Bridging the gap between self schedules and feasible schedules in staff scheduling

Eyjólfur Ingi Ásgeirsson

Abstract Every company that has employees working on irregular schedules must deal with the difficult and time consuming problem of creating feasible schedules for the employees. We introduce an algorithm that takes a partial schedule created by requests from employees and creates feasible schedule where most of the employee's requests are unchanged, while still making sure that rules and regulations are not violated. The algorithm is based on independent modules, which can be executed in any order, and each module tries to emulate some action taken by a staff manager.

Our goal is to create a transparent and fair system that creates feasible schedules of high quality, but also a system where the employees can get an explanation and justification for every change that the algorithm makes to the employee requests. By emulating the actions of staff managers, the algorithm is easily understood by staff managers and, using detailed logs of any action, make any decision easy to explain to the employees.

We will present the algorithm and show results from four real world companies and institutions. The results show that a simple module based heuristic can get good results and create fair and feasible schedules that encourage employees to participate in the self-scheduling process.

Keywords Staff scheduling · Rostering · Heuristics · Local search

1 Introduction

Staff scheduling is a difficult and time consuming problem that every company or institution that has employees working on shifts or on irregular workdays must solve. A variant of the general staff scheduling problem focuses on scheduling the working hours for nurses in the health industry, so called nurse rostering or nurse scheduling

Work done in collaboration with Vaktabestun ehf.

E. I. Ásgeirsson
Reykjavik University
School of Science and Engineering.
Menntavegur 1, 101 Reykjavik, Iceland.
E-mail: eyjo@ru.is

[6]. The nurse rostering problem is well known and has been studied for over 45 years. The nurse rostering problem can include many types of constraints and covers a large set of staff scheduling problems, so even though our work is not specific for hospitals or nurses, most of the previous work focuses on nurse rostering.

There are three major approaches used in nurse rostering: cyclical scheduling [15], self scheduling and preference scheduling [3]. In cyclical scheduling, several sets of schedules are generated and then the nurses are assigned to a schedule that best fits their preferences so that collectively they satisfy the manpower requirements. The cyclical scheduling approach is rather rigid and therefore difficult to use in a flexible and changing environment.

In preference scheduling, each employee gives a list of preferences to the personnel manager who then creates a schedule that satisfies the demand for personnel and work restrictions while trying to fulfill as many preferences as possible. Some form of preference scheduling is widely used in real world environments and it has many benefits, such as flexibility, individual tailoring of the schedule and so forth. The major drawback to preference scheduling is the time required to create a high quality schedule that fulfills as many preferences as possible.

The self scheduling approach moves the responsibility of creating a schedule to the employees. The employees are given the required minimum and maximum number of employees that should be on duty at each time. The employees are then asked to sign up for the shifts that they want to work on, with the requirement that the resulting schedule must be a feasible schedule. Pure self-scheduling is difficult to implement fairly, in many instances it becomes easy for someone to manipulate the system, the sign-up order is important, new employees are likely to be at a disadvantage due to unfamiliarity with the system and, due to any number of reasons, some employees might not sign up for any shifts. There are however potentially many motivational benefits of self scheduling, such as improved co-operation, greater staff satisfaction and commitment, and reduced staff turnaround [16]. In real-life, self-scheduling can be implemented as a mixture of pure self-scheduling and preference scheduling. In this approach the employees sign up for shifts, creating a preliminary schedule and the staff manager then turns the preliminary schedule into an feasible staffing schedule, making sure that no rules have been violated and the staffing load fits the manpower need at each time. The final responsibility of creating a good schedule lies with the staff manager, but the employees see the manpower need and the current staff levels when signing up for shifts, so they share the responsibility of creating the schedule. This approach can include an incentive scheme, such as priority on popular shifts, to encourage the employees to create high quality preliminary schedules.

Mathematical programming techniques are in many cases too rigid to deal with the multiple and often changing, objectives and goals of staff scheduling. The research on staff scheduling is now mainly focused on more flexible metaheuristic approaches such as genetic algorithms [1, 2, 13, 18] and variable neighborhood search [7], with Tabu-Search [5, 12] and Simulated Annealing [4] being particularly successful [14]. Burke et. al. [8] introduced a multi-criteria metaheuristic approach based on Tabu-search, which is used in Belgian hospitals. Their system allows for user defined parameters, giving the users the opportunity to adjust the algorithm to their need and to the specifics of their problem.

There are two major problems with using black-box methods such as meta-heuristics and mathematical programming methods with the self-scheduling scheme introduced above. The first problem is that it is difficult to incorporate the experience and exper-

tise of the staff managers into such techniques [17]. Staff managers often have highly valuable knowledge, experience, and detailed understanding of their specific staffing problem, which will vary from company to company. The variation in the problem specifics between the different companies makes it also difficult to incorporate a single solution that fits all.

