# KHE24: Towards a Practical Solver for Nurse Rostering

Jeffrey H. Kingston

The University of Sydney, Australia
`jeff@it.usyd.edu.au`
http://jeffreykingston.id.au

**Abstract.** Nurse rostering—assigning nurses to the shifts of a hospital ward—is one of the most studied of all timetabling problems. This paper describes ongoing work on the KHE24 nurse rostering solver. KHE24 is an improved version of the KHE18 solver published previously. It uses a time sweep algorithm to find an initial solution, which it then repairs using ejection chains and optimal reassignment by dynamic programming. Results are presented for several well-known data sets. They show that KHE24 is making progress towards success in practice, taking running time and breadth of application into account as well as solution cost.

**Keywords:** Nurse rostering, Time sweep, Ejection chains, XESTT

## 1 Introduction

*Nurse rostering*—assigning nurses to the shifts of a hospital ward—is one of the most studied of all timetabling problems. It is an NP-complete problem, and exact algorithms are out of reach in general, although many smaller instances have been solved to optimality using integer programming [4,36].

Many inexact methods have been tried. Recent work covers a wide range; it includes integer programming [15,29,32,36,40], weighted maxSAT [10,11], simulated annealing [9,38], hyper-heuristics [2,16,33], variable neighbourhood search [39], and constraint programming [34]. For older work, see [41].

The solver presented here, KHE24, is the 2024 version of the main solver built by the author on his KHE solver platform [20]. It is an improved version of the KHE18 solver described in [22]. It runs in polynomial time and aims to find competitive but not optimal solutions quickly, across a wide range of instances. It finds an initial solution using a time sweep algorithm (Section 3.1). This timetables the first day of the cycle, then the second, and so on. It then tries two repair methods: ejection chains (Section 3.3) and optimal reassignment using dynamic programming (Section 3.4).

The author's intention is to test KHE24 on four well-known data sets, to demonstrate its ability to solve a wide range of practical instances. At present, however, it has been tested on only two data sets and half of a third (Section 4). Although KHE24 has not found any new best solutions, on its own terms (that is, considering running time and breadth of application as well as cost) it promises to be successful. The work is ongoing.

## 2 Nurse rostering and its XESTT formulation

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.

Perhaps the most characteristic feature of the problem is the large array of requirements that each nurse's timetable must satisfy. In addition to workload limits, there are rules such as 'a nurse must have a day off after a sequence of night shifts', 'a nurse may work at most four days in a row', and so on. These requirements are different at different institutions.

Instead of the usual formulas, this paper's formal definition of the nurse rostering problem is supplied by the XESTT nurse rostering data format [23]. XESTT is an XML format which is capable of representing the instances found in all the well-known data sets. It is based on the XHSTT high school timetabling format [30,31]; the name 'XESTT' was chosen to be reminiscent of 'XHSTT', with 'employee scheduling' replacing 'high school'. Full details of XESTT appear online [17] and will not be repeated here. Instead, this section offers an overview, and explains the importance of XESTT to the present work.

An XESTT instance consists of the *cycle* (the sequence of *times* that events may be assigned); a set of *resources* (entities that attend events); a set of *events* (meetings); and a set of *constraints*, specifying conditions that solutions should satisfy, and penalties to impose when they don't.

Each event contains a *starting time*, which may either be preassigned a time or left open for a solver to assign; a *duration*, which is a fixed positive integer giving the number of consecutive times, starting at the starting time, that the event is running; an optional *workload*, which is a fixed non-negative integer representing the workload of the event in arbitrary units, for example in minutes; and any number of *event resources*, each specifying one resource which attends the event for the full duration, which may either be preassigned a resource or left open for a solver to assign.

In nurse rostering instances, each resource represents one nurse, and each event represents one shift. Each event has duration 1; its actual duration in minutes can be expressed as a workload, if needed. Its starting time is preassigned to a time unique to the shift. For example, if on each day there is a morning, afternoon, and night shift, then each day will contain three times, one for each shift. Within an event, each event resource represents a demand for one nurse.

A referee has commented that the time aspect of this mapping from nurse rostering to XESTT seems forced. And indeed it is fair to ask why times are needed, if they are in one-to-one correspondence with shifts. It would in fact be possible to omit times, but here are several reasons why that would not necessarily be an improvement. Timetabling in general is about assigning times and resources to events, and in nurse rostering it is a fact that the events have preassigned times. If times were omitted, the properties that times naturally have (such as belonging to a day, and chronological order) would have to be passed on to the shifts. Also, the use of times allows constraints from other subdisciplines of timetabling (unavailable times, for example) to be applied without change to nurse rostering.

Arbitrary sets of times, resources, and events may be defined, called *time groups*, *resource groups* and *event groups*. Each resource has one *resource type*, saying what type of resource it is. In nurse rostering there is just one type, `Nurse`.

XESTT offers 18 constraint types, but 9 are not used in nurse rostering, mainly because all the events have preassigned times. Of the 9 types that are used, 3 are *cover constraints*, specifying the number of resources that should attend each event, and the skills they need. The remaining 6, called *resource constraints* here, constrain the timetables of individual resources, specifying unavailable times, workload limits, unwanted patterns (for example, a day shift immediately following a night shift), limits on the number of consecutive busy days, and so on. Considerable flexibility is available, owing to the use of arbitrary time groups in constraints. For example, by defining one time group holding the times of the first weekend, another holding the times of the second weekend, and so on, one can construct a constraint which places limits on the number of busy weekends, despite weekends not being built-in to XESTT.

Each constraint contains a Boolean *required* flag indicating whether it is *hard* or *soft*, and an integer *weight*. When a constraint is violated, the degree of violation is multiplied by the weight to give a cost. Algorithms aim to minimize firstly the total cost of hard constraints (the *hard cost*), and secondly the total cost of soft constraints (the *soft cost*). In nurse rostering, solutions with non-zero hard cost are usually considered to be *infeasible*, that is, useless.

XESTT is important here for two reasons. First, it makes it easy to test KHE24 on a wide range of instances, because all these instances have been converted from their original formats to XESTT.

Second, XESTT uses just 9 types of constraints to represent the constraints found in other models. It can do this because its constraints are very flexible, as instanced above by the example of limiting busy weekends. Algorithms like the ejection chain algorithm in this paper, which handle each constraint type explicitly, have only 9 types to handle. Without XESTT or something like it, the number of constraint types would be much larger, and such an approach would hardly be feasible.

## 3 The KHE24 solver

The KHE24 solver presented here is built on the author's KHE solver platform and is available from the KHE web site [20]. It is an improved version of the KHE18 nurse rostering solver [22], which itself was descended from the KHE14 high school timetabling solver [19]. Complete details appear online, in the KHE documentation [20]; this section covers the main points.

