Universiteit Utrecht

Master Thesis
ICA-3418936

On the Complexity of Nurse Scheduling Problems

Author:
Steven den Hartog
steven.den.hartog@gmail.com

Supervisor:
Dr. J.A. Hoogeveen

Secondary supervisor:
prof. dr. H.L. Bodlaender

Department of Information and Computing Sciences
Faculty of Science
Utrecht University, The Netherlands

April 2016

Abstract

The Nurse Scheduling Problem is a specific rostering problem for situations that need 24-hour coverage. There is a wide variety of constraints, and while some combinations have been proven NP-complete in the past, there is still a large number of problems that remain unclassified. We have defined a number of constraints for a version of the Nurse Scheduling Problem that are common in literature and are able to model a wide variety of problems. For most combinations of these constraints we have shown whether the problem is NP-complete or whether it is polynomially solvable.

Contents

1 Introduction

Manpower scheduling is a computational field that has been worked on for over 40 years. Hospitals have planning problems that are more difficult than other planning problems, because wards must be staffed 24 hours a day, 7 days a week. This has given rise to the `Nurse Scheduling Problem (NSP)`.

The `NSP` aims to plan nurses in a 24 hour ward. However, there are often many constraints on these schedules, as there are many safety and legal regulations that must be conformed to (Burke et al. (2004)).

For example, consider the following problem. There is a ward with a number of nurses assigned to it. These nurses have a variety of tasks: Washing patients, bringing food, administering medicine and occasional checking on patients with more severe situations. In addition to all of these, nurses should be able to respond in case of an emergency. Therefore, it is vital that there are always enough nurses active on the ward. This ward has three working shifts: An early morning shift from 6:00 until 14:00, an afternoon shift from 14:00 until 22:00 and a night shift from 22:00 until 6:00. With these three 8-hour shifts, the entire 24 hours of a day are covered. In practice these shifts would be slightly longer to cover the transfer of work, but in most cases 3 shifts is enough to cover the entire 24 hours. Lastly, a nurse can be assigned a day off, the `Day-off` shift.

Next, there are a variety of constraints. For example, there must always be enough nurses present at the ward, and there always needs to be one headnurse. Each nurse should get sufficient days off in each fortnight, but no nurse is allowed to work more than 3 night shifts in a row. Lastly, a nurse should have at least 16 hours of rest between the end of his or her shift and the beginning of the next shift. We have provided a small example of a solution of such a problem in Table 1.

	Day 1	Day 2	Day 3
Nurse 1	14:00-22:00	22:00-6:00	Day off
Nurse 2	14:00-22:00	Day off	6:00-14:00
Nurse 3	6:00-14:00	14:00-22:00	14:00-22:00
Nurse 4	22:00-6:00	Day off	22:00-6:00
Nurse 5	Day off	6:00-14:00	22:00-6:00

Table 1: *A small example of a possible solution for a `Nurse Scheduling Problem`. There are 4 possible shifts, a planning period of 3 days and 5 nurses. Every day there is at least one nurse assigned to each of the shifts.*

The goal of the `NSP` is to find a roster for each nurse where all of these constraints are satisfied. The solution assigns exactly one shift to each nurse on each day of the planning period. In many real-life situations, this solution is made by hand. The problems have high complexity, so this takes a lot of time. It would help a lot many hospitals and clinics if each problem could be solved with a computer-algorithm. Furthermore, this

problem is not only limited to the titular nurse scheduling: Many of the same results also apply to other fields where employees have to be scheduled with 24-hour coverage.

The example that has been given is just one instance of the `Nurse Scheduling Problem`. As was mentioned before, there are many different variations of the problem, and each problem needs its own algorithm to be solved efficiently.

While a lot of work has been done on the field, the theoretical side is still incomplete. Most research is focused on solving difficult problems, either with an approximation algorithm like simulated annealing or finding an exact solution by solving an elaborate ILP or expensive branch-and-price. However, because most instances of the problem are very specific to one hospital or ward, due to laws, policies or subject matter, not a lot of the results can be used for other problems. Sometimes it is proven that an instance of `NSP` is NP-complete, but then this is done on complex instances such that the result is not useful for other research.

We wish to elaborate on the theoretical side, and most notably on when an instance of `NSP` is polynomial or NP-complete. The goal is not to find a real life instance of the `NSP` that is polynomial. Real life instances are very complex, and it is very unlikely that one such instance is polynomially solvable. However, it could be useful to find subproblems that are polynomial, by removing constraints from the problem. A polynomial subproblem can be used as a subroutine in an algorithm. An example of an algorithm which frequently uses polynomial subproblems is 2-phase decomposition.

Our goal is to look at a variety of popular constraints, and then find out in what combinations they are polynomial, or when the related problems become NP-complete. First, we give an overview of literature as background information in Section 2. Then we will take a look at a variety of constraints and formally define them in Section 3. Afterwards, we will present a number of combinations of these constraints and elaborate on these problems in Section 4 and Section 5. Lastly we present our conclusions in Section 6

2 Related Work

Burke et al. (2004) present an extensive literature overview of nurse scheduling and related fields. It contains a wide variety of problem descriptions and algorithms to solve these. Van den Bergh et al. (2013) present a more recent literature overview of the entire personnel scheduling field, which also includes papers on nurse scheduling.

In De Causmaecker and Vanden Berghe (2011) a framework for categorizing nurse scheduling problems was suggested. We will use this categorization as well. It is similar to the $\alpha|\beta|\gamma$ classification for scheduling problems, and as such each problem will have an $\alpha|\beta|\gamma$ notation. We present this NSP-categorization scheme in Section 3.1

`NSP` has often been said to be proven NP-complete, for example by Osogami and Imai (2000), Even et al. (1976) and Lau (1994). These all look at different sub-problems, which will be discussed later. Osogami and Imai (2000) discuss a neighborhood-search algorithm with many different k-opt operations. Even et al. (1976) is a theoretical paper on time tabling and multi-commodity flow problems. Lau (1994) presents an approximation algorithm for nurse scheduling with cyclic rosters.

Della Croce and Salassa (2014) solve the nurse rostering problem for an Italian private hospital with variable neighborhood search. This paper focuses on an existing hospital in Italy, and as such only has constraints which appear in real-life situations. In Brucker et al. (2010), a neighborhood search is introduced where the components are first created depending on the problem. In essence, this is a two-phase decomposition. The algorithms are then tested on a number of benchmark sets. van Veldhoven et al. (2014) suggest a two-phase decomposition where it is first decided who gets a day off on which day, followed by an optimization of the assignment of the rest of the shifts with mathematical programming. They also show a situation where in the first phase they decide not only on days off, but also on night shifts.

Weelden (2013) solves a nurse rostering problem for a ward in a large Dutch hospital. They do so with an ILP based on columns, and then change their solution with heuristics.

While most papers focus on solving difficult problems, Smet et al. (2014) is one of the few papers which brings forth polynomial subproblems. It discusses algorithms where one of the main objective functions to be maximized is employee satisfaction: Each nurse can give preference to certain shifts for each day. These are then solved with a min-cost flow algorithm.

An example on how to solve an instance of nurse rostering with branch-and-price can be found in Baeklund (2014). This paper is based on a ward in Denmark. Bruni and Detti (2014) present a method to use branch-and-cut to a nurse scheduling problem, putting a lot of focus on the different skill-sets of employees.

An international nurse rostering competition was held in 2010 (Haspeslagh et al. (2014)). It was the first competition where multiple teams tested their algorithms against each other on the same datasets. It included both deterministic and stochastic

algorithms.

3 Problem definition

3.1 NSP-categorization

As mentioned before, De Causmaecker and Vanden Berghe (2011) present a framework to categorize nurse scheduling problems. We refer to this in the rest of the paper as the *NSP-categorization*. We will explain the framework in this subsection.

As mentioned before in Section 2, the NSP-categorization uses an $\alpha|\beta|\gamma$ notation to describe instances of the NSP.

- The α category describes the constraints put on nurses. For example, skill requirements, availability of nurses and sequence constraints that limit what shifts can be worked over a sequence of multiple days all belong to the α category.
- The β category describes coverage constraints. This includes what amount of nurses is required on which day, whether this can fluctuate between days or not, shift structure and whether coverage constraints are shift-related or time-related.
- The γ category describes the optimization objective. This objective can for example be to minimize costs or to maximize nurse happiness.

Each category has multiple constraint categories that can be included. These are all in capital letters. Some categories have sub-classes to distinguish more distinction between problems. These will be listed in parentheses behind the constraint category. An example of this categorization is *S(b,c,d)|NRV|LP*. The constraints belonging to the α category are the sub-classes *b, c* and *d* of the *S* constraint. The three constraints *N, R* and *V* belong to the β category, and the optimization objective γ is formed by the objectives *L* and *P*. As there is an objective function, at least one of the constraints needs some kind of penalty function. If there are no optimization objectives in the γ category, the goal is to find any feasible solution.

It should be noted that because of the wide variety of constraints in nurse scheduling, this categorization is not necessarily injective. This means that two different problems could be classified under the same categorization due to minute differences the framework does not necessarily cover. Therefore, the framework is best used as a reference to quickly find related problems.

The categorization of each constraint is listed in Table 10 in Appendix A, and the categorization of each problem we discuss is listed in Table 3 of Section 4.3.

3.2 Introduction to problem definition

In this section, we will present and define the problems we will discuss in Section 4. This will be done by first defining the base Nurse Scheduling Problem with no constraints, and then presenting each constraint that is dealt with in this paper. For each constraint we will supply an informal interpretation, followed by real-life motivations for the constraint. Then a formal definition and a constraint classification from De Causmaecker

and Vanden Berghe (2011) will be given. Lastly, we will note alternate related versions of the constraint.

All problems that will be introduced can be found in Table 10 in Appendix A with their abbreviation and categorization. All variables that will be introduced can also be found in Table 11 in Appendix A. This table lists all introduced variables, what constraint they are from and what their role is.

We define the `Nurse Scheduling Problem (NSP)` as follows: We are given a set of nurses, shifts and days. For each day, we must assign exactly one shift to each nurse.

The set of shifts are all possible work-time assignments that can be assigned to a nurse, including a day off. A generic shift-set S is {`Early, Late, Night, Day-off`}, where each working shift is 8 hours long. Together the 3 working shifts cover all 24 hours of a day, and the fourth shift is a day off.

While some models model a day off by not assigning any shift to a nurse/day combination, in this paper we assume a day off is one of the shifts, to generalize constraints. When exceptions are made, it is explicitly stated.
A more mathematical definition (though essentially the same) is given below:

Definition 1. *Define p_{ik} as the shift that nurse i works on day k. We are given a set of nurses N, a set of days D and a set of shifts S. For each day, we must assign to each nurse one shift. In other words, assign one shift $j \in S$ to each p_{ik}, $i \in N, k \in D$. Call the complete assignment $\tilde{p} := \{p_{ik} | \forall i \in N, \forall k \in D\}$.*

Definition 2. *We refer to the set $\{p_{ik} | \forall k \in D\}$ as the* personal roster *assigned to nurse i.*

We expand *Definition 1* with a constraint-set C. By adding different constraints to constraint-set C, the problem can be customized to more accurately reflect the real problem or the wishes of the user.

Each constraint is either a hard constraint or a soft constraint. A hard constraint only puts restrictions on what kind of solutions $\tilde{p}$ are allowed. A soft constraint only penalizes certain choices, and has a corresponding penalty function $c(\tilde{p})$, which assigns a penalty to assignment $\tilde{p}$. For all soft constraints discussed in this paper, function c is assumed to be (piecewise) linear.

We define the `NSP` with constraint-set C as follows: We are given a set of nurses, shifts and days. For each day, we must assign exactly one shift to each nurse in such a way that each constraint is satisfied. If at least one constraint is a soft constraint, the optimal solution minimizes the total penalty.

More formally:

Definition 3. *Consider an instance of `NSP` with a constraint-set $C := C_1 \cup C_2$, where all constraints in C_1 are hard constraints and all constraints in C_2 are soft constraints. If $C_2 = \emptyset$, then the `NSP` is a decision problem where each solution $\tilde{p}$ must satisfy all constraints $c \in C_1$. Otherwise, the `NSP` is a minimization problem where the goal is to minimize $\sum_{c_i \in C_2} \tilde{c}_i(\tilde{p})$ while satisfying all constraints $c \in C_1$.*

The base `Nurse Scheduling Problem` without constraints is trivially solvable by assigning random shifts to each nurse for every day.

3.3 Shift Coverage

One of the most vital things for a nurse ward is to make sure there are enough nurses at all times. However, to save costs, there is often a maximum number of nurses desired. Furthermore, these parameters can be influenced by time and day. For example, on the `Night`-shift, generally fewer people are needed than in the afternoon. However, it is likely that more nurses are needed on New Year's Eve. So we would like to put an upper bound and a lower bound on each shift/day combination.

We call the lower-bound of shift j on day k L^s_{jk}, and the upper-bound of shift j on day k is called U^s_{jk}. As these are not the only bounds we will refer to in this paper, L^s and U^s are designated with the s for shift coverage.

Define `Shift Coverage` as follows:

Definition 4. *$\forall j \in S, \forall k \in D$, define lower-bound L^s_{jk} and upper-bound U^s_{jk}, where $0 \leq L^s_{jk} \leq U^s_{jk}$. Furthermore, define p^j_{ik} as the amount of nurses that are assigned to shift j on day k.*

As a hard constraint, $\forall j \in S, \forall k \in D$, the following inequality must hold: $L^s_{jk} \leq p^j_k \leq U^s_{jk}$.

When shift coverage is used as a soft constraint, one could imagine the penalty assigned to breaking the upper bound as a penalty for wastefully rostering someone who is not needed. On the other hand, the cost penalty for breaching the lower bound could be the monetary cost of hiring extra personnel to make sure the ward is staffed.

In this case, assign penalties q^L_{jk} and q^U_{jk} to the lower bound and upper bound respectively of each j, k. Then, for each j, k where $p^j_k < L^s_{jk}$, increase the result of the penalty-function by $(L^s_{jk} - p^j_k) * q^L_{jk}$, and if $p^j_k > U^s_{jk}$ increase the result of the penalty-function by $(p^j_k - U^s_{jk}) * q^U_{jk}$.

According to the `NSP`-categorization by De Causmaecker and Vanden Berghe (2011) introduced in Section 3.1, `Shift Coverage` is from category β, and as we assume that the constraint can be different for every day, specifically a type-RV constraint.

Sometimes we will refer to the `Strict Shift Coverage` constraint. This is the same as the `Shift Coverage` constraint, but for each $j \in S, k \in D$, $U^s_{jk} = L^s_{jk}$. We refer to the combination of `Strict Shift Coverage` and `Shift Coverage` as the `Coverage` class of constraints.

3.4 Balanced Rosters

We wish to make sure the workload is decently balanced across employees. For example, it is unfair (and often also illegal) if one employee only works night shifts, or if all days off are stacked on one nurse. Over the planning period, we would like to put a maximum and a minimum on the number of times a single nurse is allowed to work a specific shift. For each nurse i and shift j we introduce a lower bound L^b_{ij} and upper bound U^b_{ij} on the amount of times nurse i should work shift j during the planning period. These bounds are marked with a b for balanced rosters.

Definition 5. *We define for each $i \in N, j \in S$ the lower bound L_{ij}^b and upper bound U_{ij}^b, with $0 \leq L_{ij}^b \leq U_{ij}^b$. L_{ij}^b and U_{ij}^b are the lower bound and upper bound of the number of times nurse i is allowed to be scheduled for shift j.*

Similarly to `Shift Coverage`, we can turn this hard constraint into a soft constraint by assigning a penalty to breaking the upper or lower bound for each combination of nurse and shift.

According to the `NSP`-categorisation, this constraint is from category α, specifically the b sub-class of the A-type constraint.

The `Balanced Roster` constraint has many problems that it cannot fix. For example, consider the problem where our planning period $|D| = 30$, shift 4 is a day off and we have lower and upper bounds $L_{14}^b = 8, U_{14}^b = 9$. The `Balanced Roster` constraint does not prevent an algorithm from scheduling all 9 days off on the first 9 days, which is probably not the wanted outcome of the algorithm.

We can instead create a new constraint which puts upper and lower bounds on subsets of D. However, we do not pursue further in this direction in this paper, as the `Balanced Roster` constraint is hard by itself.

3.5 Forbidden Sequences

There are certain sequences of shifts taken by the same employee that are either infeasible, break some kind of labor law or go against company regulations. For example, consider the basic example with 3 work-shifts and a shift for a day off. Assume the three shifts are all 8 hours long and cover the 24 hour period of a day. It is infeasible for a nurse to work a `Night`-shift, followed by an `Early`-shift the next day. The starting time of the `Early`-shift is immediately after the nurse's previous `Night`-shift ended, giving the nurse a roster where he has to work 16 hours straight!

We would like to prevent some sequences from happening by presenting a set of forbidden sequences.

Definition 6. *Define a forbidden sequence f as an ordered set of shifts. Then define the set F of all forbidden sequences. For each $f \in F$, no personal roster may contain a sequence of $|f|$ shifts for $|f|$ consecutive days identical to f.*

As a hard constraint, none of the forbidden sequences are allowed. As a soft constraint, we can put a penalty on each forbidden sequence which is incurred whenever the forbidden sequence appears. This can signify an imaginary penalty for breaking a company policy or a monetary penalty that has to be paid for breaking labor laws if the unions ever find out.

According to the `NSP`-categorization, this constraint is from category α, and incorporates the b and c sub-classes of the S-type constraint.

