse

  maximum number of consecutive free days :
  the maximum number of consecutive days on which a nurse has no shift assigned

  minimum number of consecutive free days :
  the minimum number of consecutive days on which a nurse has no shift assigned

  maximum number of consecutive working weekends

  maximum number of working weekends in four weeks

  the number of days off after a series of night shifts

  complete weekends : true if a nurse has to work on all days in a working weekends

  identical shift types during the weekend : true if a nurse has to work the same shift types on all days of a working weekend

unwanted shift patterns :
e.g. a nurse does not want to work the following shifts in a row: L-E-L (late-evening-late)

- the nurses of the ward
- the nurses' requests:
  - day on/off requests: a nurse can request (not) to work on a certain day
  - shift on/off requests: a nurse can request (not) to work a particular shift type on a certain day

2.2 Constrains and Evaluation Function

We now describe the hard and soft constraints. Note that our decision on which constraints are hard and which are soft is rather arbitrary. In practice many different combinations will be found. Furthermore, wards may assign different weights to certain soft constraints in an attempt to produce solutions that are more appropriate for their particular needs. Different instances of the problem can have different weights.

Below we specify the hard constraints. We then give a formal description of the soft constraints described in section 2.1 by using numberings as defined in [1]. The solution evaluator we provide implements this numbering method.

2.2.1 Hard Constraints

There are two hard constraints:

- all demanded shifts must be assigned to a nurse;
- a nurse can only work one shift per day, i.e. no two shift can be assigned to the same nurse on a day.

A feasible solution is a solution that does not violate any of those two constraints.

2.2.2 Introduction to numberings

The evaluation method for real-world nurse rostering problems[1] allows for a quick and transparent evaluation of constraints. Moreover, it provides clear feedback about the amount of violations of constraints. To describe numberings and the evaluation method, we need to introduce time units, events and non-events. Time units represent time intervals of minimum allocation. In our case, shift types determine those intervals. So, each shift type has a corresponding time unit. In the case of the nurse rostering problems we consider within this competition, the number of time units equals the number of shift types times the number of days in the planning period. We denote the set of time units by T. Now we can define a numbering:

A numbering N_i is a mapping of the set of time units to a set of numbers i.e. $N_i : T \rightarrow \{-M, -M+1, \ldots, 0, 1, \ldots, M-1, M, U\}$ where $i = 1, \ldots, I$ and I is the total number of numberings. M is a positive integer and U is a

symbol introduced to represent time units where the numbering is undefined. The mapping does not need to be into or onto, nor does it need to conserve sequence.

An event is a time unit for which a nurse has a shift type assigned. The idea of the evaluation method is to go through the set of events for which the time units do not have U (undefined) as value. We call these the 'numbered' events.

We define 8 constraint types:

1. **max_total** is an upper limit for the number of events
2. **min_total** is a lower limit for the number of events
3. **max_consecutive** is the maximum number of consecutive events
4. **min_consecutive** is the minimum number of consecutive events
5. **max_between** is the maximum gap between two non-consecutive events
6. **min_between** is the minimum gap between two non-consecutive events
7. **max_pert** is an array of size M representing for each number in the numbering the maximum number of events that can be mapped to it
8. **min_pert** is an array of size M representing for each number in the numbering the minimum number of events that must be mapped to it

More information about the numbering method we implemented can be found in appendix A

2.2.3 Soft constraint description

Maximum number of assignments Consecutive time units have consecutive numbers assigned. The variable max_total is set to the maximum number of assignments. last_nr and future_nr are Undefined.

Table 1: Maximum number of assignments

Mon		Tue		Wed		Thu		Fri		Sat		Sun	
E	L	E	L	E	L	E	L	E	L	E	L	E	L
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Minimum number of assignments The same numbering as for the maximum number of assignments constraint can be used. The variable min_total is set to the minimum number of assignments. last_nr and future_nr are Undefined.

Maximum and minimum number of consecutive working days Each time unit on the same day has the same number. The first day is assigned 0. Consecutive days have consecutive numbers. Max_consecutive is set to the maximum number of consecutive working days. Min_consecutive is set to the minimum number of consecutive working days. last_nr and future_nr are Undefined.

Table 2: Number of consecutive working days

Mon		Tue		Wed		Thu		Fri		Sat		Sun	
E	L	E	L	E	L	E	L	E	L	E	L	E	L
0	0	1	1	2	2	3	3	4	4	5	5	6	6

Maximum and minimum number of consecutive free days These two constraints use the same numbering as the number of consecutive working days constraints. Max_between is set to the maximum number of free days. Min_between is set to the minimum. last_nr is assigned -1 and future_nr is assigned $(numberofdays)$, 7 for the numbering in table 2.

Maximum number of consecutive working weekends Each time unit of the days of the same weekend has the same number. Consecutive weekends have consecutive numbers. All other time units are assigned $Undefined$. Max_consecutive is set to the maximum number of consecutive working weekends. last_nr and future_nr are Undefined.

Table 3: Number of consecutive working days

Fri		Sat		Sun		Mon		Tue	
E	L	E	L	E	L	E	L	E	L
U	U	0	0	0	0	U	U	U	U
Wed		Thu		Fri		Sat		Sun	
E	L	E	L	E	L	E	L	E	L
U	U	U	U	U	U	1	1	1	1

Complete weekends Time units on the same day are assigned the same number. Consecutive days in the same weekend have consecutive numbers. There must be a difference of at least two between the number of a weekend and the last day of the previous weekend. Min_consecutive is set to the number of days in a weekend, in our case this is three. last_nr and future_nr are Undefined. A higher penalty is raised if the working days in the weekend are not consecutive. In this case, with uniform cost, X 0 X (X = working, 0 = not working), raises a penalty of 4.