In the KHE platform, instances and solutions are distinct objects. Instances are immutable after creation; solutions change as solving proceeds. A solution consists of a set of *meets*, each representing one event. Within each meet is a variable holding the meet's starting time, plus a set of *tasks*, one for each event resource of the meet's event. Each task is a variable to which one resource may be assigned; it represents an indivisible piece of work for one resource.

We will use ‘task’ in the following in preference to ‘shift’, because ‘shift’ is ambiguous: it can mean one shift for one nurse (‘Smith and Jones swapped shifts’), or one shift type (‘he prefers the night shift’), or one shift on one day (‘the shift passed quietly’).

KHE24 is a general timetabling solver: it assigns both times and resources. Nurse rostering events have preassigned times, so KHE24’s time assignment part merely converts the time preassignments in the events of the instance into time assignments in the meets of the solution. This takes almost no time.

After time assignment comes resource assignment, which assigns resources to tasks. KHE24 first carries out a *construction phase* which builds an initial assignment, then continues with a *repair phase* which tries several kinds of repairs that improve that assignment. The repair phase is run twice, to enable repairs of one kind to open the way to repairs of other kinds.

KHE24 has many parameters (called *options*), settable from the command line, determining such things as how to divide up the available running time, the maximum number of days to include when swapping the timetables of two resources, and so on. However, in this paper, the default values of all these options are used. There is no tuning of parameters for different instances, or even different data sets. The only options passed to the calls to KHE24 reported on here are those that set its time limit, the number of solutions to make for each instance, and the number of solves to run in parallel; and those options (except the time limit) are the same for all runs. This is an important aspect of our claim that KHE24 could become a *practical* solver.

Although we usually speak of finding one solution using one run of KHE24, the intention is that in practice several independent runs will occur in parallel, with the final reported solution being the best of those found by the different runs. Our results (Section 4) present the best of 24 runs, for example. We stress that multiple solves are not done just for testing; they are an integral part of the use of KHE24 in practice, included to take advantage of the multiple cores usually found in contemporary desktop computers.

## 3.1 Construction using time sweep

Because of the high density of constraints in nurse rostering, it may be easier to avoid introducing a problem than to remove it later. So it makes sense for the construction phase to try hard to produce a very good solution [27].

Many nurse rostering constraints concern what happens over consecutive days within nurses’ timetables. This suggests that the initial solution should be constructed day by day—the tasks of the first day assigned first, then the tasks of the second day, and so on. As each day is assigned, these kinds of constraints can often be satisfied. KHE24 uses this *time sweep* method.

The only other use of time sweep known to the author is [27], which finds initial solutions one week at a time in chronological order. It cites an aircrew scheduling paper [35] as its inspiration. The time sweep idea seems to be well known, however. A referee mentioned a similar algorithm from vehicle routing, which sweeps through drop-off points in order of their angle from the base.

To carry out the assignments for one day, KHE24 uses weighted bipartite matching. Each task $s$ running on that day is a demand node, each resource $r$ is a supply node, and an edge joins $s$ to $r$ when assigning $r$ to $s$ is possible. The edge is weighted by the change (often negative) in solution cost that the assignment would produce. A matching of minimum total cost is found and used to determine the assignments of resources to tasks on that day. Further details may be found in the earlier paper [22].

### 3.2 Task grouping

The KHE platform allows tasks to be *grouped*. Assigning a resource to one element of a group assigns it to all. Grouping was included to support high school timetabling: the lessons of one course should be assigned a common teacher. It has also proved useful in nurse rostering.

For example, in instance `COI-GPost` (Section 4.1), a nurse who takes a night task on a Friday should also take night tasks on the next two nights. This is because constraints penalize Friday night tasks before free weekends, incomplete weekends (working on Saturday or Sunday but not both), and day tasks after night tasks. Then the Monday and Tuesday night tasks can be grouped, as can the Wednesday and Thursday night tasks, owing to limits (minimum 2 and maximum 3) on the number of consecutive night tasks.

KHE24 has a grouping phase which precedes time sweep. It runs quickly but is general enough to be widely useful. For example, it groups the tasks of solutions of instance `COI-GPost` following the line of argument just given. The full details are rather intricate. They appear in the earlier paper [22].

A grouping will occasionally be inadvisable for some unexpected reason. So KHE24 applies the groupings during the construction and first repair phases, but then removes them, so that if something else is needed there is a chance to find it during the second repair phase. The author has observed a few cases where omitting to remove groupings led to infeasible solutions.

### 3.3 Repair using ejection chains

After constructing an initial solution using time sweep, KHE24 repairs it using two methods. The most effective of these is *ejection chain repair*. Although KHE18 also used ejection chain repair, the details were not settled and the published description was quite sketchy. KHE24's version is much more settled, so this section presents it in detail.

A *defect* in a solution is one violation of a constraint. For example, if nurse `N2` should work at most two weekends but in fact works three, that is a defect.

A *repair* is a change to a solution which removes a defect. For example, unassigning one of nurse `N2`'s busy weekends repairs the defect just described.

An ejection chain is like a path in a graph where each node represents one defect and each edge represents one repair. Starting from some defect, the first repair removes that defect but introduces one new defect. The second repair removes that defect but introduces another new defect, and so on. If some repair removes a defect without introducing a new defect, then the chain ends successfully: the solution has been improved. Or if the repair of one defect introduces two or more new defects, then the chain ends unsuccessfully: the repair has to be undone, with no improvement. (More precisely, a chain ends successfully whenever the current solution cost is less than it was at the start of the chain; new defects are acceptable, if their cost is low. A chain may be extended whenever a repair introduces a new defect whose removal would make the current solution cost less than it did at the start of the chain.)

There are usually several ways to repair a defect. For example, nurse `N2`'s defect can be repaired by unassigning any one of the three busy weekends. So finding a successful chain involves a search tree: if the first repair does not begin a successful chain, then the second is tried, and so on recursively.

The main loop of the ejection chain repair algorithm visits each defect of the current solution and attempts to remove it by searching a tree of ejection chains, stopping at the first successful chain, if any. It cycles around the defects until a complete cycle of attempts has failed, at which point it terminates.

There are several ways to limit the method to polynomial time. The usual one, which KHE24 uses, is to refuse to visit the same part of the solution twice while repairing a given defect [18,19]. There is also a time limit (Section 3.5).

Each type of constraint gives rise to one type of defect (or two: maximum and minimum limit violations are repaired differently). For each defect, a set of repairs suited to its type is tried, making the chains *polymorphic*.