The standard `Forbidden Sequences` description could use some limitations. For example, there is a big difference between a problem where all forbidden sequences are a week long, and a problem where all forbidden sequences are 2 days long. We define

a special notation, where the constraint is followed by an (in)equality sign and a number, indicating how long all forbidden sequences are. For example, consider `Forbidden Sequences=2`, the case where all forbidden sequences are two days long, and `Forbidden Sequences`$\geq$`3`, the case where all forbidden sequences are three or more days long.

F is defined as a generic set. Storage of this could be interpreted as a list or array. For some of these cases, this storage implementation is rather inefficient. If set F grows large, the lookup time could be noticeable in the running time of an algorithm. Most notably, the case of `Forbidden Sequences=2` can also be encoded as a 2-dimensional array with length and width S. In this array, each field contains a 1 if the first shift can be followed by the second shift, and a 0 if this is not allowed. In such a situation the look-up time is constant, though the array always takes up $O(n^2)$ space.

A popular constraint is the 'Forward Rotating Roster', which does not allow a nurse to start a shift earlier than his shift started on the previous day. This is encoded as follows: $F = \{\{\texttt{Late, Early}\}, \{\texttt{Night,Early}\}, \{\texttt{Night,Late}\}\}$

3.6 Personal Requests

It is important to be able to implement the specific wishes of nurses. For example, a nurse should be able to ask for a specific day off, or that he or she wishes to work a certain shift on said day. This request is a simple but very important one, as a system where nobody can at least ask for a day off is very inflexible. Furthermore, employees will request days off anyway, and then it is the task of a scheduler to manually change the output of an algorithm.

Definition 7. *Define a request $r \in R$ as a combination of a nurse, day and shift: $r \in N \times D \times S$. If nurse i has requested a specific shift j on day k, we refer to j as the value of r_{ik}. For each request $r_{ik} \in R$, p_{ik} must be equal to r_{ik}.*

It should be noted that r_{ik} does not necessarily exist for all possible values of i and k. If nurse i did not request a shift on day k, then r_{ik} is empty.

According to the `NSP`-categorization, this constraint is from category α, and specifically the e sub-class of the A-type constraint.

As a hard constraint, all requests have to be fulfilled. As a soft constraint, a penalty can be assigned to each request which is incurred if the request is not fulfilled.

There is also a variation of this problem where each nurse is allowed to designate shift and day combinations where he or she does *not* want to work. We have chosen to ignore this variation as it does not impact algorithms and complexity very much. Most notably, if nurses are only allowed to designate which shifts they do not want to work, we can still model a shift request by designating all other shifts on that day. In this way, each reduction of a constraint-set including `Personal Requests` still holds, even if

nurses can only designate shifts they do not want to work.

This constraint can be restricted to the `Days-off Requests` constraint, where a nurse can only request a day off. We refer to the combination of `Personal Requests` and `Days-off Requests` as the class of `Requests` constraints.

The `Personal Requests` constraint can also be used to model the shifts that nurses worked before the given scheduling period. For example, given the problem that 3 weeks should be scheduled, but due to other constraints like `Forbidden Sequences` or `Balanced Roster` the previous shifts matter. The scheduling period D could be extended to be 4 weeks, and the entire first week is already filled in with `Personal Requests` that reflect the last week of work before the scheduling period. In this case the constraint should be hard, or the penalty on each of these shifts should be so high that it is too expensive to not fulfill these requests.

3.7 Identical Weekends

A common occurring constraint is to have all nurses work the same shift on both Saturday and Sunday.

Definition 8. *Assign a day to each $k \in D$, according to the natural day order. Then, on each subsequent Saturday/Sunday pair, each nurse must work the same shift on Saturday and Sunday.*

According to the `NSP`-categorization by De Causmaecker and Vanden Berghe (2011), this constraint is only mentioned in the 'other' (d) sub-class of the S-type constraint (α) The constraint can be used as a soft constraint by assigning a penalty to breaking this rule.

A softer version is the `Full Free Weekends` constraint, which says that each nurse must either be free for the entire weekend, or work the entire weekend. It does not matter which working shifts are assigned in these weekends. We refer to the combination of `Identical Weekends` and `Full Free Weekends` as the class of `Weekends` constraints.

3.8 Skill Requirements

Not all nurses are homogeneous in what tasks can be assigned to them. For example, it is common to have designated 'Head nurses', of which one should be active at all times. Therefore, we wish to be able to assign a set of skills to each nurse, and to put a lower bound on how many nurses with these skills should be available. We only need a lower bound, as a nurse can choose not to use his skill-set and act as a normal nurse without his or her skills. However, in practice we will also use an upper bound, to keep this constraint similar to `Shift Coverage`.

There are two distinctive methods of implementation, depending on how the skills are used in the ward. There are generally two different ways, which we will now discuss.

In the first method of implementation, a skill represents a task that the nurse will fulfill for the entirety of his shift, and in one shift a nurse can only work with one skill. For example, if a nurse can work as the aforementioned 'Head nurse', we can assume that working as a head nurse involves different tasks than working as a normal nurse. For example, maybe a head nurse cannot spend prolonged time with a patient, because he needs to always be available for questions from other nurses. If we consider a second task with a prerequisite skill, for example 'Reception' or 'Training new nurses', it is likely that one nurse cannot do both tasks at once during his shift.

In this implementation method, each nurse must also be assigned a task associated with a skill during each shift.

In the second method of implementation, a skill represents a small task for which a nurse needs to be qualified, but which does not take up the entire shift. For example, the ability to administer a specific medicine or the qualification to greet new patients. It is likely that a nurse can do multiple of these during the same shift, but there always needs to be someone available who can do the task. In this implementation method, each nurse does not have to be assigned a task associated with a skill during each shift.

In the rest of this paper, we will always assume the first implementation method. Though it is likely that in a real environment the actual situation is a combination of the two, we feel that the first implementation is more important to model. Therefore, when the `Skill Requirements` constraint is active, each nurse is not only assigned to a shift on each day, but also a task associated with a skill.

Definition 9. *Define the set of skills as set L. Assign to each nurse i a skill-set $l_i \subseteq L$, the skills nurse i has. It should be noted that each nurse can only use one of their skills at a time on their shift. When this constraint is active, assign to each p_{ik} not only a shift j, but also a skill assignment l with $l \in l_i$.*

Then, define for each combination of day $k \in D$, shift $j \in S$ and skill $l \in L$, the lower-bound L^l_{jkl} of the amount of nurses with skill l as their active skill on day k and shift j.

According to the `NSP`-categorization, this constraint is from category α, the I constraint.

As a soft constraint, we can again assign a penalty for breaking labor regulations.

4 Complexity classification

4.1 Introduction

In this section, we will list a number of `Nurse Scheduling Problems` and, if possible, classify their time-complexity. First we will go through some preliminaries on time-complexity of various constraints. Then we present a table of all problems handled in this section and their result. Finally, we will go through each of the problems in this table.

For each problem, we will state the constraints belonging to the problem, and any additional notes on these constraints. Together with this list we present the classification according to De Causmaecker and Vanden Berghe (2011). Then we show the time complexity by either outlining a polynomial algorithm, a proof of NP-hardness or a citation of a paper showing one of these. If the problem is open, we present various findings that could help future research.

In this section, we will usually talk about a constraint-set C, a set of `NSP` constraints. As shorthand, we will refer to any `Nurse Scheduling Problem` with constraints equal to constraint-set C as the problem associated with set C.

We assume that each constraint can only appear once in C. This also means that if a hard version of a constraint is in C, the soft version cannot also be in C. Furthermore, we do not consider any situation where two constraints that share a constraint-class are present at once. For example, we do not consider a problem with both `Strict Shift Coverage` and `Shift Coverage`, as they are both in the `Coverage` class. We will now explain why.

When two of the same class constraints are both in the problem, one of two things can happen. First, it is possible that the two constraints contradict each other. For example, in the case where both `Strict Shift Coverage` and `Shift Coverage` are present, they might not agree on lower bounds and upper bounds for one shift. This is an unlikely and ridiculous scenario, so we will not consider it.

Secondly, it is possible that the two constraints can coexist without causing any problems. In that case, one of the constraints can be completely disregarded while still being the same problem, and if needed this constraint can be slightly changed. For example, if both `Full Free Weekends` and `Identical Weekends` are present as hard constraints, any solution that satisfies `Identical Weekend` will automatically satisfy `Full Free Weekends`. In the soft versions, only the penalty function of the remaining constraint would have to be changed.

It should be noted that we did not consider a mix of soft and hard constraints for this. For example, hard `Shift Coverage` and soft `Strict Shift Coverage`. While these do not necessarily contradict each other, the resulting problem is still strange enough that we do not consider it, as the problem could probably be modeled better in another way. Furthermore we are only interested in the complexity of these problems, and a mix of hard and soft versions of the same constraint should not impact the complexity.

We only consider problems with a constraint from the `Coverage` class. A problem

without any `Coverage` constraints does not even fulfill the most basic requirements of any real-life scheduling problem, as the ward cannot function, and if needed the bounds on `Shift Coverage` can be set to 0 and $|N|$.

Remember that the problem of C is either a decision problem, if all constraints included in C are hard constraints, or a minimization problem if some constraints included in C are soft constraints. We refer to these respectively as the decision problem of C and the minimization problem of C. However, to prove NP-hardness, a minimization problem is transformed into a decision problem, where the problem is to find a valid solution where the value of $\tilde{c}(p) \leq K$, for some K. Call this the $K-$decision problem of C.

Lastly, if the `NSP` with constraint-set C is considered NP-hard, we call C NP-hard.

4.2 Preliminaries

4.2.1 Complexity and additional constraints

Lemma 1. *Consider the `NSP` with constraint-set C. Given any constraint c' not in C, where c' does not belong to the `Weekends` or `Strict Shift Coverage` constraints. If C is NP-complete, then so is the problem with both C and c'. Define C' as the problem with constraint-set C and also constraint c'.*

Proof of Lemma 1. For most constraints c', it is easy to see that any reduction from an NP-hard problem that holds for C can also hold for C', by coding the constraints in such a way that they do not influence the problem at all. We will now show for all relevant constraints c' how to do this.

- `Shift Coverage`: For each $j \in S, k \in D$, set lower-bound $L^s_{jk} = 0$ and upper-bound $U^s_{jk} = |N|$. Now C' is identical to C.
  This constraint is given for completeness, as we assume C always includes either `Shift Coverage` or `Strict Shift Coverage`.
- `Balanced Rosters`:For each $i \in N, k \in D$, set lower-bound $L^b_{ij} = 0$ and upper-bound $U^b_{ij} = |D|$. Now C' is identical to C.
- `Forbidden Sequence`: Set the set of all sequences $F = \emptyset$. Now C' is identical to C. This proof also holds for any restrictions placed on `Forbidden Sequences`.
- `Requests`: Set the set of all requests $R = \emptyset$. Now C' is identical to C.
- `Skill Requirements`: Use only one skill, and give this skill to all nurses. C' is now identical to C.

□

There are only 3 constraints that cannot easily be made inconsequential.

The first is `Strict Shift Coverage`. The soft version can be made inconsequential by setting the penalties to zero, but the hard version has no such method. However, we always consider only problems with one of the `Coverage` constraints. This means that C will already contain either `Shift Coverage` or `Strict Shift Coverage`, and as such c' cannot be `Strict Shift Coverage`.

The second and third constraints are `Identical Weekends` and `Full Free Weekends`. Later on in Section 4.18 we will show that it is likely that Lemma 1 still holds if either of these two constraints are c'.

Because of Lemma 1, we do not list all problems in this thesis: If a given set of constraints C is proven NP-hard, we do not list all supersets, as it is clear that these are also NP-hard.

4.2.2 Algorithmic relation between hardness of constraints

Lemma 2. *Consider the NP-complete non-empty constraint-set C. If any of the hard constraints are changed to a soft constraint, the new constraint-set $\tilde{C}$ is still NP-complete.*

Proof. Consider the constraint-set $C := C_1 \cup C_2$, where all constraints in C_1 are hard constraints, and all constraints in C_2 are soft constraints. Assume that $C_1 \neq \emptyset$.

Remember that the minimization problem is transformed into a K-decision problem, for a given K. Take any constraint c_i from C_1. Turn this constraint into a soft constraint and call it $\tilde{c_i}$, by putting the penalties associated with its penalty function $\tilde{c_i}(p)$ strictly greater than K. Now, $\tilde{c_i}$ is effectively still a hard constraint, as any solution that would be unfeasible with c_i as a hard constraint will have a penalty $\tilde{c}(p) > K$ and thus be disregarded. Define the new constraint-set as $\tilde{C} := (C_1 - \{c_i\}) \cup (C_2 + \{\tilde{c_i}\})$.

As has been shown, the solution of C will always be identical to the solution of $\tilde{C}$, unless C has no solution, in which case $\tilde{C}$ will either have no solution or a solution with a penalty larger than K. Since the assignment of the penalties can be done in linear time, C is polynomially reducible to $\tilde{C}$. This means that if C is NP-hard, then $\tilde{C}$ is also NP-hard.

□

In short, if a problem with the hard version of a constraint is NP-hard, then the problem with the soft version of this same constraint is also NP-hard. Ironically, hard constraints are 'easier' than soft constraints. This means that in theory, a problem with hard constraints can have a polynomial algorithm while the problem with the soft versions of these same constraints can be NP-hard. If a problem is proven NP-hard with hard constraints, we will not consider the same problem with soft constraints: After all, by proving the problem to be NP-hard with hard constraints, we have also proven that the same problem with soft constraints is NP-hard.

4.2.3 Algorithmic difficulty ordering on constraints

Similarly to the algorithmic hierarchy between hard and soft constraints, most constraints with multiple versions also have an algorithmic hierarchy.

Lemma 3. *We can reduce* `Shift Coverage` *to* `Strict Shift Coverage`*.*

Proof of Lemma 3. Set $L^s_{jk} = U^s_{jk}$ for each j, k. □

Lemma 4. *We can reduce* `Personal Requests` *to* `Days-off Requests`*.*

Proof of Lemma 4. Only consider requests for days off. □

Lemma 5. *Consider constraint $c :=$ {`Forbidden Sequences=n`}, and any $C \ni c$. If C is NP-hard, then we can replace c with {`Forbidden Sequences=m`}, for any $m > n$, and C will still NP-hard.*

Proof of Lemma 5. First, we will prove that Lemma 5 holds for $m = n + 1$. We show a polynomial-time reduction from constraint {`Forbidden Sequences=n`} to {`Forbidden Sequences=(n+1)`}.

Consider an instance of C that is NP-hard that includes constraint {`Forbidden Sequences=n`} with sequence-set F. Create an instance C' that is identical for all constraints, except that it includes {`Forbidden Sequences=(n+1)`} instead. We will now show how to create a suitable set F' for C'.

First, we assume that $n < |D|$, as otherwise the length of sequences in C' would be longer than the number of days. For each $f_i \in F$, do the following: For each $j \in S$, create two new forbidden sequences, $f_i^j := \{f_{i1}, \ldots, f_{in}, j\}$ and $\tilde{f}_i^j := \{j, f_{i1}, \ldots, f_{in}\}$. Then add each f_i^j and $\tilde{f}_i^j$ to set F'.

For each forbidden sequence f we have created $2 \times |S|$ new sequences, all of length $(n+1)$. The new problem is only polynomially as large. Additionally, it should be clear that every nurse-specific roster that was forbidden by F is also forbidden by F'. Since the roster was forbidden by F, it includes a forbidden sequence. Call this forbidden sequence f_i. If this forbidden sequence is succeeded by a shift, call this shift j. Then the roster is forbidden by sequence f_i^j. If the forbidden sequence f_i is not succeeded by a shift in the roster, it means that the last day of the forbidden sequence is the last day of the scheduling period. Since $n < |N|$, there must be a preceding shift. We can call this shift j again, and then the roster is forbidden by forbidden sequence $\tilde{f}_i^j$.

It should be clear that the reverse is also true: If a roster is forbidden by F', it means that there was a roster containing a forbidden sequence. Call this forbidden sequence f_i^j. This forbidden sequence also contains forbidden sequence f_i. The same is true for any forbidden sequence $\tilde{f}_i^j$.

With this method, there is a polynomial-time reduction from `Forbidden Sequences=n` to `Forbidden Sequences=(n+1)`. Due to induction, there is a reduction to `Forbidden Sequences=n'` for every $n \leq n' \leq |N|$. □

If NP-hard	then	also NP-hard
Hard constraint		Soft constraint
`Strict Shift Coverage`		`Shift Coverage`
`Days-off Requests`		`Personal Requests`
`Forbidden Sequences=n`		`Forbidden Sequences=n'`, $n' > n$
`Forbidden Sequences≥n`		`Forbidden Sequences≥n'`, $n' > n$
Is also polynomially solvable	if	is polynomially solvable

Table 2: *Summary of easiness ordering on constraints*

Corollary 1. *The same method can be used for* ***Forbidden Sequences≥n****, by only transforming those forbidden sequences with length* n*, so there is due to induction also a reduction from* ***Forbidden Sequences≥n*** *to* ***Forbidden Sequences≥n'****,* $n \leq n' \leq |N|$*.*

Lemma 6. *Consider an NSP* $C \ni \{$***Forbidden Sequences≥n***$\}$*. If* C *is hard, then so is* C *-* $\{$***Forbidden Sequences≥n***$\}$ *+* $\{$***Forbidden Sequences≥n'***$\}$ *for all* $n' \geq n$*.*

If constraint c can be reduced to c' and $C \ni c$ is proven NP-hard, then we do not list $C - \{c\} + \{c'\}$, as it is also implied NP-hard.

The results of Section 4.2.2 and 4.2.3 are summarized in Table 2. As noted there, the relation is also the other way around: If C being NP-hard also implies that C' is also NP-hard, then if C' is polynomially solvable, C is also polynomially solvable.

