The minimum shift design problem

Luca Di Gaspero[*] Johannes Gärtner[†] Guy Kortsarz[‡] Nysret Musliu[§]
Andrea Schaerf[¶] Wolfgang Slany[‖]

Abstract

The *min*-SHIFT DESIGN problem (MSD) is an important scheduling problem that needs to be solved in many industrial contexts. The issue is to find a minimum number of shifts and the number of employees to be assigned to these shifts in order to minimize the deviation from workforce requirements.

Our research considers both theoretical and practical aspects of the *min*-SHIFT DESIGN problem. This problem is closely related to the minimum edge-cost flow problem ($MECF$), a network flow variant that has many applications beyond shift scheduling. We show that MSD reduces to a special case of $MECF$ and, exploiting this reduction, we prove a logarithmic hardness of approximation lower bound for MSD. On the basis of these results, we propose a hybrid heuristic for the problem which relies on a greedy heuristic with a min-cost max-flow subroutine based on the network flow analogy followed by a local search algorithm that makes use of multiple neighborhood relations.

An experimental analysis on structured random instances shows that the hybrid heuristic clearly outperforms our previous commercial implementation and highlights the respective merits of the composing heuristics for different performance parameters.

Introduction

The typical process of planning and scheduling a workforce in an organization is inherently a multi-phase activity (Tien and Kamiyama, 1982). First, the production or the personnel management have to determine the temporal staff requirements, i.e., the number of employees needed for each timeslot of the planning period. Afterwards, it is possible to proceed to determine the shifts and the total number of employees needed to cover each shift. The final phase consists in the assignment of the shifts and days-off to employees.

In the literature, there are mainly two approaches to solve the latter two phases (see Burke et al., 2004). The first approach consists of solving the shift assignment as one single optimization problem (e.g., Glover and McMillan, 1986; Jackson et al., 1997). A second approach, instead, proceeds in stages by considering the design of shifts (or staffing) and the assignment as separate problems (Balakrishnan and Wong, 1990; Lau, 1996; Musliu et al., 2002). This is computationally easier, although there is no guarantee that the optimal solution to the overall problem can be found.

[*] l.digaspero@uniud.it, DIEGM, University of Udine – via delle Scienze 208, I-33100 Udine, Italy
[†] gaertner@ximes.com, Ximes Inc – Schwedenplatz 2/26, A-1010 Wien, Austria
[‡] guyk@camden.rutgers.edu, Computer Science Department, Rutgers University – Camden, NJ 08102, USA
[§] musliu@dbai.tuwien.ac.at, Inst. for Information Systems, Vienna University of Technology, A-1040 Wien, Austria
[¶] schaerf@uniud.it, DIEGM, University of Udine – via delle Scienze 208, I-33100 Udine, Italy
[‖] wsi@ist.tugraz.at, Inst. for Software Technology, Graz University of Technology – A-8010 Graz, Austria

In this work we follow the latter approach and focus on the problem of designing the shifts only. We propose the *min*-SHIFT DESIGN formulation where the issue is to construct a set of work shifts to use, their number being minimal (hence the name), and to find the number of workers to assign to each shift in order to meet (or minimize the deviation from) pre-specified staff requirements. The selection of shifts is subject to constraints on the possible start times and lengths of shifts. Our work is motivated by practical considerations. It differs in particular from other literature dealing with similar problems by the explicit constraint of minimizing the number of shifts. The existence of less shifts leads to schedules that are easier to read, check, manage and administer.

The MSD problem arose in a project at Ximes Inc, a consulting and software development company specializing in shift scheduling. The goal of this project was, among others, to produce a software end-product called OPA (short for 'OPerating hours Assistant'). OPA was introduced mid 2001 to the market and has since been successfully sold to end-users besides of being heavily used in the day to day consulting work of Ximes Inc at customer sites (mainly European, but Ximes recently also won a contract with the US ministry of transportation). OPA has been optimized for "presentation"-style use where solutions to many variants of problem instances are expected to be more or less immediately available for graphical exploration by the audience. Speed is of crucial importance to allow for immediate discussion in working groups and refinement of requirements, especially if such a solver is used during a meeting with customers. Without quick answers, understanding of requirements and consensus building would be much more difficult. OPA and the underlying heuristics have been described in (Gärtner et al., 2001; Musliu et al., 2004).

In this work we aim at giving a deeper insight into the problem. We first establish the complexity of the *min*-SHIFT DESIGN problem by means of a reduction to a network flow problem, namely the cyclic multi-commodity capacitated fixed-charge *min*-COST *max*-FLOW problem (Garey and Johnson, 1979, Prob. ND32, page 214). As we will show, even a logarithmic approximation of the problem is NP-hard. However, if the issue of minimizing the number of shifts is neglected, the resulting problem becomes solvable in polynomial time.

Based on the above theoretical results, we propose an hybrid solver for the MSD problem, composed of two heuristics. The first is a greedy construction heuristic based on the *min*-COST *max*-FLOW analogy. The second is a local search algorithm that makes use of the multi-neighborhood search approach proposed by Di Gaspero and Schaerf (2003a).

We evaluate the performances of the proposed heuristics on different settings and we compare them and the resulting hybrid solver with OPA. The outcome of the comparison shows that our heuristics significantly outperforms the previous approach.

1 Problem statement

The *min*-SHIFT DESIGN (MSD) consists in the selection of which work shifts to use and how many people to assign to each such shift in order to meet pre-specified workforce requirements.

The requirements are given for d *planning days* $D = \{1, \ldots, d\}$ (the so-called *planning horizon*), where a planning day can start at a certain time on a regular calendar day and ends 24 hours later, usually on the next calendar day. Each planning day j is split into n equal-size smaller intervals $t_i = [\tau_i, \tau_{i+1})$, called *timeslots*, which have the same length $h := \|\tau_{i+1} - \tau_i\| \in \mathbb{N}$ expressed in minutes. The time point τ_1 on the first planning day represents the start of the planning horizon, whereas time point τ_{n+1} on the last planning day is the end of the planning horizon. In this work we deal with cyclic schedules, thus τ_{n+1} of the last planning day coincides with τ_1 of the first planning day of the next cycle, and the requirements are repeated in each cycle.

