KHE18: A Solver for Nurse Rostering

Jeffrey H. Kingston

Received: date / Accepted: date

Abstract The problem of assigning nurses to the shifts of a hospital ward, known as *nurse rostering*, has been studied for many years. This paper presents work in progress on the KHE18 nurse rostering solver. A time sweep algorithm is used to make an initial assignment, followed by repair using several methods, including ejection chains. Results are presented for several standard data sets.

Keywords Nurse rostering · Time sweep · Ejection chains · XESTT

1 Introduction

The problem of assigning nurses to the shifts of a hospital ward, known as *nurse rostering*, has been studied for many years. It is an NP-complete problem, and exact algorithms are out of reach in general, although many smaller instances have recently been solved to optimality with integer programming [4,30].

Many inexact methods have been tried. Even very recent work covers a wide range: integer programming [14,24,27,30], variable neighbourhood search [32], simulated annealing [9], weighted maxSAT [10], hyper-heuristics [2,15,28], and constraint programming [29]. For less recent work, consult [33].

The solver presented here, KHE18, is the 2018 version of the main solver built by the author on his KHE timetabling platform [19]. It runs in polynomial time and aims to find a good but not optimal solution quickly. It uses a time sweep algorithm for constructing an initial solution (Section 3.1). This assigns nurses to the first day of the cycle, then to the second, and so on. After that, it tries several repair methods, including ejection chains (Section 3.2).

KHE18 is work in progress. It has been tested on three data sets (Section 4). It runs quickly and has found some solutions as good as, or nearly as good as, the best in the literature; but overall its results are not yet competitive.

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

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

Instead of the usual formulas, this paper's formal definition of the nurse rostering problem is supplied by the XESTT [21] nurse rostering data format. 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 [25,26]; the name 'XESTT' was chosen to be reminiscent of 'XHSTT', with 'employee scheduling' replacing 'high school'. Full details of XESTT appear online [16] 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 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. This arrangement is somewhat artificial, but, as [21] explains, most nurse rostering constraints concern shifts, not workload in minutes, and this works best in practice. Within an event, each event resource represents a demand for one nurse.

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, `Nurses`.

XESTT offers 18 constraint types, but 9 are not used in nurse rostering, mainly because all the events have preassigned times and duration 1. Of the 9 types that are used, 3 are *cover constraints*, specifying the number of resources

that should attend each event, and the skills those resources should have. The other 6, called *resource constraints* here, constrain the timetables of individual resources, specifying unavailable times, workload limits, unwanted patterns (such as a day shift immediately following a night shift), and so on.

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 KHE18 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 all of the constraints found in other models. An algorithm like the ejection chain repair algorithm in this paper, which handles each constraint type explicitly, has only 9 kinds to handle. Also, all XESTT constraints are represented in a uniform way (in object-oriented terms, the 18 constraint types are subtypes of an abstract `Constraint` superclass). Without XESTT or something like it, the number of constraint types would be effectively endless, and an approach that handles each constraint type explicitly would hardly be feasible.

3 The KHE18 solver

The KHE18 solver presented here is built on the author's KHE solve platform and is available from the KHE web site [19]. It is descended from the KHE14 high school timetabling solver [18]. It is in fact a general timetabling solver, but because in nurse rostering the times of all events are preassigned, its time assignment part does nothing here except to convert the time preassignments in the instances into assignments in the solutions, taking almost no time.

After time assignment comes resource assignment—the assignment of nurses to shifts. KHE18 first constructs an initial assignment using time sweep, then continues with a number of repairs which improve that initial assignment. Space precludes a complete presentation, although that is available online, in the KHE documentation [19]. Instead, this section focuses on the main points.

3.1 Time sweep assignment

Because of the high density of constraints in nurse rostering, it is often easier to avoid introducing a problem during construction of the initial solution than to remove it later. So it makes sense to try hard to produce a very good solution during the initial construction phase.

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 shifts of the first day assigned first, then the shifts of the second day, and so on. As each day is assigned, these kinds of constraints can usually be satisfied. This we call the *time sweep* method. The assignment of resources to the shifts of one day can often be made in a locally optimum manner in polynomial time using weighted bipartite matching. The rest of this section describes KHE18's implementation of this approach.