4.2.4 Notation

When referring to constraints, we will sometimes use a shorthand notation from now on. The shorthand notation is an abbreviation formed by the first letters of all words in the constraint, preceded by small 's' or 'h', denoting whether the constraint is soft or hard. For example, a shorthand notation of a hard `Shift Coverage` constraint is noted with `hSC`.

Though we have listed for each constraint how they are categorized in the `NSP`-categorization by De Causmaecker and Vanden Berghe (2011), we have not yet mentioned all relevant categories. The last category, γ, is all about the optimization function. γ has 2 categories we will use in this paper: If category P is included, it means there is some minimization function involving α categories. If category L is included, it means there is some minimization function involving β categories. If they are missing, those categories are assumed to all contain hard constraints. If the minimization function involves both α and β categories, then both P and L are included.

There are also 2 categories in β that have yet to be explained. The first is N. This symbolizes that the amount of shifts is a variable. Though we use a simple 4-shift situa-

tion to illustrate many examples, we always assume the amount of shifts to be variable. Therefore, we will always include this category unless explicitly mentioned. If we do, we use the category of a constant amount of shifts. This is normally marked by a number equal to the amount of shifts. We will signify this as a lower-case k if the amount of shifts is equal to a constant k.

4.3 Overview table

An overview of the problems that will be handled is given in Table 3.

Section	Abbreviation	Categorization	Complexity	Page nr.
4.4	hSC	$\|NRV\|$	Polynomial	21
4.5	sSC	$\|NRV\|L$	Polynomial	21
4.6	sSC-hIW	$S(d)\ \|NRV\|L$	Polynomial	22
4.7	hSC-hBR-hDOR	$A(b,e)\|NRV\|$	NP-complete	24
4.8	hSC-hBR	$A(b)\|NRV\|$	NP-complete	26
4.9	hSSC-hFS=3	$S(b,c)\|NRV\|$	NP-complete	27
4.10	hSC-hFS=2	$S(b,c)\|NRV\|$	Polynomial	28
4.11	sSC-sFS=2	$S(b,c)\|NRV\ \|LP$	Polynomial	30
4.12	sSC-sFS=2-sIW	$S(b,c,d)\|NRV\|LP$	Polynomial	31
4.13	hSC-hPR-hSR	$A(b)I\|NRV\|$	Polynomial	32
4.14	hSC-hSR-hFFW	$A(b,e)I\|NRV\|$	Polynomial	33
4.15	hSC-hPR-hSR	$A(b,e)I\|NRV\|$	Polynomial	35
4.16	hSSC-hFS=2-hDOR	$A(b)S(b,c)\|NRV\|$	NP-complete	36
4.17	hSSC-hFS=2-hSR	$IS(b,c)\|NRV\|$	NP-complete	42
5	hSC-hFS=2-hPR-kShifts	$A(b)S(b,c)\ \|kRV\ \|$	Open	44

Table 3: *Overview of all problems*

4.4 hSC

Constraints:	• hard `Shift Coverage`
Categorisation:	$\lvert NRV\rvert$
Time complexity:	Polynomial

Algorithm. Remember that our goal is to assign nurses to shifts, according to a lower- and upper-bound L^s_{jk} and U^s_{jk} for each $j \in S, k \in D$.
We start out with the simplest problem. This problem is trivial, and can be easily solved in a greedy way.
Repeat the following subroutine for each day $k \in D$:

Take the set of nurses and keep a counter. Then, start with the first shift. Assign L^s_{1k} nurses to shift 1, and increase the counter for each to keep track of how many nurses have been assigned already. Then, do this for all the other shifts in order, each time assigning L^s_{jk} nurses to shift j. If not enough nurses can be assigned, return `FALSE`: The lower-bound cannot be reached for day k.
After this has been done, there may still be nurses left. Then, start with the first shift again, and assign $U^s_{jk} - L^s_{jk}$ nurses to shift j. If there are still nurses left after this has been done for all shifts j, then the upper-bound must be breached for at least one shift j on day k, so return `FALSE`. If there are not enough nurses left to assign $U^s_{jk} - L^s_{jk}$, instead assign how many nurses are left to shift j and continue with the next day.

After all $|D|$ days we have a final solution $\overline{p}$. For each day and shift of the solution, the lower bound has been reached and no upper bound has been breached, or the algorithm would have terminated prematurely. We have also assigned a shift to each nurse on each day, so now we have a valid solution.

For each day, we go through each nurse once, and go through each shift up to 2 times. Therefore, our time-complexity is $O\Big(|D| * (|S| + |N|)\Big)$. □

4.5 sSC

Constraints:	• soft `Shift Coverage`
Categorisation:	$\lvert NRV\,\rvert L$
Time complexity:	Polynomial

Algorithm. The only change from the previous problem is that `Shift Coverage` is now a soft constraint, which means that we now allow breaking the upper and lower bounds,

but penalize doing so. We can slightly alter the algorithm we used for hard `Shift Coverage` in Subsection 4.4.
For each day $k \in D$ repeat the following subroutine:

Take the set of nurses and keep a counter. Order the shifts according to penalty q^L_{jk}. Start with the shift j that has the largest q^L_{jk}, and assign L^s_{jk} nurses to it. Then, continue with the next-largest q^L_{jk}, until the smallest penalty is reached. If at any point there are not enough nurses left, assign as many as possible to the current shift, and continue on with the next day.
Then, if there are nurses left, assign them until all upper bounds U^s_{jk} are reached. Then, if there are still nurses left, assign all of them to the shift with the lowest penalty q^U_{jk}. Then continue with the next day.

For each day, we have found the optimal solution. If there are not enough nurses to reach all lower bounds, we first focus on the lower bounds with the highest penalty per missing nurse, so we minimize the penalty. If there are too many nurses to keep within the upper bounds, we assign all leftovers to the shift with the lowest penalty per extra nurse.

For each day, we go through each nurse once, go through each shift up to 2 times and order the shifts up to twice. Therefore, our time-complexity is $O\Big(|D| * \big(|S| + |N| \log |N|\big)\Big)$.

This problem and the previously handled hard version are rather trivial, and unlike any real problems. The main reason these are easy is because there is no relation at all between different days, and it does not matter which nurse is assigned to which shift, as long as there are enough nurses. These trivial cases were mostly handled as an introduction. □

4.6 sSC-hIW

Constraints:	• soft `Shift Coverage` • hard `Identical Weekends`
Categorisation:	*S(d) |NRV |L*
Time complexity:	Polynomial

Algorithm. This problem is exactly the same as `sSC`, except for the constraint that each nurse should work the same shift on both days of each weekend, and therefore we will edit the previous algorithm for the special weekend case. For each weekend, we will

decide on one assignment of nurses for both Saturday and Sunday. This means that we will first decide the exact amount of nurses for each shift, by calculating the aggregated cost of an assignment of nurses over the complete weekend. Then the nurses can be assigned as in `sSC`.

Call the given Saturday day k and Sunday $(k+1)$. First, we assume that for each shift, the lower bound on Saturday is less than or equal to the lower bound on Sunday ($L^s_{jk} \leq L^s_{j(k+1)}$). If this is not the case, swap both bounds for that shift with each other. As we will choose one assignment for both days of the weekend, it does not matter which day is denoted Saturday and which is denoted Sunday, and there are no ties between 2 subsequent days.

There are 2 cases that can occur. Both cases are shown in Figure 1.

In the first case, assume $U^s_{jk} \leq U^s_{j(k+1)}$. This case is shown in (a) of Figure 1. Now, for each shift j define the following intervals: $[0, L^s_{jk}]$, $(L^s_{jk}, L^s_{j(k+1)}]$, $(L^s_{j(k+1)}, U^s_{jk}]$, $(U^s_{jk}, U^s_{j(k+1)}]$ and $(U^s_{j(k+1)}, \infty)$. We can respectively assign a penalty per shift assignment: $(q^L_{jk} + q^L_{j(k+1)}), q^L_{j(k+1)}, 0, q^U_{jk}$ and $(q^U_{jk} + q^U_{j(k+1)})$.

Now, instead of ordering on the penalty for not making the lower bound, order on the penalties corresponding to the first 2 intervals, and assign nurses accordingly. Then, assign all possible nurses to the middle intervals, and again order the last 2 intervals, starting with assignment to the intervals with lowest penalty.

In the second case, $U^s_{jk} > U^s_{j(k+1)}$. This case is shown in (b) of Figure 1. The intervals are mostly similar, but are now $[0, L^s_{jk}]$, $(L^s_{jk}, L^s_{j(k+1)}]$, $(L^s_{j(k+1)}, U^s_{j(k+1)}]$, $(U^s_{j(k+1)}, U^s_{jk}]$ and (U^s_{jk}, ∞) while the associated penalties per shift assignment are $(q^L_{jk}+q^L_{j(k+1)})$,$q^L_{j(k+1)}$, 0, $q^U_{j(k+1)}$ and $(q^U_{jk}+q^U_{j(k+1)})$. The rest of the algorithm works the same as in the first case.

The algorithm works as fast as that of `sSC`. □

It should be noted that in the first case we implicitly assumed $L^s_{j(k+1)} \leq U^s_{jk}$. If this is not true, then this means that the minimum amount of nurses needed on Sunday is larger than the maximum amount of nurses for Saturday. In this case, it is probably better to change the formulation of the problem. However, the problem is still solvable in polynomial time.

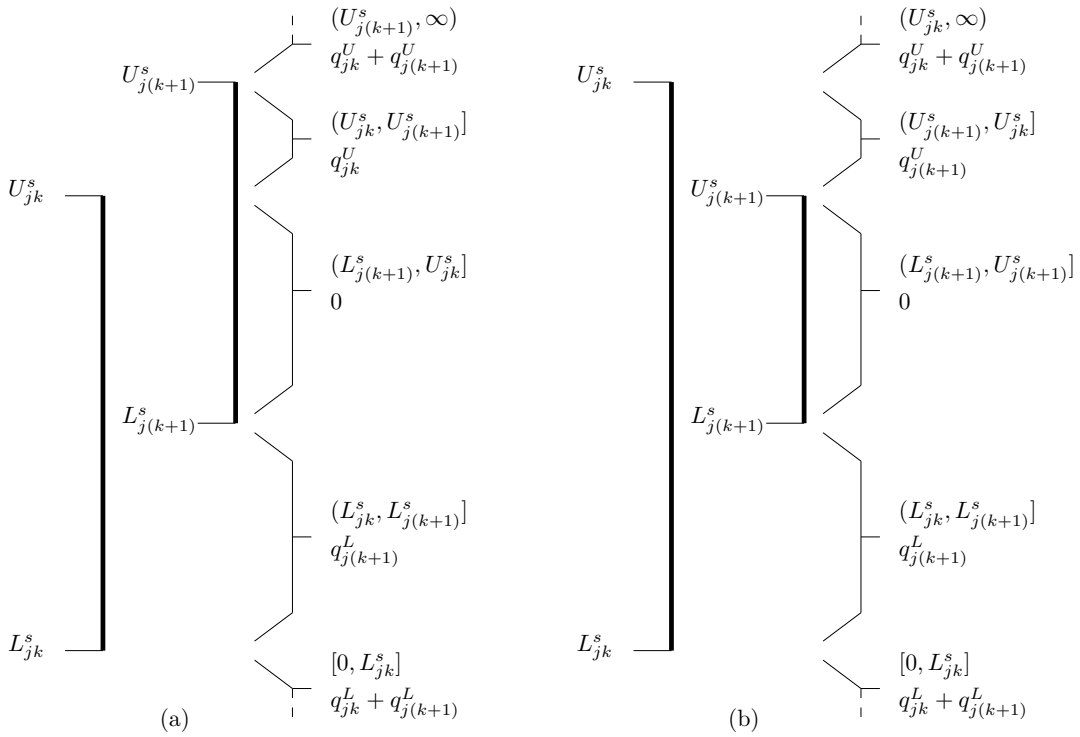

Figure 1: *Both situations for* `sSC-hIW` *are shown here. For each interval, the length of the interval and the associated penalty are listed, with the interval length on top and the penalty on bottom.*

4.7 hSC-hBR-hDOR

Constraints:	• hard `Shift coverage` • hard `Balanced Rosters` • hard `Days-off Requests`
Categorization:	*A(b,e) |NRV |*
Time complexity:	NP-complete

We will now include the `Balanced Roster` constraint, which states that there is a lower bound L^b_{ij} and an upper bound U^b_{ij} for each nurse i and shift j. Over the entire scheduling period, nurse i must work shift j at least L^b_{ij} times and at most U^b_{ij} times.

The proof for this is given in Osogami and Imai (2000)[1] . However, due to notation and implementation differences, some edits need to be made. Furthermore, in Osogami

[1]The proof can be found in Appendix A, which can be found on `https://sites.google.com/site/takayukiosogami/publications-in-english/conference-papers-refereed`

and Imai (2000) every nurse can also request to not work certain shifts on each day. However, this is not necessary for an NP-completeness proof. We will now present a full proof adapted from Osogami and Imai (2000).

Proof. We will perform a reduction from the timetabling (`TT`) problem, which was shown to be NP-complete in Even et al. (1976). We will use the same definition as in Osogami and Imai (2000).

`TT` is defined as follows. We are given a set H of work hours, n teachers and m classes. For each teacher i a subset $T_i \subseteq H$ is given, containing the hours in a week that teacher i can teach. Furthermore, for each class j a $C_j \subseteq H$ is given, which contains the hours in a week in which class j is available for following class. Then for each combination of i and j we are given the positive integer R_{ij}, which is the number of times that teacher i must teach class j during the week.

The timetabling problem is the problem of determining an assignment of teachers to hours and classes, such that the following constraints are met:

1. For each i and j, teacher i gives a lecture to class j exactly R_{ij} times
2. A teacher can give at most one lecture per hour
3. A class can follow at most one lecture each hour
4. If teacher i gives a lecture to class j on hour k, then k must be in both T_i and C_j

We are given an instance of the `TT` problem. To perform the reduction to an instance of the `NSP`, correspond the set of nurses N to the set of teachers and the set of days D to the set of work hours H. Our set of shifts S has $m + 1$ shifts. The first m shifts each correspond to one of the m classes, while the last shift is our `Day Off` shift. In the original problem a teacher does not always have to be assigned a class at an hour, so we must include an extra shift to put all our inactive nurses in: The day-off shift.

For each nurse i and shifts $j \in \{1, ..., m\}$, set both the lower bound L^b_{ij} and upper bound U^b_{ij} to R_{ij}, so that each nurse works each of these shifts exactly R_{ij} times. For the day off, shift $m + 1$, set $L^b_{i(m+1)} = 0$ and $U^b_{i(m+1)} = |D| = |H|$ for each nurse i, so that every nurse can always have a day off.

For each day k and shift j set $L^s_{jk} = 0$. If in the `TT` a class j is unavailable on hour k (or in other words, $k \notin C_j$), set $U^s_{jk} = 0$. Otherwise, set $U^s_{jk} = 1$.

To complete the transformation, if a teacher i is not available during hour k (or in other words, $k \notin T_i$), add a day off request for nurse i on day k.

This completes our reduction. First we will prove that a solution for our reduced problem is a valid solution for the timetabling problem.

Assume that we have a solution for our reduced problem. We will now check all of the constraints we set earlier. First, because we set a lower and upper bound on each

shift and nurse combination, we can make sure that constraint 1 holds. Because each nurse is assigned to exactly one shift each day (*Definition* 1), constraint 2 holds. As each shift that corresponds to a class has an upper bound of at most 1, constraint 3 holds as well. A nurse can only be assigned to a shift that corresponds to a class if it does not have a request for a day off, which implies $k \in T_i$. A shift can only have a nurse assigned to it if the upper bound is larger than 0, and as such if $k \in C_j$. This fulfills the last constraint. Therefore, all four constraints hold, and a solution for our reduced problem is also an admissible solution for the corresponding timetabling problem.

Now we will check whether a solution for the timetabling problem is an admissible solution for our reduced problem. Firstly, because of constraint 2 a teacher can give at most 1 lecture an hour, and if a teacher is not assigned to a class on hour k, then we can assume that he is assigned to the day off shift $m + 1$ on day k. As each teacher could be assigned to the day off shift on every day, we are guaranteed that on each day each teacher is assigned to exactly one shift.

Because of constraint 1, `Balanced Roster` constraints hold. Because of constraint 3, we are guaranteed that on the hours that a class can follow lectures the upper bound of 1 holds on those days. Because of constraint 4, we are both guaranteed that the `Days-off Requests` constraint holds and that the upper bound of 0 holds on days that classes cannot follow lectures. It is trivial that the shift coverage lower bound constraints always hold.

The reduction can be completed in polynomial time. Creating the nurses, shifts and days can be done in linear time, and the upper bounds, lower bounds and days off requests also take linear time.

As a result, *Definition* 1 and all constraints in C hold, so the solution is admissible. It is trivial to show that the problem is in NP. Therefore, C is NP-hard.

□

4.8 hSC-hBR

Constraints:	• hard `Shift Coverage` • hard `Balanced Rosters`
Categorization:	*A(b)* $\|NRV\|$
Time complexity:	NP-complete

Proof. We can show the even stronger result that the problem is NP-complete even without `Days-off Requests`. The most basic timetabling problem is even NP-complete if $T_i = H$ for all i and all $R_{ij} \in \{0, 1\}$((Garey and Johnson, 1979, page 243)). We call this version of timetabling `TTB`.