The second major problem is that the employees have spent time and effort into creating the preliminary schedule, so they have an emotional attachment to their selection. Since the staff manager has the final responsibility of the schedule, the staff manager is also responsible to the employees for any changes that are made to the preliminary schedule. The employees whose schedules are changed and whose wishes are ignored will demand to know why. Having a black-box method makes it impossible to see exactly why each choice is made, which makes it impossible to justify or explain individual decisions made by the algorithm.

2 Staff Scheduling

The staff scheduling problem we look at is a mixture of self scheduling and preference scheduling. The employees sign up for shifts and indicate periods where they cannot work. During the selection process, every employee has full information of how many employees are signed up for each shift and the required minimum and maximum staffing levels. The resulting plan, or preliminary schedule, is then used by the management as a foundation to create a feasible schedule. The quality of the preliminary schedule determines how close we are to either pure self scheduling or preference scheduling. If the preliminary schedule is close to feasible, then we have a self scheduling system, but on the other hand, if the preliminary schedule is far from being feasible, then the system is closer to preference scheduling, which entails the time consuming process of creating a feasible schedule from the preliminary schedule.

Our goal is to automate the time consuming process of creating a feasible schedule which is the drawback of preference scheduling systems. We will use a combination of local search methods and heuristics to emulate the manual process of creating a high quality schedule based on the preliminary schedule. The preliminary schedule is unlikely to satisfy the minimum and maximum required number of employees on duty, there might be some employees that have not signed up for any shifts or too few shifts and there might even be employees with too many scheduled hours.

In the real world instances that we've analyzed, the schedules that are created manually from the preliminary schedule are usually accepted as fair by the employees, the problem is how time consuming the process is. Our system is designed to preserve the perceived fairness of the current manual system, which we do by ensuring that the decisions of the algorithm are transparent and easily justifiable using the available data. There are no constrictions on the shifts that we use, each company or institution will have a set of allowed shifts and the shifts can have different length, they can overlap and different days of the week can have different set of allowed shifts.

We will present the algorithm and give results using real data to show how effectively the local search methods and heuristics can create feasible schedules with high quality, while still satisfying personnel preferences and preserving the trust that the employees have in the current system.

3 Constraints

Staff scheduling problems have a large number of constraints, such as minimum or maximum number of employee on duty at each time, the requests of employees, union rules and other regulations that must be satisfied. We partition the constraints into hard and soft constraints, where the hard constraints must always be satisfied while we allow the soft constraints to be violated if necessary.

3.1 Hard Constraints

The hard constraints, i.e. the constraints that must be satisfied at all times, are mostly based on union contracts and contracts with the employees. The system allows for having different constraints for different employees. In our examples the main constraints are usually the same for all the employees, with the exception of work limits. The hard constraints that we use are the following:

- **Restrictions on working hours and rest periods from employee contracts and union regulations.** Each employee can have restrictions on when they can work, such as employees that will never work nights or weekends. Additionally, there will be union regulations on rest periods, maximum lengths of continuous work, minimum length of continuous rest between shifts and other limits.
- **Vacation requests.** We treat requests for vacations as hard constraints, so the algorithm will never assign work duty to people on vacation.
- **Working weekends.** There can be limits on how many weekends particular employees are working. Common limits include working at most 2 or 3 out of every 4 consecutive weekends.
- **Requests for time off.** Each employee can have a some hours where they wish to be off-duty. We treat such wishes as hard constraints, so the algorithm will never violate such wishes. The number of such hours depends on the company and on the employee contract.
- **Special shifts, training sessions or meetings.** In many instances, employees have work related duties that are not flexible and are not necessarily included in the number of people on duty. Such instances include training sessions, meetings or other special functions. Since training sessions and meetings are often not flexible, we make sure that the algorithm will not make any changes to such functions.
- **Other limits on shifts or working hours, such as double shifts**. Double shifts are defined as two separate shifts in the same 24 hour period, where the interval between the shifts is less than the minimum resting period between shifts.

The set of constraints that are used is flexible and different from one company to the next. In one of our examples, the use of double shifts is not only allowed but actually encouraged while other companies might consider schedules with double shifts as infeasible.

To evaluate constraints such as working weekends or minimum rest, the actual schedule from the previous period must be included in the input.

