# The Days On Days Off Scheduling Problem

Fabien Nießen\*

Paul Paschmanns<sup>†</sup>

October 31, 2024

#### Abstract

Personnel scheduling problems have received considerable academic attention due to their relevance in various real-world applications. These problems involve preparing feasible schedules for an organization's employees and often account for factors such as qualifications of workers and holiday requests, resulting in complex constraints. While certain versions of the personnel rostering problem are widely acknowledged as NP-hard, there is limited theoretical analysis specific to many of its variants. Many studies simply assert the NP-hardness of the general problem without investigating whether the specific cases they address inherit this computational complexity.

In this paper, we examine a variant of the personnel scheduling problems, which involves scheduling a homogeneous workforce subject to constraints concerning both the total number and the number of consecutive work days and days off. This problem was claimed to be NP-complete by Brunner, Bard, and Köhler [BBK13]. In this paper, we prove its NP-completeness and investigate how the combination of constraints contributes to this complexity. Furthermore, we analyze various special cases that arise from the omission of certain parameters, classifying them as either NP-complete or polynomial-time solvable. For the latter, we provide easy-to-implement and efficient algorithms to not only determine feasibility, but also compute a corresponding schedule.

## 1 Introduction

In personnel scheduling problems the task is to compute a schedule for a given set of workers subject to certain constraints, such as daily requests on the number of active workers or an upper limit on the number of consecutive night shifts, possibly aiming to optimize a given objective function. Such problems are well-studied from a pratical point of view, due to its many applications. Real-world instances often differentiate staff members based on their qualifications or personal preferences for days off. Given the numerous constraints involved, personnel rostering problems are frequently addressed using integer linear programming (ILP) or LP-based heuristics [Ngo+22]. For examples of modeling nurse rostering – one of the most widely studied applications within personnel scheduling – see [Sme+13; Sme+14].

For a comprehensive overview of nurse rostering, we refer to [Bur+04]. Additionally, efforts to categorize nurse rostering problems in a manner similar to scheduling are presented by De Causmaecker and Vanden Berghe [DV11]. On the complexity front, the NP-hardness of specific cases within personnel rostering has been established in several studies, including key contributions by Lau [Lau96], Osogami and Imai [OI00], and Hartog, Hoogeveen, and Zanden [HHZ23]. In contrast, polynomially solvable cases – where particular constraints or simplified structures allow for efficient solutions – are for example examined by Brucker, Qu, and Burke [BQB11] and Smet et al. [Sme+16]. A comprehensive summary covering both NP-hard and polynomially solvable cases is available in the thesis by Den Hartog [Den16].

In this paper we are focusing on the decision variant of the DAYS ON DAYS OFF SCHEDULING PROBLEM (DODOSP), a personnel scheduling problem involving a homogeneous set of workers, introduced by Brunner, Bard, and Köhler [BBK13]. The DODOSP involves a sequence of days

<sup>\*</sup>Department of Computer Science, KU Leuven, Gebroeders De Smetstraat 1, 9000 Gent, Belgium (fabien.niessen@kuleuven.be)

<sup>†</sup>Research Institute for Discrete Mathematics, University of Bonn, Lennéstr. 2, 53113 Bonn, Germany (paschmanns@dm.uni-bonn.de)

with specified minimum requests for the number of workers needed each day, along with the following parameters: upper limits on the total number of work days and days off for each worker, as well as upper and lower bounds on the number of consecutive work days and days off.

These constraints are derived from practical considerations. For example, an upper limit on consecutive working days could stem from ensuring compliance with labor regulations. For example, in Germany, exceeding 19 consecutive days of work violates the "Arbeitszeitgesetz" [Arb24]. Similarly, the total number of working days of each worker during the given time horizon might be limited. On the other hand, employees usually only have a limited number of vacation days. A lower bound on the number of consecutive work days or days off can prevent wasting too much time to commute and might therefore be especially interesting for industries that involve visiting customers on site.

While Brunner, Bard, and Köhler [BBK13] primarily focused on practical results using a branch-and-price approach, they claimed the NP-completeness of the DODOSP. However, the proof they provided was unfortunately incorrect (cf. Appendix A.1). In this paper, we prove that the problem is indeed NP-complete. The complexity primarily arises from combining lower bounds on consecutive work days or days off with an upper bound on each worker's total number of work days or days off. Additionally, we classify each special case that results from omitting certain constraints as either polynomial-time solvable or NP-complete. All hardness results are derived from reductions from the 3-Partition Problem. For the polynomial-time solvable cases, we reduce the DODOSP to the problem of checking for negative cycles in a directed graph.

The remainder of this paper is organized as follows. We formally introduce the DODOSP and the special cases we study in Section 2. Furthermore, we outline the main ideas used throughout the paper. The hardness results are given in Section 3. Then, in Sections 4 and 5, we provide polynomial-time algorithms for the special cases that are not NP-complete. In Section 6, we then briefly discuss how the results can be used in case of an optimization variant of the problem. Finally, Section 7 concludes the paper by summarizing our results and providing guidance on how they can be used, while also identifying open questions that remain for future research.

## 2 Problem introduction

This chapter begins by first providing a formal definition of the DAYS ON DAYS OFF SCHEDULING PROBLEM (DODOSP) in Section 2.1 and introduces the terminology needed to understand the problem. Furthermore, we establish a set of special cases of the DODOSP, which will come in handy to analyze where the hardness originates from. Finally, in Section 2.2 we present the main concepts used throughout the paper.

#### 2.1 Formal definition

In this paper we are always given a number of workers  $N \in \mathbb{N}$  and a number of days  $D \in \mathbb{N}$ . We then denote the set of workers by  $\mathcal{N} := \{n_1, \dots, n_N\}$  and the set of days by  $\mathcal{D} := [D] = \{1, \dots, D\}$ .

**Definition 1.** Given a number of workers N and a number of days D, a mapping  $f: \mathcal{N} \times \mathcal{D} \to \{\text{ON}, \text{OFF}\}$  is called a *schedule*. We say worker  $n_i$  works or is on duty on day  $d \in [D]$  if  $f(n_i, d) = \text{ON}$ . Otherwise, we say that worker  $n_i$  has day d off.

**Definition 2.** Given a worker  $n_i$ , we call every inclusion-wise maximal set of consecutive days on which  $n_i$  works a work period (of worker  $n_i$ ). Similarly, we define an off period (of worker  $n_i$ ) as an inclusion-wise maximal set of days on which  $n_i$  does not work.

Figure 1 provides an example schedule and illustrates the concept of work periods and off periods. We define the Days On Days Off Scheduling Problem as the following decision problem:

DAYS ON DAYS OFF SCHEDULING PROBLEM (DODOSP)

Input: A number of days  $D \in \mathbb{N}$ , a number of workers  $N \in \mathbb{N}$ , bounds

 $l_w, u_w, l_o, u_o, U_w, U_o \in \mathbb{N}$  on periods and bounds  $0 \le r_l^d \le r_u^d \le N$  for each

 $d \in (D)$  on requests for all days.

**Question:** Does there exist a *feasible* schedule with N workers? A schedule is called feasible if on every day d at least  $r_l^d$  and at most  $r_u^d$  workers work and the

feasible if on every day d at least  $r_i^*$  and at most  $r_u^*$  workers work and bounds for every worker are respected, i.e.,

• every work period of every worker is at least  $l_w$  and at most  $u_w$  days long,

• every off period of every worker is at least  $l_o$  and at most  $u_o$  days long,

• every worker works in total at most  $U_w$  days and

• every worker has in total at most  $U_o$  many days off.

The DODOSP was first introduced in [BBK13], where a branch-and-price algorithm was proposed to solve real-world instances optimally. We have expanded their problem definition by adding an upper bound  $U_o$  on the total number of days off as well as day-specific upper bounds  $r_u^d$  on the number of workers that should be scheduled to work on the given day.

We distinguish the problem's bounds in the following way: We call  $l_w$  and  $l_o$  the local lower bounds,  $u_w$  and  $u_o$  the local upper bounds,  $U_w$  and  $U_o$  the global (upper) bounds and  $r_l^d$  and  $r_u^d$  the request bounds.

Remark. Note that the definition of the DODOSP does not introduce global lower bounds. This is because they can easily be transformed into global upper bounds of the opposite type: A global lower bound of  $L_w$  on the work days is equivalent to a second upper bound of  $D-L_w$  concerning the number of days off. Thus, instead of using  $L_w$ , we would use  $\min\{U_o, D-L_w\}$  as a global upper bound on the number of days off. The same argument holds for a global lower bound  $L_o$  for the number of days off. Further, we point out that the schedule is not cyclic, i.e., is not repeated after D days. Thus, a worker who works on the first day has to work for the first  $l_w$  days. Nevertheless, all our hardness results from Section 3 can be extended to the CYCLIC DODOSP, where we assume that the schedule is repeated. This affects the impact of local bounds, i.e., it would be feasible to work for only two days at the beginning of the schedule and  $l_w - 2$  days at the very end of the schedule.

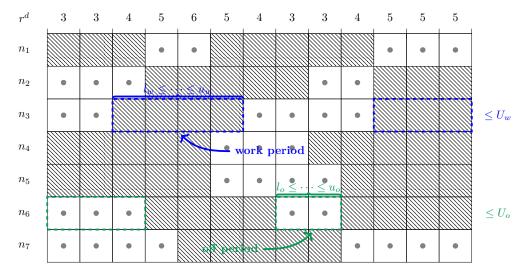


Figure 1: A feasible solution for the instance of the DODOSP with exact demands (i.e.,  $r^d := r_l^d = r_u^d$  for all d) given in the first row, work period bounds of  $l_w = 3$  and  $u_w = 5$ , off period bounds of  $l_o = 2$  and  $u_o = 4$ , as well as global bounds  $U_w = 10$  and  $U_o = 9$ . Squares with a dot indicate a day off, while shaded squares indicate a day on duty.

Next, we introduce two special cases which arise from ignoring a set of bounds. Note that

instead of removing a bound from the problem definition we can fix it to the value given in Table 1, which makes it trivially fulfilled.

**Definition 3.** We call the DODOSP restricted to instances with  $l_w = l_o = 1$  the ONLY UPPER BOUNDED DAYS ON DAYS OFF SCHEDULING PROBLEM (UDODOSP).

**Definition 4.** We call the DODOSP restricted to instances with  $U_w = U_o = D$  the LOCAL DAYS ON DAYS OFF SCHEDULING PROBLEM (LDODOSP).

We are interested in these two special cases since they can be decided in polynomial time, which we prove in Sections 4 and 5. On the other hand, in Theorem 8, we show that the DODOSP is NP-complete as soon as it features both global bounds and local lower bounds.