Since $T_i = H$ for all teachers i, there are no hours k where teacher i is unavailable. Therefore, in our reduced problem of Section 4.7 there are no days where nurse i is unavailable. This means that there would be no days-off requests, and by default the request-set R would be empty. We can conclude that we can choose to get rid of the constraint, and the reduction as outlined in Section 4.7 still holds for `TTB`. As a result, C is NP-hard. □

Since we require at least hard `Shift Coverage` in each problem, we have shown that each `NSP` with `Balanced Rosters` is NP-hard.

Note: In Smet et al. (2014) it is shown that a related subproblem can be solved polynomially. By considering a day off not as a shift but as a lack of assignment, it is possible to put a lower bound and upper bound on the total amount of days worked (or, interpreted differently, a lower bound and upper bound on the day off shift). In this way only one shift can have bounds for the planning period.

Naturally, any shift can be substituted for the day-off shift, so the problem is polynomial if there are bounds for exactly one shift. The problem is open if there are k shifts with bounds, where k is a constant ≥ 2.

4.9 hSSC-hFS=3

Constraints:	• hard `Strict Shift Coverage` • hard `Forbidden Sequences=3`
Categorization:	*S(b,c)|NRV|*
Time complexity:	NP-complete

We will now include the `Forbidden Sequences` constraint. As we use `Forbidden Sequences=3`, we have a set of sequences of length 3, and no personal roster may contain 3 consecutive days on which a sequence of shifts is worked that is equal to a sequence in this set.

Proof. We show a reduction from `3-Dimensional Matching`(Karp (1972)), also referred to as `3DM`. In `3DM`, given a finite set T and a set $U \subseteq T \times T \times T$, the goal is to find a set $W \subseteq U$ such that $|W| = |T|$ and no two elements of W agree in any coordinate. This means that each element of T is chosen exactly once in each coordinate.

We are given an instance of `3DM`. To perform a reduction to an instance of the `NSP`, correspond a shift-set S with set T, giving us $|S| = |T|$. Furthermore, we have $|N| = |S| = |T|$ nurses, and a planning period of $|D| = 3$ days. Each day corresponds to one of the coordinates of `3DM`. For each $j \in S, k \in D$ set $L^s_{jk} = U^s_{jk} = 1$, such that each shift is chosen exactly once on each day. This is also possible because we have exactly $|N| = |S| = |T|$ nurses. Our goal is to find an assignment where each nurse corresponds

to a member of W, and as such each shift must be chosen exactly once by all nurses together.

3DM works with a set of allowed sequences, while set F lists forbidden sequences. We can build the reverse set by first assigning $F' = S \times S \times S$, and then defining $F := \{f | f \in F', f \notin U\}$. This can be found by removing all entries belonging to U from F'. Now all sequences that are forbidden in 3DM are also forbidden in our instance of the NSP.

Now we have a NSP with an amount of nurses equal to the amount of shifts, with all shifts needing to be fulfilled exactly once each day. This means that there will be $|N| = |T|$ personal rosters, and on no day more than one person is assigned to the same shift. Now all personal rosters together form the solution-set W. Since each personal roster is not in F, the solution is valid.

Building the full set of sequences is done in $|S|^3$ time. Even if this set of sequences is searched through, assuming that this set is ordered, this costs at most $|S|^3 \log(|S|)$ time. All other operations are also done in polynomial time, so the reduction is a polynomial-time reduction.

It is not completely trivial that a solution can be checked with respect to feasibility in polynomial time in case of `Forbidden Sequences`. In general, for a forbidden sequence of length n, it takes $O(|D| * n)$ time to check whether a personal roster contains the forbidden sequence. Assuming that n' is the maximum forbidden length, it takes $O(|D| * n' * |F| * |N|)$ time to check all personal rosters for all forbidden sequences. This is still polynomial. All other constraints are trivial.

As C is in NP, C is NP-complete. □

Since `Forbidden Sequences=3` is NP-complete, we consider `Forbidden Sequences=n` to be NP-complete for all $n \geq 3$ with any form of `Shift Coverage` (Section 4.2.3, Lemma 5). This in turn means that `Forbidden Sequences`$\geq$`n` is NP-complete for all n.

4.10 hSC-hFS=2

Constraints:	• hard `Shift Coverage` • hard `Forbidden Sequences=2`
Categorization:	*S(b,c)|NRV |*
Time complexity:	Polynomial

Algorithm. We can solve this problem by building a max-flow network and solving it. Consider the following directed network. We have a source s and a sink t. Add for each day k and shift j a vertex v^1_{jk} and v^2_{jk}. The amount of flow that is sent through these two vertices represents the amount of nurses assigned to the day and shift combination.

For each $j \in S, k \in D$ add an arc from v^1_{jk} to v^2_{jk} with minimal flow L^s_{jk} and maximal flow U^s_{jk}. This enforces that all flow through the combined vertices v^1_{jk} and v^2_{jk} is between the bounds on that shift and day combination.

Then, for each $j, j' \in s$ add an arc from v^2_{jk} to $v^1_{j'(k+1)}$ if $\{j, j'\} \notin F$. In other words, there is only an arc from a day to the following day if the sequence is not forbidden. Since all forbidden sequences are of length 2, we eliminate all forbidden sequences in this way. There are no capacity constraints on these arcs.

Lastly, create a dummy-source s'. Add an arc from s to s' with both minimal and maximal flow $|N|$, to enforce a total of $|N|$ flow through the network. Then, for each shift $j \in s$, add an arc from s' to v^1_{j1} and an arc from $v^2_{j|D|}$ to t.

An admissable flow through this network consists of a series of paths that go from s' to t through $|D|$ vertices. The amount of nurses assigned to each shift j on day k is equal to the amount of flow passing through vertex n_{jk}. The amount of nurses assigned is between the corresponding lower and upper bound, due to the flow constraints on each vertex n_{jk}. Lastly, the maximum flow through the network is equal to $|N|$, due to the maximum flow on the arc $\{s, s'\}$. Therefore there can only be $|N|$ nurses assigned to each day.

We can see that an admissable flow gives a solution as to how many nurses should be assigned to each shift, together with how many nurses assigned to shift j on day k are assigned to shift j' on day $(k+1)$, by looking at $f(n_{jk}, n_{j'(k+1)})$. We can now extract a solution in the following way. Take the maximum flow. Until the network is empty, find any path from s' to t such that the flow through each vertex is strictly positive. Assign this path (and thus, for each day, a shift) to a nurse with no roster assigned, and subtract 1 from the flow through each arc on the path. We can always find $|N|$ paths, as we have set both a minimum and a maximum flow of $|N|$, so we can assign a roster to each nurse.

The flow network has $2 \times |S| \times |D| + 3$ vertices. Since there are only arcs between two subsequent days and the two vertices belonging to a day/shift combination, there are a maximum of $|D| \times |S| + |D| \times |S|^2$ arcs. If a network has $|E|$ vertices and $|V|$ arcs, a maximum flow can be found in $O(|V||E|^2)$ time (Edmonds and Karp (1972)). This network can thus be solved in $O(|D|^3|S|^5)$ time. Lastly, we can extract the full solution from the network in $O(|N||D|)$ time, giving us a running time of $O(|D|^3|S|^5+|N||D|)$. $\square$

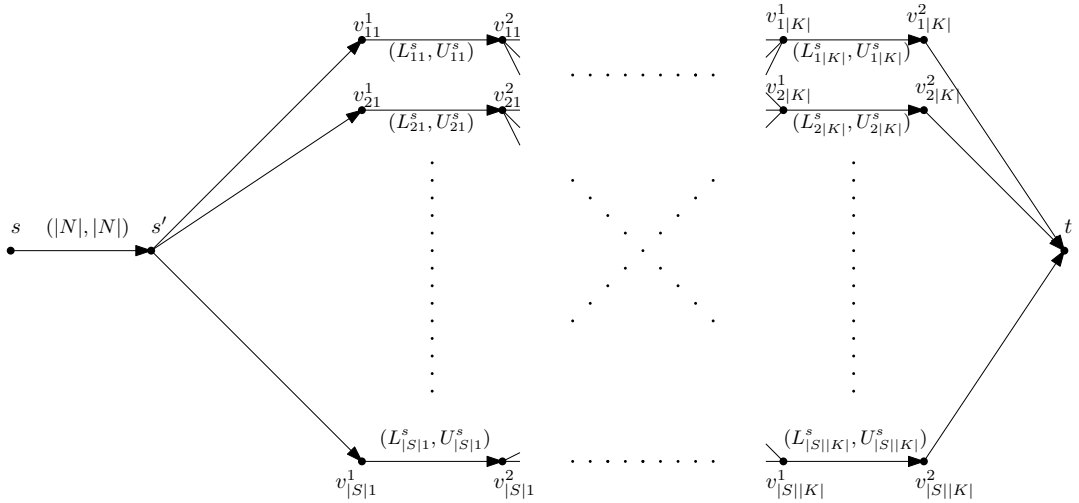

Figure 2: *An example graph for the algorithm of Section 4.10 that shows how to split up vertices. Each arc is labeled (Upper bound, Lower bound).*

4.11 sSC-sFS=2

Constraints:	• soft `Shift Coverage` • soft `Forbidden Sequences=2`
Categorization:	*S(b,c)|NRV |LP*
Time complexity:	Polynomial

Algorithm. The algorithm used for the version with hard constraints (Section 4.10) can easily be edited to accommodate for various penalties by changing it to a min-cost max-flow algorithm, instead of just a max-flow algorithm. Instead of only allowing arcs that are not forbidden sequences, for all $j, j' \in S$ add an arc from v^2_{jk} to $v^1_{j'(k+1)}$ for each k. Then, if $\{j, j'\} \in F$, add the corresponding penalty to arc $\{v^2_{jk}, v^1_{j'(k+1)}\}$ for each day k. If $\{j, j'\} \notin F$, keep the penalty 0.

Previously, we split each vertex v_{jk} into v^1_{jk} and v^2_{jk} to add capacities to the vertex. Now, we go further and change the split some more. This is also shown in Figure 3. For each vertex v_{jk}, perform the following transformation. Split the vertex into v^1_{jk} and v^2_{jk}. Next, add 3 arcs from v^1_{jk} to v^2_{jk} (if needed by adding dummy vertices). On the first arc, put a capacity of L^s_{jk}, and a penalty of $-q^L_{jk}$. On the second arc, put a capacity of $(U^s_{jk} - L^s_{jk})$, with no penalty. On the final arc, put infinite capacity and add a penalty of q^U_{jk}.

It should be noted that instead of giving a penalty for failing to fulfill the lower bounds, we instead give a bonus for assigning a nurse while the lower bound hasn't

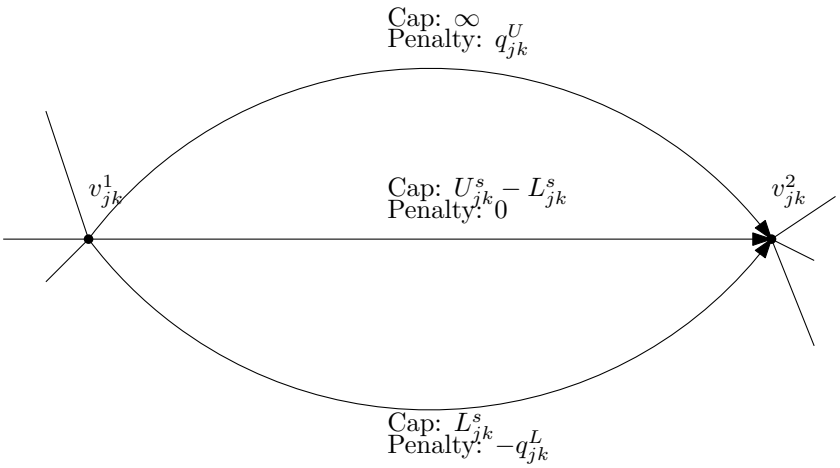

Figure 3: *This figure shows how to split each vertex for the algorithm of Section 4.11 with dummy vertices. Each arc has a capacity and a penalty associated. Any unlabeled arcs within the split vertex are assumed to have infinite capacity and zero penalty.*

been reached yet. Therefore an extra $L^s_{jk} * q^L_{jk}$ should be added for each j and k for an accurate score. However, as this number is a constant, it does not impact ordering of solutions in our minimization problem.

As we solve the problem with a min-cost max-flow algorithm, it is solvable in polynomial time (Edmonds and Karp (1972)). □

4.12 sSC-sFS=2-sIW

Constraints:	• soft `Shift Coverage` • soft `Forbidden Sequences=2` • soft `Identical Weekends`
Categorization:	*S(b,c,d)|NRV |LP*
Time complexity:	Polynomial

Algorithm. The addition of soft `Identical Weekends` can be trivially done by doing the following modification to the algorithm presented in Section 4.11. For each weekend, where the Saturday is referred to as day k, add the corresponding penalty to all arcs $\{v^2_{jk}, v^1_{j'(k+1)}\}$ if $j \neq j'$. □

4.13 hSC-hPR-hSR

Constraints:	• hard `Shift Coverage` • hard `Personal Requests` • hard `Skill Requirements`
Categorization:	*A(b)I* |*NRV* |
Time complexity:	Polynomial

We will now include the `Skill Requirements` for the first time. This constraint assigns a number of skills to each nurse, and on each day/shift combination we can also set a lower bound (and upper bound, if needed) on the amount of nurses that are assigned to each day/shift/skill combination.

Algorithm. This algorithm is presented in Smet et al. (2014). The algorithm also includes an additional preference constraint, which we will simply ignore by setting all preferences to zero. We will now present a slightly modified version of this algorithm.

The problem will be solved with a maximum flow algorithm. An example graph can also be seen in Figure 4. Our graph will roughly exist of 5 parts. First we have a source s and a sink t. Then, for each shift j, day k and skill l we have a vertex m_{jkl}. We can ignore any vertex m_{jkl} for which the upper bound on the skill requirement of skill l on day k and shift l is zero, though it is not needed. We assume that for each combination of j, k and l we have an upper bound U^l_{jkl}, lower bound L^l_{jkl} or both. There is an arc from the source s leading into each vertex m_{jkl}, with these lower and upper bounds.

For each combination of nurse i and day k, we add a vertex v_{ik}. If nurse i has skill l, there is an arc from each m_{jkl} to v_{ik} for each shift j and day k with lower bound 0 and upper bound 1. If nurse i has a personal request for shift j' on day k, instead use an upper bound of 0 on the arcs from m_{jkl} to v_{ik} where $j \neq j'$.

Lastly there is a nurse node o_i for each nurse i. For each nurse i, add an arc from each v_{ik} to o_i, for all days k. These arcs all have lower bound 0 and upper bound 1. This enforces that each nurse is scheduled at most once a day. Lastly, add an arc from each o_i to sink t. If the network has a maximum flow of $|N||K|$, then that is an admissable solution. □

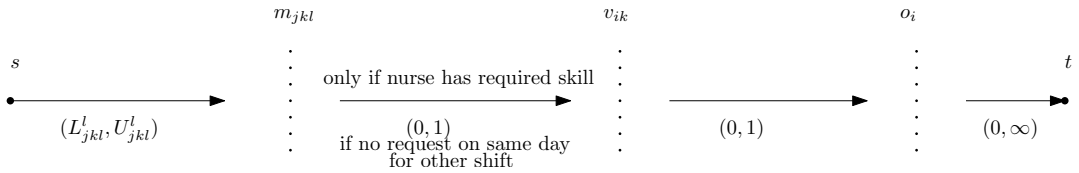

Figure 4: *Modeled after Figure 1 of Smet et al. (2014), this figure shows the structure of the flow network outlined in Section 4.13. Each arc is labeled (lower bound, upper bound).*

4.14 hSC-hSR-hFFW

Constraints:	• hard `Shift Coverage` • hard `Skill Requirements` • hard `Full Free Weekends`
Categorization:	*A(b,e)I|NRV|*
Time complexity:	Polynomial

We will use the `Full Free Weekends` constraint for this problem. This constraint states that each nurse must work an entire weekend, or have the entire weekend off.

Algorithm. Before we begin, we first assume that there are no skill requirements of any kind for the day off shift. Furthermore, as before, we assume that for each combination of j, k and l we have an upper bound, lower bound or both.

It should be clear that on the 5 weekdays, the algorithm is exactly the same as in Section 4.13. Since the only constraint that forms a dependence on different days is the `Full Free Weekends`, we can simply solve the 5 weekdays of a week independently, as in Section 4.13, while we use a different algorithm for the weekends.

As we can solve each weekend independently, we refer to the Saturday of current week as day 1, and we refer to Sunday of the current week as day 2.

Again we will solve this problem with a maximum flow algorithm. The flow network is also shown in Figure 5. First we have the source S and dummy-source S', and sink T. There is an arc from S to S' with both minimum and maximum capacity equal to $|N|$, the amount of nurses. Then, for each shift j and skill l we add a vertex m^1_{jl}, except for the day-off shift. From now on, we assume that the day-off shift is the last shift, and the associated vertex that we add is $m^1_{|J|}$. Add an arc from dummy-source S' to each vertex m^1_{jl} with the related lower and upper bounds on this arc for the Saturday.

We mirror these vertices, and also add a vertex m^2_{jl} for each shift except for the day-off shift, for which we add the vertex $m^2_{|J|}$. Add an arc from each of these vertices to the sink T with the associated lower and upper bounds for the Sunday.

