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

KHE20: An Improved Solver for Nurse Rostering

Jeffrey H. Kingston

Received: date / Accepted: date

Abstract Nurse rostering, the problem of assigning nurses to the shifts of a hospital ward, has been studied for many years. This paper describes the KHE20 nurse rostering solver, an improved version of the KHE18 solver. It uses a time sweep algorithm to find an initial solution, followed by repair using several methods, principally ejection chains. Experiments are conducted on four well-known data sets. These show that KHE20 is competitive, taking breadth of application and running time into account as well as solution cost. Other experiments are conducted which shed light on the contributions made by the various parts of the algorithm.

Keywords Nurse rostering · Time sweep · Ejection chains · XESTT

1 Introduction

Nurse rostering, the problem of assigning nurses to the shifts of a hospital ward, 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 using integer programming [4,35].

Many inexact methods have been tried. Even very recent work covers a wide range: integer programming [15,28,31,35,39], weighted maxSAT [10,11], simulated annealing [9,37], hyper-heuristics [2,16,32], variable neighbourhood search [38], and constraint programming [33]. For less recent work, see [40].

The solver presented here, KHE20, is the 2020 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 3.2).

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

This timetables the first day of the cycle, then the second, and so on. It then tries several repair methods, notably ejection chains (Section 3.3).

KHE20 has been tested on four data sets (Section 4). Although it has not found any new best solutions, on its own terms (that is, considering running time and range of applicability as well as cost) it is competitive.[1]

2 The nurse rostering problem and its XESTT formulation

Nurse rostering is the problem of assigning shifts to the nurses of a hospital ward.[2] 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.

Instead of the usual formulas, this paper's formal definition of the nurse rostering problem is supplied by the XESTT [23] 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 [29,30]; 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

[1] *Note to referees.* This paper is my previous nurse rostering solver paper updated to take account of two years' more work. My previous paper [22] was published at PATAT 2018 as an abstract of work in progress and was not submitted for journal publication. This time around I did consider producing a short paper just describing the updates and giving the new results, but such a paper would have been so un-self-contained as to be virtually unreadable. This paper is self-contained, but inevitably there is a lot of overlap with [22]. So I have placed footnotes like this one at the start of each section, indicating the major changes from [22]. If this paper is accepted for PATAT 2020 I intend to remove these footnotes from the conference version. Many small changes lie below the level of description of this paper, but have nevertheless contributed to the significantly better results.

[2] *Note to referees.* No changes to this section.

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 [23] explains, most nurse rostering constraints concern shifts, not workload in minutes, and this works well 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. 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 KHE20 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. It can do this because its constraints accept arbitrary time groups, allowing the same type of constraint to deal with days, night shifts, weekends, and so on. 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 KHE20 solver

The KHE20 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]. 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 convert the time preassignments in the instances into time assignments in the solutions, taking almost no time.

After time assignment comes resource assignment—the assignment of nurses to shifts. KHE20 first carries out a *construction phase* which builds an initial assignment using time sweep, then continues with a *repair phase* which tries several kinds of repairs that improve that assignment. This is run twice (**R2**), to enable repairs of one kind to open the way to repairs of other kinds.

A full presentation appears in the KHE documentation [20]. This section focuses on the main points. Labels, for example the '(**R2**)' just above, name aspects of the algorithm whose contributions are investigated in Section 5.

In the KHE platform, instances and solutions are distinct objects. Instances are immutable after creation, whereas solutions change as solving proceeds. A solution consists of a set of *meets*, each representing one event. Within each meet there are *tasks*, one for each event resource of the meet's event. Each task is, in effect, 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' can mean both 'task' and 'meet'.

3.1 Resource matching and rematching

Resource matching is KHE's name for a polynomial time algorithm, based on weighted bipartite matching, for assigning resources (nurses) to a set of initially unassigned tasks (shifts).[3] *Resource rematching* is a variant of resource matching in which the tasks are initially assigned; it changes the assignments. This section describes both algorithms, starting with resource matching.

Suppose hard constraints limit each resource to at most one task per day, as is usual in nurse rostering. Each call on resource matching targets one day. It begins by building a weighted bipartite graph. The graph contains one *demand node* for each task of each meet running on that day; each is a demand for one resource. It also contains one *supply node* for each resource. An edge joins a demand node to a supply node when the resource can be assigned to the task (almost always in nurse rostering; the few exceptions need not detain us). The cost of the edge is the cost of the solution containing whatever assignments are currently in force on other days, plus just this one assignment on the current day. For each demand node there is also a supply node representing not assigning any resource to its task. The cost of this edge is just the current solution cost. This usually includes a penalty for not assigning a resource, which is absent from the edges which do assign one.

If the assignments represented by the edges affect cost independently, a minimum-cost maximum matching in this graph defines an optimal assignment of resources to the tasks of the day, because each task demands one resource, and each resource can be assigned to at most one task. So this matching is found and the indicated assignments are made.

[3] *Note to referees.* This section contains a new treatment of limit resources constraints.

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 resource, for example, are affected only by assignments of that resource, 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 senior nurse to be on duty at all times of the day, which is not the same as requiring one senior nurse on every shift, because shifts may overlap in time within one day.

The author's NRConv program tries to avoid generating limit resources constraints when converting instances from other formats [23]. Only ten of the many converted instances currently available actually use them. All ten are Curtois original instances (Section 4.1).

Rather than abandoning matching when limit resources constraints are present, the algorithm approximates them by adjusting edge weights. Thus, to encourage one senior nurse to be present, the weights on the edges from one demand node to the supply nodes for non-senior nurses are increased. Optimality is preserved in simple cases. After assigning one day, if any of the limit resources constraints for that day are not satisfied, a brief ejection chain repair (Section 3.3) is run, focusing on them (**CE**).

The edge costs are adjusted slightly, to break ties in favour of assigning resources with more unused workload (**CW**), in favour of assignments that do not bring constraints from below their upper limits to their upper limits (**CL**), and in favour of shorter runs of consecutive busy days (**CB**), hoping that short runs will be more flexible when repairing the timetable later.