In Section 3 we consider further special cases that arise from removing a given set of bounds.

bound 
$$l_w$$
  $u_w$   $l_o$   $u_o$   $U_w$   $U_o$   $r_l$   $r_u$ 

default value 1  $D$  1  $D$   $D$   $D$  0  $N$ 

Table 1: Values to set bounds to in order to make them trivially fulfilled.

From an application perspective, one might be interested in the case where  $r_u^d = N$  for all  $1 \le d \le D$ . In this case we only have a minimal work request that must be covered on each day. Theorem 13 states that besides a trivial exception, the same complexity results hold as in the general case where requests are bounded from both sides. If the minimum and maximum requests are identical each day, we refer to the instance as having exact requests. Analyzing the UDODOSP with exact requests is simpler than doing so with general requests. Therefore, in Section 4, we will begin by examining the case with exact requests, followed by an analysis of the general requests case.

#### 2.2 Overview

In Section 4, we study the UDODOSP. To motivate why this special case might be easier than the general DODOSP we first take a more heuristical look at it by considering the example in Figure 2, where we are given an instance of the DODOSP with only two workers and exact requests  $(r_l = r_u)$ . For days with a request of zero or two there is no decision to be made when designing a schedule. However, on days with request one we have to choose which worker will be on duty. For the example shown in Figure 2, we are in the situation that before a certain day both workers were on duty for the same number of days. On the given day only the first worker  $n_1$  is on duty. For the following day, one worker is requested and we have to decide who will be on duty.

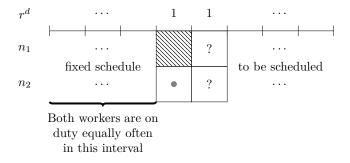


Figure 2: Assume the left part of the schedule is fixed already and we have to decide who is on duty on the following day. Given that on the first days both workers were on duty equally often all upper bounds  $(u_w, u_o, U_w \text{ and } U_o)$  "would prefer" that we switch who is on duty, while lower bounds "would prefer" us to keep  $n_1$  working.

From the perspective of the global bound it seems obvious what to do: it would be better to assign  $n_2$  to work on the following day since this equalizes the number of days off and days on duty between the workers. From the perspective of the local upper bounds, it also seems to be beneficial

to assign  $n_2$  to work. In doing so we interrupt the work resp. off periods of both workers which gives us a better chance to not violate the upper upper bounds. However, from the perspective of local lower bounds it seems more reasonable to keep  $n_1$  on duty for another day since otherwise we might end a work or off period before it reached its minimal length.

Thus, all upper bounds are "more likely" to be satisfied if we change who is on duty more often, while the lower bounds are "more likely" to be fulfilled if we change who is on duty less often.

This shared preference of all upper bounds for many changes in who is on duty will be turned into an efficient algorithm (for the case of exact requests) in Section 4.1 by simply assigning workers in a cyclic manner. Besides an algorithm that computes feasible schedules (if they exist), we also obtain feasibility criteria for the case of exact requests that are easy to handle. Using these criteria we then generalize the result in Section 4.2 to the UDODOSP with daily upper and lower bounds  $r_u$  and  $r_l$  on the number of workers on duty. This is done in two stages by first choosing the number of workers that are on duty from the interval  $\begin{bmatrix} r_l^d, r_u^d \end{bmatrix}$  for each day d in a way that preserves feasibility. This can be done by computing a feasible potential in a certain auxiliary graph. In the second stage we are left with the case of exact requests.

Since the results of Section 4 heavily rely on maximizing the number of times a worker switches between days off and being on duty, they can not be generalized to also fulfill lower bounds. In fact the combination of local lower bounds with global upper bounds is what makes the DODOSP NP-hard. We prove this in Section 3 by reducing a packing problem to the DODOSP. To do so, we interpret the different workers as containers into which we have to pack given numbers. In this setting, a global upper bound corresponds to the capacity of each container. The numbers that have to be packed are encoded in a unary way in the sequence of requests. Thus one can think of a number to be in a container whenever the corresponding worker is on duty during the days that encode the corresponding number. The local lower bounds are needed to ensure that a number is not split between two containers or, equivalently, that only one worker is on duty during the encoding of that number.

As described earlier, local lower bounds and local upper bounds have contradicting "preferences" when it comes to the number of times workers switch between being on duty and days off. However, if we would know for every day how many workers switch from being on duty to a day off and vice versa, it would be easy to construct a feasible schedule. This is because all local bounds benefit from a certain structure in the work and off periods of a schedule. That is, whenever worker  $n_1$  starts a work period earlier than another worker  $n_2$  starts their work period, the work period of  $n_1$  should not end later than the work period of  $n_2$ . The same should hold for off periods. We refer to this property as first-in-first-out (FIFO) property. Figure 3 illustrates a situation that is avoided by the FIFO property. It also shows how to modify a feasible schedule in order to obtain the property. This modification preserves feasibility regarding local bounds, but might destroy feasibility regarding global bounds.

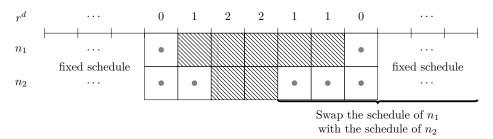
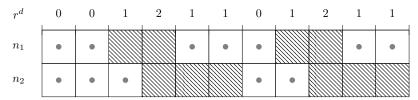


Figure 3: The depicted schedule does not satisfy the FIFO property since the work period of  $n_2$  starts later and finishes earlier than the one of  $n_1$ . If we would swap the suffixes of the schedules of  $n_1$  and  $n_2$ , the FIFO property would hold (in the given part of the schedule). Note that swapping suffixes preserves feasibility with respect to local bounds.

Since the FIFO property only distributes the number of work periods equally but does not take into account how long each work period is, it gives no guarantees in terms of global bounds (cf. Figure 4b). In fact, even in schedules that satisfy the FIFO property, it can occur that the number of days on which two different workers are on duty, can differ by an arbitrary constant independent of  $l_w$  and  $l_o$ . Figure 4a illustrates what such an instance might look like.



(a) Given  $l_w = l_o = 2$ ,  $u_w \ge 3$ ,  $u_o \ge 3$  and  $U_w = U_o = D = 10$ , the depicted schedule is the only solution of the instance with the given demands (up to symmetry in  $n_1$  and  $n_2$ ). If we expand the instance by repeating the pattern 0, 1, 2, 1, 1 in the requests c - 2 times, we receive an instance where one of the workers has to be on duty 3c times, while the other works only on 2c days.

$r^d$	0	0	1	2	1	1	0	1	2	1	1
$n_1$	•	•			•	•	•	•			
$n_2$	•	•	•				•			•	•

(b) If we instead set  $l_w = 2$ ,  $U_w = 5$  and fix all other bounds to their trivial value, the instance is feasible, but no feasible schedule fulfills the FIFO property.

Figure 4: Instances demonstrating that some worker might have to be on duty more often than others, and that the FIFO property is not compatible with global bounds.

# 3 NP-completeness

In this section, we prove that the general DODOSP is NP-complete. Furthermore, we identify the NP-complete special cases that arise from fixing certain bounds to their default values (cf. Table 1).

Before investigating the completeness, let us first check that the DODOSP is actually in NP. Naively encoding a feasible schedule as a spreadsheet takes  $\Theta(ND)$  bits, while any instance of the DODOSP can be encoded using  $\mathcal{O}(D \cdot \log N)$  bits. Thus we need a different certificate. Without any bounds on the requests, the DODOSP reduces to a flow problem. Such a flow can be used as a certificate for the general problem.

**Lemma 5.** The DODOSP restricted to instances with  $r_l^d = 0$  and  $r_u^d = N$  for each day d can be decided in polynomial time in D.

*Proof.* We reduce this problem to a network flow problem in an acyclic graph G = (V, E) with

$$V = \{s, t\} \cup \{(1, \text{ON}, 1, 1), (1, \text{OFF}, 1, 0)\}$$

$$\cup \{(d, \text{ON}, a, b) : 2 \le d \le D, 1 \le a \le u_w, d - U_o \le b \le U_w\}$$

$$\cup \{(d, \text{OFF}, a, b) : 2 < d < D, 1 < a < u_o, d - U_o < b < U_w\}$$

The work plan of a single worker can be represented as a sequence of these vertices of the form  $s, (1, SHIFT_1, a_1, b_2), \ldots, (D, SHIFT_D, a_D, b_D), t$ . Besides s and t, each vertex represents a single day d and contains three pieces of information:

- SHIFT $_d$  encodes if the worker is on duty on day d or not,
- day d is the  $a_d$ -th day of the current SHIFT $_d$  period and
- on the first d days the worker was on duty  $b_d$  times.

On the other hand such a sequence can be turned into a work plan of a single worker if and only if consecutive vertices (d, SHIFT, a, b) and (d', SHIFT', a', b') are compatible with each other. This is the case if d+1=d' and

- SHIFT = SHIFT' = ON and a + 1 = a' and b + 1 = b' or
- SHIFT = SHIFT' = OFF and a + 1 = a' and b = b' or

- SHIFT = ON, SHIFT' = OFF,  $a \ge l_w$ , a' = 1 and b' = b or
- SHIFT = OFF, SHIFT' = ON,  $a \ge l_o$ , a' = 1 and b' = b + 1.

We add an edge from (d, SHIFT, a, b) to (d', SHIFT', a', b') whenever they are compatible. We also add edges from s to all tuples with d=1 and edges from all tuples with d=D to t. Thus every feasible work plan of a single worker corresponds to an s-t-path in the constructed graph. An excerpt of such a certificate graph is illustrated in Figure 5. Note that this graph has  $\mathcal{O}\left(D^3\right)$  vertices and thus its size is polynomial in D. Furthermore, we reduce the size of the graph by imposing the conditions  $a \leq d$  and  $0 \leq b \leq d$ .

Feasible schedules correspond to (path decomposition of) integral s-t-flows of value N in G, which can be computed in polynomial time. Hence it is easy to check if a feasible schedule exists.

#### **Proposition 6.** The Days On Days Off Scheduling Problem is in NP.

*Proof.* As a certificate, we can use an integral s-t-flow of value N in the same network as in the proof of Lemma 5. The number of workers that work on day d equals the total throughput of all vertices (d, ON, a, b) with  $a, b \in [D]$  and can therefore be efficiently checked for feasibility.

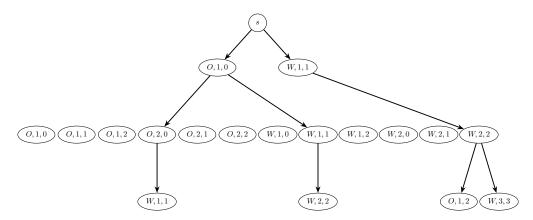


Figure 5: An excerpt of the certificate graph for instances of the DODOSP with  $D \geq 3$ ,  $l_w = 2$ ,  $u_o = 2$  and all other bounds set to their trivial value. For d = 2 we only depict the vertices which fulfill  $a \leq d$  and  $0 \leq b \leq d$ , for d = 3 only those which are connected to the previous layer. For spacing reasons we omit the first number which denotes the day.