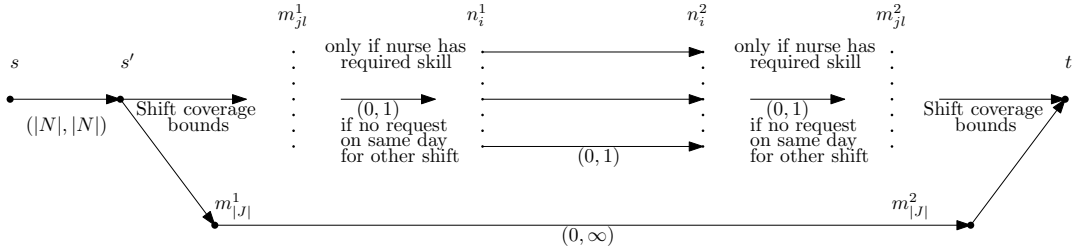

Figure 5: *Flow network for the algorithm in Section 4.14.*

Next, we add 2 vertices for each nurse i: n_i^1 and n_i^2. For each nurse i, add an arc from n_i^1 to n_i^2 with an upper bound of 1, and lower bound 0.

Now we will look at the skill requirements. For each nurse i that has skill l, for each shift j that is not the day off, add an arc from vertex m_{jl}^1 to n_i^1 and from n_i^2 to m_{jl}^2. Lastly, add an arc from $m_{|J|}^1$ to $m_{|J|}^2$.

Because of the way the graph is structured, a flow which passes through a pair of nurse-vertices assigns the shifts that that nurse will work on Sunday and Saturday. After all, if 1 flow reaches a vertex n_i^1, then 1 flow must also reach corresponding vertex n_i^2. Then this flow is sent through to a vertex belonging to a shift worked on Sunday. In the case that no flow is sent through the arc belonging to a nurse, that flow must have been sent through the nodes belonging to the day off: In other words, that nurse has a free weekend.

Because of the bounds on the arc from S to S', the total flow through the network should be exactly equal to the amount of nurses. Therefore, there must be exactly as much flow assigned to the arc between $k_{|J|}^1$ and $k_{|J|}^2$ as there are nurses that are not assigned to any shift. Because of the upper and lower bounds between the sinks/sources and the vertices belonging to each shift and skill combination, the upper and lower bounds belonging to these are also met, fulfilling the `Shift Coverage` constraints.

As the weekdays can all be solved polynomially and the weekends can all be solved with a linear amount of a polynomial time algorithm, the problem can be solved in polynomial time.

It should be noted that this algorithm for the weekends can easily accommodate soft `Shift Coverage` and `Skill Requirements` by using the 'splitting of vertices' tactic described in Section 4.11 on the arcs from S' and to T. It is also easy to add hard or soft `Personal Requests` by modifying the corresponding arcs or putting penalties on them. □

4.15 hSC-hSR-hIW

Constraints:	• hard `Shift Coverage` • hard `Skill Requirements` • hard `Identical Weekends`
Categorization:	$A(b,e)I\|NRV\|$
Time complexity:	Polynomial

As a preliminary, we never defined how the `Skill Requirements` constraint interacts with the `Identical Weekends` constraint. The `Identical Weekends` constraint states that each nurse must work the same shift on both days of the weekend. However, we never stated whether that nurse must also work the same skill on both days of the weekend.

In the case that the nurse does not necessarily have to work the same skill on both days of the week, we have run into a peculiar situation. The resulting problem is not trivial as the skill requirements on the days can vastly differ, but that is certainly odd when we consider what the `Skill Requirement` constraint usually entails. If the constraint is, for example, used to make sure there is at least one head nurse present, then it is very likely that there is at least one head nurse needed for both days. It becomes even more likely that the same skill distribution is needed on both days when we consider that there are exactly as many nurses on Saturday present in each shift as on Sunday (after all, the `Identical Weekends` constraint ensures this). Therefore, we do not consider the problem where the nurses do not have to work the same skill on both days of the week, as any roster where the skill distribution differs too much between Saturday and Sunday should probably be reevaluated.

Algorithm. In the case that the nurse must work the same skill on both days of the week, we can simply merge the two days of the week into one (as was also done in Section 4.6) and solve this with flow as in Section 4.13. Then the algorithm is in polynomial time. □

4.16 hSSC-hFS=2-hDOR

Constraints:	• hard `Strict Shift Coverage` • hard `Forbidden Sequences=2` • hard `Days-Off Requests`
Categorization:	*A(b)S(b,c) |NRV|*
Time complexity:	NP-complete

Proof. The NP-hardness proof for this is shown in Lau (1994). It should be noted that the paper is about cyclic rosters, but as is mentioned there, the proof even holds for non-cyclic rosters with $|D| \geq 5$. Furthermore, with a correct ordering of the shifts, it can be built with a forward rotating roster.

The proof as given in Lau (1994) is a reduction from the normalized `3SAT` problem. The `3SAT` problem is defined as follows. We are given a set of Boolean variables $U = \{U_1, U_2, ..., U_n\}$. We must assign either *True* or *False* to each of these variables. Given a variable X from U, then X and $\overline{X}$ are the literals over U. In other words, a literal is an appearance of a member of U or its negation.

Next, we are given a set of clauses $F = \{C_1, ..., C_m\}$. Each clause has exactly three literals, and a clause is *True* if at least one of its three literals is *True*. Our goal is to assign *True* or *False* to each variable, such that all clauses in F are *True*.

The normalized `3SAT` problem has the additional constraint that each literal U_i must appear exactly the same number of times as its negation $\overline{U_i}$. It is shown in Lau (1994) that normalized `3SAT` is also NP-complete.

Assume an instance F of the normalized `3SAT` problem. F has n variables and m clauses. For each variable U_i, p_i denotes how many times U_i appears (and thus also how many times its negation $\overline{U_i}$ appears). Call P the sum of all p_i.

We create an instance of C with 5 days and $2P$ nurses. The first P nurses will all correspond to the assignment of *True* to a literal, and the last P nurses will correspond to the assignment of *False* to a literal. We will call the first P and the last P nurses correspondingly the *True Region* and the *False Region*.

Next, we define the following $4P + m + 5$ shifts. First, for each occurrence j of variable U_i, we define the shifts x_{ij}, $\overline{x_{ij}}$, v_{ij} and w_{ij}. Shift x_{ij} directly corresponds to the jth occurrence of literal U_i. This means that if a nurse from the *True Region* is assigned shift x_{ij}, literal U_i is assigned *True*, and similarly if shift $\overline{x_{ij}}$ is assigned to a nurse form the *True Region*, literal $\overline{U_i}$ is assigned *True*. For each clause C_k we define the shift c_k. Lastly, we introduce the shifts u, y, z, d and θ. Shift d is the designated day-off shift.

	1	2	3	4	5
1	u	v_{11}	x_{11}	c_1	d
2	u	v_{12}	x_{12}	c_2	d
3	u	v_{21}	$\overline{x_{21}}$	θ	d
4	u	v_{22}	$\overline{x_{22}}$	c_4	d
5	u	v_{31}	x_{31}	θ	d
6	u	v_{32}	x_{32}	c_3	d
7	d	w_{11}	$\overline{x_{12}}$	y	z
8	d	w_{12}	$\overline{x_{11}}$	y	z
9	d	w_{21}	x_{21}	y	z
10	d	w_{22}	x_{22}	y	z
11	d	w_{31}	$\overline{x_{32}}$	y	z
12	d	w_{32}	$\overline{x_{31}}$	y	z

Table 4: *An example showing a filled in example solution of the following normalized* `3SAT` *problem:* $F = (U_1 \vee U_2 \vee U_3) \wedge (U_1 \vee \overline{U_2} \vee \overline{U_3}) \wedge (\overline{U_1} \vee U_2 \vee U_3) \wedge (\overline{U_1} \vee \overline{U_2} \vee \overline{U_3})$.
The example is based on Figure 2 of Lau (1994). The only differences are the addition of the d shift, and the erroneous v shifts of the original example were fixed. Each column corresponds to a day, while each row corresponds to a nurse. The top 6 nurses form the True Region*, while the bottom 6 nurses form the* False Region.
The solution we can extract from the table is the following assignment: U_1=True, U_2=False, U_3=True.

First, we will give a sketch of the reduction. Table 4 contains a filled in example, so the instance can be seen in practice.

The allowed sequences are listed in Table 5. Any combination of two consecutive shifts that is not explicitly mentioned is forbidden.

Furthermore, we will only use a couple of different shifts each day. This is listed in Table 6.

Whenever a shift occurs on a day, we assign it specific `Shift Coverage` bounds to form our instance. These bounds are listed in Table 7. The days-off requests are listed in Table 8.

Our end-goal is to find an assignment of shifts x_{ij} and $\overline{x_{ij}}$ to regions on day 3. From this, we can find out which literals are *True* and which are *False*, by looking at whether the nurses they are assigned to are from the *True Region* or the *False Region*.

The reduction we use is built by two different components: Days 1 and 2 are used to ensure that day 3 corresponds with a valid assignment of literals to booleans, while days 4 and 5 are used to ensure that if the normalized `3SAT` problem F is satisfiable, then the solution will correspond with the assignment on day 3.

First, we will elaborate on how days 1 and 2 are needed to force a correct structure. Afterwards, we will start to prove that C is a valid reduction of F, and in doing so we show the construction of days 4 and 5.

In days 1 and 2, we make sure that all occurrences of the same literal get the same values on day 3. First, we make sure that all nurses in the *True Region* are assigned to

Allowed sequences:
$\{u, v_{ij}\}$
$\{d, w_{ij}\}$
$\{c_k, d\}$
$\{\theta, d\}$
$\{v_{ij}, x_{ij}\}$
$\{v_{ij}, \overline{x_{ij}}\}$
$\{w_{ij}, x_{ij}\}$
$\{w_{ij}, \overline{x_{i,j+1}}\}$
$\{x_{ij}, \theta\}$
$\{\overline{x_{ij}}, \theta\}$
$\{x_{ij}, y\}$
$\{\overline{x_{ij}}, y\}$
$\{y, z\}$
$\{\{x_{ij}, c_k\}\|$ Occurrence j of literal U_i belongs to $C_k\}$
$\{\{\overline{x_{ij}}, c_k\}\|$ Occurrence j of literal $\overline{U_i}$ belongs to $C_k\}$

Table 5: *All sequences are forbidden, with the exception of the sequences listed in this table.*

Day	Occurring shifts
1	u, d
2	v_{ij}, w_{ij}
3	$x_{ij}, \overline{x_{ij}}$
4	c_k, θ, y
5	z, d

Table 6: *Each day, only a couple of shifts can occur in our construction. The other shifts have both an upper bound and a lower bound of 0.*

Shift	Lower Bound	Upper Bound
u	P	P
d	P	P
v_{ij}	1	1
w_{ij}	1	1
x_{ij}	1	1
$\overline{x_{ij}}$	1	1
c_k	1	1
θ	$P - m$	$P - m$
y	P	P
z	P	P

Table 7: *Whenever the shifts appear, these are the lower bounds and upper bounds on the number of nurses that should be assigned to this shift. Remember that P is the total amount of positive literals (and, in effect, also the total amount of negative literals). Furthermore, $|N|$ is equal to $2P$. m is the amount of clauses in our normalized `3SAT` problem.*

Day	Day-off Requests
1	Each nurse in the *False Region*
5	Each nurse in the *True Region*

Table 8: *The given day-off requests for our reduction from normalized `3SAT`*

shift u on day 1 and all nurses in the *False Region* are assigned to shift d on day 1, by filing a request for a day off. We only allow the sequences $\{u, v_{ij}\}$ and $\{d, w_{ij}\}$. As all nurses in the *True Region* are assigned to shift u on day 1, all nurses in the *True Region* are assigned to some shift v_{ij} on day 2, and similarly all nurses in the *False Region* and assigned to some shift w_{ij} on day 2.

Now, for each i, j, we allow the sequences $\{v_{ij}, x_{ij}\}$, $\{v_{ij}, \overline{x_{ij}}\}$ and $\{w_{ij}, x_{ij}\}$. Given the occurrence j of a literal i, define $j+1$ as the next occurrence of literal i, looping around. In other words, if $j = p_i$, then $j + 1 = 1$. Now, we also allow the sequence $\{w_{ij}, \overline{x_{i,j+1}}\}$.

Lemma 7. *Each occurrence of the same literal is assigned to the same region and if shift x_{ij} is in one region, then $\overline{x_{ij}}$ is in the other region.*

Proof of Lemma 7. Assume that we have literal i with p_i occurrences. We will now show that all occurrences of literal i are in the same region, and if x_{ij} is in one region, then $\overline{x_{ij}}$ is in the other. We will refer to a nurse by the name of the shift they were assigned on day 2.

It is easy to see that if x_{ij} is in one region, then $\overline{x_{ij}}$ must be in the other. Assume that x_{ij} is in the *True Region*. Then shift x_{ij} is assigned to nurse v_{ij}. However, the only other nurse that is allowed to work shift $\overline{x_{ij}}$ on day 3 is nurse $w_{i,j-1}$. Here $j-1$ is defined as the occurrence before j, looping around, such that for literal i $w_{i,0} = w_{i,p_i}$. Since $w_{i,j-1}$ belongs to the *False Region*, $\overline{x_{ij}}$ will also belong to the *False Region*.

Now assume that $\overline{x_{ij}}$ is in the *True Region* by being assigned to nurse v_{ij}. The only possible nurse left that can work shift x_{ij} is w_{ij}, which is in the *False Region*.

Next, we will show that all occurrences of literal i are all in the same region. Assume that x_{ij} is in the *True Region* by being assigned to nurse v_{ij}. Then $\overline{x_{ij}}$ is assigned to nurse $w_{i,j-1}$, as no other nurse can work this shift. However, as nurse $w_{i,j-1}$ is already assigned a shift on day 3, shift $x_{i,j-1}$ can not be assigned to nurse $w_{i,j-1}$, and only to nurse $v_{i,j-1}$. Therefore, $v_{i,j-1}$ is also in the *True Region*. This can be applied iteratively to show that if x_{ij} is in the *True Region*, then all other occurrences of literal U_i are also in the *True Region*.

Now, assume that $\overline{x_{ij}}$ is in the *True Region* by being assigned to nurse v_{ij}. Then x_{ij} can only be assigned to nurse w_{ij}. In turn, this means that $\overline{x_{i,j-1}}$ cannot be assigned to nurse w_{ij}, and in the same way it can be iteratively shown that if $\overline{x_{ij}}$ is in the *True Region*, then all other occurrences of literal $\overline{U_i}$ are also in the *True Region*. □

Before we continue, we will show the construction of days 4 and 5. Similar to before, we will assign every nurse in the *True Region* to the day-off shift on day 5 with a request. Then all the nurses in the *False Region* are assigned to z on day 5. As we only allow the sequences $\{c_k, d\}, \{\theta, d\}$ and $\{y, z\}$, this means that all nurses in the *False Region* are assigned to shift y on day 4. We put a lower and upper bound on each clause-shift c_k of 1, so there is exactly one nurse in the *True Region* assigned to each clause c_k on day 4,

and the rest are assigned to shift θ.

Call the set of occurrences of literals $\mathcal{Y} = \{Y_1, Y_2, ..., Y_{2P}\}$, and assume that our clause C_k consists of the Boolean formula $(Y_m \vee Y_n \vee Y_o)$. Then, we allow the sequences $\{Y_m, c_k\}, \{Y_n, c_k\}$ and $\{Y_o, c_k\}$.

We will now prove that C is a valid reduction of F. We will start by showing that a solution of C is also a valid solution for F.

First, we have just shown that a solution for C is at least a feasible assignment of booleans to variables: Every occurrence of the same literal can be found in the same region, and a literal is found in the opposite region of its negation.

Next, we show that a valid solution of C satisfies all clauses of F. We have already decided that we will construct our solution for F by assigning `True` to all the nurses in the `True Region`. Therefore, if we want to show that a valid solution for C satisfies all clauses of F, then we must show that for each clause, at least 1 shift corresponding to a literal from that clause is assigned to a nurse in the *True Region*.

Lemma 8. *For each clause of the normalized* `3SAT` *problem, at least 1 shift corresponding to a literal in that clause belongs to the* True Region

Proof of Lemma 8. As we have just shown in the construction of days 4 and 5, on day 4 each nurse in the *False Region* is assigned to shift y. The nurses in the *True Region* are assigned to shifts c_k and θ. Each shift c_k can be assigned exactly once, and the θ shifts can be assigned to the leftover nurses.

On day 3, the nurses belonging to the *False Region* can be assigned all shifts $x_{ij}, \overline{x_{ij}}$, as we allow sequences $\{x_{ij}, y\}$ and $\{\overline{x_{ij}}, y\}$. Therefore, these nurses can be assigned all occurring shifts on day 3. Furthermore, the nurses that have been assigned to shift θ on day 4 can also be assigned to these same shifts. However, the nurses who have been assigned a shift c_k can only be assigned to shift x_{ij} if clause C_k contains the j-th occurrence of literal U_i, and to shift $\overline{x_{ij}}$ if clause C_k contains the j-th occurrence of literal $\overline{U_i}$. As all shifts c_k can only be assigned to the *True Region*, that means that for each clause k, at least one shift corresponding to the occurrence of one of those literals also belongs to the *True Region*. If this is not possible, there is no valid assignment, as one of the forbidden sequences must be used. □

As a clarifying example of Lemma 8, we will look at Table 4. The first clause in the given example is $(U_1 \vee U_2 \vee U_3)$. The shift belonging to the clause is c_1. The shifts belonging to the literals in the clause are x_{11}, x_{21}, x_{31}. The allowed sequences ending in shift c_1 are $\{x_{11}, c_1\}, \{x_{21}, c_1\}$ and $\{x_{31}, c_1\}$. As we can see, nurse 1 (which belongs to the *True Region*) has been assigned to shift x_{11} on day 3 and c_1 on day 4. As c_1 has already been assigned to a nurse, nurse 5 (who was assigned to shift x_{31}) can no longer be assigned to shift c_1, and is therefore assigned to shift θ.