At the lowest level, just one basic operation can change a solution: the *task move*, which changes the assignment of task $s$ from resource $r_1$ to resource $r_2$, where $r_1 \neq r_2$. Here $r_1$ may be `NULL`, in which case the task move is also a *task assignment*; or $r_2$ may be `NULL`, making it also a *task unassignment*. One repair is (can only be) a set of these basic operations.

Several authors (see below) use a swap of the timetables of two resources over several consecutive days as their main repair operation. KHE24 also does this; the maximum number of days is 16. It also tries assignments to a resource $r_2$ over several days, and unassignments of a resource $r_1$ over several days. In these cases, the maximum number of days is much smaller, just 4. All these operations are sets of task moves.

Unlike metaheuristics, which choose repairs at random, KHE24's ejection chain algorithm is quite focused. It finds all *small repairs* (minimal repairs that correct the current defect, typically affecting just one or two days), and then for each small repair it tries a set of alternative *large repairs*, defined by *expanding* each small repair: making the same change over a larger set of adjacent days including the original ones. Simple checks ensure that the same large repair is almost never tried twice when repairing a given defect.

It is not hard to find suitable small repairs, as was done above for nurse `N2`'s busy weekends. If a resource $r$ is overloaded during some set of times (one day, weekend, or whatever), all small repairs are tried that either unassign $r$ during those times or swap its timetable on those times' days with another resource which is free then (or in some cases less busy). If $r$ is underloaded during some set of times, all small repairs are tried that either assign $r$ to some unassigned task during those times or swap its timetable on those times' days with some other resource which is busier then. Expanded versions of these two kinds of repairs are used to repair all kinds of resource constraint defects. For example, if $r$ has too many consecutive night shifts, that is treated as $r$ being overloaded during the times of the two endpoints of the over-long sequence.

Cover constraint defects are handled similarly. For example, a task needing assignment is handled by finding all small repairs that assign a suitable resource to the task, then expanding that assignment over adjacent days.

It is common for several tasks to be equivalent, in the sense that the effect of assigning any given resource to any one of them is the same. For example, if the Thursday night shift demands one senior nurse and three ordinary nurses, there will be one task requiring a senior nurse which is not equivalent to any other task, and three equivalent tasks demanding ordinary nurses. If the demand is for two or three ordinary nurses, then the three tasks are still equivalent but there is a preference to assign resources to the first two rather than to the third. In such cases the ejection chain solver will try one repair per equivalence class, not one repair per task, saving time.

The author knows of three other papers that use ejection chains in nurse rostering. The first two are fairly old and use data sets that are not in use today, making their results hard to evaluate. One [12] includes a repair which swaps the timetables of two resources over one week. The other [28] uses chains of task moves in a tabu search framework. The chains are rather different from those used here: each is generated at random in a way that preserves coverage, but without checking other costs until the whole chain is generated. Reference [28] cites [1] as a source for these chains—a very early use. The third previous use of ejection chains [3,4,7] is much more recent. Its basic repair swaps the timetables of two resources over a variable number of consecutive days.

### 3.4 Repair using optimal reassignment

There is a dynamic programming algorithm, well known to those who use column generation to solve nurse rostering problems, for finding an optimal assignment of tasks to a single nurse, given fixed timetables for the other nurses. The author has recently generalized this algorithm to find an optimal reassignment of an arbitrary subset of the nurses on an arbitrary subset of the days of the cycle, leaving the other nurses and the other days fixed [25].

The algorithm described in [25] was deficient in one respect: it did not support the XESTT limit workload constraint, which limits total workload measured in arbitrary units, for example minutes. This deficiency is now fixed.

Analysis shows that the running time of this algorithm is polynomial in the number of days it reassigns, but exponential in the number of resources it reassigns [25]. The author's practical experience bears this out: it costs very little to reassign a few more days, but reassigning just one more resource can increase the running time dramatically. Disappointingly, despite the author's best attempts at optimization, he has been able to reliably reassign up to only three resources, although for a practically unlimited number of days.

Using this algorithm, the author has implemented a VLSN search which chooses 3 resources and 28 consecutive days at random, optimally reassigns those resources over those days, and then repeats using new random choices until time runs out or 50 consecutive attempts produce no improvement. It is nowhere near as effective as ejection chains, but it operates on quite different principles and occasionally makes a contribution.

### 3.5 Making good use of available running time

Running time is a major issue for the larger instances, so it must be used well. Each of KHE24's phases yields diminishing returns as its running time increases, so it would be a mistake to spend all of the available time on one phase and have nothing left for the others.

KHE24 limits each day during time sweep (including repairs associated with that day) to 3 seconds. Two-thirds of the remaining time is then given to the first repair phase, and one-third to the second. Within these phases, the two repair methods (ejection chains and optimal reassignment) get equal time. If ejection chains finish early, optimal reassignment gets the surplus.

The ejection chain algorithm of [7] imposes a time limit on each search for a chain that repairs one defect. KHE24 imposes a limit of 120 on the number of recursive calls when repairing one defect, which has a similar effect.

Actual running time need not match the time limit exactly. On the one hand, a phase will often end of its own accord well before its time limit. On the other, the time limits are *soft*: instead of being interrupted, each phase consults wall clock time periodically and decides for itself whether to stop. In practice, KHE24 never significantly overruns its time limit, as the actual running times reported in the results tables of Section 4 will show.

### 3.6 Randomization

Randomization is not stressed in algorithmic solvers like it is in, for example, metaheuristic ones. Still, including some randomization allows the algorithm to be run multiple times with different seeds to obtain different results. Doing this and keeping the best solution is a simple way to utilize multiple cores and trade off solution quality and running time.

For example, ejection chain repair randomizes by trying alternative repairs starting at a random point in the sequence of possible repairs. Similar methods are used in the other phases. These methods are not deep, but they seem to work, judging from the spread of solution costs they produce.

## 4 Results

This section presents the results of running KHE24 on several well-known sets of instances, after conversion to XESTT by the NRConv program [21,23], with comparisons with previous results. The converted instances and solutions are available, in the form of XESTT archive files, at [21].

Our aim here is not to find new best solutions, but rather to achieve success in practice, which means to find good solutions to a wide variety of real-world instances quickly. This idea has been formalized by the author [26] as follows:

> *A solver is* successful in practice *if, on every instance that is likely to be encountered in practice, it finds a solution whose cost is within 10% of the best known when run for 5 minutes, and within 5% of the best known when run for 60 minutes.*

A justification appeared in [26], but since the idea is somewhat novel, yet crucial to this paper, we'll discuss it now in some detail.

The basic idea, then, is that a solver satisfying these conditions can be used by practitioners with confidence that it will work well on their instances. Running for 5 minutes will produce a solution suitable for testing a possible scenario; running for 60 minutes will produce a solution for actual use which is not easy to distinguish from the best known solution. (If all constraints have the same weight, being within 5% means that for every 20 defects in the best known solution, there are 20 or 21 defects in the solver's solution.)