Table 4: Complete weekends

Fri			Sat			Sun			Mon			Tue		
E	L	N	E	L	N	E	L	N	E	L	N	E	L	N
0	0	0	1	1	1	2	2	2	U	U	U	U	U	U
Wed			Thu			Fri			Sat			Sun		
E	L	N	E	L	N	E	L	N	E	L	N	E	L	N
U	U	U	U	U	U	3	3	3	4	4	4	5	5	5

Identical complete weekends The time units for the same shift type on every day in the weekend are assigned the same number. Different shift types are assigned different numbers. For each assigned number, min_pert is set to the number of days in the weekend. last_nr and future_nr are Undefined.

Table 5: Identical complete weekends

Fri		Sat		Sun		Mon		Tue	
E	L	E	L	E	L	E	L	E	L
U	U	0	1	0	1	U	U	U	U
Wed		Thu		Fri		Sat		Sun	
E	L	E	L	E	L	E	L	E	L
U	U	U	U	U	U	2	3	2	3

Single assignment per day Time units on the same day are assigned the same number. Time units of different days are assigned different numbers. max_pert for each assigned number is set to one. last_nr and future_nr are Undefined.

Table 6: Single assignment per day numbering

Mon		Tue		Wed		Thu		Fri		Sat		Sun	
E	L	E	L	E	L	E	L	E	L	E	L	E	L
0	0	1	1	2	2	3	3	4	4	5	5	6	6

Two free days after a night shift The time unit of the first night shift is assigned a number. The time units of all the shift types except for that of the night shift of the following days get a consecutive number. All other time units are assigned undefined. max_consecutive is set to one. last_nr and future_nr are Undefined. For this constraint, multiple numberings are required (see appendix B).

Table 7: Two free days after a night shift

Mon			Tue			Wed		
E	L	N	E	L	N	E	L	N
U	U	0	1	1	U	1	1	U

Requested day on/off All time units of a requested day off are assigned the same number. Other days of requests are assigned a different number. All other time units are assigned undefined. max_pert for each number of each requested day off is set to zero. The numbering is the same for requested days on. min_pert is then set to one. last_nr and future_nr are Undefined. The example in table 8 illustrates three day on/off requests.

Table 8: Day off requests

Mon		Tue		Wed		Thu		Fri		Sat		Sun	
E	L	E	L	E	L	E	L	E	L	E	L	E	L
0	0	U	U	U	U	1	1	U	U	2	2	U	U

Requested shift on/off The time unit of the requested shift off is assigned a number. Time units of different shift off requests are assigned different numbers. All other time units are assigned undefined. max_pert for each assigned number is set to zero. The numbering is the same for requested shifts on. min_pert is then set to one. last_nr and future_nr are Undefined.

Table 9: Shift off requests

Mon		Tue		Wed		Thu		Fri		Sat		Sun	
E	L	E	L	E	L	E	L	E	L	E	L	E	L
0	U	U	U	U	U	U	1	U	U	2	U	U	U

Alternative skill The time units for which the employee does not have the required skill are assigned a number. Different time units are assigned different numbers. All other time units are assigned undefined. max_pert for each assigned number is set to zero. last_nr and future_nr are Undefined.

Table 10: Alternative skill - Employee cannot work shift type L

Mon		Tue		Wed		Thu		Fri		Sat		Sun	
E	L	E	L	E	L	E	L	E	L	E	L	E	L
U	0	U	1	U	2	U	3	U	4	U	5	U	6

Unwanted patterns An unwanted pattern is a sequence of assignments that a nurse does not want to work. We distinguish between patterns that are unwanted on specific days (e.g. a nurse does not want to work a night shift before a free weekend, a nurse wants to work on Friday before a working weekend, ...) and patterns that are unwanted throughout the entire planning period (e.g. a nurse does not want to work a late shift before an early shift, ...).

A pattern consists of a number of pattern entries X: $[X]_{1\ldots\ n}$. A pattern entry X can be one of the following:

- ST: a specific shift type
- W: any shift type on a day
- F: free (no shift type) on a day

A pattern entry X can occur on any day in the scheduling period or on a specific day. Some patterns need multiple numberings (see appendix B).

We introduce the following pattern types:

1. $\{W, ST\} - [F]_{2...n}$: No shift or a specific shift cannot be worked before a number of free days. E.g. a night shift cannot be worked before a free weekend. We assign the time unit of the shift type (or all time units of the day) that cannot be worked a number. The free days following get a consecutive number. *min_consecutive* is set to two. last_nr is Undefined, future_nr is assigned a consecutive number of the free days, in this case two.

Table 11: $W - F - F - F - F$

W			F			F			F			F		
E	L	N	E	L	N	E	L	N	E	L	N	E	L	N
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

Table 12: $N - F - F$: No night shift before a free weekend

Fri			Sat			Sun		
E	L	N	E	L	N	E	L	N
U	U	0	1	1	1	1	1	1

2. $F - [W, ST]_{2...n}$: No free day can occur before working any of a number of consecutive days or shift types. E.g. If an employee works a shift in a weekend, the employee should also work on Friday. The free day before the working days (shift types) is assigned a 3. The working days (shift types) following get a consecutive numbers starting from 4. *last_nr* is set to 0. *min_between* is set to two.