It should be noticed that we proved something a bit stronger than stated in Lemma 8: Not only should at least 1 shift corresponding to a literal from each clause belong

to the *True Region*, but the exact occurrence of that literal must appear in the *True Region*. However, as has been shown in the proof of Lemma 7, there is no difference between these two statements. After all, if one occurrence of a literal appears in the *True Region*, then all occurrences of that literal appear in the *True Region*.

We can use this result to construct a solution for F from a solution to C. On day 3, look at which nurse is assigned to the first occurrence of each positive literal. x_{i1} is assigned to a nurse in the *True Region*, assign $X_i = True$, and if x_{i1} is assigned to a nurse in the *False Region*, assign $X_i = False$. It follows from Lemma 8 that at least one literal belonging to each clause has been assigned *True*, so each clause has been satisfied. Therefore this is a valid solution to F.

We will now show that a solution of the normalized `3SAT` problem F can be transformed into at least 1 solution for our instance C. We are given an assignment of Boolean values to each variable. First, we fill in days 1 and 5 in the only way possible: On day 1 each nurse belonging to the *False Region* is assigned to the day-off shift d (see also Table 8), and on day 5 each nurse belonging to the *True Region* is assigned to the day-off shift d.

Next, we fill in day 4. Each nurse belonging to the *False Region* is assigned to shift y. Then, we start assigning the shifts c_k. We assign shift c_1 to nurse 1, shift c_2 to nurse 2, and so on, until we assigned shift c_k to nurse k. Note that this can always be done, as $P > k$.

Next, we will fill in day 3. We go through each of the clauses in order, starting with the first clause. For each literal we keep track of how many times they have appeared. For each clause k we do the following:

Go in order through the literals. The first literal that is *True* is assigned to nurse k (which was assigned to shift c_k on day 4). That is, if clause C_k contains a literal U_i, which is the j-th appearance, then nurse k is assigned shift x_{ij} on day 3. Similarly, if clause C_k would have literal $\overline{U_i}$ as its first *True* literal (which means that variable X_i is *False*), then nurse k is assigned shift $\overline{x_{ij}}$ on day 3. Now Lemma 8 holds for our solution. Then, if there is any other *True* literal left, assign one of the nurses numbered from $k+1$ to $P-k$ the corresponding shift. All literals that are *False* are assigned to a nurse belonging to the *False Region*.
We have found a valid assignment of shifts to nurses on day 3. Furthermore, each literal that was assigned *True* is now assigned to one of the nurses belonging to the *True Region*, and each literal that was assigned *False* is now assigned to one of the nurses belonging to the *False Region*. We can see from the proof in Lemma 7 that there is now exactly one valid assignment for day 2 left. If this is filled in, we have a valid assignment of shifts to nurses for our instance of C, and the reduction holds this way.

We have now shown that a valid solution for C can be transformed into a valid solution for F, and a valid solution for F can be transformed into at least 1 valid solution of C. Therefore, C is NP-hard. As it is trivial to show that this belongs to NP, C is NP-complete. □

4.17 hSSC-hFS=2-hSR

Constraints:	• hard `Strict Shift Coverage` • hard `Forbidden Sequences=2` • hard `Skill Requirements`
Categorization:	*IS(b,c)* |*NRV* |
Time complexity:	NP-complete

Proof. We can use the same NP-hardness proof as the one shown in Lau (1994). To most shifts, assign only one basic skill that every nurse knows. However, for the shift u create a skill that is only assigned to the nurses that should start in the *True Region*, while for shift z create a skill that is only assigned to the nurses that should end in the *False Region*. That way the beginning and end positions are still forced, without needing requests. □

4.18 Effect of adding `Weekends` constraints

As promised, we will now discuss the interaction of any of the `Weekends` constraints with Lemma 1 from Section 1. As was mentioned earlier, we cannot automatically guarantee that a proof of NP-hardness still works if we add an additional `Weekends` constraint. As such, we will take a look at the proofs of NP-complete instances that have been given so far, and see how the addition of any of the `Weekends` constraints change it.

Lemma 9 (Continuation of Lemma 1)**.** *Consider the `NSP` with constraint-set C. Given any constraint c' not in C, where c' is not the `Strict Shift Coverage` constraint. If C is NP-complete, then so is the problem with both C and c'.*

Proof. We have already shown that the lemma already holds for all constraints other than `Weekends` constraints in Lemma 1. We cannot show that Lemma 9 holds for the `Weekends` constraints for all NP-completeness proofs of all possible constraints, but only that it holds for the NP-completeness proofs given in this paper, for the constraints introduced in this paper. We will do this by showing how to edit the given reductions, so that they still hold, but also include one of the two `Weekends` constraints.

- hSC-hBR-hDOR (Section 4.7, page 24): During each weekend, insert two dummy days where everybody must work shift w. These do not impact the original problem.
- hSC-hBR (Section 4.8, page 26): The same method can be used.
- hSSC-hFS=3 (Section 4.9, page 27): The proof only uses 3 consecutive days. With little work, these can be put anywhere in a 5-day workweek and surrounded by dummy days where everybody is forced to be assigned to a day off.

- hSSC-hFS=2-hDOR (Section 4.16, page 36): The proof only uses 5 consecutive days. Use these as the 5-days workweek. The weekends surrounding these can be forcefully assigned to a day off.
- hSSC-hFS=2-hSR (Section 4.17, page 42): The same method can be used.

□

5 NSP with a constant amount of shifts

5.1 Introduction

A common feature of NP-hardness proofs given so far is that the amount of days is limited, while the amount of shifts is linear with the amount of nurses. All proofs of NP-hardness assign at most one nurse to most shifts. However, in real life problems it is very likely that the amount of shifts is actually limited. For example, consider the generic shift-set {`Early`, `Late`, `Night`, `Day-off`} introduced in Section 3.2. This is a set of shifts often used in literature and yet it only consists of 4 shifts. Most of our proofs prove NP-completeness by scaling exponentially among nurses and shifts, while keeping the amount of days small. Instead, we should aspire to find proofs that model the real situation closer: Keep the amount of shifts small, and prove that the problem is NP-complete scaled in days and nurses.

We will now discuss a problem where we assume that the amount of shifts is bounded by a constant k. We will refer to this new constraint as `k Shifts`. It belongs to the β category of the `NSP`-categorization introduced in Section 3.1, and is referred to as the k category, which replaces the usual N category.

Our problem has the following constraint-set C:

Constraints:	• hard `Shift Coverage` • hard `Forbidden Sequences=2` • hard `Personal Requests` • `k shifts`
Categorization:	*A(b)S(b,c) |kRV|*
Time complexity:	NP-complete (Section 5.2.1)

We have chosen to take a closer look at the aforementioned constraint-set C for two reasons. Firstly, there are some very important real-life issues (such as the forward rotating roster introduced in Section 3.5) that can be modelled with these constraints. Secondly, we deemed this constraint-set to be a very useful start for a two-phase decomposition algorithm, if it were polynomially solvable. However, as we will now see, constraint-set C is also NP-complete.

5.2 Proof of NP-completeness

5.2.1 Proof for k=5

Proof. We will prove that C is NP-complete for the case where $k = 5$ shifts; this proof is due to van der Zanden [Zanden (2016)]. It is trivial to show that if this holds, then C

Allowed sequences:		
	$\{1,1\}$	$\{1,2\}$
$\{2,1\}$	$\{2,2\}$	$\{2,3\}$
$\{3,2\}$	$\{3,3\}$	$\{3,4\}$
$\{4,3\}$	$\{4,4\}$	$\{4,5\}$
$\{5,4\}$	$\{5,5\}$	

Table 9: *All sequences are forbidden, with the exception of the sequences listed in this table.*

is NP-complete for any $k \geq 5$.

We will perform a reduction from the `Independent Set` problem. This problem is defined as follows. Given an undirected graph $G = (V, E)$ and a constant p, is there a subset V' of V such that $|V'| = p$ and no two elements of V' are connected by an edge in E?

As our reduction, we will consider the case with 5 shifts, numbered 1 to 5. If a nurse works shift n, that nurse can only work shifts $n-1, n$ and $n+1$ the following day. This does not loop around. The set of allowed sequences can also be found in Table 9. Our set of nurses corresponds to the set of vertices V.

The general outline of our reduction will be to force a total of p nurses to work shift 1 on day 1, and $|N| - p$ nurses to work shift 5 on day 1. The p nurses assigned to shift 1 will then form the set V'. Then, for each edge belonging to graph G, we will introduce a pattern of bounds and requests that can only be completed if the p nurses assigned to shift 1 form an independent set in the original graph problem. This will be done by introducing subroutine `Independence Test` and applying this for every edge in E. Each `Independence Test` will take 9 consecutive days to perform the subroutine, and the repeated usage of this test will form our planning period.

We want our subroutine `Independence Test` to adhere to the following constraints when testing edge (V_1, V_2):

1. (Immutability constraint) The nurses assigned to shift 1 must be exactly the same at the beginning and end of the subroutine. Similarly, the nurses assigned to shift 5 must also remain the same

2. The subroutine does not have a valid solution if and only if both V_1 and V_2 belong to V'

If our subroutine fulfills these 2 constraints, then we can run it for each edge in E in sequence, and the end result is an instance of C which only has a solution if and only if V' is an independent set of size p with edges E.

We will now introduce subroutine `Independence Test` by listing the shift coverage constraints and personal requests of each day of the subroutine. If a shift is not listed on a specific day, the minimum and maximum are both 0 for that day. This information is also shown in Figure 6

`Independence test`:

On day 1, we need exactly p nurses in shift 1, and exactly $|N| - p$ nurses in shift 5.

On day 2, we need exactly $p-1$ nurses in shift 1, exactly 1 nurse in shift 2, and again $|N| - p$ nurses in shift 5.

On day 3, we need $p-1$ nurses in shift 1, 1 nurse in shift 3, and again $|N| - p$ nurses in shift 5.

On day 4, we need $p-1$ nurses in shift 1, a minimum of 0 and a maximum of 1 nurse in shift 3, 1 nurse in shift 4, a minimum of $|N| - p - 1$ and a maximum of $|N| - p$ nurses in shift 5. Furthermore, file a personal request for nurse V_1 on shift 4.

On day 5,we need $p - 1$ nurses in shift 1, 1 nurse in shift 3, and $|N| - p$ nurses in shift 5.

On day 6, we need $p-1$ nurses in shift 1, a minimum of 0 and a maximum of 1 nurse in shift 3, 1 nurse in shift 4, a minimum of $|N| - p - 1$ and a maximum of $|N| - p$ nurses in shift 5. Furthermore, file a personal request for nurse V_2 on shift 4.

On day 7, repeat day 3.

On day 8, repeat day 2.

On day 9, repeat day 1.

As can be seen more clearly in Figure 6, one nurse from V' 'travels' to multiple shifts to reach shift 3, where it is then tested whether that nurse corresponds to V_1 or V_2. After the tests are done, the nurse 'travels' back to shift 1. We will refer to this specific nurse as the *traveling nurse*.

We will now look at each possible situation, and show that `Independence Test` only has no valid solution if and only if nurses V_1 and V_2 both start in shift 1.

We will start with exactly that situation. Consider the beginning situation where nurses V_1 and V_2 start in shift 1, and belong to V'. Assume without loss of generality that our traveling nurse is nurse V_1. It travels to shift 3 on day 3, and then on day 4 it is assigned to shift 4, as there is a personal request for nurse V_1 to be assigned shift 4 on day 4. Then it moves to shift 3 on day 5, as it is the only nurse that can be assigned to that shift, and it has a lower bound of 1 on that day.

On day 6, there is a personal request for V_2 to be assigned to shift 4. The only shifts from which a nurse can 'move' to shift 4 is 3, 4 or 5. Nurse V_2 is still assigned to shift 1, so there is no valid solution. It should be noted that the same would happen if V_2 were the traveling nurse, as `Independence Test` is symmetric in day 5.

Next, we will show that there are only valid solutions that adhere to the immutability constraint in all other cases. First, we start with both V_1 and V_2 not in V'. That means that they start out in shift 5. The traveling nurse is some random nurse V_3 from V'. It is assigned to shift 3 on day 3. On day 4, V_1 moves from shift 5 to shift 4 to fulfill the personal request, while V_3 stays in shift 3. On day 5, nurse V_3 can only be assigned to shift 3, as there is no other legal move. Therefore, V_1 moves back to shift 5.

On day 6, V_3 again stays at shift 3, and V_2 moves to shift 4 to fulfill the personal

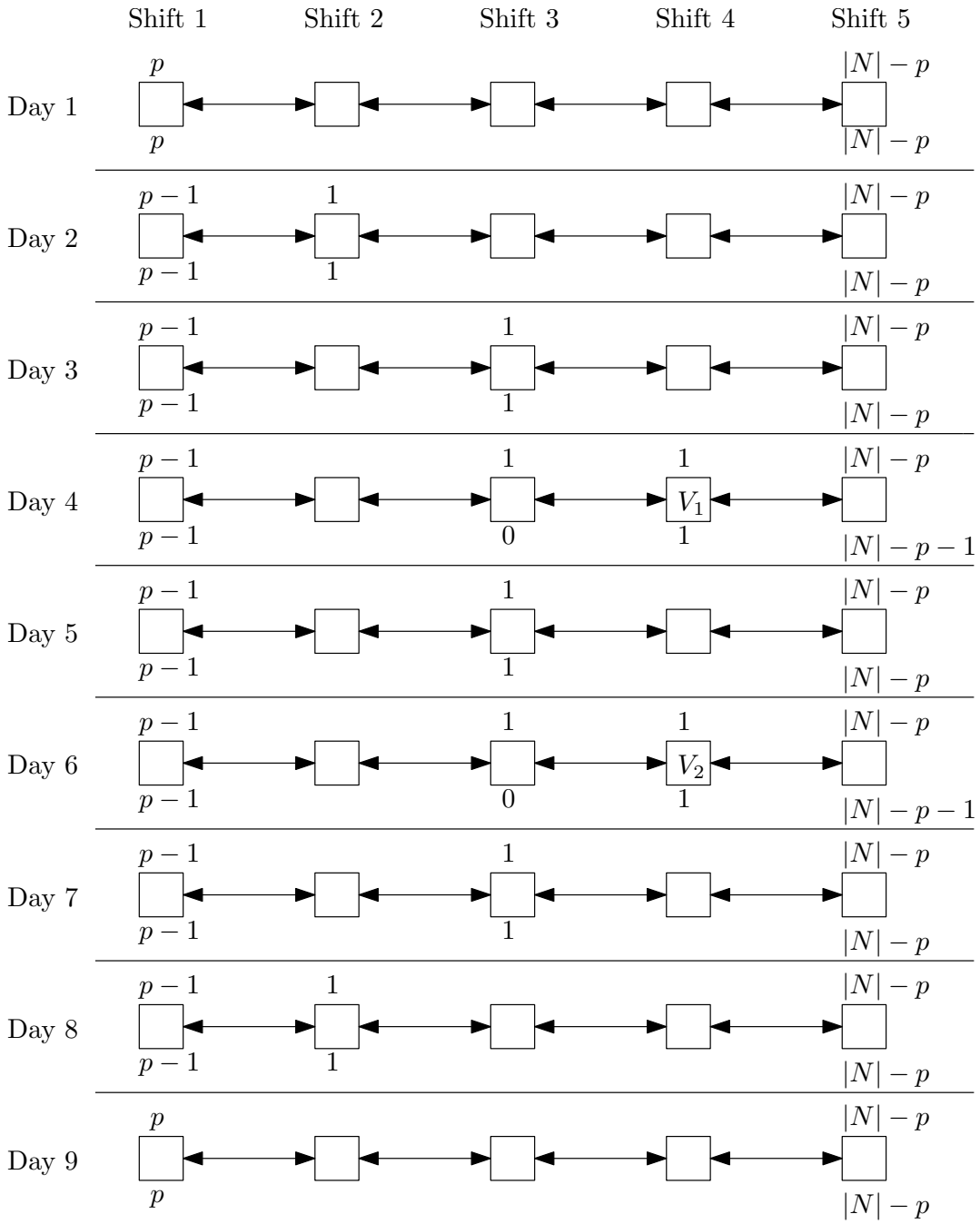

Figure 6: *Graphical view of* `Independence Test`*. Each box represents a shift on a specific day. Above the box is the corresponding upper bound and below the box is the corresponding lower bound. Inside the box, the nurse who requested that day/shift combination is listed.*

request. Then, like before, nurse V_3 stays assigned to shift 3, as there is no other legal move, while V_2 moves back to shift 5. Then V_3 can only go back to shift 1, and we have a valid solution that adheres to the immutability constraint. All other possible solutions just have a different traveling nurse, but they still adhere to the immutability constraint.

Lastly, we show the only valid solution for when V_1 belongs to V', and V_2 doesn't. Our traveling nurse is nurse V_1, which ends up at shift 3 on day 3. It moves to shift 4 to fulfill the personal request, and then to shift 3 on day 5 to fulfill the lower bound of 1. On day 6, it stays on shift 3, while V_2 fulfills their personal request at shift 4, moving there from shift 5. Then V_2 moves back to shift 5, while the traveling nurse V_1 stays on shift 3, as it is the only legal move. Afterwards it returns home to shift 1. This unique valid solution again adheres to constraint 1. Furthermore, as `Independence Test` is symmetric in day 5, this same solution can be used in reverse if V_2 belongs to V' and V_1 doesn't, by simply making V_2 the traveling nurse.

Now that we have shown that `Independence Test` satisfies the 2 constraints, we formulate the entire instance. Number the edges in $E = (e_1, e_2, ..., e_{|E|})$. Create a planning period with $|D| = 9 \times |E|$ days. The first 9 days correspond to the `Independence Test` belonging to edge e_1, complete with all shift coverage and request constraints. The next 9 days correspond to the `Independence Test` belonging to edge e_2, etcetera. This continues until all edges have been handled in the $9 \times |E|$ days, and our instance is complete.