One of the problems with the hard constraints is that the employees are allowed to be more flexible when they are creating their own schedule than staff managers or improvement algorithms. For example, an employee might sign up for a 10 hour shift, while a staff manager or an algorithm can only assign shifts of 8 hours or less to the

same employee. Since the employees have more flexibility, the preliminary schedules often contain individual employee schedules that would be considered infeasible. To make any improvements to such a schedule, the algorithm must accept the infeasibility of the current schedule, but make sure that the proposed changes to the schedule do not violate any hard constraints. We use a penalty score system to handle infeasible schedules and to check if proposed changes are feasible. Each time an individual employee schedule violates a constraint, the schedule receives a penalty. If the cumulative penalty is above a certain threshold then the schedule is considered infeasible. To handle the fact that the schedules can be infeasible to start with, the penalty for any schedule is calculated before and after a proposed change. If the penalty increase is above the threshold, then the change is not allowed. Using a penalty for each constraint violation and a threshold not only allows us to use requests that would be considered infeasible, but it is also flexible since it allows us to specify exactly the number of times a particular rule must be violated before the proposed schedule is considered infeasible.

3.2 Soft Constraints

The algorithm is designed so that the hard constraints are always satisfied. The soft constraints can be broken at any time, and represent the goals of the scheduling process. It is often impossible to satisfy all soft and hard constraints, so we must sometimes settle for satisfying as many soft constraints as possible. The soft constraints we use are the following:

- **Minimum and maximum staff levels.** An estimate for the demand for employees on duty at each time during the scheduling period is one of the prerequisites of the scheduling process. Some companies use minimum and maximum number of on-duty employees for each time slot in the scheduling period, while other companies simply state exactly how many employees should be on duty at each time. One of the goals is to have the number of on-duty employees within the minimum and maximum at all times, or as close to the exact number of employees that should be on duty. Companies and institutions often use some type of forecasting to estimate the required number of staff on duty, but the sophistication and the accuracy of the demand predictions can vary greatly from one company to the next.
- **Minimum and maximum number of on-duty hours for each employee.** Each employee is hired to work a specific number of hours per week, typically 40 hours per week for a full time employment. For each planning period, the number of hours that the employee should sign up for is calculated, based on the number of hours per week and the number of actual working hours in previous periods. Since the employees are often working irregular hours, there must be some flexibility in the system. For the scheduling period, each employee is assigned minimum and maximum duty hours, and one of the goals of the rostering process is to make sure that all employees are within the minimum and maximum duty hours.
- **Employee requests for shifts.** Before each scheduling period, the employees sign up for shifts. One of the main goals of this project is to encourage employees to create their own work schedules, so it is important to keep as much of the requests as possible.
- **Employees assigned to shifts on weekends adjacent to their vacations.** If an employee is starting his or hers vacation on a Monday, it is likely that the

employee wants the weekend free. We try to make sure that if an employee is on a vacation on a Monday or Friday, the adjacent weekend will be free, unless the employee has requested to work on that particular weekend.

Each company can have different priority rules for the soft constraints, which are reflected in the order in which various modules and functions of the algorithm are executed. A typical priority rule is that having all employees within minimum and maximum duty hours takes highest priority, then the staff levels and finally the requests of the employees.

4 The staff scheduling algorithm

The algorithm that we use is a collection of independent modules or functions, where each module takes the current schedule and tries to improve it. We can call the modules in any order, here we present the order that we usually use, i.e. we first execute Algorithm 1, then Algorithm 2 and so on. Some modules are executed more than once with different input parameters, which we will elaborate on when we give detailed description of each module. Most of these algorithms are simple and some of them have been introduced before, such as [10,11], but for completeness we will introduce and explain all the algorithms.

Algorithm 1 RepairShifts

```
Input: set of employees E
Input: set of allowed shifts A
for all e ∈ E do
  for all s ∈ e.shifts do
    s ← find closest shift to s from the shifts in A
  end for
end for
```

Algorithm 1 is used to make sure that all employees are only working on shifts that are allowed. The systems that the employees use to request shifts sometimes allow the employees to sign up for any hours. However, the collaborating companies and institutions restrict the employees to only work on certain shifts. The first step of the algorithm is to take the preliminary input and make sure that all employees are only signed up on shifts that are allowed. The variable $e.shifts$ denotes the set of shifts that employee e is signed up for. The distance between any two shifts s and t is $|s.start - t.start| + |s.end - t.end| + |s.length - t.length|$, where $s.start$ is the start time of shift s, $s.end$ is the end time of shift s and $s.length$ is the length of shift s, and similarly for shift t. The start and end times of the shifts are measured as the number of hours from the start of the scheduling period while the length of a shift is measured in hours. The length of a shift is included in the measure since we want to find an allowed shift that is both similar to the chosen shift in time and length.

The module shown in Algorithm 2 uses a priority rule to determine which employee is selected when the algorithm needs to remove an employee from an overstaffed shift. The priority rule can be different from one company to the next, in the preliminary version of the algorithm we use the number of working hours to determine which employee should be removed from an overstaffed shift.