#### 3.1 Exact requests

In this chapter we prove NP-completeness results for the special cases with exact requests. This immediately implies NP-completeness for the general DODOSP.

**Lemma 7.** The DODOSP is strongly NP-complete, even if restricted to instances where  $u_w = u_o = D$ ,  $l_o = 1$  and  $r^d := r_l^d = r_u^d \in \{0, 1\}$ .

*Proof.* For this proof, we will use a reduction from the strongly NP-complete RESTRICTED 3-PARTITION PROBLEM.

RESTRICTED 3-PARTITION PROBLEM

**Input:** A number  $m \in \mathbb{N}$  and a multiset  $A = \{a_1, \dots, a_{3m}\} \subset \mathbb{N}$  of values with  $\frac{1}{4}T < a_i < \frac{1}{2}T$  for all  $a_i \in A$ , where  $T := \frac{1}{m} \sum_{i=1}^{3m} a_i$ .

**Question:** Does there exist a partition  $A = \bigcup_{i=1}^{m} A_i$  with  $|A_i| = 3$  and  $\sum_{a \in A_i} a = T$  for all  $1 \le i \le m$ ?

Since the Restricted 3-Partition Problem is strongly NP-complete [GJ79], we may assume that  $S = \sum_{i=1}^{3m} a_i = mT \le q\left(m\right)$  for some universal polynomial q.

We construct an equivalent instance of the DODOSP. The idea of the construction is to identify the values  $a_i$  with work periods of length  $a_i$ . Furthermore each subset  $A_i$  of the partition will correspond to a worker who is on duty during the work periods that represent the elements of  $A_i$ .

We set the number of days to D=mT+3m and the number of workers to N=m. Next, define  $r^d=0$  if  $d=\sum_{i\in [k]}(a_i+1)$  for some  $1\leq k\leq 3m$  and  $r^d=1$  otherwise. Therefore, the requests are a unary encoding of the values  $a_i$  with a 0-request day serving as a separator between two consecutive values. Finally, we set  $U_w=T$ ,  $l_w=\lceil \frac{1}{4}T\rceil$ , and of course  $U_o=u_w=u_o=D$  and  $l_o=1$ . An example of such a reduction is given in Figure 6.

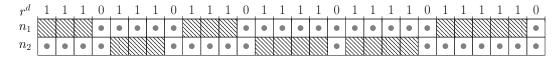


Figure 6: Given a 3-Partition Problem instance with m=2 and  $A=\{3,3,3,4,4,5\}$ , consider the DODOSP instance with the (exact) request shown in the figure as well as  $U_w=T=11$  and  $l_w=\lceil \frac{1}{4}T\rceil=3$ . Any solution to this DODOSP instance corresponds to a solution of the original problem. The shown solution correspond to the partition  $\{3,3,5\},\{3,4,4\}$ .

Any solution to the partition instance naturally translates to a feasible schedule. Given such a partition, we identify worker  $n_i$  with the set  $A_i$ . The elements of  $A_i$  correspond to three work periods. During these work periods  $n_i$  is on duty while having all other days off. Since we are given a feasible partition, the upper bound  $U_w$  is respected. Furthermore, the lower bound  $l_w$  is respected given that we consider the restricted version of the 3-Partition Problem and therefore  $a_i \geq \lceil \frac{1}{4}T \rceil = l_w$  for all i.

To prove that the converse holds as well, namely that each feasible schedule can be transformed into a feasible partition, we assume that we are given a solution to the DODOSP instance and claim the following:

**Claim.** Every interval of days with request 1 representing a value  $a_i$  will be covered by a single worker.

Proof of Claim. Assume the claim would not hold, i.e., there exists an interval representing a value  $a_i$  during which at least two workers  $n_{j_1}$  and  $n_{j_2}$  work. Denote by  $d_1 = \sum_{j < i} (a_j + 1)$  and  $d_2 = \sum_{j \le i} (a_j + 1)$  the day immediately before and immediately after the encoding of  $a_i$ . Both  $n_{j_1}$  and  $n_{j_2}$  do not work on those two days since the request for those days is 0. Hence, both of them work at least  $l_w$  days in between  $d_1$  and  $d_2$ . This is a contradiction since the total request in this time period is  $a_i < \frac{1}{2}T \le 2 \cdot l_w$ .

From the bounds of the DODOSP, we can only deduce that the number of days a single worker works is at most  $U_w = T$ , but since the total request is  $\sum_{i=1}^{3m} a_i = mT = NU_w$ , this bound has to be tight for all workers in a feasible schedule. Therefore, a feasible schedule induces a partition of A into sets  $A_i$  with sum T each. Finally, we have to prove that each  $A_i$  has exactly three elements. This is a direct consequence of  $\frac{1}{4}T < a_i < \frac{1}{2}T$ .

The size of the constructed instance and all occurring numbers are polynomially bounded in m since  $S \leq q(m)$  is bounded. Therefore, the reduction is polynomial.

This proof can be adapted to other special cases whenever we have some proper encoding of the values  $a_i$ . This does not have to be a simple unary encoding. We will consider different encodings for other special cases in Lemmata 11 and 12.

**Theorem 8.** The DODOSP is NP-complete, even if  $r_l^d = r_u^d$  for all  $d \in [D]$  and if there is only one nontrivial global bound and one nontrivial local bound.

*Proof.* We have already proven this in the case that  $U_w$  and  $l_w$  are nontrivial. The proofs for the special cases in which  $l_o$  or  $U_o$  are nontrivial work similarly, we just have to adapt the requests and the bounds.

Fix an instance  $\mathcal{I}$  of the RESTRICTED 3-PARTITION PROBLEM. If  $U_w$  and  $l_o$  are nontrivial, we represent the values  $a_i$  of  $\mathcal{I}$  by off-periods. Thus the separator days that had a request of 0 in the proof of Lemma 7 will now have a request of N, while the request on all other days is N-1.

Again, we set  $l_o = \lceil \frac{1}{4}T \rceil$ . Since for each worker the number of days off should sum up to T, we set  $U_w = D - T$ . Here we use the fact that in the proof of Lemma 7 all global bounds are fulfilled with inequality.

The case in which  $U_o$  and  $l_o$  are nontrivial is symmetric to the case in which  $U_w$  and  $l_w$  are nontrivial. Hence, we again use requests of N and N-1, set  $U_o=T$ , and  $l_o=\lceil \frac{1}{4}T\rceil$ .

Finally, in the case that  $U_o$  and  $l_w$  are nontrivial, we again represent the values  $a_i$  of  $\mathcal{I}$  by work periods. We therefore use requests of 0 and 1 to build an equivalent instance of the DODOSP. Furthermore, we set  $U_o = D - T$  and  $l_w = \lceil \frac{1}{4}T \rceil$ .

As the proof suggests, the problem remains NP-hard, even if we add some further restrictions on the requests, such as being from  $\{0,1\}$  or from  $\{N-1,N\}$ . We can also conclude the complexity of the CYCLIC DODOSP.

**Corollary 9.** The CYCLIC DODOSP is NP-complete, even if  $r_u^d = r_u^d$  for all  $d \in [D]$  and if there is only one nontrivial global bound and one nontrivial local bound.

*Proof.* In the proof of Theorem 8, the final day of the scheduling period has request 0 or N depending on whether  $l_w$  or  $l_o$  is nontrivial. Thus, in the cyclic version the two periods representing  $a_1$  and  $a_{3m}$  are also separated. Therefore, we can apply the same proof.

## 3.2 Partially bounded requests

So far we only reduced the 3-Partition Problem to the DODOSP with exact requests which then implied NP-completeness for the general DODOSP. However, if either one of  $r_l^d$  and  $r_u^d$  is trivial, meaning the requests are bounded by only one side, then the problem is not NP-hard in exactly the same cases as the general DODOSP.

In this subsection, we will determine the computational complexities for all special cases that arise from fixing certain bounds to their trivial values. Most of these can be derived from the general case, but we have to analyze some special cases. An overview of the complexities is given in Table 2.

**Lemma 10.** The DODOSP is NP-complete, even if only  $r_u^d$ ,  $U_o$  and one of the local lower bounds  $l_w$  or  $l_o$  is given.

*Proof.* Consider the instances constructed in Theorem 8. We keep  $r_u^d = r^d$ , but set  $r_l^d = 0$  for all  $d \in [D]$ . By construction, we have

$$\sum_{d \in [D]} N - r_u^d = U_o \cdot N.$$

Hence there cannot be a day d on which fewer than  $r_u^d$  workers work. Thus, the set of feasible schedules did not change and we still have a reduction from the RESTRICTED 3-PARTITION PROBLEM.

**Lemma 11.** The DODOSP is NP-complete even if only  $r_u^d, U_w, u_o$ , and  $l_w$  are given.

*Proof.* We follow the proof of Lemma 7, but use a slightly more complex encoding of the values  $a_i$ . Without loss of generality let T be divisible by 4. We set  $l_w = \frac{T}{4}$  and  $u_o = \frac{T}{2}$ . This, in particular, implies  $l_w < a_i < 2l_w = u_o$  for all i.

The encoding of  $a_i$  consists of the following sequence:  $u_o$  times zero,  $l_w$  times N,  $l_w$  times one,  $l_w$  times zero,  $l_w$  times (N-1),  $a_i$  times one; after that we repeat the sequence in reversed order by adding  $l_w$  times (N-1),  $l_w$  times zero,  $l_w$  times one,  $l_w$  times N and finally  $u_o$  times zero (see Figure 7).

$$\underbrace{0 \quad \cdots \quad 0}_{u_0} \quad \underbrace{\begin{bmatrix} l_w \times N \end{bmatrix} \quad \begin{bmatrix} l_w \times 1 \end{bmatrix} \quad \begin{bmatrix} l_w \times 0 \end{bmatrix} \quad \begin{bmatrix} l_w \times N-1 \end{bmatrix}}_{a_i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \end{matrix}}_{a_i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{matrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\ \end{bmatrix}}_{i} \quad \underbrace{\begin{matrix} \vdots \\ \vdots \\ \vdots \\$$

Figure 7: Sequence of values  $r_u$  to encode  $a_i$  (up to a mirrored part to the right). The rectangles represent the first to fourth block of that sequence.

We claim the following:

**Claim.** In any feasible schedule for such a sequence, there is one "special" worker who works on  $4l_w + a_i$  many days and all other workers work on  $4l_w$  many days. There always exists a schedule that fulfills the local bounds.