This definition of success in practice is not realistic in one case: when the best known solution has no defects, or just a few. In that case, in reality a solution would be considered successful in practice if it contained just a few more defects, whatever the percentage is. For example, the best known solution to instance `COI-GPost` (Section 4.1) has cost 5. KHE24's solution has cost 8, which in reality would be judged to be successful in practice, even though it is numerically 60% worse. Rather than complicating the rule to take account of such cases, we'll simply point them out as they arise.

Our task, then, is to evaluate the KHE24 solver against this standard of success in practice. As a proxy for 'every instance that is likely to be encountered in practice', we intend to test the solver on four widely used data sets, although in this paper we only test on two data sets and part of a third. And for 'best known solutions' we will use sets of solutions from other authors which are either the best, or among the best known.

KHE24 (as described in Section 3) is run 24 times on each instance, with a different random seed for each run, and the best of the 24 solutions is kept. We call this version of the solver KHE24x24. We stress that running multiple solves in parallel and keeping the best solution is an integral part of our solving strategy, not something we are doing merely to gain insight into the variability of the solver (although that is an interesting question worthy of investigation).

Twelve threads are used, running on the author's 12-core Intel i7-12700 processor. (The Intel web site gives several clock frequencies, from 1.6GHz to 4.9GHz, chosen depending on various factors including temperature, so it's hard to be definite about processor speed.) Each individual solve is given a time limit which is half the overall time limit (half of either 5 minutes or 60 minutes). For each instance, the reported running time is the wall clock time in seconds from the start of the first solve to the end of the last one. It does not include file read and write times; they are negligible.

Each result table was generated by the author's HSEval program from an XESTT archive file and included with no hand editing. A blank entry indicates that there is no solution for the instance of its row in the solution group of its column. If there are multiple solutions, one with minimum running time among all solutions with minimum cost is shown. The minimum solution costs in each row appear in bold. Any solutions with non-zero hard cost are shown as 'inf.' (infeasible). No costs are reported on trust; all are calculated from the solutions in the archive file, and hence verified, by HSEval.

Adjacent to each cost $c$, in the 'Rel.' column, is a relative cost: the value of $c$ divided by the best cost in the row, when the best cost is non-zero. When KHE24x24 is run for 5 minutes, a relative cost of up to 1.10 represents success in practice; when it is run for 60 minutes, 1.05 represents success in practice.

Running times are shown (in seconds) where present in the archive. HSEval cannot verify them. KHE24x24 running times are as defined above.

Care is needed with the average costs at the foot of each table, since costs are sensitive to the scale of the constraint weights. Constraints that measure workload in minutes may produce costs in the thousands.

## 4.1 The Curtois original instances

The Curtois original instances are the instances available online at [8] under the heading 'Original instances.' Their XESTT versions appear in XESTT archive file `COI.xml` at [21], along with the solutions posted with the instances. Nearly all of these solutions are optimal, according to Table 3 of [4].

Table 1 compares KHE24x24's solutions with the solutions from [8]. It is important to analyse what is happening, and not simply take these results at face value. Several cases of a phenomenon described earlier, arising when the best known solution has very low cost, occur here. For example, KHE24 is successful in practice on the Post and Ikegami instances, even though the relative costs are numerically high. Also, in every solution of `COI-WHPP` with cost below 1000, some nurses must take night shifts only, and the rest must take non-night shifts only—hardly a real-world scenario. (The author has addressed this issue with `COI-WHPP` in work carried out after this table was produced.)

In summary, KHE24 is successful in practice, or nearly so, on most of the Curtois original instances, the main exceptions being the larger ones, such as `COI-ERMGH`, `COI-CHILD`, `COI-ERRVH`, and `COI-MER`. The large differences in cost between the 5-minute and 60-minute runs on these instances suggest that their large size is overwhelming the solver, although further analysis is needed. The Curtois original instances are rarely used for testing these days, which is a pity, considering how interesting and varied they are.

## 4.2 The First International Nurse Rostering Competition

The First International Nurse Rostering Competition [14] has published many instances, still available from the competition web site [13]. XESTT versions are available in files `INRC1-Long-And-Medium.xml` and `INRC1-Sprint.xml` [21], along with the GOAL research group's virtually optimal solutions [37].

Tables 2 and 3 show KHE24's results. Running times are not given for the GOAL solutions because the GOAL solution files do not contain any; but the GOAL web site has a table of running times. About 10 instances, from the Long and Medium sets, have running times of about 4 hours. Many others have running times under one minute, often well under.

Another source of virtually optimal solutions to these instances is the branch and price algorithm whose results are reported in Table 5 of [4]. The author has not tried to obtain these solutions. Their reported running times are better than the GOAL ones, never exceeding about 10 minutes.

KHE24 is generally successful in practice on these instances, although once again there are some exceptions, including `INRC1-LH04`, `INRC1-LH05`, `INRC1-MH01`, and

Table 1: Results of running KHE24x24 on the Curtois original instances. Column Misc shows the best solutions from XESTT archive file `COI.xml`, taken from Curtois' web site [8]. The other columns show the results for KHE24x24, running for 5 minutes and 60 minutes. Here and elsewhere, running times are in seconds. This table is derived from XESTT archive file `KHE24-2024-03-07-COI.xml`, available at [21].