Algorithm 2 Overstaffing: Remove shifts from employees

```
Input: set of overstaffed shifts S
while S ≠ ∅ do
    s ← select the maximum overstaffed shift from S
    E(s) ← {e ∈ E : s ∈ e.shifts}
    e ← select lowest priority employee from E(s)
    e.shifts ← e.shifts \ {s}
    Update s
    Update S
end while
```

Algorithm 3 Understaffing: Add shifts to employees

```
Input: Set of employees E
Input: Set of understaffed shifts S
while S ≠ ∅ do
    s ← select the shift with the largest total understaffing from S
    P(s) ← select all employees that can work on shift s.
    if P(s) ≠ ∅ then
        e ← select the employee from P(s) with fewest scheduled working hours.
        e.shifts ← e.shifts ∩ {s}
        Update S and E
    else
        S ← S \ {s}
    end if
end while
```

The function described in Algorithm 3 tries to decrease understaffing by locating understaffed shifts and then find employees that are available and can work on the shift. We order the employees in ascending order of scheduled working hours to improve the schedule of employees with too few scheduled hours.

Algorithm 4 Understaffing: Swap overlapping shifts

```
Input: Set of employees E
Input: Set of understaffed shifts S
for all e ∈ E do
    for all s ∈ e.shifts do
        O(s) ← {s' ∈ S : s' ∩ s ≠ ∅ ∧ e can work on shift s'}
        if O(s) ≠ ∅ then
            s* ← select the shift with the highest understaffing from O(s)
            e.shifts ← e.shifts \ s
            e.shifts ← e.shifts ∪ {s*}
        end if
    end for
end for
```

Algorithm 4 is used to decrease understaffing while making only small changes to the schedule. If we find an understaffed shift, we try to locate employees that are working on shifts that overlap with the understaffed shift. If the understaffed hours are not in the intersection of the two shifts, then moving the employee to the understaffed shift can decrease understaffing.

After we run Algorithms 3 and 4 to improve the understaffing, we use Algorithms 5 and 6 to improve the individual schedules of employees that are below the minimum

Algorithm 5 Staff below working hours: Add shifts

```
Input: Set of employees with scheduled working hours below minimum duty hours, E
Input: Set of shifts with staffing levels below maximum staffing, S
for all e ∈ E do
  for all s ∈ S do
    if e can work on shift s then
      e.shifts ← e.shifts ∪ {s}
      if Staffing level of s is at maximum staffing then
        S ← S \ {s}
      end if
    end if
  end for
end for
```

duty hours. Algorithm 5 selects an employee below duty hours and then tries to find a feasible shift with enough space for the employee.

Algorithm 6 Move shifts from employees above duty hours to employees below duty hours

```
Input: Set of employees with scheduled working hours above duty hours, E_above
Input: Set of employees with scheduled working hours below duty hours, E_below
for all Pairs (e_a, e_b) : e_a ∈ E_above, e_b ∈ E_below do
  for all s ∈ e_a.shifts do
    if e_b can work on shift s and e_a can be removed from shift s then
      e_a.shifts ← e_a.shifts \ {s}
      e_b.shifts ← e_b.shifts ∪ {s}
      if e_a is at or below duty hours then
        E_above ← E_above \ {e_a}
      end if
      if e_b is at or above duty hours then
        E_below ← E_below \ {e_b}
      end if
    end if
  end for
end for
```

Algorithm 6 is used to improve the balance of the employees duty hours. We create a set of employees above the duty hours and another set of employees that are below their duty hours. Then we look at all pairs of employees and try to find a shift that we can move from the above duty hours employee to the below duty hour employee. We can run Algorithm 6 either by allowing the algorithm to modify the requested shifts, or only allow modifications to shifts that were added by other modules of the algorithm.

If an employee is below duty hours, we can try to increase the number of working hours by swapping shifts. Algorithm 7 looks at employees below the minimum duty hours. For each employee, the module tries to swap an existing shift with a longer overlapping shift. Since the employee is already working on this particular day, such swaps are often feasible, assuming that the new shift has room for additional staff. Algorithm 7 can be allowed to change requested shifts, or we can focus only on shifts that have been added by other modules.

The experimental results in the next section are created by running Algorithms 1 to 7 in that order. Algorithms 6 and 7 were executed twice, first focusing only on shifts

Algorithm 7 Staff below working hours: Swap shifts to add working hours

```
Input: Set of employees with scheduled working hours below minimum duty hours, E
Input: Set of shifts with staffing levels below maximum staffing, S
for all e ∈ E do
  for all s ∈ e.shifts do
    O(s) ← {s' ∈ S : s' ∩ s ≠ ∅ ∧ e can work on shift s'}
    if O(s) ≠ ∅ then
      s* ← select the largest shift from O(s)
      if |s*| > |s| then
        e.shifts ← e.shifts \ {s}
        e.shifts ← e.shifts ∪ {s*}
        Update E
        Update S
      end if
    end if
  end for
end for
```

that had been added in previous steps, and then by allowing the modules to change requested shifts.