Start	End	Mon	Tue	Wen	Thu	Fri	Sat	Sun
00:00	06:00	5	5	5	5	5	5	5
06:00	08:00	2	2	2	6	2	0	0
08:00	09:00	5	5	5	9	5	3	3
09:00	10:00	7	7	7	13	7	5	5
10:00	11:00	9	9	9	15	9	7	7
11:00	14:00	7	7	7	13	7	5	5
14:00	16:00	10	9	7	9	10	5	5
16:00	17:00	7	6	4	6	7	2	2
17:00	22:00	5	4	2	2	5	0	0
22:00	24:00	5	5	5	5	5	5	5

Table 1: Sample workforce requirements.

Shift type	min_{st}	max_{st}	min_l	max_l
M (morning)	05:00	08:00	07:00	09:00
D (day)	09:00	11:00	07:00	09:00
A (afternoon)	13:00	15:00	07:00	09:00
N (night)	21:00	23:00	07:00	09:00

Table 2: An example of shift types

For each timeslot t_i of a day j of a cycle, we are given a *required number of employees* b_{ij}, representing the number of persons needed at work in that timeslot.

An example of workforce requirements with $d = 7$ is shown in Table 1, where the planning days coincide with calendar days (for the rest of this document we will use the term 'day' when referring to planning days unless stated otherwise). In the table, the days are labeled 'Mon', 'Tue', etc., and, for conciseness, timeslots with the same requirements are grouped together (the example is adapted from a real call-center problem).

In this problem we are interested in determining a set of *shifts* for covering the workforce requirements. Each *shift* $s = [\sigma_s, \sigma_s + \lambda_s)$ is characterized by two values σ_s and λ_s that determine the *starting time* and the *length* of the shift, respectively. Since we are dealing with discrete time slots, for each shift s, the variables σ_s can assume only the τ_i values defined above, and the variables λ_s are constrained to be a multiple of the timeslot length h. The set of all possible shifts is denoted by S.

When designing shifts, not all starting times are feasible, neither are all lengths allowed. For this reason, the problem statement also includes a collection of *shift types* $V = \{v_1, \dots v_r\}$, each of them characterized by the *earliest* and the *latest starting times* (denoted by $min_{st}(v_k)$ and $max_{st}(v_k)$, respectively), and a *minimum* and *maximum length* of its shifts (denoted by $min_l(v_k)$ and $max_l(v_k)$). Each shift s *belongs* to a unique shift type, therefore its starting time and length are constrained to lie within the intervals defined by its type. We denote the shift type relation with $K(s) = v_k$. A shift s that belongs to the type v_k is a *feasible shift* if $min_{st}(v_k) \leq \sigma_s \leq max_{st}(v_k)$ and $min_l(v_k) \leq \lambda_s \leq max_l(v_k)$.

Table 2 shows an example of a set of shift types together with the ranges of allowed starting times and lengths. The times are expressed in the format *hour:minute*.

The shift types together with the time granularity h determine the quantity $m = |S|$ of possible shifts. For example, assuming a timeslot length of 15 minutes, there are 117 different shifts belonging to the morning type shift of Table 2. In fact, there are 13 distinct start times (05:00, 05:15, ..., 08:00) and 9 distinct lengths (7 hours, 7 hours and 15 minutes, ..., 9 hours) thus yielding $13 \times 9 = 117$ shifts. Altogether, there are $m = 360$ shifts belonging to all the types of the example: 117 morning shifts, 81 day shifts, 81 afternoon shifts, and 81 night shifts.

Start	Type	Length	Mon	Tue	Wed	Thu	Fri	Sat	Sun
06:00	M	08:00	2	2	2	6	2	0	0
08:00	M	08:00	3	3	3	3	3	3	3
09:00	D	08:00	2	2	2	4	2	3	2
14:00	A	08:00	5	4	2	2	5	0	0
22:00	N	08:00	5	5	5	5	5	5	5

Table 3: A solution to the *min*-SHIFT DESIGN problem

The goal of the *min*-SHIFT DESIGN problem is to select a set of q feasible shifts $Q = \{s_1, \ldots, s_q\}$ and to decide how many people $x_j(s) \in \mathbb{N}^*$ are going to work in each shift $s \in Q$ for each day j, so that b_{ij} people will be present at each timeslot t_i of the day. If we denote the collection of shifts that include the timeslot t_i by $S_{t_i} \subseteq Q$, a feasible solution consists of d numbers $x_j(s)$ assigned to each shift s so that $l_{ij} = \sum_{s \in S_{t_i}} x_j(s)$ is equal to b_{ij}. In other words, we require that the number of workers present at time t_i, for all values of i and for all days j, meets the staffing requirements.

In practical cases, however, this constraint is relaxed, so that small deviations are allowed. To this aim, each solution of the *min*-SHIFT DESIGN problem is evaluated by means of an *objective function* to be minimized, which is a weighted sum of three components. The first and second components are the *staffing excess* and *shortage*, namely, the sums $F_1(Q, x) = \sum_{j=1}^{d} \sum_{i=1}^{n} \max\{l_{ij} - b_{ij}, 0\}$ and $F_2(Q, x) = \sum_{j=1}^{d} \sum_{i=1}^{n} \max\{b_{ij} - l_{ij}, 0\}$. The third component of the objective function is the number of shifts selected $F_3(Q, x) = |Q|$. Once a shift is selected (at least one person works in this shift during any day) it is not really important how many people work at this shift nor on how many days the shift is reused. However, it is important to have only few shifts as they lead to schedules that have a number of advantages, e.g., if one tries to keep teams of persons together. Such team-building may be necessary due to managerial or qualification reasons. Besides this, there are further advantages of obtaining schedules with fewer shifts. For example, they lead to schedules that are easier to design with or without software support (see Musliu et al., 2002). Fewer shifts also make schedules easier to read, check, manage and administer; each of these activities being a burden in itself.

In summary, we look for an assignment $x_j(s)$ to all the possible shifts s that minimizes the objective function composed by a weighted sum of *excess*, *shortage* and *number* of shifts, where the components have different relative importance depending on the situation. Therefore, the weights of the three components depend on the instance at hand and can be adjusted interactively by the user.

A solution to the problem in Table 1 is given in Table 3 and is pictorially represented in Figure 1. Note that this solution is not perfect. For example, there is a shortage of workers every day in the timeslot 10:00–11:00, represented by the thin white peaks in the figure. Conversely, on Saturdays there is an excess of one worker in the period 09:00–17:00. The values of the objectives F_i are the following. The shortage of employees F_1 is 15 person-hours, the excess of workers F_2 is 7 person-hours, and the number of shifts used, F_3, is 5, as listed in Table 3.