Resource rematching of the tasks of one day is just unassignment of those tasks followed by resource matching. However, KHE offers a form of resource rematching which targets an arbitrary set of days for reassignment, not just one. The days need not be consecutive, although in KHE20 they always are.

For each resource r, the tasks initially assigned r on the targeted days are grouped into a single demand node. Each initially unassigned task on the targeted days goes into its own demand node. Then resource matching is applied. The cost of each edge is the cost of the solution containing all of the assignments on non-targeted days, plus the assignments of the edge's resource to all of the tasks of its demand node, adjusted as before.

When there are two or more days, the result is not locally optimal, because of the grouping. However, it may still be an improvement. Finding a locally optimal reassignment for an arbitrary set of days is clearly NP-complete.

Being able to target an arbitrary set of days for reassignment is something of an embarrassment of riches. Even limiting to consecutive days, for multi-week instances there are still many choices. The author has not attempted a systematic search of this large space. Instead, KHE20 tries a relatively small number of resource rematches that seem likely to do well, as explained below.

3.2 Time sweep assignment

Because of the high density of constraints in nurse rostering, it may be easier to avoid introducing a problem initially than to remove it later.[4] So it makes sense to try hard to produce a very good initial solution [26].

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 usually be satisfied. This we call the *time sweep* method. KHE20 uses it, assigning each day using resource matching (Section 3.1).

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 two, then the first assignment after a free day will attract a spurious cost. KHE20 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 unless there is good reason (such as a preassignment, or a request for a day off) and make its best estimate of cost (a lower bound on the true cost) accordingly (**CS**). See [24] for a full description.

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, that day and the preceding day are jointly reassigned using resource rematching (Section 3.1), then that day and the two preceding days are jointly reassigned, and so on, up to a small limit (currently four days) and subject to a time limit (**CR**). Also, after the whole time sweep has ended, each day from first to last is individually rematched (**CI**). This may produce small changes that coordinate better with the assignments now present on following days. It is acknowledged that the choice of sets of days to rematch is fairly arbitrary here.

The only previous use of time sweep known to the author is [26], which finds initial solutions one week at a time in chronological order. It cites a recent aircrew scheduling paper [34] as its inspiration. The general idea seems to be well known, although not the details described here and in Section 3.1.

3.3 Ejection chain repair

After constructing an initial solution using time sweep, KHE20 repairs it using several methods.[5] Only two are significant in either the amount of improvement achieved or in running time. One of these is resource rematching (Section 3.1), applied to a small number of fairly large sets of consecutive days, up to seven (**RM**). The other is the subject of this section: *ejection chain repair* (**EE**).

[4] *Note to referees.* Only minor changes here.

[5] *Note to referees.* Major changes here. The ejection chain repairs were still evolving at the time of the previous paper, and the description there was impressionistic. This time around there is a full description of a settled set of repairs.

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 was 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 KHE20 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*.

It is not hard to discover suitable repairs, as was just done for nurse `N2`'s busy weekends. We do this now in general, working from the bottom up.

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. KHE20 also has a single main repair operation, called the *task-set swap*. Let r_1 and r_2 be two resources such that $r_1 \neq r_2$; either may be `NULL`, but not both. Let S_1 be a set of tasks assigned to r_1 (unassigned if r_1 is `NULL`). Let S_2 be a set of tasks assigned to r_2 (unassigned if r_2 is `NULL`). The operation moves the tasks of S_1 to r_2 and the tasks of S_2 to r_1. S_1 must be non-empty, but S_2 may be empty, in which case the operation is also called a *task-set move*.

Two conditions on when a task-set swap may be applied are imposed, to save time by avoiding swaps that are unlikely to be useful. First, none of the tasks involved may be preassigned. Second, r_1 (when non-`NULL`) must not attend two tasks on the same day afterwards, and the same for r_2.

In some ways, the task-set swap is more general than the timetable swap: since r_1 or r_2 may be `NULL`, tasks may become assigned or unassigned; and S_1 and S_2 need not be on the same days. One can also show formally that every timetable swap is also a task-set swap, by setting S_1 to r_1's tasks on some days, and S_2 to r_2's tasks on the same days. However, KHE20 usually only swaps tasks on consecutive busy days. More general timetable swaps, involving (say) a mixture of busy and free days spread over a week, can reduce cost, but they do not closely target specific defects, as needed by ejection chains.

Most defects can be repaired with a single task move. For example, if a resource is busy for too many consecutive days, moving any one of its tasks on those days to any other resource removes the defect. Occasionally more than one task move is needed—for example, to free up one of `N2`'s busy weekends.

Suppose some defect can be repaired by moving a set of tasks S from r_1 to r_2. This *initial repair* might succeed, but there are several common reasons why it might fail. KHE20 tries to anticipate these problems by *expanding* the initial repair into a set of alternative repairs. Each of these repairs is a task-set move or swap that includes the original moves, and so repairs the defect, but adds other moves that may help the repair to succeed.

One common reason for failure is that the repair is too disruptive of the timetables of r_1 and r_2 on the days near S. KHE20 counters this by a form of expansion called *widening* (**EW**), which adds task moves from r_1 to r_2 of tasks on days adjacent to the days of S. One repair is tried for each set of up to $max(|S|, 4)$ tasks on consecutive busy days for r_1 (days with unassigned tasks if r_1 is `NULL`) that include S. Often the full range cannot be used, because r_1 is free or preassigned on nearby days, or r_2 is busy.

Another common reason for failure is that r_2 is busy on some of the days of S, so that the move would give r_2 two tasks on the same day. This is countered by a form of expansion called *reversing* (**ER**), which moves r_2's tasks on the days of S to r_1, that is, swaps their timetables, although only on consecutive days when both resources are busy. Widening is used with reversing, as before except that only days when both resources are busy are included.