The algorithm first checks that a hard constraint limits each nurse to at most one shift per day (if not, it applies itself to individual shifts, rather than to days). Then, for each day it builds a weighted bipartite graph. The graph contain one *demand node* for each event resource of each event (shift) on that day, since each event resource is a demand for one nurse. It also contains one *supply node* for each nurse available for assignment. An edge joins a demand node to a supply node when the nurse can be assigned to the shift (which is practically always; the few exceptions need not detain us here). The cost of the edge is the cost of the solution containing all the assignments on previous days, plus just this one assignment on the current day. There is also one supply node for each demand node, representing non-assignment of a nurse to the demand node's event resource. The cost of this edge is just the current solution cost. This will usually include a penalty for not assigning a resource, which will be absent from the edges which do assign one. Assuming that the assignments represented by the edges have an independent effect on solution cost, a minimum-cost maximum matching in this graph represents a locally optimal assignment of nurses to the event resources of the day.

When are these assignments independent, in fact? All costs are produced by constraint violations, so this question can be answered by examining each of the 9 constraint types, to see whether the costs it generates are independent. And indeed most of them are. Constraints on the timetable of an individual nurse, for example, are affected only by assignments of that nurse, and there is at most one of those each day. XESTT has just one kind of constraint whose cost depends on multiple edges: the *limit resources constraint*. For example, a constraint of this type could require at least one nurse with a certain specialty to be on duty at all times of the day, which is not the same as requiring one such nurse on every shift, because shifts may overlap in time within one day.

Accordingly, when limit resources constraints are present, the algorithm finds a matching using a simple constructive heuristic (for each event resource in turn, calculate the cost of each possible assignment, and also the cost of non-assignment, and choose the alternative of least cost). Solution quality has been observed to deteriorate when this has to be done.

To encourage an even distribution of workload, the edge costs are adjusted slightly to break ties in favour of assigning resources with more unused workload. This has been observed to significantly improve solution quality. Two other adjustments are also included: one which favours assignments that do not bring constraints from below their upper limits to their upper limits, and another that favours shorter runs of consecutive busy days (reasoning that they will offer more flexibility when repairing the timetable later). Both have

been observed to give marginal further improvements. These observations need to be tested by large-scale experiments, which have not yet been carried out.

Constraints with minimum limits often produce spurious costs during time sweep (costs that will disappear as the time sweep moves on). For example, if busy days should come in groups of at least 2 consecutive days, then the first assignment after a free day will attract a spurious cost. KHE18 avoids this by informing constraints that the days after the current day should be treated like days after the end of the cycle often are: the constraint should not assume that they are either assigned or not assigned, and make its best estimate of cost (a lower bound on the true cost) accordingly. This turns off most minimum limits. A full description, with formulas, is given elsewhere [22].

The great weakness of time sweep—its inability to look ahead—is mitigated by including repairs in the construction stage. After each day is assigned, a few preceding days are individually unassigned and reassigned, using weighted bipartite matching if possible. This may produce small changes that coordinate better with the assignments now present on following days.

The author is unable to cite previous work using time sweep. The general idea is well known, although whether the details described above have been tried is not known. They make a significant difference.

3.2 Ejection chain repair

After constructing an initial solution using time sweep, KHE18 repairs it using several methods. The most significant, in the amount of improvement attained and in running time, is *ejection chain repair*.

A *defect* in a solution is one violation of a constraint. For example, if nurse `N2` should work at most 2 weekends but in fact works 3, that is a defect.

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

An ejection chain is a sequence of repairs. Starting from some defect, the first repair repairs that defect but introduces one new defect. The second repair repairs that defect, but introduces another new defect, and so on. If some repair repairs a defect without introducing a new defect, then the chain ends and the solution has been improved. Or if the repair of one defect introduces two or more new defects, the chain has to be undone, with no improvement.

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 no further progress is possible and the algorithm terminates. Alternatively, it terminates when one of its periodical checks reveals that the soft time limit has been reached.