2 Theoretical results

In this section we prove that a restricted version of *min*-SHIFT DESIGN is equivalent to the infinite capacities flow problem on a Direct Acyclic Graph with unitary edge costs ($UDIF$), which, in turn, is a variant of the *min*-COST *max*-FLOW problem (Garey and Johnson, 1979, Prob. ND32, page 214). The latter is one of the most fundamental network flow variants with many applications.

Thanks to the equivalency between *min*-SHIFT DESIGN and $UDIF$, a hardness of approximation result for $UDIF$ carries over to *min*-SHIFT DESIGN and, as we are going to see, we could prove a

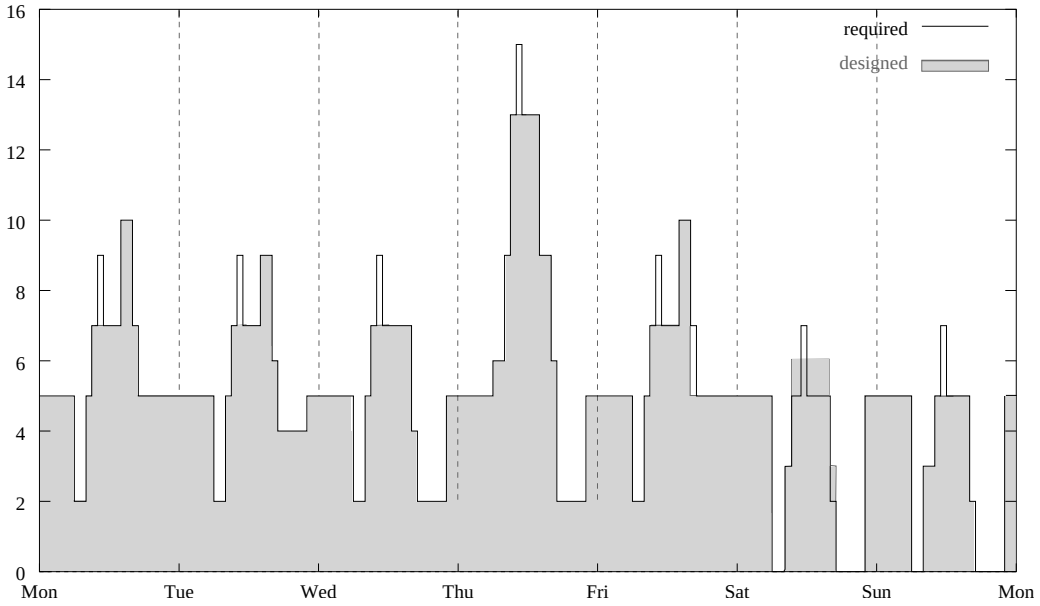


Figure 1: A pictorial representation of the solution in Table 3

logarithmic lower bound on these problems.

First we present the $UDIF$ problem which is a network flow problem (Ahuja et al., 1993) with the following features:

(1) Every edge not incident to the sink or to the source, called *proper* edge, has infinite capacity. Non-proper edges, namely edges incident to the source or to the sink, have arbitrary capacities.

(2) The costs of proper edges is 1, whereas the cost of non-proper edges is 0.

(3) The underlying flow network is a Direct Acyclic Graph (DAG).

(4) The goal is, as in the general problem, to find a maximum flow $f(e)$ over the edges (obeying the capacity and flow conservation laws) and, among all maximum flows, to choose the one minimizing the cost of edges carrying non-zero flow. Hence, in this case, the problem is to minimize the *number* of proper edges carrying non-zero flow (namely, minimizing $|\{e : f(e) > 0,\ e \text{ is proper}\}|$).

To simplify the theoretical analysis of *min*-SHIFT DESIGN, in this section we restrict to instances where $d = 1$, that is, workforce requirements are given for a single day only, and no shifts in the collection of possible shifts span over two days (i.e., each shift starts and ends on the same day). We also assume that, for the evaluation function, weights for excess and shortage are equal and are so much larger than weights for the number of shifts so that the former always take precedence over the latter. This effectively gives priority to the minimization of deviation, thereby only minimizing the number of shifts for all those feasible solutions already having minimum deviation.

It is useful to describe the shifts via $0-1$ matrices with the *consecutive ones* (c1) property on columns. We say that a matrix A obeys the c1 property on columns if all entries in the matrix are either 0 or 1 and all the 1 in each column appear consecutively. We call such a matrix a c1 matrix.

For a c1 matrix $A = (a_{ij})$, we say that a column j *starts* (respectively *ends*) at i if the topmost 1 entry in the column (respectively, the lowest 1 entry in the column) is in row i. A column with a single

1 entry in the ith place both starts and ends at i. The row in which a column j starts (respectively, ends) is denoted by β_j (respectively η_j).

We give a formal description of *min*-SHIFT DESIGN via c1 matrices as follows. We are given an $n \times m$ matrix A in which each column corresponds to a possible shift. Each entry a_{ij} in the matrix is either 1 if i is a valid timeslot for shift $s = j$ or 0 otherwise. Since the set of valid timeslots for a given shift type (and thus for a given shift belonging to a shift type) is made up of consecutive timeslots, A is clearly a c1 matrix. Furthermore, we are given a vector b of length n of positive integers; each entry b_i corresponds to the workforce requirement for the timeslot i. Within these settings, the *min*-SHIFT DESIGN problem can be stated as a system of inequalities: $Ax \geq b$ with $x \in \mathbb{Z}^n$, $x \geq 0$, where the vectors x correspond to the shift assignments.

The optimization criteria are represented as follows. Let A_i be the ith row in A, and $\|x\|_1$ denote the L_1 norm of x. We are looking for a vector $x \geq 0$ with the following properties:

(1) The vector x minimizes $\|Ax - b\|_1$ (i.e., the deviation from the staffing requirements).

(2) Among all vectors minimizing $\|Ax - b\|_1$, x has the minimum number of non-zero entries (corresponding to the number of selected shifts).

Claim 1 *The restricted one-day noncyclic variant of* min-SHIFT DESIGN *where a zero deviation solution exists (i.e., $d = 1$, all shifts start and finish on the same day, and $Ax = b$ admits a solution), is equivalent to the UDIF problem.*

The proof below is followed by an explanation of how shortage and excess can be handled by a small linear adaptation of the network flow problem. This effectively allows us to find the minimum (weighted) deviation from the workforce requirements (without considering minimization of the number of shifts) by solving a *min*-COST *max*-FLOW ($MCMF$) problem, an idea that will be reused in Section 3.1. It is well known that the problem of finding such a maximum flow minimizing $\sum_e p(e)f(e)$ is solvable in polynomial time (see, e.g., Papadimitriou and Steiglitz, 1982).