A third common reason for failure is that all nurses are working at or near their global or local workload limits, so that the move underloads r_1 or overloads r_2. This is countered by a third form of expansion called *balancing* (**EB**), which applies to each unreversed repair after widening. It identifies a set of tasks currently assigned r_2 (unassigned tasks if r_2 is `NULL`) on consecutive days when r_1 is free, and moves those tasks from r_2 to r_1. The number of tasks is the same as the number being moved the other way, since the aim is to keep the workloads of the two resources unchanged. If r_2 is not `NULL`, only tasks at the ends of sequences of consecutive busy days for r_2 are tried. For each widened task-set move, the number of alternative repairs tried by balancing

is at most 12. Days as close as possible to (but not overlapping with) S are tried, since they are more likely to keep local workload limits balanced.

To summarise: whenever a move of S from r_1 to r_2 is tried, widening and balancing are also tried when r_2 is free on the days of S, and widening and reversing are also tried when r_2 is busy on the days of S. If S contains several tasks and r_2 is free on some of their days and busy on others, the repair is *blocked*, that is, not tried. It is also blocked when r_2 is busy but the reverse move would reinstate the defect that the initial move removed. While not strictly necessary, this is in the spirit of ejection chains and saves time by avoiding evidently poor repairs. These cases are detailed below.

Nurse rostering often has *optional tasks*: tasks that may be left unassigned without cost. When moving tasks from one resource to another, optional tasks in the way of the move are treated like free time, and unassigned as part of it.

It remains to describe how, for each kind of defect, a set of alternative initial repairs is found.

If some constraint's cost function is a step function (if the constraint has the same cost for any non-zero degree of violation), then a repair must reduce the degree of violation to zero to reduce cost. The following repairs do not yet understand this. Constraints with step cost functions are rare, and violations of them of degree 2 or more are even rarer, which excuses this deficiency.

XESTT has three cover constraints (Section 2). Their defects always consist of some tasks and resources, such that too few or too many of the tasks are assigned the resources. If a meet is understaffed, the tasks are the meet's and the resources are all nurses. If there are too many trainees at 3pm, the tasks are those running at 3pm and the resources are the trainees. And so on.

A task move is needed which changes the assignment of one of the tasks, so as to increase or decrease the number assigned the resources, as required. All such moves are tried, expanded using widening, reversing, and balancing as above. Reversing blocks when the task it moves back is part of the defect.

Suppose that a task which prefers to be unassigned is assigned resource r. Then, proceeding as just described, the only repair tried is the unassignment of the task, possibly widened. But the task was probably assigned because r needed to be busy at its time, in which case these repairs will lead to nothing if r needs to be busy for more than one reason. So in this case, for each other task running at the same time, KHE20 tries to move that task to r. The original task will be treated like free time and unassigned, as explained above.

XESTT offers six types of resource constraints, but most resource defects consist of a resource r and a set of sets of times, such that r is busy for too many or too few sets. (A resource is busy for a set of times when it is busy at one or more of the times.) For example, when r is busy six days in a row, but the limit is five, for each day d the times of d make one set. For nurse `N2` and the three busy weekends, for each weekend w the times of w make one set.

What is needed here is one task move which makes r busy during a free set of times, or one or more task moves which make r free during a busy set. All such repairs are tried, expanded as usual. Reversing blocks when the reverse move makes r busy during the set of times that the initial move aims to free

up. For example, if the aim is to make r free on some day, all reversing is blocked; but if the aim is to ensure that r does not work the night shift on some day, only reverse moves that assign a night shift to r are blocked.

In some unwanted pattern constraints, some of the sets of times need to be free and others need to be busy. Some repairs need to increase or decrease the number of busy times within a set of times, not make the set busy or free. These are simple variants and cause no problems.

The author knows of three other uses of 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. The first [12] includes a repair which swaps the timetables of two resources over one week. The second [27] uses chains of task moves as repairs 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 [27] 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. It allocates equal running time to each top-level search. Its results are compared with KHE20's in Section 4.

The present paper's ejection chain algorithm is adapted from recent work in high school timetabling [18, 19]. Its polymorphism seems to be new to nurse rostering. The works cited just 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 is less affected by high constraint density; on the other, it reduces the precision with which repairs can address specific defects. One paper [3] adds precision by selecting repairs which focus on parts of the resource's timetable where defects lie.

3.4 Task grouping

The KHE platform allows tasks to be *grouped* (**GR**).[6] Assigning a resource to one element of a group assigns it to all. This 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 Friday night task should also take the following Saturday and Sunday night tasks, 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.

[6] *Note to referees.* In this section the description of profile grouping is new, although KHE18 used it. The special treatment of equal minimum and maximum limits is new, as is the division into separate instances where possible.

KHE20 has a grouping phase which runs before time sweep. Its first step is *combinatorial grouping*. For each sequence of k consecutive days (k is arbitrary; KHE20 tries 2 and 3), and for each task s on the first or last of these days, carry out one trial as follows. Find all combinations of tasks and free time over the k days which include s. If only one combination has zero cost, group its tasks. Only count costs that depend only on the days of the trial and are the same for all resources.

If nurses must have a certain number of weekends off, that may only be achievable if all their working weekends are complete, even without an explicit 'complete weekends' constraint. KHE20 detects this and similar cases and uses them to reduce the number of combinations, making grouping more likely.

One grouping may lead to another. For example, if complete weekends are required, and a day task cannot follow a night task, then the Saturday night tasks may be grouped with the Sunday night tasks, but only after that does it become possible to group the Saturday and Sunday day tasks.

The second step is *profile grouping*. Suppose there is a constraint, applicable to every nurse, requiring each nurse's tasks of a certain type (night tasks, for example) to occur in consecutive sequences of length at least k, where $k \geq 2$. Now suppose that n_i tasks of this type need to be assigned on day i, and n_{i+1} of them need to be assigned on day $i+1$. (The n_i form the *profile* of the constraint.) If $n_{i+1} > n_i$, then $n_{i+1} - n_i$ sequences of k or more night tasks must begin on day $i+1$, and so this many night tasks may be grouped with night tasks on the following $k-1$ days. Similary, if $n_{i+1} < n_i$, then $n_i - n_{i+1}$ sequences of k or more night tasks must end on day i, and so this many night tasks may be grouped with night tasks on the preceding $k-1$ days.