Table 13: $F - W - W - W - W$

F			W			W			W			W		
E	L	N	E	L	N	E	L	N	E	L	N	E	L	N
3	3	3	4	4	4	5	5	5	6	6	6	7	7	7

3. $[ST]_{2...n}$: Unwanted shift type successions. E.g. L-E-L. Consecutive shift types in the pattern are assigned consecutive numbers. The other shift types are assigned undefined. max_consecutive is set to the length of the pattern. last_nr and future_nr are Undefined.

3 Data Format

All instances will be available in both an XML-format and a text-format. Participants are free to work with the format they prefer. The same is valid for the output format. Participants can choose the output format in which they want to submit their solutions.

3.1 Input format

Each instance comes in a single file, and the files are named comp01.xml, comp02.xml, ... or comp01.txt, comp02.txt, ...

Table 14: $F - W - W$: Working on Friday if working a weekend

Fri			Sat			Sun		
E	L	N	E	L	N	E	L	N
0	0	0	1	1	1	1	1	1

Table 15: $L - E - L$

Mon			Tue			Wed		
E	L	N	E	L	N	E	L	N
U	0	U	1	U	U	U	2	U

Each xml or text file contains the following elements and attributes:

- SchedulingPeriod: representing the a rostering problem with an unique ID per instance, within a planning period.
- StartDate: the start date of the planning period
- EndDate: the end date of the planning period
- Skills: the skills nurses can have
- ShiftTypes: for each shift type an ID, StartTime, EndTime, Description and required skills are given
- Patterns: the different (unwanted) patterns used in the instance
- Contracts: representing the working regulations of employees. This sections contains the values of most of the constraints described earlier. For clarity, not all constraints are shown in the example below.
- Employees: an enumeration of the nurses that are available. Each employee has an unique ID and works according to a Contract.
- CoverRequirements:
  - Day of week cover: for each day of the week and for each shift type, the required amount of nurses per day is specified.
  - Date specific cover: for a specific date and a shift type, the required amount of nurses per day is specified. This amount has preference over the day of week cover requirement!
- DayOffRequests: for each request not to work, the employee and date is given.
- DayOnRequests: for each request to work, the employee and date is given
- ShiftOffRequests: for each request, the employee, the shift type and date is given.
- ShiftOnRequests: for each request, the employee, the shift type and date is given.

3.1.1 XML

An XML-schema (.xsd) is available to describe the structure of the instances:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:annotation>
  <xs:documentation>
      Schema for personnel scheduling problems.
    </xs:documentation>
 </xs:annotation>
 <xs:element name="SchedulingPeriod">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="StartDate" type="xs:date"/>
    <xs:element name="EndDate" type="xs:date"/>
    <xs:element name="Skills" type="Skills" minOccurs="0"/>
    <xs:element name="ShiftTypes" type="ShiftTypes"/>
    <xs:element name="Patterns" type="Patterns" minOccurs="0"/>
    <xs:element name="Contracts" type="Contracts"/>
    <xs:element name="Employees" type="Employees"/>
    <xs:element name="CoverRequirements" type="CoverRequirements"/>
    <xs:element name="DayOffRequests" type="DayOffRequests" minOccurs="0"/>
    <xs:element name="DayOnRequests" type="DayOnRequests" minOccurs="0"/>
    <xs:element name="ShiftOffRequests" type="ShiftOffRequests" minOccurs="0"/>
    <xs:element name="ShiftOnRequests" type="ShiftOnRequests" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="ID" type="xs:string" use="required"/>
   <xs:attribute name="OrganisationID" type="xs:string" use="optional"/>
  </xs:complexType>
 </xs:element>
 <xs:complexType name="Skills">
  <xs:sequence>
   <xs:element name="Skill" type="ID" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="ShiftTypes">
  <xs:sequence>
   <xs:element name="Shift" maxOccurs="unbounded">
    <xs:complexType>
     <xs:all>
      <xs:element name="StartTime" type="xs:time"/>
      <xs:element name="EndTime" type="xs:time"/>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
      <xs:element name="Skills" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Skill" type="ID" maxOccurs="unbounded"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
```

```
    </xs:all>
    <xs:attribute name="ID" type="ID" use="required"/>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="Patterns">
 <xs:sequence maxOccurs="unbounded">
  <xs:element name="Pattern">
   <xs:complexType>
    <xs:all>
     <xs:element name="PatternEntries">
      <xs:complexType>
       <xs:sequence minOccurs="2" maxOccurs="unbounded">
        <xs:element name="PatternEntry">
         <xs:complexType>
          <xs:all>
           <xs:element name="ShiftType" type="xs:string"/>
           <xs:element name="Day" type="WeekDayOrAny"/>
          </xs:all>
          <xs:attribute name="index"/>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:all>
    <xs:attribute name="weight"/>
    <xs:attribute name="ID" type="xs:string"/>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="Contracts">
 <xs:sequence>
  <xs:element name="Contract" maxOccurs="unbounded">
   <xs:complexType>
    <xs:all>
     <xs:element name="SingleAssignmentPerDay"
                 type="WeightOnly" minOccurs="0"/>
     <xs:element name="MaxNumAssignments"
                 type="OnAndWeight" minOccurs="0"/>
     <xs:element name="MinNumAssignments"
                 type="OnAndWeight" minOccurs="0"/>
     <xs:element name="MaxConsecutiveWorkingDays"
                 type="OnAndWeight" minOccurs="0"/>
     <xs:element name="MinConsecutiveWorkingDays"
                 type="OnAndWeight" minOccurs="0"/>
     <xs:element name="MaxConsecutiveFreeDays"
                 type="OnAndWeight" minOccurs="0"/>
```