Proof. We are following here a path similar to the one by Hochbaum (2000) in order to get this equivalence (see also, e.g., Ahuja et al., 1993).

First note that in the special case when $Ax = b$ has a feasible solution, by the definition of MSD the optimum x^* satisfies $Ax^* = b$.

Let $\mathcal{T}$ denote the matrix:

$$\mathcal{T} = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 & & 0 \\ 0 & 1 & -1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & -1 & 0 & & 0 \\ & \vdots & & \ddots & & & \vdots \\ 0 & 0 & 0 & & 1 & -1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & -1 \\ 0 & 0 & 0 & & 0 & 0 & 1 \end{pmatrix}$$

The matrix $\mathcal{T}$ is a $n \times n$ matrix which is regular. In fact, $\mathcal{T}^{-1}$ is the upper diagonal matrix with 1 along the diagonal and above, whereas all other elements are equal to 0.

As $\mathcal{T}$ is regular, the two sets of feasible vectors for $Ax = b$ and for $\mathcal{T}Ax = \mathcal{T}b$ are equal. The matrix $\mathcal{F} = \mathcal{T}A$ is a matrix with at most two non-zero entries in each column: one being a 1 and the other being a -1. In fact, all columns j in A create a column in $\mathcal{F} = \mathcal{T}A$ with exactly one -1 entry

and exactly one 1 entry except for columns j with 1 in the first row (namely, so that $\beta_j = 1$). These columns leave one 1 entry in row η_j, namely, in the row where column j ends. We call these columns the *special* columns.

The matrix $\mathcal{F}$ can be interpreted as a flow matrix (see, e.g., Bar-Ilan et al., 2001). We start our construction with a graph consisting of only two vertices s[1] and t, representing the source and the sink of the flow graph. Then we assign a vertex u_i to each row i of the matrix and we add an extra vertex u_0.

Each column j of the matrix $\mathcal{F}$ is represented by an edge e_j. Each e_j with $\mathcal{F}_{ij} = 1$ and $\mathcal{F}_{kj} = -1$ goes out of u_k into u_i. Note that the existence of this column in $\mathcal{F}$ implies the existence in A of a column of ones starting at row $k + 1$ (and not k) and ending at row j.

For all special columns j ending at η_j, we add an edge from u_0 into u_{η_j}. In addition, we add an edge of capacity b_1 from the source s to u_0.

Let $\bar{b} = \mathcal{T}b$. This vector determines the way all vertices (except u_0) are joined to the sink t and source s. If $\bar{b}_i > 0$ then there is an edge from u_i to t with capacity $\bar{b}_i$. Otherwise, if $\bar{b}_i < 0$, there is an edge from s to u_i with capacity $-\bar{b}_i$. Vertices with $\bar{b}_i = 0$ are not joined to the source or sink. All edges not incident to the source or sink have infinite capacity.

Note that the addition of the edge from s into u_0 with capacity b_1 makes the sum of capacities of edges leaving the source equal to the sum of capacities of edges entering the sink. It is easy to see that if there exists a saturating flow (namely a flow saturating all the edges entering the sink), then the feasible vectors for the flow problem are exactly the feasible vectors for $\mathcal{F}x = \bar{b}$. Hence, these are the same vectors feasible for the original set of equations $Ax = b$.

As we assumed that $Ax = b$ has a solution, there exists a saturating flow, namely, there is a solution saturating all the vertex-sink edges (and, in our case, all the edges leaving the source are saturated as well). Therefore, the problem is transformed into the following question: Given a network G constructed as above, find a maximum flow in G and among all maximum flows find the one that minimizes the number of proper edges carrying non-zero flow.

The resulting flow problem is in fact a $UDIF$ problem. Indeed, the network G is a DAG since all edges go from u_i to u_j with $j > i$. In addition, all capacities on edges which are not incident to the sink or source are infinite (see the above construction).

On the other hand, given a $UDIF$ instance with a saturating flow it is possible to find an inverse function that maps it to an MSD instance. The MSD instance is described as follows.

Assume that the vertices u_i are ordered in increasing topological order. Given the DAG G, the corresponding matrix $\mathcal{F}$ is defined by taking the edge-vertices incidence matrix of G. As it turns out, we can find a c1 matrix A so that $\mathcal{T}A = \mathcal{F}$. Indeed, for any column j with non-zeros in rows p, q with $p < q$, necessarily, $\mathcal{F}_{pj} = -1$ and $\mathcal{F}_{qj} = 1$ (if there is a column j that does not contain an $\mathcal{F}_{pj} = -1$, let set $p = 0$). Hence, add to A the c1 column j' with $\beta_{j'} = p + 1$ and $\eta_{j'} = q$.

We note that the restriction of the existence of a flow saturating the flow along edges entering the sink t is not essential. It is easy to guarantee this as follows. Add a new vertex u to the network and an edge (s, u) of capacity $\sum_{(v,t)} c(v, t) - f^*$ (where f^* is the maximum flow value). By definition, the edge (s, u) has cost 0. Add a directed edge from u to every source v. This makes a saturating flow possible, at the increase of only 1 in the cost.

It follows that in the restricted case when $Ax = b$ has feasible solutions the MSD problem is equivalent to $UDIF$. □

In order to understand how this result can be employed to find solutions to MSD instances where

[1] In this proof and in the following one the symbol s denotes the source of the flow graph instead of a generic shift.

no zero deviation solution exists, we need to explain how to find a vector x so that $Ax \geq b$ and $\|Ax - b\|_1$ is minimum.

When $Ax = b$ does not have a solution, we introduce n dummy variables y_i. The ith inequality is replaced by $A_i x - y_i = b_i$, that is y_i is set to the difference between $A_i x$ and b_i (and $y_i \geq 0$). Let $-I$ be the negative identity matrix, namely the matrix with all zeros except -1 in the diagonal entries. Let $(A; -I)$ be the A matrix with $-I$ to its right and let $(x; y)$ be the column of x followed by the y variables. The above system of inequalities is represented by $(A; -I)(x; y) = b$. Multiplying the inequality by $\mathcal{T}$ (where $\mathcal{T}$ is the matrix defined above) gives $(\mathcal{F}; -\mathcal{T})(x; y) = \mathcal{T}b = \bar{b}$.