5 Experimental Results

To evaluate the performance of our algorithm, we use actual data from four companies and institutions. These companies and institutions include a nursing home, call centers and airport services. We will present the details of each problem instance and show examples of the preliminary schedule and the improved schedule. The scheduling period is usually 6 weeks, but we will plot the preliminary schedule and the improved schedule for only a single week for each problem instance. We tried to select a typical week for each instance. For each employee we measure the percentage of requested hours that are still in the improved schedule using the formula

$$\text{Requested hours granted} = \frac{\sum_{t=1}^{t=N} I_{prel}(t) \times I_{improved}(t)}{\sum_{t=1}^{t=N} I_{prel}(t)}$$

where N is the number of time slots in the scheduling period, $I_{prel}(t) = 1$ if the employee has requested to be working in time slot t, and 0 otherwise. Similarly the indicator function $I_{improved}(t)$ is equal to 1 if the employee is working in time slot t in the improved schedule and 0 otherwise. The percentage of requested hours is not defined for the employees that do not make any requests, so those employees are not included when we calculate the average requested hours granted.

We decided to use the number of hours instead of focusing on the shifts when measuring how close the final schedule is to the requested schedule from the employees. The reason is that we felt that if an employee has signed up for a $8 - 16$ shift, and we change it to $9 - 17$, then the employee is getting a shift that's very close to what was requested. Therefore, measuring hours is in our mind often more fair than focusing on shifts. However, this is not necessarily always the case, in some instances the measure should focus on the shifts instead of the hours. For the collaborating companies and institutions, the requested hours granted measure was considered both fair and appropriate.

	Preliminary schedule	Improved schedule
Scheduled hours	4669	5198
Man-hours overstaffed	478	220
Man-hours understaffed	777	21
Employees below minimum duty hours	3	0
Total unscheduled duty hours	905	545

Table 1 Results for the nursing home instance.

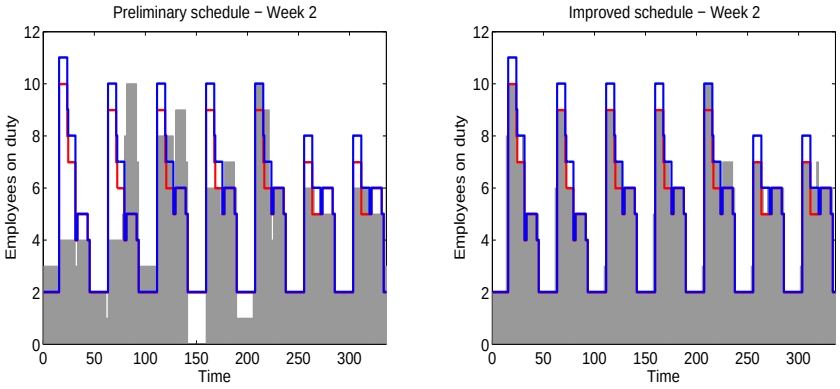

Fig. 1 Staffing levels for the nursing home problem instance. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

Using the notation introduced by De Causmaecker [9], we can describe the following problem instances as (AS|TVNO|PLGO).

5.1 Problem instance: Nursing home

The first problem instance comes from a nursing home with 55 employees. The scheduling period is 6 weeks with 30 minute intervals. There are 1428 different shifts that are allowed in the scheduling period. The length of each shift ranges from 4 hours up to 12 hours. If we sum up the total maximum working hours over the scheduling period, we get that the maximum number of hours that we can assign, without any overstaffing, is 5217 hours. However, the total duty hours of the employees is 5333 hours, so unless we violate the overstaffing constraint, we can never satisfy all duty hour requirements for the employees. The improved schedule has 5198 man-hours scheduled, not counting vacations and shifts that do not contribute to the staffing requirements. The results for the nursing home problem instance are shown in Table 1 while Figure 1 shows the preliminary schedule and the improved schedule for a typical week in the scheduling period.

The nursing home has the following hard constraints. An employee cannot work on more than 6 consecutive days, the maximum length of a shift is 9 hours while the minimum length of a shift is 4 hours. In any 24 hour period, each employee must get at least 8 consecutive hours of rest, while the maximum number of working hours in any 24 hour period is 9 hours.