The grouping of the Monday and Tuesday (and Wednesday and Thursday) night tasks in instance `COI-GPost`, mentioned earlier, is the result of profile grouping. The previously grouped Friday, Saturday, and Sunday night tasks already have maximum length and so are omitted from the profile.

When the constraint concerns several types of tasks, for example when it constrains the number of consecutive busy days, profile grouping is not usually tried, because it is not clear which types of tasks to group with which. An exception is made, however, when the constraint's minimum and maximum limits are equal. Such constraints can have a profound effect, as the reader can see by imagining repairing one sequence of busy days of incorrect length in an otherwise perfect timetable, when there is no freedom to vary the number of resources assigned to each meet: the defect has to be moved, step by step, to the start or end of the timetable, where typically the constraint does not apply. So KHE20 performs profile grouping in these cases, using a form of combinatorial grouping to choose the tasks to group together.

Grouped tasks must accept similar resources. It would not do, for example, to group one which requires a senior nurse with one which requires a trainee. In practice, this is easy to ensure; if not, the grouping is omitted. In some cases, assigning certain nurses (trainees, perhaps) can be seen to be an independent problem. In those cases KHE20 divides the instance into (in effect) separate instances. Full details appear in the KHE documentation [20].

This paper's algorithms handle grouped tasks correctly. For example, each task-set swap (Section 3.3) contains either all the tasks of a group, or none.

A grouping will occasionally be inadvisable for some unexpected reason. So KHE20 applies the groupings during the initial time sweep and first repair phase, 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.

3.5 Making good use of available running time

Running time is a major issue for the larger instances, so it must be used well.[7] KHE20 is not suited to a single global time limit, because it has many repair phases: two for each day during construction, and two at the end. These yield diminishing returns as running time increases, so it would be a mistake for KHE20 to spend all its time on one and have nothing left for the others.

KHE20 limits each day during time sweep (including that day's repairs) to 2 seconds, each day's rematch at the end of time sweep to 1 second, the first repair phase to 120 seconds, and the second to 60 seconds. In both repair phases, resource rematching is limited to half the available time, leaving the rest for the other repair methods, mainly ejection chains. A six-week instance should take about 5 minutes to solve at most, which seems reasonable, given KHE20's aim of finding good but not optimal solutions quickly.

The ejection chain algorithm of [7] imposes a time limit on each search for a chain that repairs one defect. KHE20 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 limits exactly. On the one hand, many phases end of their own accord well before their 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. Actual running times are reported in the results tables of Section 4.

An unfortunate consequence of having many small time limits is a loss of repeatability in the experiments. One can always verify that the solutions found have the costs claimed, but when a large instance is solved twice with the same random seed the solutions are usually different, owing (as the author has verified) to time limits being reached at slightly different points in the run.

3.6 Randomization

Randomization is not stressed in algorithmic solvers like it is in, for example, metaheuristic ones.[8] 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.

[7] *Note to referees.* This section is new. KHE18 had only a single time limit.

[8] *Note to referees.* Nothing new here.

Resource matching randomizes by cycling through the resources starting from a random point when creating supply nodes, causing ties in edge weights to be broken differently, at least at the start of time sweep when many resources have equal available workload. Ejection chain repair randomizes by trying alternative repairs starting at 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 KHE20 on several well-known sets of instances, after conversion to XESTT by the NRConv program [21,23], with comparisons with previous results.[9] The converted instances and results are available, in the form of XESTT archive files, at [21].

Two versions of the solver are tested. The first, KHE20, runs the solver once and produces one solution. The second, KHE20x8, runs it 8 times, with a different random seed on each run, and keeps the best solution. All solves were run on the author's machine, a quad-core 3.6GHz Intel Core i7-4790.

The KHE20x8 results are produced in parallel using 4 threads. For each instance, four solves are started initially, one per thread. New solves are started in these threads as old ones end, until 8 solves have been started. Once started, a solve runs until it stops itself. After all solves are complete, the running time of the best solution is set to the total wall clock time since solving the instance began, and the four threads start again on the next instance.

The KHE20 results are produced by saving separately the result of the first solve initiated by each KHE20x8 test, with its individual running time. When done this way, the average running time per instance is about one second longer than when the instances are solved sequentially. This is presumably due to contention for shared resources, such as the memory bus or memory allocator. It is small enough to ignore for the sake of speeding up the testing.

Time spent reading and writing the instances and solutions is not included in the running time, since all are read together at the start of the run, and all are written together at the end. The largest archive, CQ14, takes 8.1 seconds to read at the start and 1.9 seconds to write at the end. When distributed fairly over its 24 instances this extra time is 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

[9] *Note to referees.* There are only minor changes to the text here, but the results are much better. For example, in Table 1, the average cost of KHE20x8's solutions is 665, whereas it was 4637 before; 11 of KHE20x8's solutions are best solutions now, whereas only 3 were before. The results overall are competitive now; before, they were not. Section 4.3 is new.

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 solver described in this paper. Here and elsewhere, running times are in seconds. This table is derived from XESTT archive file `KHE20-2020-01-20-COI.xml`, available at [21].