The matrix $(\mathcal{F}; -\mathcal{T})$ is a flow matrix and its corresponding graph is the graph of $\mathcal{F}$ with the addition of an infinite capacity edge from u_i into u_{i-1} $(i = 1, \ldots, n)$. We call these edges the y edges, whereas the edges originally in G are called the x edges. The sum $\sum_i y_i$ clearly represents the excess L_1 norm $\|Ax - b\|_1$. Hence, we give a cost $C(e) = 1$ to each edge corresponding to a y_i. We look for a maximum flow minimizing $\sum_i C(e)f(e)$, namely, a min-cost max-flow solution. As we have assumed (without loss of generality) that all time intervals $[t_i, t_{i+1})$ $(i = 1, \ldots, n)$ have equal length, this gives the minimum possible excess. Shortage can be handled in a similar way.

We next show that unless $P = NP$, there is some constant $0 < c < 1$ such that approximating $UDIF$ within a $c \ln n-$ratio is NP-hard.

Since the case of zero excess MSD is equivalent to $UDIF$ (see Claim 1), similar hardness results follow for this problem as well.

Theorem 2.1 *There is a constant $c < 1$ so that approximating the $UDIF$ problem within $c \ln n$ is NP-hard.*

Proof. We prove a hardness reduction for $UDIF$ under the assumption $P \neq NP$ using a reduction from Set-Cover. We need a somewhat different proof than the one reported in (Krumke et al., 1998) to account for the extra restriction imposed by $UDIF$. For our purposes it is convenient to formulate the set cover problem as follows. The set cover instance is an undirected bipartite graph $\mathcal{B}(U_1, U_2, A)$ with edges only crossing between U_1 and U_2. We may assume that $|U_1| = |U_2| = n$. We look for a minimum sized set $Q \subseteq U_1$ so that $N(Q) = U_2$ (namely, every vertex in U_2 has a neighbor in Q). If $N(Q) = U_2$ we say that Q covers U_2. We may assume that the given instance has a solution. The following result is proven in (Raz and Safra, 1997).

Theorem 2.2 *There is a constant $c < 1$ so that approximating Set-Cover within $c \ln n$ is NP-hard.*

We prove a similar result for $UDIF$ and thus for MSD.

Let $\mathcal{B}(U_1, U_2, \mathcal{E})$ be the instance of the set cover problem at hand, so that $|U_1| = |U_2| = n$. Add a source s and a sink t to the graph. Connect s to all the vertices of U_2 and assign capacity 1 to the resulting edges. Direct all the edges of $\mathcal{B}$ from U_2 to U_1. Now, create n^2 copies U_1^i of U_1 and for convenience denote $U_1 = U_1^0$. For each $i \in \{0, \ldots, n^2 - 1\}$, connect in a directed edge the copy $u_1^i \in U_1^i$ of each $u_1 \in U_1$ to the copy $u_1^{i+1} \in U_1^{i+1}$ of u_1 in U_1^{i+1}. Hence, a perfect matching is formed between contiguous U_1^i via the copies of the $u_1 \in U_1$ vertices. The vertices of $U_1^{n^2}$ are all connected to t via edges of capacity n. Note that, by definition, all other edges (which are edges not incident to the source nor to the sink) have infinite capacity.

It is straightforward to see that the resulting graph is a DAG and that the graph admits a flow saturating the source edges, and can be made to saturate the sink edges as described before.

We now inspect the properties of a "good" solution. Let Q be the set of vertices $Q \subseteq U_1$ so that for every vertex $u_2 \in U_2$ there exists a vertex $q \in Q$ such that edge (u_2, q) carries positive flow.

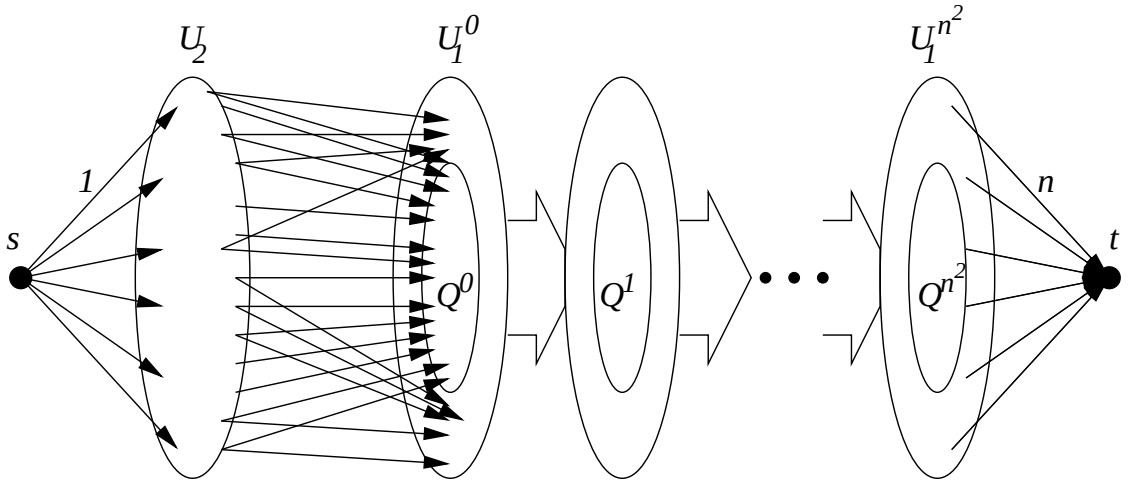


Figure 2: Schematic illustration of the reduction from the Set-Cover problem to the $UDIF$ problem.

Note that for every $u_2 \in U_2$ there must be such an edge for otherwise the flow is not optimal. Further note that the flow units entering Q must be carried throughout the copies of Q in all of the U_1^i sets $i \geq 1$ using the matching edges as this is the only way to deliver the flow into t. Hence, the number of proper edges in the solution is exactly $n^2 \cdot |Q| + n$. The n term comes from the n edges touching the vertices of u_2.