There are several strategies for limiting the method to polynomial time. The usual one, which KHE18 uses, is to refuse to visit the same part of the solution twice while trying to remove a given defect [17,18].

Each of the 9 kinds of constraints gives rise to one type of defect (or one might say two types, because a maximum limit violation is repaired differently from a minimum limit violation). For each defect, a set of repairs tailored to its type is tried, making the chains *polymorphic*. Space precludes a full description of these sets of repairs, which in any case are still evolving. They will be defined at [19] when settled. Meanwhile, here is a general overview.

The basic operations underlying all repairs are *shift assignment*, which assigns resource r to unassigned shift s, *shift unassignment*, which unassigns resource r from shift s, and *shift move*, which changes the assignment of shift s from resource r_1 to resource r_2. One repair is a set of these basic operations.

For example, if the defect is the appearance of day shift s_2 immediately following night shift s_1 in the timetable of resource r, then the obvious repairs to try are shift moves of s_1 from r to any other resource, and shift moves of s_2 to any other resource. If the problem is the appearance of 6 consecutive busy days in the timetable of resource r when the maximum limit is 4, the most likely repairs are moving the first two shifts to some other resource, and moving the last two shifts to some other resource. And so on.

The author is currently investigating some quite complex repairs, such as swapping two sequences of days so as not to change total workloads. Unlike high school timetabling, in nurse rostering the constraint density is very high, so a repair that removes one defect often introduces several others. When that happens the ejection chain algorithm terminates that branch of its search immediately, so one aims to design repairs that are likely to produce few new defects. This is challenging and suggests that ejection chains might be less effective in nurse rostering than in high school timetabling [18].

The author is aware of three previous uses of ejection chains in nurse rostering. The first [11] is older than the data sets in use today, making its results hard to evaluate. It includes a repair which swaps one week's work between two resources. The second [3,4,7] is recent. Its basic repair swaps a variable number of consecutive days' work between two resources. It allocates equal running time to each top-level repair. Its results are compared with ours in Section 4. The third [23] uses ejection chains as individual moves within a tabu search framework. Each chain is generated at random in a way that preserves coverage, but otherwise without checking cost until the whole chain has been generated. It cites [1] as a source for these kinds of chains—a very early use.

This paper's ejection chain algorithm derives from recent work in high school timetabling [17,18]. Its polymorphism seems new to nurse rostering. The works cited above bundle defects together by resource and search for repairs which reduce the total cost of one bundle. It is hard to say *a priori* whether this is advantageous or not: on the one hand, it reduces the precision with which repairs can address specific defects; on the other, it is less affected by high constraint density. One paper [3] adds precision by selecting repairs which focus on those parts of the resource's timetable where defects lie.

3.3 Grouping event resources

The KHE solve platform allows event resources to be grouped together, in which case assigning a resource to one of them automatically assigns it to all. This feature was added in support of high school timetabling, where the lessons of one course spread through the week need to be assigned a common teacher. However, it has proved useful in nurse rostering as well.

For example, in instance `COI-GPost` (Section 4.1), a nurse who takes a Friday night shift should also take the following Saturday night and Sunday night shifts, owing to constraints which penalize Friday night shifts before free weekends, incomplete weekends (working on Saturday or Sunday but not both), and day shifts after night shifts. Then whoever takes the Monday night shift should also take the Tuesday night shift, and whoever takes the Wednesday night shift should also take the Thursday night shift, owing to a minimum limit of 2 and a maximum of 3 on the number of consecutive night shifts.

KHE18's method of finding such combinations is very simple. For each shift s, find all combinations of shifts and free time over 2 and 3 consecutive days starting with s. If only one of these combinations has zero cost, group its shifts together so that they will always be assigned the same resource. Of course, such groupings might turn out to be unadvisable for some deep reason. So KHE18 applies the groupings during the initial time sweep and the first repair cycle, but then it removes them, so that if something else is needed there is a chance to find it in a later repair cycle.

3.4 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 with different seeds to obtain different results. Doing this and keeping the best solution is a simple way to trade off solution quality and running time.