Instances (27)	**Misc**		**KHE20**		**KHE20x8**	
	Cost	**Time**	**Cost**	**Time**	**Cost**	**Time**
COI-Ozkarahan	**0**	-	**0**	0.0	**0**	0.0
COI-Musa	**175**	-	**175**	0.4	**175**	0.8
COI-Millar-2.1	**0**	1.0	**0**	0.0	**0**	0.1
COI-Millar-2.1.1	**0**	-	**0**	0.0	**0**	0.1
COI-LLR	**301**	10.0	302	1.2	302	4.0
COI-Azaiez	**0**	600.0	1	0.4	**0**	0.8
COI-GPost	**5**	-	13	0.5	12	0.9
COI-GPost-B	**3**	-	5	0.5	5	1.0
COI-QMC-1	**13**	-	18	0.8	16	2.7
COI-QMC-2	**29**	-	**29**	2.3	**29**	4.3
COI-WHPP	**5**	-	3003	5.8	3000	12.7
COI-BCV-3.46.2	**894**	17840.0	896	6.5	**894**	16.0
COI-BCV-4.13.1	**10**	-	**10**	0.5	**10**	1.3
COI-SINTEF	**0**	-	**0**	0.2	**0**	0.5
COI-ORTEC01	**270**	105.0	350	1.8	330	5.3
COI-ORTEC02	**270**	-	355	2.3	300	7.4
COI-ERMGH	**779**	124.0	809	299.5	808	598.0
COI-CHILD	**149**	-	**149**	238.6	**149**	494.1
COI-ERRVH	**2001**	-	2248	292.1	2148	597.2
COI-HED01	**136**	-	167	3.0	151	10.6
COI-Valouxis-1	**20**	-	100	0.4	80	1.2
COI-Ikegami-2.1	**0**	13.0	**0**	5.4	**0**	11.7
COI-Ikegami-3.1	**2**	21600.0	18	10.5	8	21.2
COI-Ikegami-3.1.1	**3**	2820.0	8	10.2	8	31.9
COI-Ikegami-3.1.2	**3**	2820.0	13	12.2	7	25.6
COI-BCDT-Sep	**100**	-	270	2.8	230	5.1
COI-MER	**7081**	36002.7	9282	282.0	9282	566.4
Average	**454**		675	43.7	665	89.7

as 'inf.' (infeasible). No costs are reported on trust; all are calculated from the solutions in the archive file, and hence verified, by HSEval.

Running times are shown (in seconds) where present in the archive. HSEval cannot verify them. KHE20 and KHE20x8 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. For example, 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 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 [36], 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 `KHE20-2020-01-20-INRC1-Long-And-Medium.xml`, available at [21].

Instances (30)	**GOAL**		**KHE20**		**KHE20x8**	
	Cost	**Time**	**Cost**	**Time**	**Cost**	**Time**
INRC1-L01	**197**	-	200	64.4	**197**	170.5
INRC1-L02	**219**	-	228	39.3	225	96.2
INRC1-L03	**240**	-	242	52.2	241	109.6
INRC1-L04	**303**	-	306	54.8	306	108.3
INRC1-L05	**284**	-	287	37.0	286	97.9
INRC1-LH01	**346**	-	377	111.8	364	249.9
INRC1-LH02	**89**	-	105	73.5	95	150.2
INRC1-LH03	**38**	-	46	45.8	46	122.0
INRC1-LH04	**22**	-	37	71.8	31	102.9
INRC1-LH05	**41**	-	57	37.7	51	113.4
INRC1-LL01	**235**	-	258	91.5	253	234.5
INRC1-LL02	**229**	-	262	100.9	247	213.8
INRC1-LL03	**220**	-	272	97.8	256	210.9
INRC1-LL04	**222**	-	258	93.8	256	202.1
INRC1-LL05	**83**	-	88	13.4	85	35.1
INRC1-M01	**240**	-	245	6.0	245	19.4
INRC1-M02	**240**	-	248	3.7	245	10.1
INRC1-M03	**236**	-	243	6.4	242	12.1
INRC1-M04	**237**	-	244	5.0	241	12.3
INRC1-M05	**303**	-	315	8.2	310	17.3
INRC1-MH01	**111**	-	145	19.2	121	46.4
INRC1-MH02	**221**	-	246	24.5	245	46.2
INRC1-MH03	**34**	-	43	9.5	38	25.0
INRC1-MH04	**78**	-	89	15.8	85	36.4
INRC1-MH05	**119**	-	133	9.3	133	31.8
INRC1-ML01	**157**	-	183	10.7	174	16.2
INRC1-ML02	**18**	-	31	2.4	29	4.5
INRC1-ML03	**29**	-	46	2.6	36	6.2
INRC1-ML04	**35**	-	41	5.1	41	13.3
INRC1-ML05	**107**	-	139	6.8	132	20.0
Average	**164**		180	37.4	175	84.5

Table 1 compares KHE20 with the solutions posted at [8]. It is important to analyse what is happening, and not simply take these results at face value. For example, 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. Taking this kind of analysis into account, the author rates KHE20x8 as competitive, except that in four instances the running time is slow for an algorithm with no pretensions to optimality.

Another instructive instance is `COI-BCDT-Sep`. It prefers night shifts to occur in blocks of three, but because of a slight peculiarity (a block of four night shifts starting on a Wednesday is acceptable), that is not expressed in the usual way, and so profile grouping (Section 3.4) is not triggered. By adding

Table 3 Results for the Sprint instances of the First International Timetabling Competition. This table is derived from XESTT archive file `KHE20-2020-01-20-INRC1-Sprint.xml`, available at [21]. Other details as for Table 2.