We can put the encodings of the values  $a_i$  one after another and merge trailing and leading zeros of consecutive encodings. This way, the encoding sequences are still independent and do not restrict each other. According to the claim, we define  $U_w = 3m \cdot 4l_w + T$  and we use the same arguments as in the general proof.

It remains to prove the claim.

Proof of Claim. We call the subsequence from position  $u_o + 1$  to position  $u_o + l_w$  the first block of that sequence. This subsequence contains exactly the  $l_w$  positions with request N (in the left half of the sequence). Similarly we denote by the second block the following  $l_w$  positions with request one, by the third block the following  $l_w$  positions with request zero and by the fourth block the  $l_w$  positions with request N-1 (cf. Figure 7). The sequence of  $a_i$  ones will be abbreviated to  $a_i$ .

No worker works in the third block. Due to the leading zeros and  $u_o$ , every worker is on duty on the first day of the first block. Therefore, by  $l_w$  every worker is on duty for the entire first block. Whichever worker works on any day of the second resp. the fourth block has to work on the first day of the second resp. the last day of the fourth block  $(l_w)$ . Hence at most one worker works in the second block and at most N-1 workers work in the fourth block. Again considering  $u_o$ , we conclude that exactly one worker  $n_1^*$  is on duty for the entire second block while all other N-1 workers are on duty for the entire fourth block. Additionally,  $n_1^*$  has to work the first  $l_w$  days of  $a_i$ . The same holds for the four blocks after  $a_i$ . Thus there is also a special worker  $n_2^*$  who works in the second to last block (the block with  $r_u = 1$ ) and works the last  $l_w$  days of  $a_i$ . Since  $a_i < 2l_w$ , both  $n_1^*$  and  $n_2^*$  are on duty in the middle of  $a_i$ , hence they are the same worker. Thus we have shown that every feasible schedule has one special worker who works for  $4l_w + a_i$  days (first and second block,  $a_i$ , second to last and last block) while all other workers work for  $4l_w$  days (first, fourth, fourth to last and last block). Finally, note that this indeed gives a feasible schedule.

As already mentioned this ends the proof of the lemma.

**Lemma 12.** The DODOSP is NP-complete, even if only  $r_u^d, U_w, u_o$  and  $l_o$  are given.

*Proof.* We again follow the proof of Theorem 8 and assume w.l.o.g. T to be even, otherwise we can multiply all values  $a_i$  and thus T with two. This time we set  $l_o = T$  and  $u_o = \frac{3}{2}T + 1$ . The encoding of  $a_i$  is given by Figure 8. It suffices to prove the following:

$$\underbrace{0 \ \cdots \ 0}_{u_o = \frac{3}{2}T+1} N \quad 1 \quad \underbrace{0 \ \cdots \ 0}_{0 \ N-1} \underbrace{0 \ \cdots \ 0}_{N-1} \quad \underbrace{1 \ \cdots \ 0}_{0 \ \cdots \ 0} \quad \underbrace{1 \ \cdots \ 1}_{a_i} = \underbrace{1 \ \cdots \ 1}_{a_i}$$
 line of symmetry

Figure 8: Sequence of values  $r_u$  to encode  $a_i$  (up to a mirrored part to the right)

**Claim.** In any feasible schedule for such a sequence, there is one "special" worker who works on  $4 + a_i$  many days and all other workers work on 4 days. There always exists a schedule that fulfills the local bounds.

Proof of Claim. Due to the leading zeros, everybody works on the day where  $r_u = N$ . Let  $d_1$  and  $d_2$  be the first two days where  $r_u = 1$  (meaning  $d_2$  is the first day of  $a_i$ ). There are  $u_o$  days strictly in between  $d_1$  and  $d_2$  whose  $r_u$  values sum up to N-1. Thus there has to be a special worker  $n_1^*$  who works on  $d_1$ . Furthermore, all other workers have to work on the day where  $r_u = N-1$ , and  $n_1^*$  is on duty on  $d_2$ . Since  $\frac{1}{2}T + a_i < T = l_o$ , none of the other workers can work on any of the  $a_i$ -days.

The same situation holds for the mirrored sequence to the right of  $a_i$  with a special worker  $n_2^*$ . Since none of the other workers may work at the end of  $a_i$ , we again know that  $n_1^* = n_2^*$ . Thus there is one special worker who works on the first and the last day of  $a_i$ . Since  $a_i < \frac{1}{2}T < l_o$ , this worker cannot have a break in between these days. In other words  $n_1^* = n_2^*$  is on duty on all  $a_i$  days. Furthermore, every worker works exactly two days before and two days after  $a_i$ . The described schedule is indeed feasible and thus the claim holds.

Finally we set  $U_w = 3m \cdot 4 + T$  and the statement follows by concatenating the encodings of the  $a_i$  and merging trailing and leading zeros.

**Theorem 13.** If the requests are bounded only by one side, the theoretical complexity of the DODOSP is as given in Table 2.

*Proof.* Note that a lower bound on the requests is an upper bound on the number of workers who do not work on a given day. Therefore, if only  $r_l^d$  is given, we receive the same results as in the case that only  $r_u^d$  is given, but with w and o swapped.

In Sections 4 and 5 we present polynomial algorithms for the cases that only upper bounds or only local bounds are given. Since  $r_l^d$  can be chosen arbitrary, they also apply to this case. If only  $U_w, u_o, l_o$  and  $l_w$  are given, the schedule in which no worker works at all is feasible. Thus, this case is trivially in P. The remaining cases are considered in Lemmata 10 to 12.

request bound	additional b	complexity	proof	
$r_u^d$	$U_w, U_o, u_w, u_o$	or any subset	P	Theorem 14
$r_u^d$	$u_w, u_o, l_w, l_o$	or any subset	P	Theorem 22
$r_u^d$	$U_w, u_w, l_w, l_o$	or any subset	P	Theorem 13
$r_u^d$	$U_w, u_o, l_w$	or any superset	NP-compl.	Lemma 11
$r_u^d$	$U_o, l_x$ with $x \in \{o, w\}$	or any superset	NP-compl.	Lemma 10
$r_u^d$	$U_w, u_o, l_o$	or any superset	NP-compl.	Lemma 12

Table 2: Complexity of the DODOSP if requests are bounded only by one side. A lower bound on the requests is an upper bound on the number of workers who do not work on a given day. Therefore, we receive the same results for  $r_l^d$ , but with w and o swapped.

# 4 Upper bounds

The primary goal of this section is to prove the following theorem:

**Theorem 14.** The Only Upper Bounded Days On Days Off Scheduling Problem can be decided in  $\mathcal{O}(D^2)$  time.

Because the UDODOSP can be solved by an easier and faster algorithm if restricted to instances with exact requests, we first consider this case in Section 4.1. In Section 4.2, we then adapt the arguments to the general UDODOSP.

#### 4.1 Exact requests

In this subsection, we assume that  $r^d := r_l^d = r_u^d$ . We exploit that all upper bounds "prefer" that workers swap between work and off periods as often as possible to evenly distribute the work days among the workers. If we ensure such an even distribution, it is sufficient to consider the total workload of certain consecutive days. We therefore define  $R^d = \sum_{i=1}^d r^i$  to be the partially summed requests for  $0 \le d \le D$ .

There exist some easy-to-check inequalities which must be fulfilled by feasible instances.

**Lemma 15.** Any feasible instance of the UDODOSP with exact requests must fulfill the following four inequalities:

$$R^D \le N \cdot U_w \tag{req. } U_w)$$

$$N \cdot D - R^D \le N \cdot U_o \tag{req. } U_o)$$

$$R^{d+u_w} - R^{d-1} \le N \cdot u_w \qquad \forall 1 \le d \le D - u_w \qquad (req. \ u_w)$$

$$R^{d+u_o} - R^{d-1} \ge N \qquad \qquad \forall 1 \le d \le D - u_o \qquad \text{(req. } u_o)$$

*Proof.* Given a feasible instance of the UDODOSP, we fix some feasible schedule. In this schedule each worker is at most  $U_w$  times on duty. Since feasibility implies that the total demand equals the total number of work days, summing over all workers gives Eq. (req.  $U_w$ ).

Furthermore each worker has at most  $U_o$  days off, in other words they are at least  $D - U_o$  times on duty. Again summing over all workers gives Eq. (req.  $U_o$ ).

Fix any  $1 \le d \le D - u_w$ . In our feasible schedule, no worker works on all days in the interval  $d, \ldots, d + u_w$ , meaning each worker is on duty at most  $u_w$  times. Thus, the total request of this interval is at most  $N \cdot u_w$ , proving Eq. (req.  $u_w$ ).

Fix any  $1 \leq d \leq D - u_o$ . In our feasible schedule no worker can take the entire interval  $d, \ldots, d + u_o$  off, meaning each worker is on duty at least once. Thus, the total request of this interval is at least N, proving Eq. (req.  $u_o$ ).

These inequalities are not only necessary but sufficient to characterize feasible instances of the UDODOSP with exact requests. We will show this by stating Algorithm 1 which, given an instance fulfilling the inequalities in Lemma 15, returns a corresponding feasible schedule.

In the algorithm, we expand the set of workers  $\{n_j : j \in [N]\}$  to a set of worker-representatives  $\{n_j : j \in \mathbb{N}\}$  and identify each worker  $n_i$  with the representatives  $n_{i+k \cdot N}$  for  $k \in \mathbb{N}$ . Each representative is on duty at most once. Finally, a worker is on duty whenever one of its representatives is. Figure 9 provides a visualization of how the algorithm operates.

#### Algorithm 1: UDODOSP

```
input: UDODOSP instance which fulfills the inequalities in Lemma 15 output: A feasible schedule [D] \times [N] \rightarrow \{ON, OFF\}.
```

Identify  $n_j = n_{j+N}$ 

for  $day d = 1, \dots, D$  do

return computed schedule

assign exactly the workers with representatives  $n_j$  for  $j = R^{d-1} + 1, \dots, R^{d-1} + r^d$  to work on day d

-

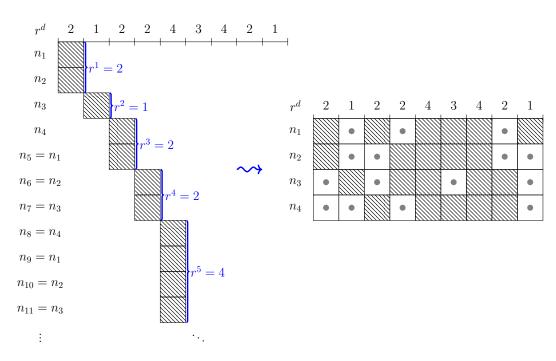


Figure 9: Illustration of Algorithm 1. The assignment of work days to representatives is shown on the left. The corresponding schedule is shown on the right.