| **Instances (27)** | **Misc** | | | **KHE24x24-05** | | | **KHE24x24-60** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **Cost** | **Rel.** | **Time** | **Cost** | **Rel.** | **Time** | **Cost** | **Rel.** | **Time** |
| COI-Ozkarahan | **0** | 1.00 | - | **0** | 1.00 | 0.1 | **0** | 1.00 | 0.1 |
| COI-Musa | **175** | 1.00 | - | **175** | 1.00 | 15.9 | **175** | 1.00 | 16.9 |
| COI-Millar-2.1 | **0** | 1.00 | 1.0 | **0** | 1.00 | 6.8 | **0** | 1.00 | 7.3 |
| COI-Millar-2.1.1 | **0** | 1.00 | - | **0** | 1.00 | 0.0 | **0** | 1.00 | 0.0 |
| COI-LLR | **301** | 1.00 | 10.0 | **301** | 1.00 | 46.5 | **301** | 1.00 | 50.8 |
| COI-Azaiez | **0** | 1.00 | 600.0 | **0** | 1.00 | 300.1 | **0** | 1.00 | 1800.0 |
| COI-GPost | **5** | 1.00 | - | 8 | 1.60 | 143.5 | 8 | 1.60 | 139.4 |
| COI-GPost-B | **3** | 1.00 | - | 5 | 1.67 | 157.5 | 5 | 1.67 | 1800.0 |
| COI-QMC-1 | **13** | 1.00 | - | 16 | 1.23 | 70.6 | 16 | 1.23 | 62.5 |
| COI-QMC-2 | **29** | 1.00 | - | 30 | 1.03 | 44.1 | 30 | 1.03 | 43.5 |
| COI-WHPP | **5** | 1.00 | - | 2001 | 400.20 | 300.1 | 2001 | 400.20 | 3600.1 |
| COI-BCV-3.46.2 | **894** | 1.00 | 17840.0 | 896 | 1.00 | 300.1 | **894** | 1.00 | 3600.1 |
| COI-BCV-4.13.1 | **10** | 1.00 | - | **10** | 1.00 | 300.1 | **10** | 1.00 | 3600.1 |
| COI-SINTEF | **0** | 1.00 | - | **0** | 1.00 | 6.9 | **0** | 1.00 | 7.4 |
| COI-ORTEC01 | **270** | 1.00 | 105.0 | 300 | 1.11 | 174.3 | 295 | 1.09 | 211.4 |
| COI-ORTEC02 | **270** | 1.00 | - | 295 | 1.09 | 154.3 | 295 | 1.09 | 158.8 |
| COI-ERMGH | **779** | 1.00 | 124.0 | 1481 | 1.90 | 301.7 | 882 | 1.13 | 3600.4 |
| COI-CHILD | **149** | 1.00 | - | 2824 | 18.95 | 324.8 | 261 | 1.75 | 3600.3 |
| COI-ERRVH | **2001** | 1.00 | - | 6739 | 3.37 | 307.7 | 2219 | 1.11 | 3829.4 |
| COI-HED01 | **136** | 1.00 | - | 138 | 1.01 | 254.3 | **136** | 1.00 | 1824.7 |
| COI-Valouxis-1 | **20** | 1.00 | - | 100 | 5.00 | 300.0 | 100 | 5.00 | 866.1 |
| COI-Ikegami-2.1 | **0** | 1.00 | 13.0 | **0** | 1.00 | 150.0 | **0** | 1.00 | 1800.1 |
| COI-Ikegami-3.1 | **2** | 1.00 | 21600.0 | 10 | 5.00 | 300.1 | 9 | 4.50 | 3600.3 |
| COI-Ikegami-3.1.1 | **3** | 1.00 | 2820.0 | 14 | 4.67 | 300.2 | 15 | 5.00 | 3600.5 |
| COI-Ikegami-3.1.2 | **3** | 1.00 | 2820.0 | 15 | 5.00 | 300.1 | 10 | 3.33 | 3600.4 |
| COI-BCDT-Sep | **100** | 1.00 | - | 230 | 2.30 | 300.1 | 210 | 2.10 | 3600.3 |
| COI-MER | **7081** | 1.00 | 36002.7 | 56048 | 7.92 | 406.4 | 13840 | 1.95 | 3616.5 |
| **Average** | **454** | 1.00 | | 2653 | 17.52 | 195.0 | 804 | 16.47 | 1801.4 |

Table 2: Results for the Long and Medium instances of the First International Timetabling Competition. The GOAL column shows the solutions from the GOAL research group web site [37], which the GOAL group has shown to be virtually optimal. The other columns show the results for KHE24x24, running for 5 minutes and 60 minutes. This table is derived from the instances and solutions stored in XESTT archive file `KHE24-2024-03-07-INRC1-Long-And-Medium.xml`, available at [21].

| **Instances (30)** | **GOAL** | | **KHE24x24-05** | | | **KHE24x24-60** | | |
|---|---|---|---|---|---|---|---|---|
| | **Cost** | **Rel.** | **Cost** | **Rel.** | **Time** | **Cost** | **Rel.** | **Time** |
| INRC1-L01 | **197** | 1.00 | 199 | 1.01 | 222.4 | 199 | 1.01 | 405.5 |
| INRC1-L02 | **219** | 1.00 | 227 | 1.04 | 219.9 | 225 | 1.03 | 775.3 |
| INRC1-L03 | **240** | 1.00 | **240** | 1.00 | 226.0 | **240** | 1.00 | 480.4 |
| INRC1-L04 | **303** | 1.00 | 305 | 1.01 | 219.1 | 304 | 1.00 | 740.4 |
| INRC1-L05 | **284** | 1.00 | 285 | 1.00 | 218.4 | 285 | 1.00 | 686.9 |
| INRC1-LH01 | **346** | 1.00 | 367 | 1.06 | 237.4 | 354 | 1.02 | 2471.2 |
| INRC1-LH02 | **89** | 1.00 | 95 | 1.07 | 249.3 | 93 | 1.04 | 2192.8 |
| INRC1-LH03 | **38** | 1.00 | 44 | 1.16 | 235.8 | 42 | 1.11 | 1779.7 |
| INRC1-LH04 | **22** | 1.00 | 33 | 1.50 | 229.8 | 30 | 1.36 | 1741.4 |
| INRC1-LH05 | **41** | 1.00 | 52 | 1.27 | 224.4 | 47 | 1.15 | 1957.4 |
| INRC1-LL01 | **235** | 1.00 | 252 | 1.07 | 235.9 | 246 | 1.05 | 2451.3 |
| INRC1-LL02 | **229** | 1.00 | 250 | 1.09 | 239.2 | 238 | 1.04 | 2455.2 |
| INRC1-LL03 | **220** | 1.00 | 262 | 1.19 | 248.3 | 244 | 1.11 | 2445.9 |
| INRC1-LL04 | **222** | 1.00 | 256 | 1.15 | 235.3 | 237 | 1.07 | 2443.1 |
| INRC1-LL05 | **83** | 1.00 | 87 | 1.05 | 245.0 | 84 | 1.01 | 1908.4 |
| INRC1-M01 | **240** | 1.00 | 243 | 1.01 | 179.2 | 243 | 1.01 | 186.1 |
| INRC1-M02 | **240** | 1.00 | 244 | 1.02 | 182.2 | 244 | 1.02 | 214.9 |
| INRC1-M03 | **236** | 1.00 | 239 | 1.01 | 180.2 | 239 | 1.01 | 197.4 |
| INRC1-M04 | **237** | 1.00 | 240 | 1.01 | 182.3 | 240 | 1.01 | 198.8 |
| INRC1-M05 | **303** | 1.00 | 308 | 1.02 | 209.3 | 308 | 1.02 | 285.8 |
| INRC1-MH01 | **111** | 1.00 | 140 | 1.26 | 219.2 | 132 | 1.19 | 1471.3 |
| INRC1-MH02 | **221** | 1.00 | 237 | 1.07 | 226.9 | 231 | 1.05 | 1040.9 |
| INRC1-MH03 | **34** | 1.00 | 41 | 1.21 | 208.9 | 41 | 1.21 | 301.0 |
| INRC1-MH04 | **78** | 1.00 | 85 | 1.09 | 218.8 | 85 | 1.09 | 458.2 |
| INRC1-MH05 | **119** | 1.00 | 130 | 1.09 | 222.7 | 132 | 1.11 | 1102.9 |
| INRC1-ML01 | **157** | 1.00 | 170 | 1.08 | 234.0 | 163 | 1.04 | 1079.7 |
| INRC1-ML02 | **18** | 1.00 | 26 | 1.44 | 139.6 | 26 | 1.44 | 241.2 |
| INRC1-ML03 | **29** | 1.00 | 35 | 1.21 | 125.7 | 35 | 1.21 | 141.5 |
| INRC1-ML04 | **35** | 1.00 | 41 | 1.17 | 183.5 | 41 | 1.17 | 330.8 |
| INRC1-ML05 | **107** | 1.00 | 114 | 1.07 | 217.6 | 114 | 1.07 | 1309.5 |
| **Average** | **164** | 1.00 | 175 | 1.11 | 213.9 | 171 | 1.09 | 1116.5 |