	Preliminary schedule	Improved schedule
Scheduled hours	9424	11920
Man-hours overstaffed	390	791
Man-hours understaffed	1560	14
Employees below minimum duty hours	19	5
Total unscheduled duty hours	2108	234

Table 2 Results for call center A.

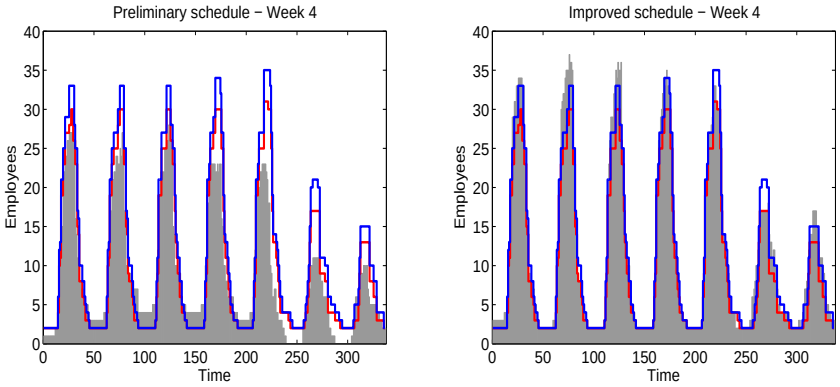

Fig. 2 Staffing levels for call center A. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

Figure 1 shows that the requirements for the minimum and maximum number of employees on duty have a very regular pattern. The nightshifts require 2 persons, while the mornings require between 8-10 people on duty. The number of on-duty employees decreases over the weekend, while Mondays require the highest number of on-duty employees. The preliminary schedule has both understaffing and overstaffing. However, in the improved schedule, the overstaffing has been reduced from a total of 478 hours to 220 hours, and the understaffing has gone from a total of 777 hours down to 21 hours. The average percentage of requested hours still in the improved schedule was 97.2%.

5.2 Problem instance: Call center A

The second problem instance is a call center. We have two call centers in our set of real world data so we will refer to them as call center A and call center B. Call center A has 92 employees and the scheduling period is 6 weeks in 30 minute intervals. The number of possible shifts in the scheduling period is 8863 and their lengths are from 4 hours up to 11 hours. The total maximum required on-duty employees is 11582, while the total duty hours for all employees is 12054, so it will be impossible to satisfy both the duty hour constraints and the overstaffing constraints. In the end, we actually schedule a total of 11920 hours, not including vacations and shifts that do not contribute to the staffing, so this instance has some overstaffing. Table 2 shows the data from the preliminary schedule and the results of the improved schedule. There are 5 employees

	Preliminary schedule	Improved schedule
Scheduled hours	6623	7554
Man-hours overstaffed	795	609
Man-hours understaffed	1306	189
Employees below minimum duty hours	15	7
Total unscheduled duty hours	1551	529

Table 3 Results for call center B.

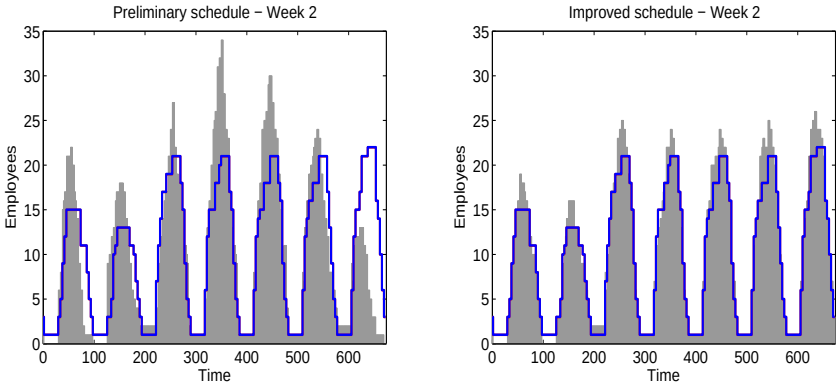

Fig. 3 Staffing levels for call center B. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

below the minimum duty hours in the improved schedule, but 3 of these employees are less than two hours below the minimum, while the remaining 2 employees are 6 hours and 22 hours below the minimum duty hours.

For call center A, the maximum number of consecutive working days is 6, the maximum number of working hours in each 24 hour period is 9 hours and the maximum length of a single shift is also 9 hours. In any 24 hour period, each employee must get at least 11 consecutive hours of rest.

Figure 2 shows a typical week in the scheduling period for call center A. The required number of on-duty employees peaks at around $30 - 35$ during the afternoon while the night shifts require only around $2 - 3$ on-duty employees. The preliminary schedule has both overstaffing and understaffing, while the improved schedule manages to almost eliminate the understaffing problem. However, the total number of overstaffed man-hours increases from 390 hours up to 791 hours. The increase in overstaffing is not surprising since having employees within the minimum and maximum duty hours has higher priority than overstaffing at this particular call center, and the call center has more staff than it needs to satisfy the maximum required on-duty personnel.