The idea behind the algorithm is to assign the work days in a cyclic and the most evenly

distributed way possible. This is also reflected in the two lemmas below.

Note that two representatives of the same worker cannot be assigned to work on the same day, since the request of each day is at most N.

**Lemma 16.** Assume that a worker  $n_j$  is on duty on days  $d_1$  and  $d_2$ , with  $d_1 < d_2$ . Then all other workers have to work on at least one of the days  $d_1, \ldots, d_2$ . Furthermore, if the representatives  $n_{j+kN}$  and  $n_{j+(k+1)N}$  are working on days  $d_1$  and  $d_2$ , every other worker has a representative  $n_i$  with j + kN < i < j + (k+1)N who works on one of these days.

*Proof.* The assigned representatives form a consecutive sequence throughout the algorithm. We may assume that the representatives of  $n_j$  that work on days  $d_1$  and  $d_2$  are consecutive. Thus they are the representatives  $n_{j+kN}$  and  $n_{j+(k+1)N}$  for some  $k \in \mathbb{N}$ . Due to the consecutiveness, the representatives  $n_{j+kN+l}$  with  $1 \leq l < N$  are on duty on a day from  $d_1$  to  $d_2$ , inclusive. Consequently, all other workers must also work at least one day.

An informal rephrasing of this lemma would be: before a worker is an duty another time, all other workers must have been on duty at least once more.

**Lemma 17.** The number of times two workers were on duty differs by at most one at any point throughout the algorithm. The same holds for the number of days off that two workers have had.

*Proof.* It is enough to show that if a worker n has worked w days at some point throughout the algorithm, then all the other workers have worked at least w-1 many days. This is true from the beginning up to and including the first day n works. Now assume it is true after we assign n to work on some day  $d_1$  for the w'th time, meaning all other workers have been on duty at least w-1 times. Due to Lemma 16, by the time n works another time, all other workers must have also been on duty once more, thus, the inequality still holds.

Note that, as a direct consequence of Lemma 17, the total number of days off for two workers differs by at most one as well.

**Theorem 18.** The schedule computed by Algorithm 1 satisfies the work requests and respects upper bounds  $u_w$ ,  $u_o$ ,  $U_w$  and  $U_o$ .

*Proof.* Assume the schedule that is computed by Algorithm 1 is not feasible and therefore violates a bound. We will show that this leads to a contradiction concerning the assumption that the instance fulfills the inequalities in Lemma 15.

 $U_w$ : Assume the computed schedule violates  $U_w$ . Denote by  $w_n$  the number of days worked by worker n. Since  $U_w$  is violated, there exists a worker n' for which  $w_{n'} > U_w$ . Given Lemma 17, this implies that  $w_n \geq U_w$  for all workers  $n \in [N]$ . Since the computed schedule fulfills all requests exactly, it holds that

$$\sum_{d=1}^{D} r^d = \sum_{n \in [N]} w_n > N \cdot U_w.$$

This contradicts Eq. (req.  $U_w$ ) from Lemma 15.

 $U_o$ : Assume the computed schedule violates  $U_o$ . With similar arguments as before, we can show that

$$ND - \sum_{d=1}^{D} r^d = \sum_{d=1}^{D} N - r^d = \sum_{n \in [N]} o_n > N \cdot U_o,$$

where  $o_n$  denotes the number of days off that worker n has. This contradicts Eq. (req.  $U_o$ ) from Lemma 15.

 $u_w$ : Assume the computed schedule violates  $u_w$ . Hence, there exists a worker n' and an interval  $\mathcal{D}' = \{d_1, \ldots, d_{u_w+1}\}$  of  $u_w + 1$  consecutive days on which worker n' works.

Lemma 16 applied to all consecutive pairs of days within  $\mathcal{D}'$  implies that every other worker is on duty at least  $u_w$  times in the interval  $\mathcal{D}'$  (note that each application of Lemma 16 gives us a different representative of each worker). The situation is outlined in Figure 10.

Hence, the summed request of days  $d_1, \ldots, d_{u_w+1}$  is

$$\sum_{d=d_1}^{d_{u_w+1}} r^d > N \cdot u_w$$

which contradicts Eq. (req.  $u_w$ ) from Lemma 15.

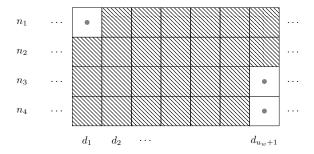


Figure 10: Illustration of the case where  $n_2$  violates the bound  $u_w = 6$  with N = 4.

 $u_o$ : Assume the computed schedule violates  $u_o$ . Hence, there exists a worker n' and an interval  $\mathcal{D}' = \{d_1, \dots, d_{u_o+1}\}$  of  $u_o + 1$  consecutive days on which worker n' takes off.

We claim that every other worker  $n \neq n'$  can work at most one of the days in  $\mathcal{D}'$ . Assume there would be a worker n'' that works at least two days  $d', d'' \in \mathcal{D}$  with d' < d''. However, in this case Lemma 16 implies that worker n' works one of the days in  $\{d', \ldots, d''\} \subseteq \mathcal{D}'$ , which is a contradiction.

Hence, the total number of work days of all workers during  $\mathcal{D}'$  is at most N-1 and therefore

$$(u_o + 1) N - \sum_{d=d_1}^{d_{u_o+1}} r^d \ge (u_o + 1) N - (N-1) = u_o \cdot N + 1.$$

Again this results in a contradiction with respect to Eq. (req.  $u_o$ ) from Lemma 15.

Therefore, the algorithm returns a feasible schedule.

As observed before, this result implies that we can characterize instances with a feasible solution and determine feasibility in linear time.

**Corollary 19.** An instance of the UDODOSP has a feasible solution if and only if it fulfills the inequalities in Lemma 15. Thus, feasibility can be checked in  $\mathcal{O}(D)$ .

Remark. Note that the running time of Algorithm 1 depends on the chosen output format. If an  $N \times D$ -matrix is returned, the running time is not polynomial but pseudopolynomial since N is super-polynomial in the input size. However, as already shown in Section 3, polynomial representations for a schedule exist. In this case, we can choose the cyclic interval  $\left(R^{d-1}, R^d\right] \mod N$  for each day as an encoding of the workers on duty. Hence, if we want to determine if a worker n works on day d, we must check if  $n \in \left(R^{d-1}, R^d\right] \mod N$ , which is equivalent to  $1 \le (n - R^{d-1} \mod N) \le r^d$ .

Note that this encoding does not work for the general DODOSP. More precisely, a feasible instance of the DODOSP may become infeasible if we restrict ourselves to solutions in which the workers on duty form a cyclic interval for each day.

#### 4.2 Request intervals

The results from Section 4.1 can be generalized to the case where we have request intervals for each day.

The goal is to compute the exact number  $w^d$  of workers that should work on a given day, namely the value which will take the place of  $r^d$ . To this end, we demand that the values  $w^d$  fulfill the inequalities in Lemma 15, together with  $r_l^d \leq w^d \leq r_u^d$ . Similar to before we again work with partial sums. Therefore, for  $0 \leq d \leq D$ , we define

$$W^d = \sum_{i=1}^d w^i.$$

This results in the following corollary of Corollary 19:

**Corollary 20.** An instance of the UDODOSP is feasible if and only if there are integers  $W^d$  for  $0 \le d \le D$  where

$$\begin{split} W^{D} - W^{0} &\leq N \cdot U_{w} \\ ND + W^{0} - W^{D} &\leq N \cdot U_{o} \\ W^{d+u_{w}} - W^{d-1} &\leq N \cdot u_{w} \\ N &\leq W^{d+u_{o}} - W^{d-1} \\ r_{l}^{d} &\leq W^{d} - W^{d-1} & \forall 1 \leq d \leq D - u_{o} \\ r_{l}^{d} &\leq W^{d} - W^{d-1} \leq r_{u}^{d} & \forall 1 \leq d \leq D. \end{split}$$

Note that any solution can be shifted such that  $W^0 = 0$ .

**Proposition 21.** The integral inequality system described in Corollary 20 can be solved in  $\mathcal{O}(D^2)$ .

*Proof.* Consider the graph G = (V, E, c) with

$$V = \{v^{d} : 0 \le d \le D\},$$

$$E = \{e_{U_{w}} = (v^{0}, v^{D}), e_{U_{o}} = (v^{D}, v^{0})\}$$

$$\cup \{e_{u_{w}}^{d} = (v^{d-1}, v^{d+u_{w}}) : 1 \le d \le D - u_{w}\}$$

$$\cup \{e_{u_{o}}^{d} = (v^{d+u_{o}}, v^{d-1}) : 1 \le d \le D - u_{o}\}$$

$$\cup \{e_{r_{l}}^{d} = (v^{d}, v^{d-1}), e_{r_{u}}^{d} = (v^{d-1}, v^{d}) : 1 \le d \le D\}$$

and

$$\begin{split} c\left(e_{U_{w}}\right) &= N \cdot U_{w}, & c\left(e_{U_{o}}\right) &= N \cdot \left(U_{o} - D\right), \\ c\left(e_{u_{w}}^{d}\right) &= N \cdot u_{w}, & c\left(e_{u_{o}}^{d}\right) &= -N, \\ c\left(e_{r_{l}}^{d}\right) &= -r_{l}^{d}, & c\left(e_{r_{u}}^{d}\right) &= r_{u}^{d}. \end{split}$$

An example of such a graph is given in Figure 11. Recall that a function  $\pi$  on the vertices of a digraph is called a feasible potential, if  $\pi(v) + c(v, w) - \pi(w) \ge 0$  for each edge (v, w). The values  $W^d$  fulfill the inequality system given in Corollary 20 if and only if  $\pi\left(v^d\right) := W^d$  for  $0 \le d \le D$  is a feasible potential of G. Each edge of the graph corresponds to one inequality. Since the number of vertices and the number of edges in G is linear in D, the Moore-Bellman-Ford algorithm [Bel58; For56; Moo59] can compute a feasible integral potential or detect a negative cycle in  $\mathcal{O}\left(D^2\right)$ .  $\square$ 

Remark. Using a Las Vegas randomized algorithm to check for negative cycles, the problem can be solved in  $\mathcal{O}\left(D\log\left(D\right)^8\log\left(DN\right)\right)$  (with high probability) [BNW22] and thus in near-linear time with respect to the input length.

The integrality and even further that the system is TDI can also be shown using a theorem by Ghouila-Houri, which was proven in [Gho62]. Another consequence

Together Corollary 20 and Proposition 21 give us the desired theorem.

**Theorem 14.** The Only Upper Bounded Days On Days Off Scheduling Problem can be decided in  $\mathcal{O}(D^2)$  time.

Remark. The theorem only considers the decision problem, but we can also compute a feasible schedule if one exists. We first compute  $W^d$  for  $0 \le d \le D$  with  $W^0 = 0$ . From then on, we continue as in Section 4.1. Again the time complexity depends on the representation of the schedule.