The time sweep algorithm randomizes by choosing a random resource as the first to be given a supply node, cycling through the resources from there. This causes ties in edge weights to be broken differently, at least at the start when many resources have equal unused workload. The ejection chain repair algorithm randomizes by trying alternative repairs starting at a random point. For example, when a shift is to be moved to some other resource, it traverses the list of resources from a random point. 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 KHE18 on several well-known sets of instances, after conversion to XESTT by the NRConv program [20,21], with comparisons with previous results. The converted instances and results

are available, in the form of XESTT archive files, at [20]. These results are work in progress and are likely to be superseded as KHE18 is improved.

Two versions of the solver are tested. The first, KHE18, runs the solver once and produces one solution. The second, KHE18x8, runs the solver 8 times, with a different random seed on each run, and keeps the best solution.

All solves are run on the author's machine, a quad-core 3.6GHz Intel Core i7-4790. All runs have a time limit of 300 seconds per instance. The limit is *soft*: instead of being interrupted, each potentially slow part of the solver consults wall clock time periodically and does as little as possible after the time limit.

KHE18x8 runs in parallel using 4 threads. Four solves are started initially, one per thread. New solves are started in these threads as old ones end, until all solves have been started or the whole run reaches the time limit. So KHE18x8 could start as few as 4 solves. Once started, a solve runs until it stops itself.

Each result table was generated by the author's HSEval program from an XESTT archive file and included with no hand editing. In each table, 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, the solution with minimum running time among all solutions with minimum cost is shown.

The costs are not reported on trust; they are calculated from the solutions in the archive file, and hence verified, by HSEval. Only soft costs are shown; any solutions with non-zero hard cost are shown as 'inf.'.

Running times in seconds are reported where present in the archive. HSEval cannot verify them. KHE18 and KH18x8's running times are wall clock times.

At the foot of each table is a row of averages. This is good for comparing running times, but for costs it is sensitive to the scale of the constraint weights. Multiplying each weight in one instance by 10, say, would greatly increase the influence of that instance on the average, without changing anything fundamental. So average cost is useful mainly for instances with similar weights, such as those of the First International Nurse Rostering Competition.

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 [20], along with the solutions posted with the instances.

This author does not know where these solutions came from originally. Most of them have the same costs as the solutions reported for the branch and price algorithm in Table 3 of [4], which also contains lower bounds showing that nearly all of them are optimal. However, their running times, where recorded in the solution files, are different. They may come from several solvers, in which case caution is needed in comparing their running times with KHE18's.

Table 1 compares KHE18's solutions with those from [8]. It shows that even KHE18x8 is not yet competitive. Still, there are hopeful signs: the optimal or near-optimal solutions for instances such as `COI-Millar-2.1.1`, `COI-LLR`, and `COI-BCV-3.46.2`, and the moderate running times. More work is needed.

Table 1 Results for the Curtois original instances. Column Misc shows the best solutions from XESTT archive file `COI.xml`. The other columns show the results from the two versions of the author's solver described in this paper. This table is derived from XESTT archive file `KHE18-2018-05-22-COI.xml`, at [20]. Here and elsewhere, running times are in seconds.