```
      <xs:element name="MinConsecutiveFreeDays"
                  type="OnAndWeight" minOccurs="0"/>
      <xs:element name="MaxConsecutiveWorkingWeekends"
                  type="OnAndWeight" minOccurs="0"/>
      <xs:element name="MinConsecutiveWorkingWeekends"
                  type="OnAndWeight" minOccurs="0"/>
      <xs:element name="MaxWorkingWeekendsInFourWeeks"
                  type="OnAndWeight" minOccurs="0"/>
      <xs:element name="WeekendDefinition"
                  type="Weekend" minOccurs="0"/>
      <xs:element name="CompleteWeekends"
                  type="WeightOnly" minOccurs="0"/>
      <xs:element name="IdenticalShiftTypesDuringWeekend"
                  type="WeightOnly" minOccurs="0"/>
      <xs:element name="NoNightShiftBeforeFreeWeekend"
                  type="WeightOnly" minOccurs="0"/>
      <xs:element name="TwoFreeDaysAfterNightShifts"
                  type="WeightOnly" minOccurs="0"/>
      <xs:element name="AlternativeSkillCategory"
                  type="WeightOnly" minOccurs="0"/>
      <xs:element name="UnwantedPatterns" minOccurs="0">
       <xs:complexType>
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
         <xs:element name="Pattern" type="ID"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="Description"/>
     </xs:all>
     <xs:attribute name="ID" type="ID" use="required"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:complexType name="Employees">
  <xs:sequence>
   <xs:element name="Employee" maxOccurs="unbounded">
    <xs:complexType>
     <xs:all>
      <xs:element name="ContractID" type="ID"/>
      <xs:element name="Name" type="xs:string" minOccurs="0"/>
      <xs:element name="Skills" minOccurs="0">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Skill" type="xs:string" maxOccurs="unbounded"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:all>
     <xs:attribute name="ID" type="ID" use="required"/>
```

```
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="ConstraintAttributes">
 <xs:attribute name="on" type="xs:boolean" use="optional"/>
 <xs:attribute name="weight" type="xs:nonNegativeInteger" use="optional"/>
</xs:complexType>
<xs:complexType name="OnAndWeight">
 <xs:simpleContent>
  <xs:extension base="xs:nonNegativeInteger">
   <xs:attribute name="on" type="xs:boolean" use="optional"/>
   <xs:attribute name="weight" type="xs:nonNegativeInteger" use="optional"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="WeightOnly">
 <xs:simpleContent>
  <xs:extension base="xs:boolean">
   <xs:attribute name="weight" type="xs:nonNegativeInteger" use="optional"/>
  </xs:extension>
 </xs:simpleContent>
</xs:complexType>
<xs:complexType name="CoverRequirements">
 <xs:choice minOccurs="0" maxOccurs="unbounded">
  <xs:element name="DayOfWeekCover">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="Day" type="WeekDay"/>
     <xs:element name="Cover" type="Cover" maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="DateSpecificCover">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="Date" type="xs:date"/>
     <xs:element name="Cover" type="Cover" maxOccurs="unbounded"/>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:choice>
</xs:complexType>
<xs:complexType name="Cover">
 <xs:sequence>
  <xs:choice>
   <xs:element name="Shift" type="ID"/>
  </xs:choice>
  <xs:element name="Preferred" type="xs:nonNegativeInteger" minOccurs="0"/>
 </xs:sequence>
```

```
</xs:complexType>
<xs:complexType name="DayOffRequests">
 <xs:sequence>
  <xs:element name="DayOff" minOccurs="0" maxOccurs="unbounded">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="EmployeeID" type="ID"/>
     <xs:element name="Date" type="xs:date"/>
    </xs:sequence>
    <xs:attribute name="weight" type="xs:nonNegativeInteger" use="required"/>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="DayOnRequests">
 <xs:sequence>
  <xs:element name="DayOn" minOccurs="0" maxOccurs="unbounded">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="EmployeeID" type="ID"/>
     <xs:element name="Date" type="xs:date"/>
    </xs:sequence>
    <xs:attribute name="weight" type="xs:nonNegativeInteger" use="required"/>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="ShiftOffRequests">
 <xs:sequence>
  <xs:element name="ShiftOff" minOccurs="0" maxOccurs="unbounded">
   <xs:complexType>
    <xs:sequence>
     <xs:element name="ShiftTypeID" type="ID"/>
     <xs:element name="EmployeeID" type="ID"/>
     <xs:element name="Date" type="xs:date"/>
    </xs:sequence>
    <xs:attribute name="weight" type="xs:nonNegativeInteger" use="required"/>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
<xs:complexType name="ShiftOnRequests">
 <xs:sequence>
  <xs:element name="ShiftOn" minOccurs="0" maxOccurs="unbounded">
   <xs:complexType>
    <xs:sequence>
     <xs:choice>
      <xs:element name="ShiftTypeID" type="ID"/>
     </xs:choice>
     <xs:element name="EmployeeID" type="ID"/>
```