Further, note that Q must be a set cover of U_2 in the original graph $\mathcal{B}$. Indeed, every vertex u_2 must have a neighbor in Q. Finally, note that it is indeed possible to get a solution with $n^2 \cdot q^* + n$ edges where q^* is the size of the minimum set cover using an optimum set cover Q^* as described above. Since all the matching edges have infinite capacities, it is possible to deliver to t the n units of flow regardless of how the cover Q is chosen. The following properties end the proof: The number of vertices in the new graph is $O(n^3)$. In addition, the additive term n is negligible for large enough n in comparison to $n^2 \cdot |Q|$ where Q is the chosen set cover. Thus, for large enough n, $|Q| \dot{n}^2 \leq c \ln n^3 \dot{(}|Q^*|n^2 + n)$ would imply that $|Q| \leq 3c \ln n \dot{(}|Q^*|)$. However, as B was arbitrarely chosen this stands in contradiction to Theorem 2.2 unless $P = NP$. Thus the result follows for $c < 1/3$, and thus also for $c < 1$. □

3 Practical heuristics

We present a hybrid solver which is divided into two stages, namely a greedy construction for the initial solution followed by a tabu search procedure (Glover and Laguna, 1997) that iteratively improves it, which are described in the following subsections. In our experiments (Sect. 4) we evaluate the behavior of each stage and of the resulting hybrid algorithm, in order to analyze the sensitivity of the tabu search procedure to the starting point used.

3.1 GreedyMCMF

On the basis of the equivalence of the (non-cyclic) *min*-SHIFT DESIGN problem to $UDIF$, and the relationship with the *min*-COST *max*-FLOW problem, we propose a new greedy heuristic GreedyM-

```
function GreedyMCMF(S,b): MSD_Solution

/* 1. Preprocessing step: where to break cyclicity? */

t := FindBestSplitOffTime(S,b);                 // Search for a split-off on the 1st day of the cycle

/* 2. Greedy part with MCMF subroutine */

f* := MCMF(EncodeMSDToFlow(S,b,t));             // Compute best flow so far for MSD instance
σ := ShiftsAndWorkforceIn(f*);                  // Shifts are edges with flow ≠ 0; workforce is edge flow
min_cost := MSD_Eval(σ);                        // Cost of the best MSD solution found so far
Q := ShiftsInUseIn(σ);                          // Shifts in the current solution
T := ∅;                                         // Shifts already tried
repeat
   s := UniformlyChooseAShiftFrom(Q \ T);       // Consider a shift s that is used but not tried yet
   f := MCMF(EncodeMSDToFlow(Q \ {s},b,t));     // Try to solve the problem without shift s
   σ := ShiftsAndWorkforceIn(f);                // Extract shifts and workforce from flow solution
   current_cost := MSD_Eval(σ);                 // Compute the cost of the current solution
   if current_cost < min_cost then
      min_cost := current_cost;                 // Solution with one shift less and lower cost
      f* := f;                                  // Update the best solution found so far
      Q := ShiftsInUseIn(σ);                    // Could be less than Q \ {s}
   endif
   T := T ∪ {s};                                // Add s to shifts already tried
until Q \ T = ∅;                                // Cycle until no shift to try is left

/* 3. Postprocessing step to recover cyclicity, perform a local search with the ExchangeStaff move */

σ := SteepestDescent(ShiftsAndWorkforceIn(f*), ExchangeStaff);
return σ;
```

Figure 3: The Greedy *min*-COST *max*-FLOW ($MCMF$) subroutine computes a solution for the *min*-SHIFT DESIGN (MSD) problem

CMF() that uses a polynomial *min*-COST *max*-FLOW subroutine (MCMF()). The pseudocode of the algorithm is reported in Figure 3. The algorithm is based on the observation that the *min*-COST *max*-FLOW subroutine can easily compute the optimal staffing with minimum (weighted) deviation when slack edges (the y edges of Section 2) have associated costs corresponding, respectively, to the weights of shortage and excess. Note that, however, the algorithm is not able to simultaneously minimize the number of shifts that are used.

Since the MCMF() subroutine cannot consider cyclicity, we must first perform a preprocessing step that determines a good split-off time at which the cycle of d days should be broken. This is done heuristically by calling MCMF() with different starting times chosen between 5:00 and 8:00 on the first day of the cycle (in practice, we can observe that there is almost always a complete exchange of workforce between 5:00 and 8:00 on Monday mornings). All possibilities in this interval are tried while eliminating all shifts that span the chosen starting point when translating from *min*-SHIFT DESIGN to the network flow instances. The number of possibilities depends on the length of the timeslots of the instance (i.e., the time granularity). The starting point with the smallest cost as determined by MCMF() is used as the split-off time for the rest of the calls to MCMF() in GreedyMCMF. This method has been shown to provide adequate results in practice.

In the main loop, the greedy heuristic then removes all shifts that did not contribute to the *min*-SHIFT DESIGN instance corresponding to the current flow computed with MCMF(). It randomly chooses one shift (without repetitions) and tests whether removal of this shift still allows the MCMF()

to find a solution with the same deviation. If this is the case, that shift is removed and not considered anymore, otherwise it is left in the set of shifts used to build the network flow instances, but it will not be considered for removal again.

Finally, when no shifts can be removed anymore without increasing the deviation, a final postprocessing step is made to restore cyclicity. It consists of a simple repair step performed by a fast steepest descent runner that uses the ExchangeStaff neighborhood relation (see below). The runner selects at each iteration the best neighbor, with a random tie-break in case of same cost. It stops as soon as it reaches a local minimum, i.e., when it does not find any improving move.

As our MCMF() subroutine, we use CS2 version 3.9 (© 1995 – 2001 IG Systems, Inc., `http://www.avglab.com/andrew/soft.html`), an efficient implementation of a scaling push-relabel algorithm (Goldberg, 1997), slightly edited to be callable as a library.

3.2 Local search heuristic solver

The second stage of the proposed heuristic is based on the local search paradigm (Aarts and Lenstra, 1997; Hoos and Stützle, 2005) and relies on multiple neighborhood relations. In order to describe it, we first define the search space, then describe the set of neighborhood relations for the exploration of this search space, followed by the search strategies we employ.

3.2.1 Search space and initial solution

We consider as a state for *min*-SHIFT DESIGN a pair (Q, X) made up of a set of shifts $Q = \{s_1, s_2, \ldots\}$ and their staff assignment $X = \{x_1, x_2, \ldots\}$. The shifts of a state are split into two categories:

- *Active* shifts: at least one employee is assigned to a shift of this type on at least one day.
- *Inactive* shifts: no employees are assigned to a shift of this type on any day. These shifts do not contribute to the solution and to the objective function. Their role is explained later.