Instance	**Misc**		**KHE18**		**KHE18x8**	
	Cost	**Time**	**Cost**	**Time**	**Cost**	**Time**
COI-Ozkarahan	**0**	-	200	0.1	100	0.2
COI-Musa	**175**	-	191	0.4	185	1.3
COI-Millar-2.1	**0**	1.0	400	0.0	400	0.1
COI-Millar-2.1.1	**0**	-	**0**	0.0	**0**	0.1
COI-LLR	**301**	10.0	302	0.4	**301**	1.4
COI-Azaiez	**0**	600.0	31	0.7	26	2.9
COI-GPost	**5**	-	25	0.5	10	2.0
COI-GPost-B	**3**	-	1021	0.3	1015	2.1
COI-QMC-1	**13**	-	37	0.6	33	2.1
COI-QMC-2	**29**	-	52	0.8	46	2.9
COI-WHPP	**5**	-	3010	20.6	3009	47.1
COI-BCV-3.46.2	**894**	17840.0	900	3.7	896	10.1
COI-BCV-4.13.1	**10**	-	20	0.2	**10**	0.6
COI-SINTEF	**0**	-	9	0.3	8	1.1
COI-ORTEC01	**270**	105.0	1485	0.9	1465	8.5
COI-ORTEC02	**270**	-	480	12.2	415	22.5
COI-ERMGH	**779**	124.0	2276	300.4	2330	300.5
COI-CHILD	**149**	-	2686	296.2	2391	504.7
COI-ERRVH	**2001**	-	67253	300.2	60139	300.5
COI-HED01	**136**	-	182	1.1	179	3.7
COI-Valouxis-1	**20**	-	1500	0.7	1500	2.6
COI-Ikegami-2.1	**0**	13.0	113	2.3	4	6.6
COI-Ikegami-3.1	**2**	21600.0	550	12.5	247	24.3
COI-Ikegami-3.1.1	**3**	2820.0	748	8.8	346	36.4
COI-Ikegami-3.1.2	**3**	2820.0	365	12.0	254	30.9
COI-BCDT-Sep	**100**	-	560	3.7	560	9.4
COI-MER	**7081**	36002.7	55774	300.3	49327	300.7
Average	**454**		5191	**47.4**	4637	60.2

4.2 The First International Nurse Rostering Competition

The instances from the First International Nurse Rostering Competition are available from the competition web site [12]. Their converted versions appear in archive files `INRC1-Long-And-Medium.xml` and `INRC1-Sprint.xml` [20], with the solutions from the GOAL research group web site [31], which the GOAL group has proved to be virtually optimal.

The results appear in Tables 2 and 3. These tables do not give running times for the GOAL solutions, because no running times appear in the solution files downloaded from the GOAL site. However, the site itself contains a table that does give running times. About 10 of the instances, from the Long and Medium sets, have running times of about 4 hours. Many others, including all the Sprint instances, 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

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 [31], which the GOAL group has shown to be virtually optimal. The other columns show the results from the two versions of the author's solver. This table is derived from XESTT archive file `KHE18-2018-05-22-INRC1-Long-And-Medium.xml`, available at [20].

Instance	**GOAL**		**KHE18**		**KHE18x8**	
	Cost	**Time**	**Cost**	**Time**	**Cost**	**Time**
INRC1-L01	**197**	-	204	13.3	199	33.9
INRC1-L02	**219**	-	235	12.4	231	31.8
INRC1-L03	**240**	-	**240**	11.4	**240**	33.3
INRC1-L04	**303**	-	307	17.3	304	43.2
INRC1-L05	**284**	-	291	26.9	286	61.1
INRC1-LH01	**346**	-	444	18.4	386	45.2
INRC1-LH02	**89**	-	118	18.7	109	43.3
INRC1-LH03	**38**	-	67	5.3	60	15.1
INRC1-LH04	**22**	-	53	4.8	40	14.1
INRC1-LH05	**41**	-	93	9.3	71	25.4
INRC1-LL01	**235**	-	290	11.2	283	33.9
INRC1-LL02	**229**	-	272	11.1	272	32.3
INRC1-LL03	**220**	-	307	7.3	296	24.0
INRC1-LL04	**222**	-	309	7.4	309	38.5
INRC1-LL05	**83**	-	103	4.4	98	12.9
INRC1-M01	**240**	-	248	7.2	247	21.4
INRC1-M02	**240**	-	250	6.8	246	21.6
INRC1-M03	**236**	-	244	6.7	244	16.1
INRC1-M04	**237**	-	245	7.4	244	18.4
INRC1-M05	**303**	-	315	9.6	311	25.7
INRC1-MH01	**111**	-	168	3.6	144	10.4
INRC1-MH02	**221**	-	270	6.5	253	17.0
INRC1-MH03	**34**	-	53	4.5	53	10.5
INRC1-MH04	**78**	-	92	3.6	89	14.6
INRC1-MH05	**119**	-	148	3.7	138	12.5
INRC1-ML01	**157**	-	190	2.7	184	9.0
INRC1-ML02	**18**	-	34	2.6	30	8.1
INRC1-ML03	**29**	-	46	2.2	44	6.7
INRC1-ML04	**35**	-	55	2.7	47	8.6
INRC1-ML05	**107**	-	150	4.1	145	23.7
Average	**164**		195	**8.4**	187	23.7

author has not tried to obtain these solutions. Their reported running times are better than the GOAL ones, never exceeding about 10 minutes.