```
      <xs:element name="Date" type="xs:date"/>
     </xs:sequence>
     <xs:attribute name="weight" type="xs:nonNegativeInteger" use="required"/>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
 <xs:simpleType name="WeekDay">
  <xs:restriction base="xs:string">
   <xs:enumeration value="Sunday"/>
   <xs:enumeration value="Monday"/>
   <xs:enumeration value="Tuesday"/>
   <xs:enumeration value="Wednesday"/>
   <xs:enumeration value="Thursday"/>
   <xs:enumeration value="Friday"/>
   <xs:enumeration value="Saturday"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="WeekDayOrAny">
  <xs:restriction base="xs:string"/>
 </xs:simpleType>
 <xs:simpleType name="Weekend">
  <xs:restriction base="xs:string">
   <xs:enumeration value="SaturdaySunday"/>
   <xs:enumeration value="FridaySaturdaySunday"/>
   <xs:enumeration value="FridaySaturdaySundayMonday"/>
   <xs:enumeration value="SaturdaySundayMonday"/>
  </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="ID">
  <xs:restriction base="xs:string">
   <xs:pattern value="([a-zA-Z0-9._])+"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

3.1.2 Text

The template for the text format is listed below. The value of `boolean` is 0 or 1, `int+` means a positive integer. Furthermore, when an element refers to another element, capital letters are used. E.g. the type of required skill(s) for a shift type is given under `SHIFT_TYPES` by the element '`Skill`', which is of type '`SKILLS.Skill`'. This means that the type of shift type skill is a string (same as Skill under SKILLS) and that the value of `SHIFT_TYPES.Skill` equals a value listed under `SKILLS`.

The percent sign, %, is used in the text template to indicate the different elements of a "mother" element (which is typically written in CAPITAL letters. All lines starting with '%' will be replaced by data in case files (see Appendix

X) of the type given in brackets '< >'.

```
SCHEDULING_PERIOD
% ID <string>,
% StartDate <YYYY-MM-DD>,
% EndDate <YYYY-MM-DD>;

SKILLS = n1
% Skill <string>;

SHIFT_TYPES = n2
% ID <string>,
% Description <string>,
% StartTime <HH:MM:SS>,
% EndTime <HH:MM:SS>,
% NumberOfRequiredSkills <int>,
% RequiredSkill <SKILLS.Skill>; // separate by space, not comma.

CONTRACTS = n3
% ID                              <string>,
% Description                     <string>,
% SingleAssignmentPerDay          (on|weight)       (<boolean>|<int>),
% MaxNumAssignments               (on|weight|value) (<boolean>|<int>|<int>),
% MinNumAssignments               (on|weight|value) (<boolean>|<int>|<int>),
% MaxConsecutiveWorkingDays       (on|weight|value) (<boolean>|<int>|<int>),
% MinConsecutiveWorkingDays       (on|weight|value) (<boolean>|<int>|<int>),
% MaxConsecutiveFreeDays          (on|weight|value) (<boolean>|<int>|<int>),
% MinConsecutiveFreeDays          (on|weight|value) (<boolean>|<int>|<int>),
% MaxConsecutiveWorkingWeekends   (on|weight|value) (<boolean>|<int>|<int>),
% MinConsecutiveWorkingWeekends   (on|weight|value) (<boolean>|<int>|<int>),
% MaxWorkingWeekendsInFourWeeks   (on|weight|value) (<boolean>|<int>|<int>),
% WeekendDefinition               <Weekend>
% CompleteWeekends                (on|weight) (<boolean>|<int>),
% Ident.ShiftTypesDuringWeekend   (on|weight) (<boolean>|<int>),
% NoNightShiftBeforeFreeWeekend   (on|weight) (<boolean>|<int>),
% TwoFreeDaysAfterNightShifts     (on|weight) (<boolean>|<int>),
% AlternativeSkillCategory        (on|weight) (<boolean>|<int>),
% NumberOfUnwantedPatterns        <int>,
% UnwantedPatterns                <PATTERNS.ID>;
//                                separate pattern IDs by space (NOT comma)

PATTERNS = n4
% ID <string>,
% Weight <int>,
% NumberOfShiftTypes <int>,
% ShiftType (<SHIFT_TYPES.ID|Weekday OR Any>);

EMPLOYEES = n5
% ID <string>,
% Name <string>,
```

```
% ContractID <CONTRACTS.ID>,
% NumberOfSkills <int>,
% EmployeeSkills <SKILLS.Skill>;

// COVER_REQUIREMENTS

DAY_OF_WEEK_COVER = n6
% Day <WeekDay>,
% Shift <SHIFT_TYPES.ID>,
% Preferred <int+>;

DATE_SPECIFIC_COVER = n7
% Date <YYYY-MM-DD>,
% Shift <SHIFT_TYPES.ID>,
% Preferred <int+>;

DAY_OFF_REQUESTS = n8
% EmployeeID <EMPLOYEES.ID>,
% Date <YYYY-MM-DD>,
% weight <int+>;

DAY_ON_REQUESTS = n9
% EmployeeID <EMPLOYEES.ID>,
% Date <YYYY-MM-DD>,
% weight <int+>;

SHIFT_OFF_REQUESTS = n10
% EmployeeID <EMPLOYEES.ID>,
% Date <YYYY-MM-DD>,
% ShiftTypeID <SHIFT_TYPES.ID>,
% weight <int+>;

SHIFT_ON_REQUESTS = n11
% EmployeeID <EMPLOYEES.ID>,
% Date <YYYY-MM-DD>,
% ShiftTypeID <SHIFT_TYPES.ID>,
% weight <int+>;
```

An example using the above template is given in Appendix X.

3.2 Output format

Output format is also available in XML and text.