#### 5 Local bounds

In this section, we consider the LDODOSP in where there are no bounds on the total number of days a single worker is on duty. Since, to our knowledge, solving this problem is not any easier for instances with exact requests compared to general instances, we immediately deal with the general case. More precisely, in Section 5.1, we show that the LDODOSP can be decided in polynomial time. In Section 5.2 we show that most of the proofs from Section 5.1 can actually be applied a generalized version of the LDODOSP.

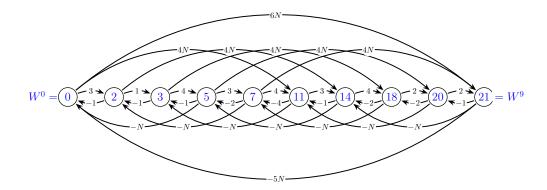


Figure 11: The graph which results from requests  $\{1,3\},\{1,1\},\{1,4\},\{2,3\},\{4,4\},\{1,3\},\{2,4\},\{2,2\},\{1,2\}$  and bounds  $u_w=4,u_o=2,U_w=6$  and  $U_o=4$  is shown in black. For N=4 one can check that the blue values in the vertices form a feasible potential and thus imply feasibility of the DODOSP instance. This potential corresponds to the demands given in Figure 9.

## 5.1 Complexity

The main goal of this section is to prove the following:

**Theorem 22.** The Local Days On Days Off Scheduling Problem can be decided in  $\mathcal{O}(D^2)$  time.

The key idea behind the proof is to consider the work periods and off periods, from a perspective similar to the one used in [Tho95]. Instead of directly computing a schedule, we first determine how many work periods start and end on a given day. More precisely we introduce variables  $S_w^d$  and  $T_w^d$  for all  $d \in [D]$ . By  $S_w^d$  we denote the number of work periods that start on some day d', with  $d' \leq d$ . By  $T_w^d$ , we denote the number of working periods that terminate before day d. Put another way, we count the work periods whose last day is at the latest d-1. An example is depicted in Figure 12.

**Definition 23.** Given D days and N workers, integers  $\left(S_w^d, T_w^d\right)_{d \in [D]}$  are called *period counters* if there exists some work schedule for N workers on D days such that for each  $d \in [D]$ 

- on the first d days  $S_w^d$  work periods begin and
- on the first d-1 days  $T_w^d$  work periods end.

We say that the period counters represent such a schedule. Note that, in general, this schedule can be any map  $[N] \times [D] \to \{ON, OFF\}$  and does not have to fulfill any bounds or requests.

One could also use  $S_o^d$  and  $T_o^d$  to count the off periods instead. Even though it is not necessary to count both, it is sometimes convenient to do so. Therefore, we begin by pointing out the connection between  $\left(S_w^d, T_w^d\right)_{d \in [D]}$  and  $\left(S_o^d, T_o^d\right)_{d \in [D]}$ .

**Lemma 24.** Given (work) period counters  $(S_w^d, T_w^d)_{d \in [D]}$  and corresponding off period counters  $(S_o^d, T_o^d)_{d \in [D]}$  the following equations hold:

$$S_w^1 + S_o^1 = N$$

$$S_o^d = T_w^d + S_o^1 = T_w^d + N - S_w^1 \qquad \forall 1 \le d \le D$$

$$T_o^d = S_w^d - S_w^1 \qquad \forall 1 < d < D$$

Proof. On the first day, each worker starts either a work period or an off period.

From the second day onward a worker starts an off period if and only if that worker terminates a work period on the previous day.

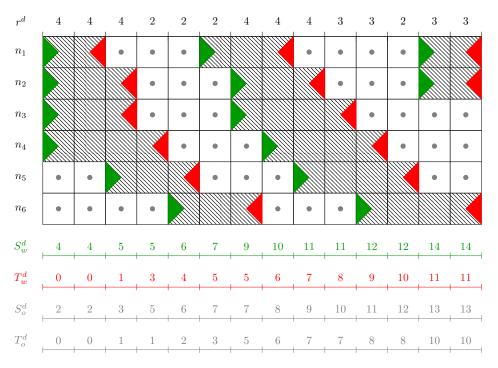


Figure 12: A schedule together with the set of period counters which represent that schedule. Note that  $T_w^d$  denotes the work periods which end on the first d-1 days. The corresponding off period counters are given in gray.

Similarly, from the second day onward, a worker starts a work period if and only if this worker terminates an off period on the previous day.

Of course not every sequence of numbers forms period counters. For example, there cannot be more than N periods starting on the same day. Some similar conditions are captured in Lemma 25.

**Lemma 25.** Period counters  $\left(S_w^d, T_w^d\right)_{d \in [D]}$  fulfill  $T_w^1 = 0$  and the following inequalities for all  $1 \le d \le D - 1$ :

$$S_w^d \le S_w^{d+1}, \quad T_w^d \le T_w^{d+1}$$
 (non decr.)

$$T_w^{d+1} \le S_w^d$$
 (pos. work)

$$T_w^{d+1} \le S_w^d$$
 (pos. work)  

$$S_w^{d+1} \le T_w^d + N$$
 (pos. off)

*Proof.* No work period can terminate before day 1, hence  $T_w^1 = 0$ . On each day a non-negative number of work periods start and terminate. As  $S_w^d$  and  $T_w^d$  are partial sums of these non-negative values, they must be non-decreasing and therefore Eq. (non decr.) must hold.

Each work period has a positive length. Since each work period counted by  $T_w^{d+1}$  ends on day d or earlier, this work period has to start on day d or earlier and is therefore also counted by  $S_w^d$ . This proves Eq. (pos. work).

For the same reason, it is the case that  $T_o^{d+1} \leq S_o^d$ . Together with Lemma 24 this gives us Eq. (pos. off). 

So far, we have considered period counters that might correspond to any assignment  $[N] \times [D] \to \mathbb{R}$ {ON, OFF}. However, we are primarily interested in feasible assignments.

**Definition 26.** Given an LDODOSP instance, period counters  $\left(S_w^d, T_w^d\right)_{d \in [D]}$  are called *feasible* if they represent a feasible schedule.

**Lemma 27.** Feasible period counters  $(S_w^d, T_w^d)_{d \in [D]}$  fulfill the following inequalities:

$$T_{w}^{l_{w}} = 0, \quad S_{w}^{D-l_{w}+1} = S_{w}^{D}$$
 (bound.  $l_{w}$ )
$$T_{w}^{d+l_{w}} \leq S_{w}^{d}$$
  $\forall 1 \leq d \leq D - l_{w}$  (count.  $l_{w}$ )
$$S_{w}^{d} \leq T_{w}^{d+u_{w}}$$
  $\forall 1 \leq d \leq D - u_{w}$  (count.  $u_{w}$ )
$$S_{w}^{1} = S_{w}^{l_{o}}, \quad T_{w}^{D-l_{o}+1} = T_{w}^{D}$$
 (bound.  $l_{o}$ )
$$S_{w}^{d+l_{o}} - N \leq T_{w}^{d}$$
  $\forall 1 \leq d \leq D - l_{o}$  (count.  $l_{o}$ )
$$T_{w}^{d} + N \leq S_{w}^{d+u_{o}}$$
  $\forall 1 \leq d \leq D - u_{o}$  (count.  $u_{o}$ )
$$r_{l}^{d} \leq S_{w}^{d} - T_{w}^{d} \leq r_{u}^{d}$$
  $\forall 1 \leq d \leq D - u_{o}$  (count.  $u_{o}$ )

*Proof.* If a work period would terminate before day  $l_w$  or start after day  $D - l_w + 1$  then it would be shorter than  $l_w$  and thus Eq. (bound.  $l_w$ ) must hold. Only the work periods that start on day d or earlier are allowed to terminate before day  $d + l_w$ , which gives Eq. (count.  $l_w$ ). Similarly, each work period that starts at day d or earlier must terminate before day  $d + u_w$ , which immediately gives Eq. (count.  $u_w$ ). With the same arguments we obtain Eq. (bound.  $l_o$ ) as well as

$$T_o^{d+l_o} \leq S_o^d$$
 and  $S_o^d \leq T_o^{d+u_o}$ .

Using Lemma 24 this gives us Eq. (count.  $l_o$ ) and Eq. (count.  $u_o$ ).

For Eq. (count. req.) it is sufficient to observe that the number of workers that are on duty on day d equals the number of work periods that contain day d. These are exactly the work periods that start on day d or earlier but do not terminate before day d.

Next, we show that the inequalities stated thus far are not only necessary but also sufficient for period counters to be feasible. To prove this, we restrict ourselves to schedules with the following FIFO property: whenever a work period  $W_1$  starts earlier than another work period  $W_2$ ,  $W_1$  does not terminate later than  $W_2$ . The same should hold for off periods.

*Remark.* Note that in the general DODOSP, we cannot restrict ourselves to such schedules. For example, the instance shown in Figure 6 has no feasible schedule in which the off periods fulfill the described property.

To prove that the inequalities in Lemma 27 are not only necessary, but also sufficient for feasibility, we explicitly construct a schedule from period counters that satisfy the inequalities and show that this schedule is feasible. To this end, we again expand the set of workers  $\{n_j : j \in [N]\}$  to a set of worker-representatives  $\{n_j : j \in \mathbb{N}\}$  and identify each worker  $n_i$  with the representatives  $n_{i+k\cdot N}$  for  $k \in \mathbb{N}$ . Each representative will obtain at most one work period. Finally, as before, a worker is on duty whenever one of its representatives is. The construction is given in Algorithm 2. For the period counters in Figure 12, Algorithm 2 would compute exactly the depicted schedule.

#### **Algorithm 2:** LDODOSP period counters to schedule

input : LDODOSP instance, period counters which fulfill the inequalities in Lemma 25 and Lemma 27

**output:** A feasible schedule  $[D] \times [N] \to \{ON, OFF\}$ 

Identify  $n_i = n_{i+N}$ 

for  $day d = 1, \dots, D$  do

assign worker/representative  $n_j$  to work if and only if  $T_w^d < j \leq S_w^d$ 

return computed schedule

**Lemma 28.** The output of Algorithm 2 has the following properties:

- 1. Only the representatives  $n_1$  to  $n_{SD}$  are assigned to work.
- 2. Each of the representatives  $n_1$  to  $n_{S_m^D}$  is assigned exactly one work period.
- 3. The i-th work period of a worker equals the work period associated with their i-th representative.