Instances (30)	**GOAL**		**KHE20**		**KHE20x8**	
	Cost	**Time**	**Cost**	**Time**	**Cost**	**Time**
INRC1-S01	**56**	-	58	1.2	57	2.2
INRC1-S02	**58**	-	60	0.5	**58**	2.5
INRC1-S03	**51**	-	52	2.1	52	2.7
INRC1-S04	**59**	-	61	1.0	**59**	3.1
INRC1-S05	**58**	-	**58**	1.2	**58**	2.3
INRC1-S06	**54**	-	**54**	1.5	**54**	2.5
INRC1-S07	**56**	-	57	1.1	57	2.0
INRC1-S08	**56**	-	**56**	1.1	**56**	2.5
INRC1-S09	**55**	-	57	0.8	56	2.1
INRC1-S10	**52**	-	53	1.6	53	3.1
INRC1-SH01	**32**	-	36	0.3	34	0.6
INRC1-SH02	**32**	-	37	0.2	37	0.5
INRC1-SH03	**62**	-	65	0.9	63	1.4
INRC1-SH04	**66**	-	68	1.1	68	2.9
INRC1-SH05	**59**	-	68	0.3	**59**	1.1
INRC1-SH06	**130**	-	139	0.4	136	1.3
INRC1-SH07	**153**	-	156	0.3	156	1.1
INRC1-SH08	**204**	-	215	1.2	210	2.9
INRC1-SH09	**338**	-	343	3.4	342	5.7
INRC1-SH10	**306**	-	332	0.5	327	1.6
INRC1-SL01	**37**	-	42	0.7	39	1.6
INRC1-SL02	**42**	-	45	0.4	44	1.3
INRC1-SL03	**48**	-	51	0.6	51	2.1
INRC1-SL04	**73**	-	88	0.7	80	1.9
INRC1-SL05	**44**	-	47	1.6	47	2.6
INRC1-SL06	**42**	-	45	0.7	**42**	1.4
INRC1-SL07	**42**	-	47	0.7	43	1.9
INRC1-SL08	**17**	-	20	0.3	20	0.8
INRC1-SL09	**17**	-	29	0.3	23	1.0
INRC1-SL10	**43**	-	47	0.3	45	1.4
Average	**78**		83	0.9	81	2.0

the 'missing' constraint by hand, the author was able to induce KHE20x8 to produce a solution of cost 180.

Other tests, not reported in detail here, show that the time limits are about right. For the instances with longer running times, halving the limits increases cost significantly. Doubling them can reduce cost; for example KHE20x8 then produces a solution to `COI-MER` of cost 8254, taking 19 minutes [25].

4.2 The First International Nurse Rostering Competition

The First International Nurse Rostering Competition instances are available from the competition web site [13]. Their converted versions appear in files `INRC1-Long-And-Medium.xml` and `INRC1-Sprint.xml` [21], with the GOAL research group's solutions [36], proved by GOAL to be virtually optimal.

Table 4 Results for the Second International Timetabling Competition 4-week instances. The LOR17 column shows the solutions obtained from the authors of [26]. This table is derived from XESTT archive file `KHE20-2020-01-20-INRC2-4.xml`, available at [21].

Instances (20)	**LOR17**		**KHE20**		**KHE20x8**	
	Cost	**Time**	**Cost**	**Time**	**Cost**	**Time**
INRC2-4-030-1-6291	**1695**	-	1980	5.6	1835	26.5
INRC2-4-030-1-6753	**1890**	-	2180	12.0	2065	29.9
INRC2-4-035-0-1718	**1425**	-	1815	17.2	1710	54.1
INRC2-4-035-2-8875	**1155**	-	1565	19.1	1440	36.4
INRC2-4-040-0-2061	**1685**	-	2095	13.3	1985	48.5
INRC2-4-040-2-6106	**1890**	-	2300	17.2	2110	40.1
INRC2-4-050-0-0487	**1505**	-	1925	28.7	1770	71.9
INRC2-4-050-0-7272	**1500**	-	1905	30.2	1825	61.8
INRC2-4-060-1-6115	**2505**	-	2970	53.7	2895	153.2
INRC2-4-060-1-9638	**2750**	-	3395	75.8	3150	149.1
INRC2-4-070-0-3651	**2435**	-	3070	59.6	2850	165.0
INRC2-4-070-0-4967	**2175**	-	2915	75.2	2695	173.3
INRC2-4-080-2-4333	**3340**	-	3900	90.3	3835	189.9
INRC2-4-080-2-6048	**3260**	-	3955	97.9	3920	189.0
INRC2-4-100-0-1108	**1245**	-	1670	103.7	1670	212.1
INRC2-4-100-2-0646	**1950**	-	2560	110.8	2300	264.8
INRC2-4-110-0-1428	**2440**	-	2995	111.0	2815	250.6
INRC2-4-110-0-1935	**2560**	-	3090	140.3	2950	261.7
INRC2-4-120-1-4626	**2170**	-	2890	113.2	2740	291.2
INRC2-4-120-1-5698	**2220**	-	2840	165.0	2800	325.4
Average	**2090**		2601	67.0	2468	149.7

Tables 2 and 3 show that KHE20x8's results are competitive in cost and faster in running time. The tables do not give running times 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, 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 author has not tried to obtain these solutions. Their reported running times are better than the GOAL ones, never exceeding about 10 minutes.

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 solve a set of conventional 4-week and 8-week instances from the competition that have been tackled by several authors [26,37]. Their converted versions appear in files `INRC2-4.xml` and `INRC2-8.xml` at [21], along with some solutions produced by the authors of [26].

The results appear in Tables 4 and 5. There is no running time information in the solution files obtained from other authors. Reference [26] gives a formula

Table 5 Results for the Second International Timetabling Competition 8-week instances. This table is derived from XESTT archive file `KHE20-2020-01-20-INRC2-8.xml`, available at [21]. Other details as for Table 4.

Instances (20)	**LOR17**		**KHE20**		**KHE20x8**	
	Cost	**Time**	**Cost**	**Time**	**Cost**	**Time**
INRC2-8-030-1-27093606	**2125**	-	2895	66.1	2695	140.4
INRC2-8-030-1-67535629	**1735**	-	2365	46.9	2255	121.1
INRC2-8-035-0-62987798	**2570**	-	3755	71.6	3745	178.3
INRC2-8-035-1-08161720	**2330**	-	3595	91.3	3505	180.7
INRC2-8-040-0-06892664	**2635**	-	4205	109.1	3700	197.6
INRC2-8-040-2-50487172	**2495**	-	3610	82.4	3605	197.1
INRC2-8-050-1-17857418	**4990**	-	6425	117.4	6110	338.6
INRC2-8-050-1-97538831	**5000**	-	6215	142.8	5865	307.3
INRC2-8-060-0-62990813	**2425**	-	3815	138.6	3670	328.6
INRC2-8-060-2-10340391	**2590**	-	4285	153.3	3920	364.3
INRC2-8-070-0-33923752	**4660**	-	6010	195.4	6010	394.2
INRC2-8-070-0-93072110	**4770**	-	6175	187.2	6175	393.7
INRC2-8-080-1-44993605	**4225**	-	6355	276.7	6020	547.4
INRC2-8-080-2-04091962	**4495**	-	6400	277.0	6400	546.5
INRC2-8-100-0-01789154	**2145**	-	4045	246.9	3900	490.3
INRC2-8-100-1-24793928	**2250**	-	4320	257.8	4240	517.8
INRC2-8-110-0-21172647	**4010**	-	5600	238.0	5290	495.7
INRC2-8-110-0-32494137	**3575**	-	5025	226.8	4805	470.0
INRC2-8-120-0-09945103	**2670**	-	4670	236.5	4455	476.3
INRC2-8-120-1-72645202	**3125**	-	5110	223.9	4890	515.1
Average	**3241**		4744	169.3	4563	360.0