KHE's results on these instances are quite good. The KHE18x8 results on `INRC1-L01` to `INRC1-L05` are very good: optimum and near-optimum costs, found in under about one minute each. This is competitive with [4] and [31].

4.3 The 2014 Curtois and Qu instances

The instances tested 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.

Table 3 Results for the Sprint instances of the First International Timetabling Competition. This table is derived from XESTT archive file `KHE18-2018-05-22-INRC1-Sprint.xml`, available at [20]. Other details as for Table 2.

Instance	GOAL		KHE18		KHE18x8	
	Cost	Time	Cost	Time	Cost	Time
INRC1-S01	**56**	-	59	0.6	59	2.5
INRC1-S02	**58**	-	60	0.6	60	2.1
INRC1-S03	**51**	-	57	0.6	54	2.1
INRC1-S04	**59**	-	64	0.6	63	1.9
INRC1-S05	**58**	-	62	0.8	60	2.7
INRC1-S06	**54**	-	60	0.7	56	1.4
INRC1-S07	**56**	-	59	0.6	59	2.3
INRC1-S08	**56**	-	59	0.8	58	2.2
INRC1-S09	**55**	-	57	0.6	57	2.3
INRC1-S10	**52**	-	54	0.6	54	1.5
INRC1-SH01	**32**	-	38	0.8	34	1.9
INRC1-SH02	**32**	-	43	0.6	37	1.8
INRC1-SH03	**62**	-	69	1.1	68	3.0
INRC1-SH04	**66**	-	75	1.5	73	4.2
INRC1-SH05	**59**	-	66	1.1	66	3.6
INRC1-SH06	**130**	-	191	0.6	183	2.4
INRC1-SH07	**153**	-	183	1.0	181	2.4
INRC1-SH08	**204**	-	242	1.1	237	3.2
INRC1-SH09	**338**	-	375	1.3	348	4.1
INRC1-SH10	**306**	-	319	0.8	319	2.3
INRC1-SL01	**37**	-	46	1.0	45	3.0
INRC1-SL02	**42**	-	46	1.0	46	2.6
INRC1-SL03	**48**	-	56	1.0	56	4.1
INRC1-SL04	**73**	-	91	0.6	91	2.3
INRC1-SL05	**44**	-	48	0.9	47	3.6
INRC1-SL06	**42**	-	46	0.6	44	1.5
INRC1-SL07	**42**	-	58	0.6	52	1.6
INRC1-SL08	**17**	-	20	0.2	18	0.5
INRC1-SL09	**17**	-	22	0.1	22	0.6
INRC1-SL10	**43**	-	51	0.3	50	1.5
Average	**78**		89	**0.8**	87	2.4

These four sets of solutions were made by the solvers reported on in Table 2 of [7], namely ejection chains (10 mins), ejection chains (60 minutes), branch and price, and Gurobi. However, they differ from the solutions presented in Table 2 of [7], for reasons explained in [21]: the solutions received were not exactly the ones reported on in the table, and some had to be omitted owing to extreme overstaffing of shifts, which is strictly speaking legal but which this author chose not to allow in the XESTT versions of the instances.

Another source of solutions to these instances is [10]. They are not competitive, so adding them to `CQ14.xml` has not been a priority.