5.3 Problem instance: Call center B

Call center B does the planning for only 4 weeks in advance, but the schedule is created down to 15 minute intervals. This instance also does not use minimum and maximum number of employees that should be on duty in each interval, but specifies only the

	Preliminary schedule	Improved schedule
Scheduled hours	3997	6670
Man-hours overstaffed	192	641
Man-hours understaffed	2464	517
Employees below minimum duty hours	23	0
Total unscheduled duty hours	2087	0

Table 4 Results for airport ground services.

exact number of employees that should be on duty at each time. Since there is no flexibility in the required number of on-duty employees, the schedule is likely to have both overstaffing and understaffing as the algorithm tries to fit the number of employees to the exact number of required on-duty employees. The total number of employees at call center B is 62. The call center is overstaffed, the total available man-hours for the scheduling period is 8134 hours while the total required man-hours over the same period is 7134 hours. Table 3 shows the difference between the preliminary schedule and the improved schedule. In the improved schedule, there are still 609 man-hours of overstaffing and 529 unscheduled duty hours. The majority of this overstaffing and the unscheduled duty hours is due to the 1000 man-hour difference between the required man-hours and the available man-hours.

The hard constraints that must be satisfied for call center B are that employees cannot be working on more than 6 consecutive days, in every 24 hour period there must be at least 11 consecutive hours of rest and at most 11 hours of work. The maximum length of a shift is 11 hours while the length of a shift must be at least 4 hours.

Figure 3 shows a single week from the 4 week planning period. We see that there is still much overstaffing and understaffing in the improved schedule, but the overstaffing is more evenly distributed than in the preliminary schedule. One of the requests from the call center was that if overstaffing is necessary then it should be distributed as evenly as possible over the busiest periods. The improved schedule has problems with understaffing, the nightshifts for the first two nights in this particular week do not have anyone on duty, while the requirements call for at least one employee on duty at all time. The average percentage of requested hours still in the improved schedule is 86%. The reason for the low ratio of requested hours still in the improved schedule is mostly due to the fact that 12% of requested hours in the preliminary schedule were overstaffed and had to be changed.

5.4 Problem instance: Airport ground service.

The fourth problem instance is an airport ground service company. The scheduling period is six weeks in 30 minute intervals. The demand for on-duty employees depends on the flight schedules at the airport. In this particular instance, there are many flights that leave during the early morning, and then there is another concentration of flights in the afternoon. Since there are almost no flights scheduled at any time apart from the morning and afternoon busy periods, the requirements for employees peaks during the two busy periods but drops sharply during other times. The airport ground service has 53 employees. Due to the structure of the manpower requirements, the employees often work a short morning shift and then another short afternoon shift with a few hour break in-between. This problem instance is understaffed compared to the previous

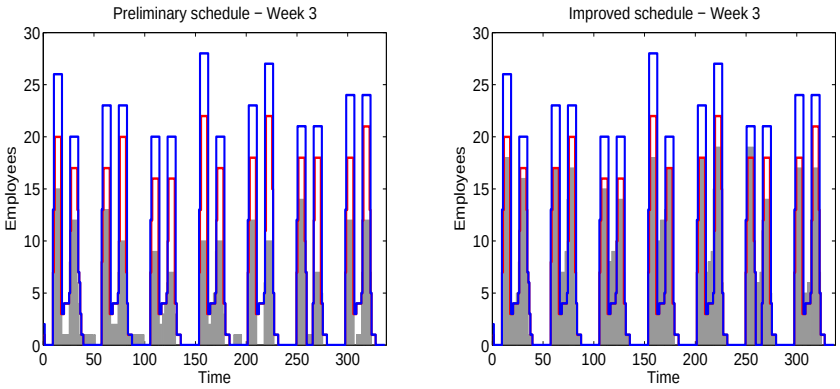

Fig. 4 Staffing levels for the airport ground services problem instance. The gray area denotes the number of employees on duty while the two lines denote the minimum and maximum required staff on duty at each time.

examples, here the total maximum required man-hours is 8152 hours over the scheduling period while the available man-hours is only 6350. Table 4 shows the results of the improvements made to the preliminary schedule.

The hard constraints for the airport ground service are that there must be a minimum continuous rest of 11 hours in any 24 hour period, each employee can not work more than 5 consecutive days, employees cannot work more than 12 consecutive hours while the number of working hours in any 24 hour period is also 12 hours.