that was used to determine the running time limit; it is about 20 minutes for the largest 4-week instances, and 40 minutes for the largest 8-week instances. Reference [37] used a running time limit of 2750 seconds (about 45 minutes).

KHE20x8 is arguably competitive for the 4-week instances (18% worse in cost on average, but much faster), but not the 8-week ones (41% worse). The algorithm of [26] makes effective use of long running times, to optimally reassign large parts of the solution. Long running times and large neighbourhoods also feature in [37]. Hand analysis suggests that repairing the remaining defects in KHE20's solutions might require larger neighbourhoods than it explores.

To further investigate this issue, another test was run in which the running time limits were doubled. This produced solutions that were 37% worse in cost on average. The detailed results of this test may be found online [25].

4.4 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.

These four sets of solutions were made by the solvers reported on in Table 2 of [7]: ejection chains (10 minutes), ejection chains (60 minutes), branch and price, and Gurobi. But they differ from the solutions presented in Table 2 of

Table 6 Solution costs for the instances published in 2014 by Curtois and Qu [7,8]. The CQ-EJ10, CQ-EJ60, CQ-BP, and CQ-GR columns show the four sets of solutions from XESTT archive `CQ14.xml`, corresponding to the ejection chain (10 mins), ejection chain (60 mins), Branch and Price, and Gurobi 5.6.3 columns of Table 2 of [7]. The last two columns show the results from the two versions of the author's solver. Owing to the fragmentary results for the last 5 instances, averages are taken over the first 19 instances only. This table is derived from XESTT archive file `KHE20-2020-01-20-CQ14.xml`, available at [21].

Instances (24)	**CQ-EJ10**	**CQ-EJ60**	**CQ-BP**	**CQ-GR**	**KHE20**	**KHE20x8**
CQ14-01	1114		**607**	**607**	**607**	**607**
CQ14-02	1019	837		**828**	831	**828**
CQ14-03	**1001**	1003		**1001**	**1001**	**1001**
CQ14-04	**1716**	1718	**1716**	**1716**	1723	1720
CQ14-05	1144	1358	1160	**1143**	1342	1239
CQ14-06	1952	2258	1952	**1950**	2153	2066
CQ14-07	**1056**	1269	1058	**1056**	1163	1078
CQ14-08	2133	1314	**1308**	1323	1429	1429
CQ14-09	1449	**439**		**439**	555	453
CQ14-10	4870	**4631**		**4631**	4750	4665
CQ14-11	**3443**	3661		**3443**	3463	3457
CQ14-12	5476	**4040**	4046	**4040**	4205	4081
CQ14-13	11432	1486		**1388**	2314	2205
CQ14-14	2286	1300		**1280**	1726	1414
CQ14-15	12050	4378		**4039**	4941	4649
CQ14-16	3926	**3225**	3323	3233	inf.	3766
CQ14-17	7801	5872		**5851**	6611	6410
CQ14-18	10002	4969		**4760**	inf.	5503
CQ14-19	14788	3715		**3218**	3461	3461
Average	4666			**2418**		2633
CQ14-20					inf.	**inf.**
CQ14-21					inf.	**inf.**
CQ14-22					inf.	**inf.**
CQ14-23		44819		**17428**	inf.	inf.
CQ14-24				**48777**	inf.	inf.

[7], for reasons explained in [23]: the solutions received were not exactly those reported in the table, and some were omitted owing to extreme overstaffing of shifts, which this author chose to disallow in the XESTT versions.

The results appear in Tables 6 and 7. KHE20x8 is competitive in cost with CQ-EJ60, CQ-BP, and CQ-GR, except on one instance (CQ14-13), and its running time is very much less than theirs. KHE20x8 is clearly superior to CQ-EJ10 on cost, and its running time is less on average.

Again, longer run times can help. In instance CQ14-13, unassigned shifts cost 100 each, and they are the cause of most of the extra cost in KHE20x8's solution. Allowing KHE20x8 to run to completion produced a solution of cost 1656 in 29.6 minutes.

There are more recent results for these instances [8,10,11]. Reference [8] has slightly better results than any shown here, 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.

Table 7 Running times in seconds for the instances published in 2014 by Curtois and Qu. Details as for Table 6, only showing running times instead of solution costs.