The results appear in Tables 4 and 5. The author has only just begun to tackle these instances, and the results are not competitive, although there is some promise in the generally modest running times, and in the fact that the first four infeasible solutions in the KHE18x8 column are only a few hard

Table 4 Results for the instances published in 2014 by Curtois and Qu [7,8]. The CQ-BP and CQ-GR columns show two of the four sets of solutions from XESTT archive `CQ14.xml`, corresponding to the Branch and Price and Gurobi 5.6.3 columns of Table 2 of [7]. The other columns show the results from the two versions of the author's solver. This table is derived from XESTT archive file `KHE18-2018-05-22-CQ14.xml`, available at [20].

Instance	CQ-BP		CQ-GR		KHE18		KHE18x8	
	Cost	**Time**	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**
CQ14-01	**607**	0.4	**607**	-	710	0.2	710	0.6
CQ14-02			**828**	-	1036	0.6	1036	2.6
CQ14-03			**1001**	-	1218	1.1	1119	3.2
CQ14-04	**1716**	1.5	**1716**	-	2439	0.6	2134	2.5
CQ14-05	1160	25.6	**1143**	-	2253	1.6	2062	5.3
CQ14-06	1952	10.5	**1950**	-	inf.	3.3	2975	11.0
CQ14-07	1058	93.7	**1056**	-	1897	4.2	1589	13.3
CQ14-08	**1308**	11831.1	1323	-	inf.	11.1	inf.	30.7
CQ14-09			**439**	-	818	20.7	544	45.9
CQ14-10			**4631**	-	inf.	17.6	5637	41.2
CQ14-11			**3443**	-	inf.	15.4	4037	53.9
CQ14-12	4046	1336.4	**4040**	-	5581	134.3	5581	250.8
CQ14-13			**1388**	-	3438	300.1	3551	300.5
CQ14-14			**1280**	-	inf.	17.5	2608	38.8
CQ14-15			**4039**	-	inf.	46.6	6471	100.6
CQ14-16	3323	265.0	**3233**	-	inf.	7.2	inf.	29.3
CQ14-17			**5851**	-	inf.	28.6	inf.	84.3
CQ14-18			**4760**	-	inf.	25.6	inf.	58.5
CQ14-19			**3218**	-	inf.	105.2	6355	346.2
CQ14-20					inf.	300.2	**inf.**	300.5
CQ14-21					inf.	301.0	**inf.**	301.2
CQ14-22					inf.	300.5	**inf.**	301.0
CQ14-23			**17428**	-	inf.	300.9	inf.	302.0
CQ14-24			**48777**	-	inf.	301.8	inf.	308.5
Average						**93.6**		122.2

constraint violations away from feasibility. The later instances are very large, and even finding a feasible solution is challenging.

5 Conclusion

This paper has presented KHE18, a solver for nurse rostering which aims to produce very good but not optimal solutions quickly across a wide range of instances. Polynomial-time methods are used, including time sweep for the initial assignment, and ejection chains for repair.

This paper reports on work in progress. At present, KHE18 is producing some very good solutions, but overall its results are not competitive, except in running time. What is needed now is more analysis of its behaviour, leading, hopefully, to new insights which can be incorporated into the algorithm.

One can see from the leading recent papers [4,30] that provably optimal solutions to realistic instances are becoming available using methods based on integer programming. There is still a role for algorithmic methods, however,

Table 5 Results for the instances published recently by Curtois and Qu [7,8]. As for Table 4, only showing the CQ-EJ10 and CQ-EJ60 columns, corresponding to the ejection chain (10 mins) and ejection chain (60 mins) columns of Table 2 of [7].