Table 3: Results for the Sprint instances of the First International Timetabling Competition. Details as for Table 2. This table is derived from XESTT archive file `KHE24-2024-03-07-INRC1-Sprint.xml`, available at [21].

| **Instances (30)** | **GOAL** | | **KHE24x24-05** | | | **KHE24x24-60** | | |
|---|---|---|---|---|---|---|---|---|
| | **Cost** | **Rel.** | **Cost** | **Rel.** | **Time** | **Cost** | **Rel.** | **Time** |
| INRC1-S01 | **56** | 1.00 | **56** | 1.00 | 42.0 | **56** | 1.00 | 48.1 |
| INRC1-S02 | **58** | 1.00 | **58** | 1.00 | 38.5 | **58** | 1.00 | 38.1 |
| INRC1-S03 | **51** | 1.00 | **51** | 1.00 | 50.2 | **51** | 1.00 | 50.4 |
| INRC1-S04 | **59** | 1.00 | **59** | 1.00 | 44.1 | **59** | 1.00 | 44.9 |
| INRC1-S05 | **58** | 1.00 | **58** | 1.00 | 49.7 | **58** | 1.00 | 50.4 |
| INRC1-S06 | **54** | 1.00 | **54** | 1.00 | 52.0 | **54** | 1.00 | 50.5 |
| INRC1-S07 | **56** | 1.00 | **56** | 1.00 | 46.9 | **56** | 1.00 | 45.3 |
| INRC1-S08 | **56** | 1.00 | **56** | 1.00 | 40.4 | **56** | 1.00 | 40.4 |
| INRC1-S09 | **55** | 1.00 | **55** | 1.00 | 59.1 | **55** | 1.00 | 61.5 |
| INRC1-S10 | **52** | 1.00 | **52** | 1.00 | 46.8 | **52** | 1.00 | 46.7 |
| INRC1-SH01 | **32** | 1.00 | 34 | 1.06 | 31.6 | 34 | 1.06 | 31.4 |
| INRC1-SH02 | **32** | 1.00 | **32** | 1.00 | 30.0 | **32** | 1.00 | 29.8 |
| INRC1-SH03 | **62** | 1.00 | **62** | 1.00 | 42.7 | **62** | 1.00 | 42.3 |
| INRC1-SH04 | **66** | 1.00 | 67 | 1.02 | 58.6 | 67 | 1.02 | 58.1 |
| INRC1-SH05 | **59** | 1.00 | **59** | 1.00 | 54.2 | **59** | 1.00 | 54.2 |
| INRC1-SH06 | **130** | 1.00 | 134 | 1.03 | 59.8 | 134 | 1.03 | 58.7 |
| INRC1-SH07 | **153** | 1.00 | **153** | 1.00 | 53.0 | **153** | 1.00 | 52.6 |
| INRC1-SH08 | **204** | 1.00 | 206 | 1.01 | 124.2 | 206 | 1.01 | 127.4 |
| INRC1-SH09 | **338** | 1.00 | **338** | 1.00 | 149.7 | **338** | 1.00 | 223.6 |
| INRC1-SH10 | **306** | 1.00 | **306** | 1.00 | 72.3 | **306** | 1.00 | 70.0 |
| INRC1-SL01 | **37** | 1.00 | 38 | 1.03 | 50.5 | 38 | 1.03 | 50.2 |
| INRC1-SL02 | **42** | 1.00 | 43 | 1.02 | 37.4 | 43 | 1.02 | 36.8 |
| INRC1-SL03 | **48** | 1.00 | 49 | 1.02 | 51.5 | 49 | 1.02 | 53.9 |
| INRC1-SL04 | **73** | 1.00 | **73** | 1.00 | 139.8 | **73** | 1.00 | 231.6 |
| INRC1-SL05 | **44** | 1.00 | 45 | 1.02 | 48.9 | 45 | 1.02 | 46.0 |
| INRC1-SL06 | **42** | 1.00 | **42** | 1.00 | 20.2 | **42** | 1.00 | 20.0 |
| INRC1-SL07 | **42** | 1.00 | 43 | 1.02 | 35.8 | 43 | 1.02 | 34.4 |
| INRC1-SL08 | **17** | 1.00 | **17** | 1.00 | 19.4 | **17** | 1.00 | 19.8 |
| INRC1-SL09 | **17** | 1.00 | **17** | 1.00 | 19.0 | **17** | 1.00 | 18.8 |
| INRC1-SL10 | **43** | 1.00 | **43** | 1.00 | 31.0 | **43** | 1.00 | 32.5 |
| **Average** | **78** | 1.00 | 79 | 1.01 | 53.3 | 79 | 1.01 | 58.9 |

INRC1-ML02. Analysis of these cases is needed. On the Sprint instances, running for 60 minutes is never better than running for 5.

### 4.3 The Second International Nurse Rostering Competition

The Second International Nurse Rostering Competition [5,6] was notable for requiring its instances to be solved week by week, as occurs in the real world. However, here we focus on a set of conventional 4-week and 8-week instances from the competition that have been tackled by several authors [27,38]. Their converted versions appear in files `INRC2-4.xml` and `INRC2-8.xml` at [21], along with some solutions produced by the authors of [27].