Proof. Property 1 follows from  $T_w^0=0$  and Eq. (non decr.). Assume for some  $1\leq j\leq S_w^D$  that the representative  $n_j$  is never assigned to work. Then for all  $d\in [D]$ , either  $j\leq T_w^d$  or  $j>S_w^d$  hold. Let d' be the largest d with  $j>S_w^d$ . This is well defined, since  $S_w^d$  is non-decreasing in d. Because  $j \leq S_w^D$ , it follows that d' < D. Then  $j \leq T_w^{d'+1}$  must hold, which contradicts Eq. (pos. work) and consequently gives Property 2.

Again by Eq. (non decr.), the days a representative is assigned to work are consecutive. Let  $n_{j_1}$  and  $n_{j_2}$  with  $1 \le j_1 < j_2 \le S_w^D$  be representatives of the same worker. Let d be the earliest day d such that  $j_2 \le S_w^{d+1}$ . By the choice of d, the representative  $n_{j_2}$  does not work before day d+1. Furthermore:

$$j_1 \le j_2 - N \le S_w^{d+1} - N \stackrel{Eq. \text{ (pos. off)}}{\le} T_w^d.$$

Hence,  $n_{j_1}$  cannot work on any day later than d-1. Therefore the work periods of  $n_{j_1}$  and  $n_{j_2}$ are not only disjoint, but they are even separated by at least one day off and in canonical order. Thus, each work period of a representative is a distinct work period of the corresponding worker, which implies Property 3.

Note that the work period associated with representative  $n_j$  starts with the first day d that satisfies  $j \leq S_w^d$  and ends on the last day d' for which  $T_w^{d'} < j$ .

Proposition 29. The work schedule computed by Algorithm 2 satisfies the work requests and respects local bounds  $l_w$ ,  $u_w$ ,  $l_o$ , and  $u_o$ .

*Proof.* The number of workers that work on day d equals the number of representatives that work on day d. By construction, there are  $S_w^d - T_w^d$  representatives on duty, which is a feasible value by Eq. (count. req.). Next we check the local bounds.

 $l_w$  Due to Eqs. (bound.  $l_w$ ) and (non decr.) no work period terminates before day  $l_w$  and no work period starts after day  $D - l_w + 1$ . Thus work periods that start on day 1 or finish on day D cover at least  $l_w$  many days.

Fix  $1 \le d \le D - l_w$  and j such that  $n_j$  is a representative that starts working on day d+1, and therefore  $S_w^d < j$  and  $j \le S_w^{d+1}$ . This implies

$$T_w^{d+l_w} \overset{Eq. \text{ (count. } l_w)}{\leq} S_w^d < j \leq S_w^{d+1} \overset{Eq. \text{ (non decr.)}}{\leq} S_w^{d+l_w}.$$

Thus the representative  $n_i$  is on duty on day  $d+l_w$  and the corresponding work period covers at least  $l_w$  many days.

 $u_w$  Given a representative  $n_j$ , let d be the day  $n_j$  starts working. This implies  $j \leq S_w^d$ . If  $d > D - u_w$ , then the work period cannot exceed  $u_w$  days anyway. If  $d \leq D - u_w$ , we get

$$j \leq S_w^d \stackrel{Eq. \text{(count. } u_w)}{\leq} T_w^{d+u_w}.$$

Thus the representative  $n_j$  does not work on day  $d + u_w$  and therefore the corresponding work period lasts at most  $u_w$  days.

 $l_o$  By Eq. (bound.  $l_o$ ), no work period starts on days  $2, \ldots, l_o$  and no work period terminates on days  $D - l_o + 1, \ldots, D - 1$ . Hence no off period terminates before day  $l_o$  and no off period starts after day  $D - l_o + 1$ . Thus, off periods that start on the first day or finish on day D last for at least  $l_o$  days.

Fix  $1 \le d \le D - l_o$  and j such that  $n_j$  is a representative whose last work day is d and thus  $T_w^d < j$ . This implies

$$S_w^{d+l_o} \overset{Eq. \text{ (count. } l_o)}{\leq} T_w^d + N < j + N.$$

Hence the next representative of the same worker does not yet work on day  $d + l_o$ . In other words: the off period in between lasts for at least  $l_o$  days.

 $u_o$  Using Eq. (count.  $u_o$ ) for d=1 we obtain that

$$N = T_w^1 + N \le S_w^{1+u_o}.$$

Hence, at least the first N representatives and therefore all workers start their first work period no later than day  $u_o + 1$ . Therefore, all off periods that start on day 1 are sufficiently short. Let  $n_j$  be a representative that finishes working on day d-1 and therefore  $j \leq T_w^d$ . If  $d > D - u_o$ , the following off period cannot exceed  $u_o$  days anyway. If  $d \leq D - u_o$ , we get

$$j + N \le T_w^d + N \stackrel{Eq. \text{(count. } u_o)}{\le} S_w^{d+u_o}.$$

Thus the next representative  $n_{j+N}$  of the same worker starts working no later than day  $d+u_o$  and therefore the off period in between lasts at most  $u_o$  days.

The correctness of Algorithm 2 implies that the inequalities stated in Lemma 25 and Lemma 27 are not only necessary, but also sufficient. This is captured by Corollary 30.

**Corollary 30.** Period counters are feasible if and only if they satisfy the inequalities in Lemma 25 and Lemma 27.

Therefore, checking the feasibility of an LDODOSP instance is equivalent to finding an *integral* solution to the system of inequalities given by Lemma 25 and Lemma 27. Using the same techniques as in Section 4.2, we can solve this integral system of inequalities in polynomial time (or even near-linear time using randomized algorithms).

**Proposition 31.** The system of inequalities given in Lemma 25 and Lemma 27 can be solved in  $\mathcal{O}(D^2)$ .

*Proof.* Consider a graph with vertices  $s_d$  and  $t_d$  for  $1 \le d \le D$ . If we consider period counters as a potential on these vertices  $(S_w^d \text{ resp. } T_w^d \text{ is the potential of } s_d \text{ resp. } t_d)$ , each inequality can be modeled as a constraint for a feasible potential by adding an edge with an appropriate weight.

This gives  $\mathcal{O}(D)$  variables (vertices) and  $\mathcal{O}(D)$  constraints (edges). Thus, using the Moore-Bellman-Ford algorithm [Bel58; For56; Moo59], we can compute a feasible potential or find a negative cycle in  $\mathcal{O}(D^2)$ .

**Theorem 22.** The Local Days On Days Off Scheduling Problem can be decided in  $\mathcal{O}(D^2)$  time.

*Proof.* By Proposition 31 we can check in  $\mathcal{O}\left(D^2\right)$  if the system of inequalities given by Lemma 25 and Lemma 27 is feasible. Due to Corollary 30, this also determines whether the LDODOSP instance is feasible.

Remark. Using Algorithm 2 we can translate feasible period counters into a feasible schedule  $[D] \times [N] \to \{\text{ON,OFF}\}$ . Once again, the running time depends on the output format we choose. One can, as before, return the cyclic interval  $\left(T_w^d, S_w^d\right]$  for each day d instead of listing all workers that are on duty.

#### 5.2 Extensions of the LDODOSP

The way in which the LDODOSP can be solved efficiently, introduced in Section 5.1, is applicable to a more general setting.

Assume we want to find a schedule where workers work for at most 5 consecutive days, but we want to decrease this bound if the work period contains (part of) the weekend. In this case, instead of a maximal length  $u_w$  for work periods, we take as input a non-decreasing function  $f_{u_w}:[D] \to [D+1]$ , and the task is to compute a schedule in which every work period that starts on day d terminates before day  $f_{u_w}(d)$ . To solve this more general problem, we can use the same approach as before but replace each  $d+u_w$  with  $f_{u_w}(d)$ . The same applies to the three other local bounds.

Note that it is crucial for  $f_{u_w}$  to be non-decreasing. Otherwise, it would not be possible to restrict ourselves to the structured schedules from Section 5.1. However, it is an interesting question which results hold for arbitrary functions  $f_{u_w}$ .

To some extent, we can also control the number of different workers that work in a given interval. For example, a constraint that requires every worker to be on duty at least once a week can be modeled by comparing  $S_w^d$  and  $S_w^{d+7}$ . However, introducing upper bounds on the number of workers that are on duty in a given interval might again destroy the structure of our schedules.

# 6 Optimizing parameters/bounds

Until now, we have only considered the decision version of the DODOSP. However, in practice, the information that an instance is infeasible is often not enough, but one wants to relax certain constraints to make it feasible. On the other hand, besides the information that an instance is feasible, it is interesting to know if the instance remains feasible after the strengthening of some constraints. Both cases are covered if we switch to the task of optimizing a parameter.

### 6.1 Optimizing bounds

In this section we fix the number of nurses and all bounds except one, which we want to optimize. We assume that the bound that is to be optimized is  $u_w$ , all other cases can be handled analogously.

If the instance is feasible for some value of  $u_w$ , increasing  $u_w$  preserves feasibility. Thus, our goal is to find the smallest value of  $u_w$  such that the instance is feasible or to decide that no such value exist. To this end, we can use binary search on the interval  $[1, \ldots, D]$  which relies on a black box algorithm that checks feasibility for a given value of  $u_w$ . If the instance is feasible, we decrease  $u_w$ , otherwise we increase it.

### 6.2 Optimizing N

Since an instance of the DODOSP can be infeasible due to the value of N being either too large or too small, both minimizing and maximizing N appear reasonable. Given that both can be achieved using the same techniques, we only consider minimizing the number of workers. Since optimizing N is as least as hard as checking feasibility, we restrict ourselves to the special cases of the DODOSP we can solve in polynomial time: the UDODOSP and the LDODOSP.

For both special cases, we can classify feasible instances by checking if a negative cycle (or a feasible potential) in a weighted digraph exists. In the following, we refer to this graph as the potential graph.

**Lemma 32.** Given an instance of the UDODOSP or the LDODOSP, the set of (fractional) values of N for which the potential graph contains no negative cycle is a possibly empty or one-sided interval.

*Proof.* The set of feasible values for N can be expressed as one-dimensional polyhedron with a linear constraint for each cycle in the potential graph (the sum of the weights of that cycle is linear in N and should be non-negative).

This immediately implies an efficient algorithm to compute the minimum (integral) value for N.

**Corollary 33.** Given an instance of the UDODOSP or the LDODOSP (without a given number of workers), the minimum number of workers required to obtain a feasible schedule can be computed by solving one LP with  $\mathcal{O}(D)$  variables and  $\mathcal{O}(D)$  constraints and checking feasibility of one UDODOSP/LDODOSP instance (with the given number of workers).

*Proof.* We consider the set of inequalities given in the potential graph as a set of inequalities in the values of the potential and N. We then minimize N subject to this system of inequalities. Next, we simply round up the optimal (fractional) value of N to the next integer. Finally, we check whether this results in a feasible number of workers. If so, this is the optimal solution. If not, there is no feasible number of workers, implying that the interval from Lemma 32 is contained strictly in between two consecutive integers.