3.2.1 XML

An example of the solution XML format is provided in appendix E.

3.2.2 Text

The text format for solutions is:

```
// The ID of the instance that this is a solution to.
ProblemInstance = <SCHEDULING_PERIOD.ID>
// Name of the competitor. Use the same name for all solutions you provide.
% Competitor <string>,
% SoftConstraintsPenalty <int>;

ASSIGNMENTS = n1 // The number of assignments that follow.
% Date <YYYY-MM-DD>,
% Employee <EMPLOYEES.ID> ,
% ShiftType <SHIFT_TYPES.ID>;
```

An example of several solutions in one file, using this format, is given in Appendix F.

4 Solution Ranking

The following illustrates how the finalists of each track will be chosen. Let m be the total number of Early and Late Instances and k be the number of participants that produce a solution for all m instances.

Let X_{ij} be the result supplied (and verified) by participant i for instance j. Each X_{ij} is the value of the objective function s, for participant i on instance j.

The matrix X of results is transformed into a matrix of ranks R assigning to each R_{ij} a value from 1 to k. That is, for instance j the supplied X_{1j}, X_{2j}, $\ldots$,X_{kj} are compared with each other and the rank 1 is assigned to the smallest observed value, the rank 2 to the second smallest, and so on to the rank k, which is assigned to the largest value for instance ij. Ranks are assigned in all of the instances. We use average ranks in case of ties.

Consider the following example with $m = 6$ instances and $k = 7$ participants.

Table 16: Solutions

Instance	1	2	3	4	5	6
Solver 1	34	35	42	32	10	12
Solver 2	32	24	44	33	13	15
Solver 3	33	36	30	12	10	17
Solver 4	36	32	46	32	12	13
Solver 5	37	30	43	29	9	4
Solver 6	68	29	41	55	10	5
Solver 7	36	30	43	58	10	4

The ranks are the following:

We define for each solver the mean of the ranks. The finalists of the competition will be the 5 solvers with the lowest mean ranks. In case of a tie for entering the last positions, all the last equal-mean solvers are included in the final (in this case the finalists will be more than 5). In the example, the mean ranks are:

In this case the finalists would be solvers 1, 3, 5, 6 and 7.

Table 17: Solution ranks

Instance	1	2	3	4	5	6
Solver 1	3	6	3	3.5	3.5	4
Solver 2	1	1	6	5	7	6
Solver 3	2	7	1	1	3.5	7
Solver 4	4.5	5	7	3.5	6	5
Solver 5	6	3.5	4.5	2	1	1.5
Solver 6	7	2	2	6	3.5	3
Solver 7	4.5	3.5	4.5	7	3.5	1.5

Table 18: Mean ranks

Solver 1	3.83
Solver 2	4.33
Solver 3	3.58
Solver 4	5.17
Solver 5	3.08
Solver 6	3.92
Solver 7	4.08

The organizers will check the runs of the candidate finalist with the submitted seed to make sure that the submitted runs are repeatable. If they are not then another entrant will be chosen for the final.

For the final, the same evaluation process is repeated for the finalists with the following differences:

1. All instances, including hidden ones, will be used.
2. The solvers will be run by the organisers, thus the finalist should give support to the organisers in the process of compiling and running the solvers.
3. For each instance, the organisers will run 10 independent trails with seeds chosen at random. For each trial, we will compute the ranks and average them on all trials on all instances.

The winner is the one with the lowest mean rank. In case of a tie, 1 trial is added for all instances until a single winner is found.

5 Benchmarking

The benchmark program is designed to test roughly how fast your machine is at doing the sort of things that are involved in rostering. The program will tell you how long you can run your algorithm on the competition rostering problem instances. It is not possible to provide perfectly equitable benchmarks across many platforms and algorithms, and we know that the benchmark may be kinder to some people than others. It is pointed out that all the finalists will be run on a standard machine therefore creating a 'level playing field'.

The benchmark is only suitable for individual, single processor machines. It is not suitable, for example, for specialist parallel machines or clusters. Only individual, single processor machines are allowed to be used for the competition.

If you are using a PC, then please get the executable and data files from the INRC2010 web site[3]. If you are using a non-PC system then please contact the organizers to obtain the program.

The program should be run when the machine is not being used for anything else. Please check for that:

- There are no unnecessary windows open
- There are no significant OS background processes going on (e.g. back-up)
- There are no remote users on the computer
- There are no CPU sharing processes running (e.g. SETI at home, United Devices, DREAM)

The program will report how long it took, and hence the length of time you can run your rostering algorithm for (for each instance).

On a relatively modern PC, the benchmark program will grant the participant approximately 10 seconds for Track 1, 10 minutes for Track 2, and 10 hours for Tracks 3 . If the results you get are not in line with this, then please get in touch with us. However, please bear in mind that the exact speed of your computer depends on a number of factors including the memory and the operating system, in addition to the clock speed.

A Changes to numbering evaluation method

The original evaluation method consists of three phases. The initialisation phase sets the start values induced by the solution of the previous planning period. The intermediate phase evaluates the current planning period. When the evaluation has reached the last event in the planning period, a final evaluation on the constraints is required. More details on these phases can be found in the original paper[1].

As we do not consider the previous planning period within the competition, we can simplify the initialisation phase. It is sufficient to initialise last_nr to a given value instead of running the initialisation algorithm.