Table 4: Results for the Second International Timetabling Competition 4-week instances. The LOR17 column shows the solutions obtained from the authors of [27]. Table derived from archive file `KHE24-2024-03-07-INRC2-4.xml`, available at [21].

| **Instances (20)** | **LOR17** | | **KHE24x24-05** | | | **KHE24x24-60** | | |
|---|---|---|---|---|---|---|---|---|
| | **Cost** | **Rel.** | **Cost** | **Rel.** | **Time** | **Cost** | **Rel.** | **Time** |
| INRC2-4-030-1-6291 | **1695** | 1.00 | 2040 | 1.20 | 300.1 | 1865 | 1.10 | 3600.1 |
| INRC2-4-030-1-6753 | **1890** | 1.00 | 2230 | 1.18 | 300.1 | 2005 | 1.06 | 3600.2 |
| INRC2-4-035-0-1718 | **1425** | 1.00 | 1835 | 1.29 | 300.1 | 1590 | 1.12 | 3600.2 |
| INRC2-4-035-2-8875 | **1155** | 1.00 | 1535 | 1.33 | 300.1 | 1360 | 1.18 | 3600.1 |
| INRC2-4-040-0-2061 | **1685** | 1.00 | 2125 | 1.26 | 300.1 | 1875 | 1.11 | 3600.1 |
| INRC2-4-040-2-6106 | **1890** | 1.00 | 2360 | 1.25 | 300.1 | 2020 | 1.07 | 3600.1 |
| INRC2-4-050-0-0487 | **1505** | 1.00 | 2015 | 1.34 | 300.1 | 1745 | 1.16 | 3600.1 |
| INRC2-4-050-0-7272 | **1500** | 1.00 | 1975 | 1.32 | 300.1 | 1690 | 1.13 | 3600.1 |
| INRC2-4-060-1-6115 | **2505** | 1.00 | 3335 | 1.33 | 300.2 | 2840 | 1.13 | 3600.2 |
| INRC2-4-060-1-9638 | **2750** | 1.00 | 3785 | 1.38 | 300.2 | 3165 | 1.15 | 3600.2 |
| INRC2-4-070-0-3651 | **2435** | 1.00 | 3225 | 1.32 | 300.1 | 2850 | 1.17 | 3600.1 |
| INRC2-4-070-0-4967 | **2175** | 1.00 | 3035 | 1.40 | 300.1 | 2565 | 1.18 | 3600.1 |
| INRC2-4-080-2-4333 | **3340** | 1.00 | 4015 | 1.20 | 300.1 | 3775 | 1.13 | 3600.3 |
| INRC2-4-080-2-6048 | **3260** | 1.00 | 4230 | 1.30 | 300.1 | 3750 | 1.15 | 3600.2 |
| INRC2-4-100-0-1108 | **1245** | 1.00 | 2100 | 1.69 | 300.2 | 1670 | 1.34 | 3600.2 |
| INRC2-4-100-2-0646 | **1950** | 1.00 | 2700 | 1.38 | 300.2 | 2275 | 1.17 | 3600.2 |
| INRC2-4-110-0-1428 | **2440** | 1.00 | 3245 | 1.33 | 300.2 | 2755 | 1.13 | 3600.2 |
| INRC2-4-110-0-1935 | **2560** | 1.00 | 3550 | 1.39 | 300.2 | 3015 | 1.18 | 3600.2 |
| INRC2-4-120-1-4626 | **2170** | 1.00 | 3060 | 1.41 | 300.2 | 2600 | 1.20 | 3600.3 |
| INRC2-4-120-1-5698 | **2220** | 1.00 | 3140 | 1.41 | 300.2 | 2650 | 1.19 | 3600.2 |
| **Average** | **2090** | 1.00 | 2777 | 1.34 | 300.1 | 2403 | 1.15 | 3600.2 |

The results for the 4-week instances appear in Table 4. There is no running time information in the solution files obtained from other authors. Reference [27] gives a formula that was used to determine the running time limit; it is about 20 minutes for the largest 4-week instances. Reference [38] used a running time limit of 2750 seconds (about 45 minutes).

KHE24's results are disappointing, relative to the best known solutions and to the author's own testing. For example, the cost of the 5-minute solution to instance INRC2-4-030-1-6291 reported here is 2040 (Table 4), but in other 5-minute tests the author has obtained solutions whose cost is as low as 1785. This problem will probably be easy to fix, but the paper submission deadline intervened before the the author had time to fix it.

Sadly, the publication deadline has prevented the author from tackling the 8-week instances with KHE24x24. A few preliminary tests have revealed that KHE24x24's solutions are not currently competitive. The author's experience with other data sets suggests that detailed analysis will reveal concrete areas in which the algorithm is performing poorly, whose improvement will lead to more competitive results. However this is speculation at present.

### 4.4 The 2014 Curtois and Qu instances

The instances here are the 24 instances published in 2014 by Curtois and Qu [7,8]. Converted versions appear in file `CQ14.xml`, which also holds four sets of solutions received from Curtois via private correspondence.

Again, the publication deadline has prevented the author from presenting results for this data set. He does not have even preliminary results to report.

There are more recent results for these instances [8,10,11]. Reference [8] has slightly better results than the ones in file `CQ14.xml`, with lower bounds. The bounds show that many of the results are optimal, although the last five instances, which are much larger than the others, still have significant optimality gaps. We regard those last five instances as not relevant to the concerns of this paper, since they have cycles of 26 and 52 weeks, much longer than is encountered in practice.

## 5 Conclusion

This paper has presented KHE24, a nurse rostering solver which aims to find very good but not optimal solutions quickly across a wide range of instances. Polynomial-time methods are used: time sweep for the initial assignment, and ejection chains and optimal reassignment using dynamic programming for repair. No parameter tuning is needed.

The results so far give hope that the solver will eventually be successful in practice (that is, taking cost, running time, and breadth of application into account). At the time of writing, however, many tests are missing, and for some of those the author's preliminary results are uncompetitive. There are some worrying tendencies: relative solution cost seems to deteriorate as instance size increases, and the solver does not always make effective use of the longer (60-minute) running time. The author is actively continuing this work.

## References

1. J. L. Arthur and A. Ravindran A multiple objective nurse scheduling model, AIIE Transactions 13(1), pages 55–60 (1981).