Remark. The number of nurses is always bounded from below by zero. Thus, the minimization problem for N is never unbounded. However, there is no general upper bound on N, therefore it may be that the maximization problem for N is unbounded. There are two possible reasons for this: first, it may be feasible to add workers who do not work at all; second, the requests may not be bounded from above.

However, using a general LP solver is not necessary but one can minimize N with a binary search as well. For this, we must find an upper and a lower bound on the optimal value of N, if existant. And we must be able to decide if a given value of N is (i) too small (ii) feasible (iii) too large or (iv) the instance is infeasible independent of N.

**Lemma 34.** The optimal value for N is bounded from above by  $\sum_{d \in [D]} r_l^d$ .

*Proof.* Fix some feasible schedule for the minimal value of N. For each day d mark  $r_l^d$  many workers that are on duty on day d. There are at most  $\sum_{d \in [D]} r_l^d$  many marked workers. If we remove all workers that are not marked, we still have a feasible schedule. Thus, each worker was marked, and therefore  $N \leq \sum_{d \in [D]} r_l^d$ .

Using the Moore-Bellman-Ford algorithm [Bel58; For56; Moo59] on the potential graph, we can distinguish between cases (i) to (iv): if the algorithm finds a feasible potential, we are in case (ii). If it returns a negative cycle, consider the affine function  $a \cdot N + b$  that describes the total cost of this cycle in terms of N. Depending on the sign of a, we know whether N is too large or too small. If a = 0, the instance is infeasible for every value of N.

Remark. Note that the binary search to optimize N is essentially the ellipsoid method on the one-dimensional polyhedron described in Lemma 32. Distinguishing cases (i) to (iii) is essentially a separation oracle for this polyhedron.

# 7 Summary

In this paper, we have considered the DODOSP and examined its complexity. We have shown that if bounds on the requests from both sides are given, then the problem is NP-complete if and only if one global and one local lower bound is nontrivial. Furthermore, there are two special cases in which the DODOSP is polynomial-time solvable, as shown in Table 3. The algorithms we introduced to solve these cases are easy to implement and fast. In case the requests are bounded only from one side, the complexity results only differ slightly (cf. Table 2).

bounds		complexity	proof
$U_x, l_y \text{ with } x, y \in \{w, o\}$	or any superset	NP-complete	Theorem 8
$U_w, U_o, u_w, u_o$	or any subset	Р	Theorem 14
$u_w, u_o, l_w, l_o$	or any subset	Р	Theorem 22

Table 3: Computational complexity of the DODOSP

Our results provide a straightforward method for distinguishing between computationally easy subcases and those that are more complex. This distinction is particularly valuable in real-world applications, where our results may help guide decisions concerning which constraints to relax or drop in order to make the problem computationally more manageable. Knowing the specific constraints that drive complexity is not only helpful to determine parameters to relax, but is a valuable information when designing a model in the first place. Even though the solution quality may decrease by choosing a weaker model, this is made up for by the speedup in running time, which is especially important if schedules must be computed quickly.

Even though our algorithms can only solve special cases of the DODOSP they can be of use for the general case, and even other variants of personell rostering. Since the algorithms are very efficient, one can use them, as starting point or as a subroutine for more sophisticated algorithms.

Future research could explore several interesting directions. One potential extension is the introduction of heterogeneous workers, where each worker's work pattern is subject to different constraints, such as varying limits on consecutive work days or total days off. This would make the problem more reflective of real-world scenarios, where employees often have individual preferences. Another important extension would involve handling predetermined entries in the schedule, such as pre-scheduled holidays or leave requests for certain workers. This might include continuing an existing schedule to a larger timer period. From the perspective of Section 6, one can ask if the DODOSP can be approximated regarding certain parameters such as the minimal number of required workers. Finally, one could analyze real-world instances hoping to find a more restricted problem definition that covers practical instances, without being NP-hard.

# Acknowledgments

We would like to express our gratitude to Erik Saule for highlighting that the problem is not trivially in NP, and to Maximilian von Aspern for proposing an NP-certificate. We also thank Sigrid Knust, Sebastian Lüderssen, Meike Neuwohner, Pieter Smet, Vera Traub, and Greet Vanden Berghe for their valuable comments on various aspects of the content and overall development of the paper. Finally, we thank Luke Connolly (KU Leuven) for his editorial consultation. This research was supported by KU Leuven C24E/23/012 'Human-centred decision support based on new theory for personnel rostering'.

## References

- [Arb24] Landesinstitut für Arbeitsschutz und Arbeitsgestaltung Nordrhein-Westfalen KOMNET. Wieviele Tage darf man nach dem Arbeitszeitgesetz hintereinander arbeiten? Dialog 4493. Sept. 25, 2024. URL: https://www.komnet.nrw.de/\_sitetools/dialog/4493.
- [Bar81] John J. Bartholdi. "A Guaranteed-Accuracy Round-off Algorithm for Cyclic Scheduling and Set Covering". In: Operations Research 29.3 (June 1981), pp. 501– 510
- [BBK13] Jens O. Brunner, Jonathan F. Bard, and Jan M. Köhler. "Bounded flexibility in days-on and days-off scheduling". en. In: Nav. Res. Logist. 60.8 (2013), pp. 678– 701.
- [Bel58] Richard Bellman. "On a Routing Problem". In: Quarterly of Applied Mathematics 16.1 (1958), pp. 87–90.
- [BNW22] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. "Negative-Weight Single-Source Shortest Paths in Near-linear Time". In: 2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS). 2022, pp. 600–611.
- [BQB11] Peter Brucker, Rong Qu, and Edmund Burke. "Personnel scheduling: Models and complexity". In: European Journal of Operational Research 210.3 (2011). Publisher: Elsevier, pp. 467–473.
- [Bur+04] Edmund K. Burke et al. "The State of the Art of Nurse Rostering". en. In: Journal of Scheduling 7.6 (2004), pp. 441–499. ISSN: 1099-1425.
- [Den16] S. J. M. Den Hartog. "On the complexity of nurse scheduling problems". MA thesis. 2016. (Visited on 02/14/2024).
- [DV11] Patrick De Causmaecker and Greet Vanden Berghe. "A categorisation of nurse rostering problems". en. In: *Journal of Scheduling* 14.1 (2011), pp. 3–16. ISSN: 1094-6136, 1099-1425.
- [For 56] Lester Randolph Ford. Network Flow Theory. Santa Monica, CA: RAND Corporation, 1956.
- [Gho62] A Ghouila-Houri. "Caractérisation des matrices totalement unimodulaires". In: Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences (Paris) 254 (1962), pp. 1192–1194.
- [GJ79] Michael R. Garey and David S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. A Series of Books in the Mathematical Sciences. San Francisco: W. H. Freeman and Co., 1979. 338 pp.
- [HHZ23] Steven JM den Hartog, Han Hoogeveen, and Tom C. van der Zanden. "On the complexity of Nurse Rostering problems". In: *Operations Research Letters* 51.5 (2023). Publisher: Elsevier, pp. 483–487.
- [Lau96] Hoong Chuin Lau. "On the complexity of manpower shift scheduling". In: Computers & operations research 23.1 (1996). Publisher: Elsevier, pp. 93–102.

- [Moo59] Edward F. Moore. "The Shortest Path through a Maze". In: *Proc. of the International Symposium on the Theory of Switching*. Harvard University Press, 1959, pp. 285–292.
- [Ngo+22] Chong Man Ngoo et al. "A Survey of the Nurse Rostering Solution Methodologies: The State-of-the-Art and Emerging Trends". In: *IEEE Access* 10 (2022). Conference Name: IEEE Access, pp. 56504–56524. ISSN: 2169-3536.
- [OI00] Takayuki Osogami and Hiroshi Imai. "Classification of Various Neighborhood Operations for the Nurse Scheduling Problem". In: Algorithms and Computation. Ed. by Gerhard Goos et al. Vol. 1969. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 72–83
- [Sme+13] Pieter Smet et al. "Nurse Rostering: A Complex Example of Personnel Scheduling with Perspectives". en. In: Automated Scheduling and Planning. Ed. by A. Sima Uyar, Ender Ozcan, and Neil Urquhart. Vol. 505. Series Title: Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 129–153.
- [Sme+14] Pieter Smet et al. "Modelling and evaluation issues in nurse rostering". en. In: Annals of Operations Research 218.1 (2014), pp. 303–326.
- [Sme+16] Pieter Smet et al. "Polynomially solvable personnel rostering problems". In: European Journal of Operational Research 249.1 (2016). Publisher: Elsevier, pp. 67–75.
- [Tho95] Gary M. Thompson. "Improved Implicit Optimal Modeling of the Labor Shift Scheduling Problem". In: Manage. Sci. 41.4 (1995).

# A Appendix

## A.1 NP-completeness proof in [BBK13]

In their proof Brunner, Bard, and Köhler aim to reduce the CIRCULANT PROBLEM to the optimization version of the DODOSP. The CIRCULANT PROBLEM was shown to be strongly NP-complete by Bartholdi in [Bar81].

CIRCULANT PROBLEM (CP)

**Input:** A circulant  $m \times n$ -matrix A and a nonnegative m-dimensional vector b.

**Question:** Find an *n*-dimensional nonnegative integer vector x which minimizes  $\mathbf{1} \cdot x$  with

 $Ax \geq b$ .

Here, an  $m \times n$ -matrix A is called *circulant* if there exists an m-dimensional vector a such that all columns of A are only copies of a whose entries were shifted in a circular way.

However, Brunner, Bard, and Köhler ultimately use the wrong direction of reduction and try to solve certain instances of the DODOSP using the CP. More precisely, they consider instances of the optimization version of the DODOSP with  $u_w = l_w$ ,  $u_o = l_o$ ,  $r_u^d = D$  for all days (meaning the requests are only bounded from below) and  $D = c(u_w + u_o)$  for some non-negative integer c. Indeed, if one uses the cyclic definition of the optimization version of the DODOSP, which was not originally done by Brunner, Bard, and Köhler, one can reduce such an instance I to an instance I' of CP in the following way.

Let a be the vector of length  $c(u_w + u_o)$  which consists of alternating sequences of  $u_w$  ones and  $u_o$  zeros. Furthermore, define A to be the circulant matrix which arises from rotating a by onre entry each time. In fact, A will only have  $u_w + u_o$  columns, since after this many rotations the vector a is sent to itself. Finally, we set  $b_d = r_l^d$ . The columns in A denote the only possible schedules for a worker in the given instance of the optimization version of the DODOSP. Therefore, any solution to I' naturally provides a solution to I and vice versa.

However, since this reduction only works for a particular kind of instances of the (circular) optimization version of the DODOSP, we do not get a statement on the complexity of the CP depending on the complexity of the (circular) optimization version of the DODOSP.