More formally, we say that a shift $s_i \in Q$ is active (resp. inactive) if and only if $\sum_{j=1}^{d} x_j(s_i) \neq 0$ $(= 0)$.

For the purpose of analyzing the behavior of the local search heuristic alone, we provide also a mean to generate a random initial solution for the local search algorithm. That is, we create a fixed number of random distinct active and inactive shifts for each shift type. Afterwards, for the active shifts, we assign a random number of employees for each day. The parameters needed to build a solution are the number of active and inactive shifts for each shift type and the range of the number of employees per day to be assigned to each random active shift.

For example, in the experimental session described in Section 4, we build a solution with four active and two inactive shifts per type, with one to three employees per day assigned to each active shift. If the possible shifts for a given shift type are less than six, we reduce the generated shifts accordingly, giving precedence to the inactive ones.

3.2.2 Neighborhood exploration

Local search methods rely on the definition of neighborhood relation, which is the core feature for the exploration of the search space. The neighborhood of a solution Q is the set of solutions which are obtained applying a set of local perturbations, called *moves*, on Q.

In this work we consider three different neighborhood relations that are combined in the spirit of the multi-neighborhood search (Di Gaspero and Schaerf, 2003a). The way these relations are employed during the search is thoroughly explained in Section 3.2.3. In the following, we formally describe each neighborhood relation by means of the attributes needed to identify a move, the preconditions for its applicability, the effects of the move and, where necessary, some rules for handling special cases.

Given a state (Q, X) of the search space the types of moves considered in this work are the following:

ChangeStaff (CS): The staff of a shift is increased or decreased by one employee

Attributes: $\langle s_i, j, a \rangle$, where $s_i \in Q$ is a shift, $j \in \{1, \ldots, d\}$ is a day, $a \in \{\uparrow, \downarrow\}$.

Preconditions: $w_j(s_i) > 0$ if and only if $a = \downarrow$.

Effects: if $a = \uparrow$ then $w'_j(s_i) := w_j(s_i) + 1$, else $w'_j(s_i) := w_j(s_i) - 1$

Special cases: if s_i is an inactive shift (and $a = \uparrow$, by precondition), then s_i becomes active and a new randomly created inactive shift of type $K(s_i)$ is inserted (distinct from the other shifts).

ExchangeStaff (ES): One employee in a given day is moved from one shift to another one of the same type.

Attributes: $\langle s_{i_1}, s_{i_2}, j \rangle$, where $s_{i_1}, s_{i_2} \in Q$, and $j \in \{1, \ldots, d\}$.

Preconditions: $w_j(s_{i_1}) > 0,\ K(s_{i_1}) = K(s_{i_2})$.

Effects: $w'_j(s_{i_1}) := w_j(s_{i_1}) - 1$ and $w'_j(s_{i_2}) := w_j(s_{i_2}) + 1$.

Special cases: If s_{i_2} is an inactive shift, s_{i_2} becomes active and a new random distinct inactive shift of type $K(s_{i_1})$ is inserted (if such a distinct shift exists). If the move makes s_{i_1} inactive then, in the new state, the shift s_{i_1} is removed from the set Q.

ResizeShift (RS): The length of the shift is increased or decreased by 1 time-slot, either on the left-hand side or on the right-hand side.

Attributes: $\langle s_i, l, p \rangle$, where $s_i = [\sigma_i, \sigma_i + \lambda_i) \in Q$, $l \in \{\uparrow, \downarrow\}$, and $p \in \{\leftarrow, \rightarrow\}$.

Effects: We denote with δ the size modification to be applied to the shift s_i, that is $\delta = +1$ when the shift is enlarged by one timeslot and $\delta = -1$ when the shift is shrunk.

If $p = \leftarrow$ the action identified by l is performed on the left-hand side of s_i, that is $\sigma'_i := \sigma_i + \delta h$ and λ_i does not change. Conversely, if $p = \rightarrow$ the move takes place to the right-hand side, therefore σ_i remains unchanged and $\lambda'_i := \lambda_i + \delta h$.

Preconditions: For this kind of move we require that the shift s'_i, obtained from s_i by the application of the move must be feasible with respect to the shift type $K(s_i)$.

In a previous work of some of us (Musliu et al., 2004), we define many neighborhood relations for this problem including CS, ES, and a variant of RS. In this paper, instead, we restrict ourselves to the above three relations for the following two reasons.

CS and RS represent the most atomic changes, so that all other move types can be built as chains of moves of these types. For example an ES move can be obtained by a pair of CS moves that decreases one employee from a shift and assigns him/her in the same day to the other shift.

Even though ES is not a basic move type, it turned out to be very effective for the search. In fact, the move that passes one employee from a shift to another one makes a very small change to the current state, which represents a fine grain adjustments that could not be found by the other move types.

Inactive shifts are conceived to provide a single uniform way to move staff between shifts and to new shifts. This approach limits the insertion of employees in new shifts only to the current inactive ones, rather than considering all possible shifts belonging to the shift types (which are many more). Obviously, we could also insert as many inactive shifts as compatible with the shift type, thus allowing to insert any possible shift. Preliminary experimental results, though, show that there is a trade-off between computational cost and search quality, which seems to have its best trade-off in having 2 inactive shifts per type.

3.2.3 Search strategies

In a preliminary test phase, we experimented CS, ES, and RS neighborhoods driven by three different meta-heuristics, namely randomized hill climbing, tabu search and simulated annealing. The one that gave best results is tabu search, and in this work we report only the results with tabu search.

A full description of tabu search is out of the scope of this paper and we refer to (Glover and Laguna, 1997) for a general introduction. We later in this section describe its specialization to our problem.

Differently from Musliu et al. (2004), that use tabu search as well, we employ the three neighborhood relations selectively in various phases of the search, rather than exploring the overall neighborhood at each iteration.

Our strategy is to combine the neighborhood relations CS, ES, and RS, according to the following scheme made of compositions and interleaving. That is, our algorithm interleaves three different tabu search *runners* using the following neighborhoods:

- the ES alone
- the RS alone
- the set-union of the two neighborhoods CS and RS

The runners are invoked sequentially and each one starts from the best state obtained from the previous one. The overall process stops when a full round of all of them does not find any improvement. Each single runner stops when it does not improve the current best solution for a given number of iterations (called *idle iterations*).