Instances (24)	**CQ-EJ10**	**CQ-EJ60**	**CQ-BP**	**CQ-GR**	**KHE20**	**KHE20x8**
CQ14-01	0.0		0.4	-	0.1	0.3
CQ14-02	30.0	3600.0		-	0.7	2.5
CQ14-03	2.6	3600.0		-	1.9	2.8
CQ14-04	8.2	3600.0	1.5	-	1.3	3.2
CQ14-05	76.8	3600.0	25.6	-	3.6	10.1
CQ14-06	88.8	3600.0	10.5	-	11.6	13.9
CQ14-07	262.1	3600.0	93.7	-	9.8	24.0
CQ14-08	600.0	3600.0	11831.1	-	31.2	62.1
CQ14-09	0.6	635.2		-	51.6	91.1
CQ14-10	600.0	863.2		-	39.4	103.2
CQ14-11	171.4	3600.0		-	48.7	130.7
CQ14-12	600.0	1526.1	1336.4	-	67.3	147.1
CQ14-13	30.1	3600.1		-	194.5	385.4
CQ14-14	600.0	3600.3		-	128.5	294.8
CQ14-15	30.1	3600.0		-	180.5	360.9
CQ14-16	30.0	2965.7	265.0	-	166.6	316.3
CQ14-17	600.0	3600.1		-	180.4	360.9
CQ14-18	30.0	3600.0		-	180.3	360.7
CQ14-19	30.0	3600.0		-	180.8	361.8
Average	199.5				77.8	159.6
CQ14-20					183.1	369.9
CQ14-21					192.9	386.7
CQ14-22					187.1	378.2
CQ14-23		3601.0		-	207.8	420.5
CQ14-24				-	294.4	578.5

5 Aspects of KHE20

This section investigates eleven aspects of KHE20, to see whether they make useful contributions.[10] For each aspect, the results of running KHE20x8 with and without the aspect are compared. Only the Curtois original instances are tested, but they are very varied instances, so that if some aspect benefits the algorithm significantly, the results on them should make that clear.

There is a supplement to this paper online, containing the detailed results of these tests [25]. Only summaries of those results appear here. In what follows, 'KHE20' refers to the algorithm without omissions.

The first test investigates omitting task grouping (aspect GR from Section 3.4), and of omitting suppression of spurious costs during time sweep (CS from Section 3.2). The results are similar to KHE20's. Omitting GR never produces a lower cost solution. Omitting CS produces a lower cost solution in six instances, but overall, KHE20 produces more best solutions. The effect of GR and CS was larger earlier in this project; as the ejection chain algorithm has improved, the need for them has diminished.

[10] *Note to referees.* This section is new.

The second test investigates omitting three repair steps during construction: ejection chain repair of limit resources defects after assigning each day (CE from Section 3.1), resource rematching after each day (CR from Section 3.2), and resource rematching each day individually at the end (CI from Section 3.2). The results are similar to KHE20's.

The third test investigates omitting the three edge adjustments in resource matching (Section 3.1): favouring nurses with more available workload (CW), assignments that bring fewer constraints to their maximum limits (CL), and shorter runs of consecutive busy days (CB). Again the results are similar to KHE20's, although KHE20 does best.

The fourth test investigates omitting aspects of the repair phases: the whole second repair phase (R2 from the start of Section 3), resource rematching from both phases (RM from Section 3.3), and ejection chain repair from both phases (EE from Section 3.3). This shows that ejection chains have a major impact, and that the second repair phase is also important on some instances. There is some evidence that, for some instances, time spent on resource rematching might be better spent on ejection chains.

The fifth test investigates omitting widening (EW), reversing (ER), and balancing (EB) from ejection chain repair (Section 3.3). KHE20 is clearly best here, producing the lowest average cost and the largest number of best solutions. Omitting balancing speeds up the algorithm significantly and is beneficial in a few instances, but it is clearly inferior overall.

The sixth test tries different amounts of widening (where KHE20 uses 4). It shows that KHE20 is doing about the right amount of widening.

The seventh test tries different amounts of balancing (where KHE20 uses 12). There is some evidence that KHE20 does too much balancing, although this area needs more study. Balancing is likely to be of more benefit to some instances—those whose workload limits are tight—than others. For solves that reach the time limit, balancing repairs lack variety compared with other repairs that could be tried in the limited time.

To summarize the results. Several of the aspects have not demonstrated any value in these tests. Testing them on all archives might prove definitively that they could be removed. On the other hand, all aspects seem to be doing no harm, and, given the wide variation among instances, most of them probably do good sometimes. 'Biodiversity' produces robustness in algorithms as it does in the natural world, so the author is in no hurry to remove any aspects.

The supplementary paper also contains some tests which run KHE20 with its running time limits doubled. These are the source of some of the remarks in Section 4 about the effect of longer running times. Doubling the running time produces significantly better solutions for some large instances. More often, however, the improvement is only modest.

In any case, the author does not advocate longer running times, because KHE20 is intended to run quickly. The real value of these experiments is in showing what the current ejection chain repair operations are capable of, when more time is available.

6 Conclusion

This paper has presented KHE20, 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 for repair. There are no parameters to tune. The algorithm has many detailed aspects which, taken together, improve it significantly.

Tests on four well-known data sets have shown that KHE20, or rather KHE20x8, is always competitive in running time, and usually in cost. There are some weak results, notably on the INRC2 8-week instances. But it must be remembered that none of the algorithms against which KHE20 is compared here have been tested as widely as KHE20 has. It is not known how any of them would perform on all four data sets.

The author has not yet investigated in detail the poor performance of KHE20 on the INRC2 8-week instances. This is the obvious next step. Other investigations of this kind, carried out over the last two years, have driven the improvements which have brought the algorithm to its current level.

Very often, one can observe the solver working hard on repairs that an intelligent observer can dismiss as hopeless. Future work could usefully focus on targeting repairs better. That would improve cost as well as running time, especially for solves that reach the time limit.

One can see from the leading recent papers [4,35] 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, in finding good solutions for large instances with reliably fast running times.

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.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 Stlevik, and Andrea Schaerf, First international nurse rostering competition website, URL: http://www.kuleuven-kortrijk.be/nrpcompetition (2010)
14. 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)
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, Additional experiments with the KHE20 nurse rostering solver, unpublished technical report, `http://jeffreykingston.id.au/tt_papers/` (2020)
26. 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`.
27. 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.
28. 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

29. Gerhard Post, XHSTT web site, http://www.utwente.nl/ctit/hstt/ (2011)
30. 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)
31. 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)
32. 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)
33. 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
34. 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.
35. 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)
36. Haroldo G. Santos, Túlio A. M. Toffolo, Rafael A. M. Gomes, and Sabir Ribas, http://www.goal.ufop.br/nrp/
37. 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
38. 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
39. 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
40. 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).