If there is a valid solution V' for the `Independent Set` problem, there is a valid solution for our instance of C by assigning all vertices belonging to V' to shift 1 on day 1. If there is a valid solution for our instance of C, we can find a valid solution for the `Independent Set` problem by calling the group of nurses assigned to shift 1 on day 1 our set V'. The reduction can also be done in linear time, so we have formed a valid polynomial time reduction and have shown that C is NP-hard. If we have a solution we can check in linear time whether it is a valid solution for our problem, so it is also NP-complete. □

5.2.2 Proof for k=4

We have shown proof of NP-completeness for the case with 5 shifts. However, the smallest practical set of shifts has only 4 shifts. We have also found a proof that shows NP-completeness for the smallest practical case, 4 shifts.

Proof. We will prove that C is NP-complete for the case where $k = 4$ shifts in Proof 5.2.1. Since 4 shifts is the smallest possible problem, we therefore show that C is NP-complete.

We will prove NP-completeness in the same way as we did for $k = 4$. We perform a reduction from the `Independent Set` problem introduced in Section 5.2.1. We define a new problem with a new subroutine, the `Revised Independence Test`. Like before,

we perform this test on each edge in the set E in sequence.

As our reduction, we now use 4 shifts. These are labeled $\{1, 2, 3, d\}$, where d is the day-off shift. If a nurse is assigned to a numbered shift, on the next day it can only be assigned to either that same shift or the day-off shift. If a nurse is assigned to the day-off shift, it can be assigned to any other shift on the following day. In other words, the forbidden sequences are as follows: $F = \{\{1, 2\}, \{1, 3\}, \{2, 1\}, \{2, 3\}, \{3, 1\}, \{3, 2\}\}$. This can also be seen in Figure 7.

Our set of nurses corresponds to the set of vertices V again. This time, the p nurses that correspond to the set of vertices V' start in shift 1, while the $|N| - p$ nurses not in V' start in shift 3. We want our new subroutine, `Revised Independence Test` to adhere to the following constraints as `Independence Test`. That is, while testing edge (V_1, V_2) the following constraints should hold for `Revised Independence Test`:

1. (Immutability constraint) The nurses assigned to shift 1 must be exactly the same at the beginning and end of the subroutine. Similarly, the nurses assigned to shift 3 must also remain the same

2. The subroutine does not have a valid solution if and only if both V_1 and V_2 belong to V'

We will now introduce subroutine `Revised Independence Test` by listing the shift coverage constraints and personal requests of each day of the subroutine. If a shift is not listed on a specific day, the minimum and maximum are both 0 for that day. This information is also shown in Figure 7.

`Revised Independence Test`:

On day 1, we need exactly p nurses in shift 1, and exactly $|N| - p$ nurses in shift 3.

On day 2, we need $p-1$ nurses in shift 1, 1 nurse in shift d, and again exactly $|N|-p$ nurses in shift 3.

On day 3, we need $p-1$ nurses in shift 1, 1 nurse in shift 2, and again exactly $|N|-p$ nurses in shift 3.

On day 4, we need $p-1$ nurses in shift 1, a minimum of 0 and a maximum of 1 nurse in shift 2, and now a minimum of $|N| - p - 1$ nurses and a maximum of $|N| - p$ nurses in shift 3. We need 1 nurse in shift d, and file a personal request for nurse V_1 on shift d as well.

On day 5, we need $p-1$ nurses in shift 1 again, exactly 1 nurse in shift 2, and exactly $|N| - p$ nurses in shift 3.

On day 6, we need $p-1$ nurses in shift 1, a minimum of 0 and a maximum of 1 nurse in shift 2, and now a minimum of $|N| - p - 1$ nurses and a maximum of $|N| - p$ nurses in shift 3. Weneed 1 nurse in shift d, and this time file a personal request for nurse V_2 on shift d.

On day 7, we need exactly $p - 1$ nurses in shift 1 again, exactly 1 nurse in shift 2, and exactly $|N| - p$ nurses in shift 3.

On day 8, repeat day 2.

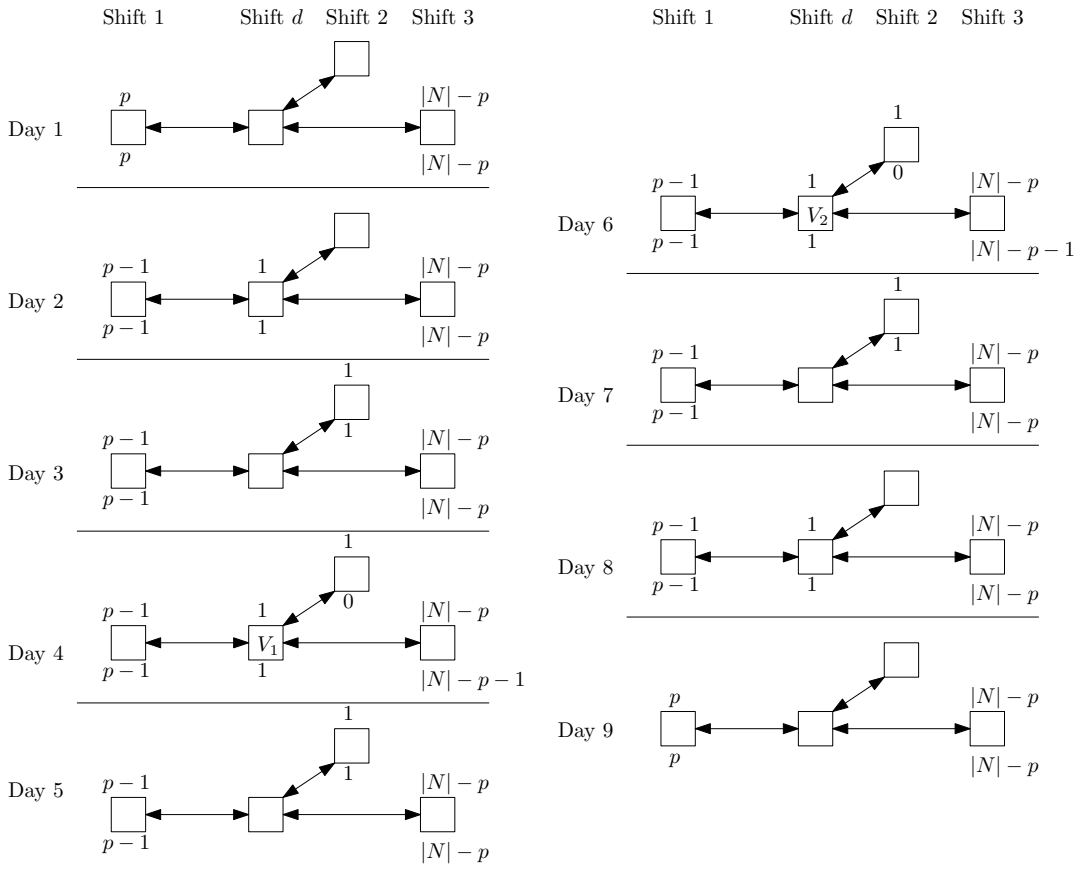

Figure 7: *Graphical view of* `Revised Independence Test`*. Each box represents a shift on a specific day. Above the box is the corresponding upper bound and below the box is the corresponding lower bound. Inside the box, the nurse who requested that day/shift combination is listed. Arrows represent allowed sequences.*

On day 9, repeat day 1.

Like in `Independence Test`, we again have a nurse 'traveling', this time from shift 1 to shift 2 through shift d. We again refer to this nurse as the *traveling nurse*. We will now look at each possible situation, and show that `Revised Independence Test` has no solution if and only if V_1 and V_2 both belong to V', and that the immutability constraint holds for each valid solution.

We will start by proving that there is no solution if V_1 and V_2 both belong to V'. Remember that all nurses belonging to V' start in shift 1, while all nurses not belonging to V' start in shift 3. Therefore, on day 1, V_1 and V_2 are assigned shift 1.

Assume without loss of generality that our traveling nurse is V_1. It travels to shift d on day 2. It should be pointed out that the traveling nurse can only belong to shift 1 on the first day. After all, both the lower and upper bound drop by 1 from day 1 to day 2, which means that one nurse must be thrown out. Furthermore, the lower bound

on shift 3 stays the same, so no nurse that was assigned to shift 3 on day 1 can possibly work any other shift on day 2.

On day 3, the only valid move is to assign the traveling nurse to shift 2. The other nurses have the same shifts assigned to them as on previous days.

On day 4, there is a request for nurse V_1 on shift d. Therefore, we move the traveling nurse V_1 to shift d. As the maximum amount of nurses on shift d is 1, it is impossible for any nurse in shift 3 to move there, even though the lower bound on shift 3 drops by 1.

On day 5, there is a minimum of 1 nurse at shift 2. Therefore, our traveling nurse V_1 moves there.

On day 6, there is a request for nurse V_2 on shift d. Nurse V_2 was in shift 1 on day 5. However, it cannot possibly move to shift d on day 6: The upper and lower bounds on shift 1 stay the same, so nurse V_2 moving to shift d would violate the shift coverage constraints. The only way in which V_2 could leave shift 1 is if another nurse took its place. However, there are no nurses in shift d on day 5, so there is no possible nurse that could move to shift 1 on day 6. Therefore, there is no valid solution. Again, it should be noted that the same would happen if V_2 were the traveling nurse, as `Revised Independence Test` is symmetric in day 5.

We will now show that there is always a solution if V_1 and V_2 do not both belong to V', and that that solution does not violate the immutability constraint.

We will start with the case where V_1 and V_2 both start out in shift 3. Our traveling nurse V_3, which is any random member from V', moves to shift 2 on day 3. On day 4, it stays assigned to shift 2 while V_1 moves from shift 3 to shift d to fulfill the personal request. On day 5, the lower bound on shift 3 ensures that V_1 must move back to shift 3, while V_3 stays assigned to shift 2.

Then, on day 6, V_2 moves from shift 3 to shift d to fulfill the personal request while V_3 stays assigned to shift 2. V_2 must move back to shift 3 again on day 7 to satisfy the lower bound on shift 3. Then the traveling nurse V_3 moves back to shift 1 on the following days. Therefore, V_3 is again back at shift 1, and V_1 and V_2 are back at shift 3. All other possible solutions have a different traveling nurse, but all of them adhere to the immutability constraint.

Next, we will look at the case where V_1 belongs to V', and V_2 does not. As before, this also shows the case where V_2 belongs to V' while V_1 does not, as `Revised Independence Test` is symmetric in day 5.

Clearly, our traveling nurse is V_1. It can only move to shift 2 on day 3, as always. Then it moves down to shift d to fulfill the personal request, and up again to shift 2 on day 5 to satisfy the lower bound of 1. On day 6, V_2 moves from shift 3 to shift d to fulfill the personal request, while V_1 stays assigned to shift 2. V_2 moves back to shift 3, while V_1 stays in shift 2 on day 7, after which it moves back to shift 1. Again the immutability constraint is satisfied. Now we have shown that for every possible situation there is a valid solution if and only V_1 and V_2 do not both belong to V', while the immutability

constraint is satisfied for all possible valid solutions.

We can finish our reduction by formulating the entire instance. We are given an instance of the `Independent Set` problem. As in the proof of Section 5.2.1, number the edges $E = (e_1, e_2, ..., e_{|E|})$. Create a planning period with $|D| = 9 \times |E|$ days. The first 9 days correspond to the `Revised Independence Test` belonging to edge e_1, complete with all shift coverage and request constraints. The next 9 days correspond to the `Revised Independence Test` belonging to edge e_2, etcetera. This continues until we have test all edges in $9 \times |E|$ days. This complete our instance.

If there is a valid solution V' for the `Independent Set` problem, there is a valid solution for our instance of C by assigning all nurses corresponding to vertices belonging to V' to shift 1 on day 1. If there is a valid solution for our instance of C, we can extract a valid solution for `Independent Set` by checking which nurses are assigned to shift 1 on day 1, and calling the set of corresponding vertices our set V'. This reduction can be done in linear time, so we have formed a valid polynomial time reduction and have shown that C is NP-hard. As the solution can be validated in linear time, it is also NP-complete. □

5.2.3 Miscellaneous results

We have now shown that C is NP-complete. However, we have actually shown something far stronger: We have shown C to be NP-complete with `Days-off Requests` instead of `Personal Requests`, as all requests were filed on the shift d. In `Independence Test` (Section 5.2.1) we actually also filed all requests on the same shift, shift 4. Therefore we could also consider shift 4 to the be day-off shift.

Furthermore, in the proof of Section 5.2.2, we have shown C to be NP-complete with a forward rotating roster. This term was first introduced in Section 3.5. A forward rotating roster is defined as a roster where a nurse cannot be assigned to a shift that starts earlier than the starting time of his shift on the previous day. It should be clear that this holds for the instance used in `Revised Independence Test` (Section 5.2.2): After all, a nurse cannot be assigned to any working shift other than the one he worked the previous day!

However, as there is no single allowed sequence where a nurse works a shift that starts strictly *later* than the shift he worked the previous day, this claim is diminished by a bit. It can be checked that removing any of the forbidden sequences would invalidate `Revised Independence Test`, so the reduction would no longer hold. Nonetheless, the instance in the proof of Section 5.2.2 still satisfies the forward rotating roster definition.

5.3 Miscellaneous subproblems

5.3.1 Fixed amount of nurses

Consider the subproblem of C where we assume that not only the shifts are limited by a constant k, but the amount of nurses is also limited by a constant k'. Refer to this problem as C'. In other words, C' is defined as the following constraint-set:

Constraints:	• hard `Shift Coverage`
	• hard `Forbidden Sequences=2`
	• hard `Personal Requests`
	• `k shifts`
	• `k' nurses`

C' can be solved in polynomial time with dynamic programming. We consider our problem as an assignment problem over the days $t = 0, 1, \ldots, |D|$. On each day t, we must assign a shift to each nurse. We call such an assignment p_t.

For each day, define the set Θ_t. This set consists of all valid assignments p_t. This means that all $p_t \in \Theta_t$ satisfy the constraints of C'. Personal requests up to day t must be fulfilled, but also none of the forbidden sequences may be present. Therefore, it is clear that the state Θ_t must be dependent on Θ_{t-1}. Since we actually need no more information to calculate Θ_t, we can define a function that is only dependent on Θ_{t-1}.

First, define the subset $v(p_{t-1}) := \{p_t | p_t \text{ is a valid successor assignment of } p_{t-1}\}$. This subset can be calculated naively by enumerating all possible assignments in $O(|S|^{|N|}) = O(k^{k'})$ time. Note that under the assumption that k and k' are constants, this is constant time.

$v(p_1)$ is equal to all possible assignments where `Shift Coverage` and `Personal Requests` hold, as there is no preceding day.

Next, define $\Theta_t = \bigcup_{p_{t-1} \in \Theta_{t-1}} v(p_{t-1})$. Since there can be up to $k^{k'}$ valid assignments at a time, Θ_t can be calculated in $O(k^{k'} k^{k'}) = O(k^{2k'})$ time. By keeping track of which p_{k-1} are succeeded by p_k, we can then find a valid assignment $\tilde{p}$ in $O(|D| * k^{2k'})$ time. Since we consider k and k' to be constant, this is linear in $|D|$.

So far we have not actually used dynamic programming a lot. If any soft constraint is used, consider the penalty function $\tilde{c}(\tilde{p}_t)$ as the penalty over days $1 \ldots t$. Instead of just saving the possible assignments p_k, we can also save the total minimum value $\tilde{c}(\tilde{p}_t)$ of all assignments ending in p_t. Then we can pick the assignment $p_{|N|}$ with lowest penalty $\tilde{c}(\tilde{p}_{|N|})$. This gives us the optimal result.

While the algorithm is linear in $|D|$, the constant factor is likely to be huge. Nonetheless, this could be used in smaller instances.

5.3.2 Fixed amount of days

We can consider a similar subproblem, one where instead of a limited amount of nurses, we consider a limited amount of days. Limit the amount of days by a constant $\bar{k}$, and refer to this problem as $\bar{C}$. In other words, $\bar{C}$ is defined as the following constraint-set:

Constraints:	• hard `Shift Coverage`
	• hard `Forbidden Sequences=2`
	• hard `Personal Requests`
	• `k shifts`
	• $\bar{C}$ `days`

We start out analogously to the dynamic programming algorithm introduced in Section 5.3.1. We consider our problem as an assignment problem over the nurses $u = 1, 2, \ldots, |N|$. For each nurse u, we must assign a shift to each day. We will start out with the situation with only 1 nurse, and add one additional nurse in each step of the dynamic programming.

For each nurse u, define the assignment p_u as the number of nurses, up to and including nurse u, assigned to each shift on each day. Call p^u_{jz} the amount of nurses assigned to shift j on day z in assignment p_u. Then $\sum_j p^u_{jz} = u$ for each day $z \in D$.

We can define the subset $w(p_{u-1}) := \{p_u | p_u \text{ is a sufficient successor assignment of } p_{u-1}\}$. We call p_u a sufficient successor of p_{u-1} if there exists a personal roster for nurse u that can be added to p_{u-1} to form p_u . This personal roster must satisfy any personal requests from nurse u and `Forbidden Sequences`. Furthermore, p_u must conform to the upper bounds of `Shift Coverage`. p_u is called sufficient, instead of valid, as p_u is likely to break the lower bounds of `Shift Coverage`.