The original paper only considers history. Some constraints within the Nurse Rostering Competition, however, also require future. Similar to last_nr for history, we introduce future_nr to represent the number associated with the first event in the future. The final evaluation phase that allows evaluation future event is given below.

```
FOR i=1, ... , I
    IF (total > max_total) THEN
        penalty_max_total = penalty_max_total +
        cost_max_total * (total - max_total)

    IF (total < min_total) THEN
        penalty_min_total = penalty_min_total +
        cost_min_total * (total - min_total)

    IF (future_nr != U)
```

```
        IF (last_nr = future_nr - 1)
            consecutive = consecutive + 1;

    IF (consecutive > max_consecutive) THEN
        penalty_max_consecutive = penalty_max_consecutive +
        cost_max_consecutive * (consecutive - max_consecutive)

    IF (consecutive < min_consecutive) THEN
        penalty_min_consecutive = penalty_min_consecutive +
        cost_min_consecutive * (min_consecutive - consecutive)

    FOR EACH number t in numbering i
        IF (pert[t] > max_pert[t]) THEN
            penalty_max_pert = penalty_max_pert +
            cost_max_pert * (pert[t] - max_pert[t])
        IF (pert[t] < min_pert[t]) THEN
            penalty_min_pert = penalty_min_pert +
            cost_min_pert * (min_pert[t] - pert[t])

    IF (last_nr != U AND future_nr != U) THEN
     between = future_nr - last_nr - 1
        IF (between > max_between) THEN
            penalty_max_between = penalty_max_between +
            cost_max_between * (future_nr - last_nr - 1)

    IF (last_nr != U AND future_nr != U) THEN
     between = future_nr - last_nr - 1
        IF (between > 0 AND between < min_between) THEN
            penalty_min_between = penalty_min_between +
            cost_min_between * (future_nr - last_nr - 1)

i=i+1
```

B Constraints with Multiple Numberings

Some constraints cannot be expressed using only one numbering. Mostly constraints that do not occur on fixed days, need multiple numberings. The number of numberings needed is equal to the length of the pattern. For example consider the unwanted pattern, L-E-L, of length 3 and a scheduling period of 7 days. The pattern can start on any day, we need 3 numberings to achieve this.

Table 19: Multiple numberings for unwanted pattern $L - E - L$

	Mon		Tue		Wed		Thu		Fri		Sat		Sun	
	E	L	E	L	E	L	E	L	E	L	E	L	E	L
Numbering 1	U	0	1	U	U	2	U	4	5	U	U	6	U	U
Numbering 2	U	U	U	0	1	U	U	2	U	4	5	U	U	6
Numbering 3	U	U	U	U	U	0	1	U	U	2	U	U	U	U

C Example of an XML Instance File

Below is an example of an xml instance file.

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
```

```
<SchedulingPeriod ID="EXAMPLE"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="competition.xsd">
    <StartDate>2010-01-01</StartDate>
    <EndDate>2010-01-29</EndDate>
    <Skills>
        <Skill>Nurse</Skill>
    </Skills>
    <ShiftTypes>
        <Shift ID="E">
            <StartTime>06:30:00</StartTime>
            <EndTime>14:30:00</EndTime>
            <Description>Early</Description>
            <Skills>
                <Skill>Nurse</Skill>
            </Skills>
        </Shift>
        <Shift ID="L">
            <StartTime>14:30:00</StartTime>
            <EndTime>22:30:00</EndTime>
            <Description>Late</Description>
            <Skills>
                <Skill>Nurse</Skill>
            </Skills>
        </Shift>
    </ShiftTypes>
    <Patterns>
        <Pattern ID="0" weight="1">
            <PatternEntries>
                <PatternEntry index="0">
                    <ShiftType>L</ShiftType>
                    <Day>Any</Day>
                </PatternEntry>
                <PatternEntry index="1">
                    <ShiftType>E</ShiftType>
                    <Day>Any</Day>
                </PatternEntry>
            </PatternEntries>
        </Pattern>
    </Patterns>
    <Contracts>
        <Contract ID="0">
            <Description>fulltime</Description>
            <SingleAssignmentPerDay weight="1">true</SingleAssignmentPerDay>
            <MaxNumAssignments on="1" weight="1">16</MaxNumAssignments>
            <UnwantedPatterns>
                <Pattern>0</Pattern>
            </UnwantedPatterns>
        </Contract>
        <Contract ID="1">
```

```
        <Description>75_time</Description>
        <SingleAssignmentPerDay weight="1">true</SingleAssignmentPerDay>
        <MaxNumAssignments on="1" weight="1">12</MaxNumAssignments>
        <UnwantedPatterns>
            <Pattern>0</Pattern>
        </UnwantedPatterns>
    </Contract>
</Contracts>
<Employees>
    <Employee ID="0">
        <ContractID>0</ContractID>
        <Name>0</Name>
        <Skills>
            <Skill>Nurse</Skill>
        </Skills>
    </Employee>
    <Employee ID="1">
        <ContractID>0</ContractID>
        <Name>1</Name>
        <Skills>
            <Skill>Nurse</Skill>
        </Skills>
    </Employee>
</Employees>
<CoverRequirements>
    <DayOfWeekCover>
        <Day>Monday</Day>
        <Cover>
            <Shift>E</Shift>
            <Preferred>2</Preferred>
        </Cover>
    </DayOfWeekCover>
    <DayOfWeekCover>
        <Day>Tuesday</Day>
        <Cover>
            <Shift>L</Shift>
            <Preferred>2</Preferred>
        </Cover>
    </DayOfWeekCover>
</CoverRequirements>
<DayOffRequests>
    <DayOff weight="1">
        <EmployeeID>0</EmployeeID>
        <Date>2010-01-03</Date>
    </DayOff>
</DayOffRequests>
<ShiftOffRequests>
    <ShiftOff weight="1">
        <ShiftTypeID>L</ShiftTypeID>
        <EmployeeID>1</EmployeeID>
```

```
            <Date>2010-01-15</Date>
        </ShiftOff>
    </ShiftOffRequests>
</SchedulingPeriod>
```

D Example of a Text Instance File

Below is an example of a text instance file based on the same case as the xml example in Appendix C.