Table 4 shows that there are 641 hours of overstaffing in the improved schedule, even though the total available hours is much lower than the total maximum required hours. Figure 4, which shows the preliminary schedule and the improved schedule for a single week, gives an insight into why there is so much overstaffing. Due to the narrow peaks of busy periods in the morning and the afternoon, it's difficult to fit the employees exactly to the manpower requirements. The hours between the busy periods are often overstaffed due to employees working long shifts that cover both busy periods. The average percentage of requested hours that are in the improved schedule is 94%.

Figure 4 shows that the preliminary schedule is very understaffed. One of the reasons for the understaffing in the preliminary schedule is that out of 53 employees, 20 did not sign up for any shifts. However, the improved schedule does not have any employees under the minimum duty hours, so the algorithm managed to create feasible schedules for all the employees.

6 Conclusions

In this paper we have introduced an algorithm that is designed to emulate the behavior of staff managers when creating a high quality feasible staff schedule from a partial staff schedule based on requests from employees. Having a transparent system that the employees can trust is important since it encourages the employees to complete their own scheduling as well as possible, in the belief that the system will behave in a fair manner, while also completing the schedules for the employees that haven't completed their schedule.

Using real data from four different companies and institutions, we have shown that that the proposed algorithm does well in real world situations. The algorithm manages to decrease understaffing and make sure that the working hours of almost all employees are within the minimum and maximum bounds. Three of the examples that we presented have the problem of having more staff than the manpower requirements call for, so there is some overstaffing. The examples and the results demonstrate how difficult the staff scheduling problem is and highlight the challenge of maintaining a balance between overstaffing, understaffing, employee requests and the size of the staff versus the demand for employees. The presented algorithm is simple, transparent and easily understood, but still manages to perform well in our real world examples.

There is still more work that needs to be done, such as adding additional modules to the algorithm and analyzing the order in which they are executed. An ideal system based on this algorithm would have the option that each company would be able to define exactly in which order the modules are executed, in order to emulate exactly the behavior of the staff managers of the company. Being able to tailor-make the software for individual companies would allow for seamless implementation of such a staff scheduling system into companies that are currently spending time and effort into finding manual solutions to the staff scheduling problem.

References

1. J. Ahmad, M. Yamamoto, and A. Ohuchi. Evolutionary algorithms for nurse scheduling problem. *Proceedings of CEC00, San Diego*, pages 196–203, 2000.
2. U. Aickelin and K. Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3(3):139–153, 2000.
3. J. F. Bard and H. W. Purnomo. Preference scheduling for nurses using column generation. *European Journal of Operational Research*, 164, 2005.
4. M. J. Brusco and L. W. Jacobs. Cost analysis of alternative formulations for personnel scheduling in continuously operating organisations. *European Journal of Operational Research*, 86:249–261, 1995.
5. E. K. Burke, P. De Causmaecker, and G. Vanden Berghe. A hybrid tabu search algorithm for the nurse rostering problem. *SEAL98, LNCS 1585*, pages 187–194, 1999.
6. E. K. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of Scheduling*, 7:441–499, 2004.
7. E. K. Burke, P. De Causmaecker, S. Petrovic, and G. Vanden Berghe. Variable neighborhood search for nurse rostering problems. *Metaheuristics: computer decision-making*, pages 153–172, 2004.
8. E. K. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence special issue on Simulated Evolution and Learning, Springer*, 15(3):199–214, 2001.
9. P. De Causmaecker and G. Vanden Berghe. Towards a reference model for timetabling and rostering. *Annals of Operations Research*, 2010.
10. P. De Causmaecker, P. Demeester, and G. Vanden Berghe. Relaxation of coverage constraints in hospital personnel rostering. *Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling*, pages 187–206, 2002.
11. P. De Causmaecker, P. Demeester, Y. Lu, and G. Vanden Berghe. Agent technology for timetabling. *Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling*, pages 215–220, 2002.
12. K. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operations Research*, 106:393–407, 1998.
13. F. Easton and N. Mansour. A distributed genetic algorithm for employee staffing and scheduling problems. *Conference on Genetic Algorithms, San Mateo*, pages 360–367, 1993.
14. A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, February 2004.

15. J. P. Howell. Cyclical scheduling of nursing personnel. *Hospitals*, 40(2):77–85, 1966.
16. R. Hung. Improving productivity and quality through workforce scheduling. *Industrial Management*, 34(6), 1992.
17. S. Petrovic, G. Beddoe, and G Vanden Berghe. Storing and adapting repair experiences in employee rostering. *Selected Papers from PATAT, LNCS 2740. Springer-Verlag.*, pages 149–166, 2002.
18. J. Tanomaru. Staff scheduling by a genetic algorithm with heuristic operators. *Proceedings of CEC95*, pages 456–461, 1995.