$w(p_{u-1})$ can be calculated by considering all possible rosters for nurse u, and checking whether they are a sufficient successor to p_{u-1}. Calculating every possible roster for a single nurse takes $O(|S|^{|D|})$ time, by trying every single shift on each day. While these rosters are built, as long as forbidden sequence set F and personal requests set R have a constant look up time, it can be checked in constant time whether the roster violates either of these constraints. It can be checked in $O(|D| * |S|)$ time whether a roster is a sufficient successor to p_{u-1}. As there are $O(|S|^{|D|})$ different rosters, this takes $O(|D| * |S| * |S|^{|D|})$ time. So this can be accomplished in $O(|D| * |S|^{|D|+1}) = O(\bar{k} * k^{\bar{k}+1})$ time.

The set Φ_u consists of all sufficient assignments p_u. We can define $\Phi_u(\Phi_{u-1}) := \bigcup_{p_{u-1} \in \Phi_{u-1}} w(p_{u-1})$. By again keeping track of which assignments p_u are preceded by which p_{u-1}, we can select an assignment $p_{|N|}$ that conforms to the upper bound constraint of `Shift Coverage`, and get a valid solution from this.

Each $w(p_{u-1})$ can be calculated in $O(\bar{k} * k^{\bar{k}+1})$ time. However, set Φ_u of sufficient assignments p_u is very large. For a given Φ_u, the amount of sufficient assignments for each day is the amount of different ways in which u nurses can be divided over k sets, disregarding any assignments which break constraints. This is up to $\binom{u+k-1}{u}$ different combinations. As there are $\bar{k}$ days, Φ_u can be as large as $\binom{u+k-1}{u}^{\bar{k}}$. This equals

$\left(\frac{(u+k-1)!}{u!(k-1)!}\right)^{\bar{k}}$. Therefore the algorithm takes factorial time to run, and is not polynomial.

5.4 Multi-commodity integral flow

In this section we will use the following constraint-set C again:

Constraints:	• hard `Shift Coverage`
	• hard `Forbidden Sequences=2`
	• hard `Personal Requests`
	• `k shifts`

In Sections 4.10 and 4.11, we showed how to formulate an instance of a nurse scheduling program as a network flow problem. However, we cannot easily transform the limited problem C to this form. In both 4.10 and 4.11, we treated the amount of nurses as a commodity. However, by anonymizing the nurses, we can no longer enforce any personal requests. Instead we can formulate it as a version of the multi-commodity integral flow problem. As C was already proven NP-complete in Section 5.2, it is only natural that the multi-commodity integral flow problem is NP-hard (Garey and Johnson, 1979, page 216).

In the Multi-commodity integral flow problem (`MCIFP`), we are given a directed graph $G = (V, A)$ and n commodities $K_1, \ldots, K_n$. Each commodity K_i has its own source, sink and demand s_i, t_i and d_i. For each commodity i, we must give a flow function f_i, following flow constraints and capacity constraints among all arcs and making sure the demand d_i of flow is sent through. There is some cost-function $a(u, v)$ that assigns a cost to each arc (u, v). The goal is to minimize the total cost.

`MCIFP` is usually formulated as an Integer Linear Programming problem. Here, we

formulate it as follows:

$$\text{Minimize} \sum_{(u,v)\in A} \left(a(u,v) \sum_{i=1}^{n} f_i(u,v) \right) \qquad \text{s.t.} \tag{5.4.1}$$

$$\sum_{(u,v)\in A} f_i(u,v) - \sum_{(v,w)\in A} f_i(v,w) = 0 \qquad \forall i \in \{1,\ldots,n\}, \forall v \in V, v \neq s_i, t_i \tag{5.4.2}$$

$$\sum_{i=1}^{n} f_i(u,v) \leq c(u,v) \qquad \forall (u,v) \in A \tag{5.4.3}$$

$$\sum_{(s_i,v)\in A} f_i(s_i,v) = \sum_{(w,t_i)\in A} f_i(w,t_i) = d_i \qquad \forall \{i \in 1,\ldots,n\} \tag{5.4.4}$$

$$f_i(u,v) \in \mathbb{Z}_+ \qquad \forall i, \forall (u,v) \in A \tag{5.4.5}$$

Equation 5.4.2 describes the flow constraint. Given a capacity-function $c(u,v)$ for all $(u,v) \in A$, Equation 5.4.3 describes the capacity constraints. Equation 5.4.4 ensures that the demand is satisfied for each commodity whereas Equation 5.4.5 ensures integrality of the solution.

A single-commodity flow network guarantees that if capacity and demand constraints are integral, then the solution is also always integral. Therefore, it can be freely solved by a normal, non-integer linear program. However, multi-commodity gives no such guarantees(Ahuja et al., 2013, page 651). Therefore, integrality must be enforced through additional constraints.

We will use roughly the same network as we used in Section 4.10. For each shift j and day k, create a vertex v_{jk}. Create a master-source s and a master-sink t. For two subsequent days k and $k+1$, add arc $(v_{jk}, v_{j'(k+1)})$ if $\{j,j'\} \notin F$. Add arcs (s, v_{j1}) and $(v_{j|D|}, t)$ for all shifts j.

Each nurse will serve as a commodity. Create $|N|$ commodities. For each commodity K_i, create source s_i and sink t_i, set the demand $d_i = 1$, and add arcs $(s_i, s), (t, t_i)$. Now each commodity will be given a path among our vertices, and for each day exactly one shift will be assigned. Vertices of subsequent days only have an arc between them if this is allowed, so there are no forbidden sequences in the personal rosters of each nurse (the soft version of the `Forbidden Sequences` constraint can be handled the same way as in Section 4.11). Now, the only constraints left to tackle are `Shift Coverage` and `Personal Requests`.

We split each vertex v_{jk} into 3 vertices, v^1_{jk}, v^2_{jk} and v^3_{jk}. As before, we redirect all incoming arcs to v^1_{jk}, and all outgoing arcs start at v^3_{jk}. We add an arc (v^1_{jk}, v^2_{jk}) with not

only a capacity $c(v^1_{jk}, v^2_{jk}) = U^s_{jk}$, but also a lower bound on the flow, $d(v^1_{jk}, v^2_{jk}) = L^s_{jk}$. The capacity ensures that the upper bounds L^s_{jk} are satisfied, while the lower bound on the flow ensures that the lower bounds L^s_{jk} are satisfied. The soft version can be handled as in Section 4.11.

For `Personal Requests`, we define a new constraint. $d_i(u, v)$ is defined as the demand of commodity i over (u, v). In other words, $f_i(u, v) \geq d_i(u, v)$. Create an arc (v^2_{jk}, v^3_{jk}). If shift j was requested on day k by nurse i, set $d_i(v^2_{jk}, v^3_{jk}) = 1$, otherwise 0. Now we can guarantee that nurse i is assigned to this shift and day combination. For the soft version of `Personal Requests`, we would have to define arcs which only a specific commodity is allowed to cross, and assign the negative of the penalty for breaking the request to such an arc. If the demand or capacity of an arc has not been mentioned, it is assumed that the capacity is infinite and the demand zero.

The full ILP of the hard version of C is as follows:

$$\text{Minimize} \sum_{(u,v)\in A} \left(a(u, v) \sum_{i=1}^{n} f_i(u, v) \right) \qquad \text{s.t.} \quad (5.4.6)$$

$$\sum_{(u,v)\in A} f_i(u, v) - \sum_{(v,w)\in A} f_i(v, w) = 0 \qquad \forall i\{\in 1, \ldots, n\}, \forall v \in V, v \neq s_i, t_i \quad (5.4.7)$$

$$\sum_{i=1}^{|N|} f_i(v^1_{jk}, v^2_{jk}) \leq c(v^1_{jk}, v^2_{jk}) \qquad \forall j \in S, k \in D \quad (5.4.8)$$

$$\sum_{i=1}^{|N|} f_i(v^1_{jk}, v^2_{jk}) \geq d(v^1_{jk}, v^2_{jk}) \qquad \forall j \in S, k \in D \quad (5.4.9)$$

$$f_i(v^2_{jk}, v^3_{jk}) \geq d_i(v^2_{jk}, v^3_{jk}) \qquad \forall j = r_{ik} \in R \quad (5.4.10)$$

$$f_i(s_i, S) = f_i(T, t_i) = 1 \qquad \forall i \in \{1, \ldots, n\} \quad (5.4.11)$$

$$f_i(u, v) \in \{0, 1\} \qquad \forall i, \forall (u, v) \in A \quad (5.4.12)$$

As we did not define any cost function, Equation 5.4.6 is always equal to 0, and the first admissible solution is chosen.

Equation 5.4.7 is the same as Equation 5.4.2. Equation 5.4.8 is almost the same as Equation 5.4.3, except only the arcs with demands on them have been written down. Equation 5.4.9 lists the demand constraints, and Equation 5.4.10 the personal demand constraints. We can set the demands for each commodity to 1 as in Section 5.4.11. As the demand is 1, the flow can never succeed 1, so the flow over an arc is now a $\{0, 1\}$ constraint instead of a general positive integer.

We will now present the ILP of the version of C where only `Shift Coverage` is soft, and the other constraints are hard. As mentioned before, for each j, k we create 3 new vertices, n^L_{jk}, n^U_{jk} and n^0_{jk}. Instead of an arc (v^1_{jk}, v^2_{jk}), create arcs (v^1_{jk}, v^L_{jk}), $(v^1_{jk}, v^U_{jk}), (v^1_{jk}, v^0_{jk}), (v^L_{jk}, v^2_{jk}), (v^U_{jk}, v^2_{jk})$ and (v^0_{jk}, v^2_{jk}). Set the capacity $d(v^1_{jk}, v^L_{jk}) =$

$L^s_{jk}, d(v^1_{jk}, v^0_{jk}) = U^s_{jk} - L^s_{jk}$. Lastly, set the cost $a(v^1_{jk}, v^L_{jk}) = -q^L_{jk}, a(v^1_{jk}, v^0_{jk}) = 0$ and $a(v^1_{jk}, v^U_{jk}) = q^U_{jk}$. This is almost identical to the situation portrayed in Figure 3 of Section 4.11.

All incoming flow first goes through the node v^L_{jk}, due to its negative penalty. When the lower bound of the shift has been reached, the flow goes through the node v^0_{jk}, and once the upper bound has also been reached all flow goes through v^U_{jk}, where a penalty is incurred for each assignment.

The ILP of this version with soft `Shift Coverage` is as follows:

$$\text{Minimize} \sum_{j \in S, k \in D} \left(a(v^1_{jk}, v^L_{jk}) \sum_{i=1}^{n} f_i(v^1_{jk}, v^L_{jk}) + a(v^1_{jk}, v^U_{jk}) \sum_{i=1}^{n} f_i(v^1_{jk}, v^U_{jk}) \right) \quad \text{s.t.} \tag{5.4.13}$$

$$\sum_{(u,v) \in A} f_i(u,v) - \sum_{(v,w) \in A} f_i(v,w) = 0 \qquad \forall i \in \{1, \ldots, n\}, \forall v \in V, v \neq s_i, t_i \tag{5.4.14}$$

$$\sum_{i=1}^{|N|} f_i(v^1_{jk}, v^L_{jk}) \leq c(v^1_{jk}, v^L_{jk}) \qquad \forall j \in S, k \in D \tag{5.4.15}$$

$$\sum_{i=1}^{|N|} f_i(v^1_{jk}, v^U_{jk}) \leq c(v^1_{jk}, v^U_{jk}) \qquad \forall j \in S, k \in D \tag{5.4.16}$$

$$f_i(v^2_{jk}, v^3_{jk}) \geq d_i(v^2_{jk}, v^3_{jk}) \qquad \forall j = r_{ik} \in R \tag{5.4.17}$$

$$f_i(s_i, S) = f_i(T, t_i) = 1 \qquad \forall i \in \{1, \ldots, n\} \tag{5.4.18}$$

$$f_i(u,v) \in \{0,1\} \qquad \forall i, \forall (u,v) \in A \tag{5.4.19}$$

6 Conclusion and future work

We have shown for a large number of constraints in what combination they are polynomially solvable, and when they become NP-complete. For the polynomial problems, we have given rudimentary algorithms and for the NP-complete problems we have shown proofs. We have looked at one instance of a problem with a limited amount of shifts, which was still NP-complete, even with forward rotating rosters. We have also shown additional results of our work on this problem to help future research into this subject. For future research it could be a good idea to look at more constraint-sets and test how limiting the amount of shifts affects their polynomiality.

With this overview of constraints, theory of nurse scheduling is more complete. Polynomial subproblems can be used for algorithms such as branch and price, which could potentially give a big computational boost. More work could be done to see what combinations of constraints are efficient to use in subproblems.

Lastly, there are more constraints which could be looked at. For example, more limitations could be put on a variety of constraints to see if they are still NP-complete in some cases, like limiting the amount of forbidden sequences or personal requests by a constant. It is possible that some of these are fixed-parameter tractable. Other possibilities are non-linear penalty functions for `Shift Coverage`.

A Tables

Section	Constraint	Abbreviation	Categorization	Page nr.
3.3	`Shift Coverage`	`SC`	$\cdot\|RV\|\cdot$	10
3.3	`Strict Shift Coverage`	`SSC`	$\cdot\|RV\|\cdot$	10
3.4	`Balanced Rosters`	`BR`	$A(b)\|\cdot\|\cdot$	10
3.5	`Forbidden Sequences`	`FS`	$S(b,c)\|\cdot\|\cdot$	11
3.6	`Personal Requests`	`PR`	$A(e)\|\cdot\|\cdot$	12
3.6	`Days-off Requests`	`DOR`	$A(e)\|\cdot\|\cdot$	12
3.7	`Identical Weekends`	`IW`	$S(d)\|\cdot\|\cdot$	13
3.7	`Full Free Weekends`	`FFW`	$S(d)\|\cdot\|\cdot$	13
3.8	`Skill Requirements`	`SR`	$I\|\cdot\|\cdot$	13

Table 10: *Overview of all problems introduced in Section 3*

Variable	Constraint	Description
N	`NSP`	Set of nurses
D	`NSP`	Set of days
S	`NSP`	Set of shifts
$\tilde{p}$	`NSP`	Shift assignment to all nurses
p_{ik}	`NSP`	Shift assignment to nurse i on day k
$c(\tilde{p})$	`NSP`	Penalty function of constraint
$\tilde{c}(\tilde{p})$	`NSP`	Total penalty function over all constraints
L^s_{jk}	`Shift Coverage`	Lower bound on each shift/day combination
U^s_{jk}	`Shift Coverage`	Upper bound on each shift/day combination
q^L_{jk}	`Shift Coverage`	Cost assigned to breaking of the lower bound in soft version
q^U_{jk}	`Shift Coverage`	Cost assigned to breaking of the upper bound in soft version
L^b_{ij}	`Balanced Rosters`	Lower bound on each nurse/shift combination
U^b_{ij}	`Balanced Rosters`	Upper bound on each nurse/shift combination
F	`Forbidden Sequences`	Set of all forbidden sequence
R	`Personal Requests`	Set of all requests
r_{ik}	`Personal Requests`	Shift requested by nurse i on day k
L	`Skill Requirements`	Set of all skills
l_i	`Skill Requirements`	Set of skills assigned to nurse i
L^l_{jkl}	`Skill Requirements`	Lower bound on the amount of nurses assigned to skill l
U^l_{jkl}	`Skill Requirements`	Upper bound on the amount of nurses assigned to skill l

Table 11: *Overview of all variables introduced in Section 3*

Bibliography

R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Pearson New International Edition: Theory, Algorithms, and Applications*. Always learning. Pearson Education, Limited, 2013. ISBN 9781292042701.

J. Baeklund. Nurse rostering at a danish ward. *Annals of Operations Research*, 222(1):107–123, 2014.

J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, 2013.

P. Brucker, E. Burke, T. Curtois, R. Qu, and G. V. Berghe. Adaptive construction of nurse schedules: A shift sequence based approach. *Journal of Heuristics*, 16(4):559–573, 2010.

R. Bruni and P. Detti. A flexible discrete optimization approach to the physician scheduling problem. *Operations Research for Health Care*, 3(4):191–199, 2014.

E. K. Burke, P. De Causmaecker, G. V. Berghe, and H. Van Landeghem. The state of the art of nurse rostering. *Journal of scheduling*, 7(6):441–499, 2004.

P. De Causmaecker and G. Vanden Berghe. A categorisation of nurse rostering problems. *Journal of Scheduling*, 14(1):3–16, 2011.

F. Della Croce and F. Salassa. A variable neighborhood search based matheuristic for nurse rostering problems. *Annals of Operations Research*, 218(1):185–199, 2014.

J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691, 1976.

M. Garey and D. Johnson. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman and company, 1979.

S. Haspeslagh, P. De Causmaecker, A. Schaerf, and M. Stølevik. The first international nurse rostering competition 2010. *Annals of Operations Research*, 218(1):221–236, 2014.

R. M. Karp. *Reducibility among combinatorial problems*. Springer, 1972.

H. C. Lau. Manpower scheduling with shift change constraints. In *Algorithms and Computation*, pages 616–624. Springer, 1994.

T. Osogami and H. Imai. Classification of various neighborhood operations for the nurse scheduling problem (extended abstract). In *Algorithms and Computation*, pages 72–83. Springer, 2000.

P. Smet, P. Brucker, P. De Causmaecker, and G. Vanden Berghe. Polynomially solvable formulations for a class of nurse rostering problems. In *Proceedings of the 10th international conference on the practice and theory of automated timetabling*, pages 408–419. 2014.

S. van Veldhoven, G. Post, E. van der Veen, and T. Curtois. An assessment of a days off decomposition approach to personnel shift scheduling. *Annals of Operations Research*, pages 1–17, 2014.

T. v. Weelden. Nurse rostering through linear programming and repair heuristics. 2013.

T. v. d. Zanden. Personal Communication, 2016.