```
// Comments can be added by prefixing the comment with '//'.

// Text following '//' should not be treated.


//////////////////////////////////////////////////////////////////////

SCHEDULING_PERIOD;

//////////////////////////////////////////////////////////////////////

EXAMPLE, 2010-01-01, 2010-01-29;


//////////////////////////////////////////////////////////////////////

SKILLS = 1;

//////////////////////////////////////////////////////////////////////

Nurse;


//////////////////////////////////////////////////////////////////////

SHIFT_TYPES = 2;

//////////////////////////////////////////////////////////////////////

E, Early, 06:30:00, 14:30:00, 1, Nurse;

L, Late, 14:30:00, 22:30:00, 1, Nurse;


//////////////////////////////////////////////////////////////////////

CONTRACTS = 2;

//////////////////////////////////////////////////////////////////////
```

```
0, fulltime, (1|1), (1|1|16), , , , , , , , , , , , , , , , , 0, , 1, 0;

1, 75_time, (1|1), (1|1|12), , , , , , , , , , , , , , , , , 0, , 1, 0;

//////////////////////////////////////////////////////////////////////

PATTERNS = 1;

//////////////////////////////////////////////////////////////////////

0, 1, 0, L Any E Any;

//////////////////////////////////////////////////////////////////////

EMPLOYEES = 2;

//////////////////////////////////////////////////////////////////////

0, 0, 0, 1, Nurse;

1, 1, 0, 1, Nurse;

//////////////////////////////////////////////////////////////////////

DAY_OF_WEEK_COVER  = 2;

//////////////////////////////////////////////////////////////////////

Monday, , E, 2;

Tuesday, , L, 2;

//////////////////////////////////////////////////////////////////////

DATE_SPECIFIC_COVER  = 0;

//////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////

DAY_OFF_REQUESTS = 1;

//////////////////////////////////////////////////////////////////////

0, 2010-01-03, 1;
```

```
//////////////////////////////////////////////////////////////////

DAY_ON_REQUESTS = 0;

//////////////////////////////////////////////////////////////////


//////////////////////////////////////////////////////////////////

SHIFT_OFF_REQUESTS  = 1;

//////////////////////////////////////////////////////////////////

1, 2010-01-15, L, 1;


//////////////////////////////////////////////////////////////////

SHIFT_ON_REQUESTS  = 0;

//////////////////////////////////////////////////////////////////
```

E Example of a XML Solution File

```
<Solution>
  <SchedulingPeriodID>EXAMPLE</SchedulingPeriodID>
  <Competitor>PlanCo</Competitor>
  <SoftConstraintsPenalty>3</SoftConstraintsPenalty>
  <Assignment>
    <Date>2010-01-01</Date>
    <Employee>1</Employee>
    <ShiftType>L</ShiftType>
  </Assignment>
  <Assignment>
    <Date>2010-01-01</Date>
    <Employee>2</Employee>
    <ShiftType>E</ShiftType>
  </Assignment>
  <Assignment>
    <Date>2010-01-01</Date>
    <Employee>3</Employee>
    <ShiftType>D</ShiftType>
  </Assignment>
  <Assignment>
    <Date>2010-01-01</Date>
```

```
    <Employee>4</Employee>
    <ShiftType>N</ShiftType>
  </Assignment>
  <Assignment>
    <Date>2010-01-01</Date>
    <Employee>5</Employee>
    <ShiftType>E</ShiftType>
  </Assignment>
  <Assignment>
    <Date>2010-01-01</Date>
    <Employee>6</Employee>
    <ShiftType>N</ShiftType>
  </Assignment>

  ....

</Solution>
```

F Example of a Text Solution file

Here is an example of a text solution file based on the same solution as in the xml example in Appendix E.

```
////////////////////////////////////////////////////////////////////
SOLUTION = EXAMPLE;
////////////////////////////////////////////////////////////////////
PlanCo, 3;

////////////////////////////////////////////////////////////////////
ASSIGNMENTS = 6;
////////////////////////////////////////////////////////////////////
2010-01-01, 1, L;
2010-01-01, 2, E;
2010-01-01, 3, D;
2010-01-01, 4, N;
2010-01-01, 5, E;
2010-01-01, 6, N;

...
```

References

[1] E.K. Burke, P. De Causmaecker, S. Petrovic, G. Vanden Berghe. Fitness Evaluation for Nurse Scheduling Problems. In Proceedings of the 2001 IEEE Congress on Evolutionary Computation, Seoul, Korea, 2001, 1139–1146

[2] B. McCollum, A. Schaerf, B. Paechter, P. McMullan, R. Lewis, A. J. Parkes, L. Di Gaspero, R. Qu, and E. K. Burke. Setting the Research Agenda in Automated Timetabling: The Second International Timetabling Competition. INFORMS Journal on Computing, Vol. 22, Issue 1. In press.

[3] INRC2010 website: `http://www.kuleuven-kortrijk.be/nrpcompetition`