Instance	**CQ-EJ10**		**CQ-EJ60**		**KHE18**		**KHE18x8**	
	Cost	**Time**	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**
CQ14-01	1114	0.0			**710**	0.2	**710**	0.6
CQ14-02	1019	30.0	**837**	3600.0	1036	0.6	1036	2.6
CQ14-03	**1001**	2.6	1003	3600.0	1218	1.1	1119	3.2
CQ14-04	**1716**	8.2	1718	3600.0	2439	0.6	2134	2.5
CQ14-05	**1144**	76.8	1358	3600.0	2253	1.6	2062	5.3
CQ14-06	**1952**	88.8	2258	3600.0	inf.	3.3	2975	11.0
CQ14-07	**1056**	262.1	1269	3600.0	1897	4.2	1589	13.3
CQ14-08	2133	600.0	**1314**	3600.0	inf.	11.1	inf.	30.7
CQ14-09	1449	0.6	**439**	635.2	818	20.7	544	45.9
CQ14-10	4870	600.0	**4631**	863.2	inf.	17.6	5637	41.2
CQ14-11	**3443**	171.4	3661	3600.0	inf.	15.4	4037	53.9
CQ14-12	5476	600.0	**4040**	1526.1	5581	134.3	5581	250.8
CQ14-13	11432	30.1	**1486**	3600.1	3438	300.1	3551	300.5
CQ14-14	2286	600.0	**1300**	3600.3	inf.	17.5	2608	38.8
CQ14-15	12050	30.1	**4378**	3600.0	inf.	46.6	6471	100.6
CQ14-16	3926	30.0	**3225**	2965.7	inf.	7.2	inf.	29.3
CQ14-17	7801	600.0	**5872**	3600.1	inf.	28.6	inf.	84.3
CQ14-18	10002	30.0	**4969**	3600.0	inf.	25.6	inf.	58.5
CQ14-19	14788	30.0	**3715**	3600.0	inf.	105.2	6355	346.2
CQ14-20					inf.	300.2	**inf.**	300.5
CQ14-21					inf.	301.0	**inf.**	301.2
CQ14-22					inf.	300.5	**inf.**	301.0
CQ14-23			**44819**	3601.0	inf.	300.9	inf.	302.0
CQ14-24					**inf.**	301.8	inf.	308.5
Average						**93.6**		122.2

in finding good solutions for large instances with reliably fast running times. It is not clear to this author that *very* large instances like the last few of the 2014 Curtois and Qu instances really do need to be solved in the real world. However, if they do, that would be another role for algorithmic methods.

Note. Shortly after submitting the original version of this paper the author inadvertently allowed his software to overwrite the archive files holding its results. Those results cannot now be verified, so in this version of the paper he has replaced them with results produced on 22 May 2018. These can be verified, using archive files accessible from the XESTT web site [20]. The algorithm has not changed significantly, but code tuning has allowed it to do more within the same time limit, producing better results on the larger instances.

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), 411419 (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.cs.nott.ac.uk/~psztc/NRP/ (2016)
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. Kathryn A. Dowsland, Nurse scheduling with tabu search and strategic oscillation, European Journal of Operational Research 106, 393–407 (1998)
12. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stlevik, and Andrea Schaerf, First international nurse rostering competition website, URL: http://www.kuleuven-kortrijk.be/nrpcompetition (2010)
13. Stefaan Haspeslagh, Patrick De Causmaecker, Martin Stlevik, and Andrea Schaerf, The first international nurse rostering competition 2010, Annals of Operations Research, 218, 221–236 (2014)
14. 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)
15. 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)
16. Jeffrey H. Kingston, The HSEval High School Timetable Evaluator, URL http://jeffreykingston.id.au/cgi-bin/hseval.cgi (2010)
17. Jeffrey H. Kingston, Repairing high school timetables with polymorphic ejection chains, Annals of Operations Research, DOI 10.1007/s10479-013-1504-3
18. 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
19. Jeffrey H. Kingston, KHE web site, http://jeffreykingston.id.au/khe (2014)
20. Jeffrey H. Kingston, XESTT web site, http://jeffreykingston.id.au/xestt (2017)
21. 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)
22. Jeffrey H. Kingston, Modelling history in nurse rostering, PATAT 2018 (Twelfth international conference on the Practice and Theory of Automated Timetabling, Vienna, August 2018)
23. 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.

24. Antonn Novák, Roman Václavk, 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
25. Gerhard Post, XHSTT web site, http://www.utwente.nl/ctit/hstt/ (2011)
26. 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)
27. 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)
28. 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)
29. 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
30. 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)
31. Haroldo G. Santos, Túlio A. M. Toffolo, Rafael A. M. Gomes, and Sabir Ribas, http://www.goal.ufop.br/nrp/
32. 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
33. 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).