2. Shahriar Asta and Ender Özcan, A tensor-based approach to nurse rostering, In Ender Özcan, Edmund Burke, and Barry McCollum (eds.), PATAT 2014 (Tenth international conference on the Practice and Theory of Automated Timetabling, York, UK, August 2014), 442–445 (2014)
3. Edmund K. Burke, Tim Curtois, Rong Qu, and Greet Vanden Berghe, A time predefined variable depth search for nurse rostering, INFORMS Journal on Computing, 25(3), 411–419 (2013), accessed via `http://eprints.nottingham.ac.uk/28283/1/JOC12vds.pdf`
4. Edmund K. Burke and Tim Curtois, New approaches to nurse rostering benchmark instances, European Journal of Operational Research 237, 71–81 (2014)
5. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stephaan Haspeslagh, and Andrea Schaerf, Second international nurse rostering competition (INRC-II), problem description and rules. oRR abs/1501.04177 (2015). URL http://arxiv.org/abs/1501.04177
6. Sara Ceschia, Nguyen Thi Thanh Dang, Patrick De Causmaecker, Stephaan Haspeslagh, and Andrea Schaerf, Second international nurse rostering competition (INRC-II) web site, URL http://mobiz.vives.be/inrc2/.
7. Tim Curtois and Rong Qu, Computational results on new staff scheduling benchmark instances URL http://www.cs.nott.ac.uk/ psztc/NRP/ (2014)
8. Tim Curtois, Employee Shift Scheduling Benchmark Data Sets, `http://www.schedulingbenchmarks.org/` (2019)
9. Nguyen Thi Thanh Dang, Sara Ceschia, Andrea Schaerf, Patrick De Causmaecker, and Stefaan Haspeslagh, Solving the multi-stage nurse rostering problem, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, Italy, August 2016), 473–475 (2016)
10. Emir Demirovíc, Nysret Musliu, and Felix Winter, Modeling and solving staff scheduling with partial weighted maxSAT, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, Italy, August 2016), 109–125 (2016)
11. Emir Demirovíc, Nysret Musliu, Felix Winter, and Peter J. Stuckey, Solution-based phase saving and MaxSAT for employee scheduling: a computational study, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 453–457 (2018)
12. Kathryn A. Dowsland, Nurse scheduling with tabu search and strategic oscillation, European Journal of Operational Research 106, 393–407 (1998)
13. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stølevik, and Andrea Schaerf, First international nurse rostering competition website, URL: http://www.kuleuven-kortrijk.be/nrpcompetition (2010)
14. 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)
15. Han Hoogeveen and Tim van Weelden, Personalized nurse rostering through linear programming, In Ender Özcan, Edmund Burke, and Barry McCollum (eds.), PATAT 2014 (Tenth international conference on the Practice and Theory of Automated Timetabling, York, UK, August 2014), 476–478 (2014)
16. Ahmed Kheiri, Ender Özcan, Rhyd Lewis, and Jonathan Thompson, A sequence-based selection hyper-heuristic: a case study in nurse rostering, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, Italy, August 2016), 503–505 (2016)
17. Jeffrey H. Kingston, The HSEval High School Timetable Evaluator, URL `http://jeffreykingston.id.au/cgi-bin/hseval.cgi` (2010)
18. Jeffrey H. Kingston, Repairing high school timetables with polymorphic ejection chains, Annals of Operations Research, DOI 10.1007/s10479-013-1504-3

19. Jeffrey H. Kingston, KHE14: An algorithm for high school timetabling, In Ender Özcan, Edmund Burke, and Barry McCollum (eds.), PATAT 2014 (Tenth international conference on the Practice and Theory of Automated Timetabling, York, UK, August 2014), 269–291
20. Jeffrey H. Kingston, KHE web site, `http://jeffreykingston.id.au/khe` (2014).
21. Jeffrey H. Kingston, XESTT web site, `http://jeffreykingston.id.au/xestt` (2017)
22. Jeffrey H. Kingston, KHE18: a solver for nurse rostering, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)
23. Jeffrey H. Kingston, Gerhard Post, and Greet Vanden Berghe, A unified nurse rostering model based on XHSTT, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)
24. Jeffrey H. Kingston, Modelling history in nurse rostering, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018). Also Annals of Operations Research, DOI 10.1007/s10479-019-03288-x
25. Jeffrey H. Kingston, Improving the dynamic programming algorithm for nurse rostering, PATAT 2022 (Thirteenth International Conference on Practice and Theory of Automated Timetabling, Leuven, Belgium, August 2022).
26. Jeffrey H. Kingston, Timetabling research: a progress report, PATAT 2022 (International Conference on the Practice and Theory of Automated Timetabling, Leuven, Belgium, August 2022).
27. 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`.
28. M. J. Louw, I. Nieuwoudt, and J. H. Van Vuuren, Finding good nursing duty schedules: a case study. Technical report, Department of Applied Mathematics, Stellenbosch University, South Africa (2005). Received from Tim Curtois and held by this author.
29. Antonın Novák, Roman Václavık, Premysl Sucha, and Zdenek Hanzálek, Nurse rostering problem: tighter upper bound for pricing problem in branch and price approach, MISTA 2015 (7th Multidisciplinary International Conference on Scheduling: Theory and Applications), Prague, August 2015, 759–763
30. Gerhard Post, XHSTT web site, `https://www.utwente.nl/en/eemcs/dmmp/hstt/` (2011)
31. 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)
32. Florian Mischek and Nysret Musliu, Integer programming and heuristic approaches for a multi-stage nurse rostering problem, PATAT 2016 (Eleventh international conference on the Practice and Theory of Automated Timetabling, Udine, Italy, August 2016), 245–262 (2016)
33. Christopher Rae and Nelishia Pillay, Investigation into an evolutionary algorithm hyper-heuristic for the nurse rostering problem, In Ender Özcan, Edmund Burke, and Barry McCollum (eds.), PATAT 2014 (Tenth international conference on the Practice and Theory of Automated Timetabling, York, UK, August 2014), 527–532 (2014)
34. Erfan Rahimian, Kerem Akartunali, and John Levine, A Hybrid Constraint Integer Programming Approach to Solve Nurse Scheduling Problems, MISTA 2015 (7th Multidisciplinary International Conference on Scheduling: Theory and Applications), Prague, August 2015, 429–442
35. M. Saddoune, G. Desaulniers, and F. Soumis, Aircrew pairings with possible repetitions of the same flight number, Computers and Operations Research **40**, 3 (2013), 805–814.
36. 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)

37. Haroldo G. Santos, Túlio A. M. Toffolo, Rafael A. M. Gomes, and Sabir Ribas, `http://www.goal.ufop.br/nrp/`
38. 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
39. Pieter Smet and Greet Vanden Berghe, Variable neighbourhood search for rich personnel rostering problems, MISTA 2015 (7th Multidisciplinary International Conference on Scheduling: Theory and Applications), Prague, August 2015, 928–930
40. Pieter Smet, Constraint reformulation for nurse rostering problems, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018), 69–80
41. J. Van den Bergh, J. Belien, P. De Bruecker, E. Demeulemeester and L. De Boeck, Personnel scheduling: a literature review, European Journal of Operational Research, 226(3), 367–385, (2013).