The reason for using limited neighborhood relations is not to improve the computational efficiency, which could be obtained in other ways, for example by a clever ordering of promising moves. The main reason, instead, is the introduction of a certain degree of *diversification* in the search. In fact, certain move types would be selected very rarely in a full-neighborhood exploration strategy, even though they could help to escape from local minima. For example, a runner that uses all three neighborhood relations together would almost never perform a CS move that worsens the objective function, simply because it can always find an ES move that worsen it by a smaller amount. On the other hand, the neglected CS move could lead to a more promising region of the search space. This intuition is supported by the experimental analysis that shows that our results are much better than those in (Musliu et al., 2004).

Parameter	TS(ES)	TS(RS)	TS(CS∪RS)
Tabu range	10-20	5-10	20-40 (CS) 5-10 (RS)
Idle iterations	300	300	2000

Table 4: Tabu search parameter settings

This composite solver is further improved by performing a few changes on the final state of each runner, before handing it over as the initial state of the following runner. In details, we make the following two adjustments:

- Identical shifts are merged into one. When the procedure applies RS moves, it is possible that two shifts become identical. This situation is not detected at each move, because it is a costly operation, and is therefore left to this inter-runner step.
- Inactive shifts are recreated. That is, the current inactive shifts are deleted, and new distinct ones are created at random in the same quantity. This step, again, is meant to improve the diversification of the search algorithm.

Concerning the prohibition mechanism of tabu search, for all three runners, the size of the tabu list is kept dynamic by assigning to each move a number of tabu iterations randomly selected within a given range. The ranges vary for the three runners, and they were selected experimentally. The ranges are roughly suggested by the cardinality of the different neighborhoods, in the sense that a larger neighborhood deserves a longer tabu tenure. According to the standard aspiration criterion defined in (Glover and Laguna, 1997), the tabu status of a move is dropped if it leads to a state better than the current best.

As already mentioned, each runner stops when it has performed a fixed number of iterations without any improvement.

Tabu lengths and idle iterations are selected once for all, and the same values were used for all the instances. The selection turned out to be robust enough for all tested instances. The selected parameter values are reported in Table 4.

4 Experimental results

In this section, we describe the results obtained by our solvers on a set of benchmark instances. First, we introduce the instances used in this experimental analysis, then we illustrate the performance parameters that we want to highlight and we present the outcomes of the experiments. We conclude the section with an analysis that aims at classifying the instances in terms of their computational hardness.

4.1 Description of the instances

The benchmark consists of three different sets, each containing thirty randomly generated instances. Instances were generated in a structured way so as to ensure that they look as similar as possible to real instances, while allowing for the construction of arbitrarily difficult cases.

Set 1 contains the 30 instances that were described and investigated in (Musliu et al., 2004). They vary in their complexity and we mainly include them to be able to compare the new heuristics with the results reported in (Musliu et al., 2004) for the OPA implementation. These instances were

basically generated by first constructing a feasible solution, called the *seed* solution, and then taking the resulting staffing numbers as workforce requirements. This implies that a solution with zero deviation from workforce requirements is known in advance. In a few cases, our heuristics could find better solutions for some instances, so the seed solution may be non-optimal. In the following we refer to the best solutions we could come up with for these instances, either the seed solution or the improved one, as 'best known' solutions.

Set 2 contains instances similar to Set 1, but here the best known solutions of instances 1 to 10 were constructed to feature 12 shifts, those of instances 11 to 20 to feature 16 shifts, and those of instances 21 to 30 to feature 20 shifts. This allows us to study the relation between the number of shifts in the best known solutions and the running times of the heuristics.

While knowing these best known solutions eases the evaluation of the proposed heuristics, it also might form a biased preselection toward instances where zero deviation solutions exist for sure, thereby letting all or some of the heuristics behave in ways that are unusual for instances for which no such solution can be constructed. The Set 3 is therefore composed of instances where presumably solutions without deviations do not exist. Instances of Set 3 were constructed with the same random instance generator as the two previous sets but allowing the constructed solutions to contain invalid shifts that deviate from normal starting times and lengths by up to 4 timeslots. The number of shifts is similar to those in Set 2, i.e., instances 1 to 10 feature 12 shifts (invalid and valid ones) etc. This construction ensures that it is unlikely that zero deviation solutions exist for these instances. It might also be of interest to see whether a significant difference in performance for some of the heuristics can be recognized compared to Set 2, which would provide evidence that the way Sets 1 and 2 were constructed constituted a bias for the heuristics.

All sets of instances are available in self-describing text files from `http://www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html`. A detailed description of the random instance generator used to construct them can be found in (Musliu et al., 2004).

4.2 Experimental setting

In this work we make two types of experiments, aiming at evaluating two different performance parameters:

1. median time necessary to reach the best known solution,
2. average objective value obtained within a time bound.

The latter parameter consists of the weighted sum of the three components F_1 (excess), F_2 (shortage) and F_3 (number of shifts) (see Section 1). The components F_1 and F_2 are measured in number of workers in excess/shortage per minute, whereas the component F_3 is multiplied by 60 minutes. This way, the penalty of each shift is the same as one worker in excess/shortage for a whole hour.

Our experiments have been run on different machines. The local search solvers are implemented in C++ using the EASYLOCAL++ framework (Di Gaspero and Schaerf, 2003b) and they were compiled using the GNU g++ compiler version 3.2.2 on a 1.5 GHz AMD Athlon PC running Linux kernel 2.4.21. The greedy *min*-COST *max*-FLOW algorithm, instead, was coded in MS Visual Basic and runs on a MS Windows NT 4.0 computer.

The running times have been normalized according to the DIMACS netflow benchmark[2] to the times of the Linux PC (calibration timings on that machine for above benchmark: `t1.wm: user`

[2] `ftp://dimacs.rutgers.edu/pub/netflow/benchmarks/c/`

`0.030 sec t2.wm: user 0.360 sec`). Because of the normalization the reported running times should be taken as indicatory only.

Our experiments deal with the following three heuristic solvers:

LS The local search procedure repeated several times starting from different (random) initial solutions. The procedure is stopped when the time granted is elapsed or the best solution is reached.

GrMCMF GreedyMCMF() is called repeatedly until the stopping criterion is reached. Since the selection of the next shift to be removed in the main loop of GreedyMCMF() is done randomly, we call the basic heuristic repeatedly and use bootstrapping as described in (Johnson, 2002) to compute expected values for the computational results (counting the preprocessing step only once for each instance since it computes the same split-off time for all runs).

GrMCFC+LS The two solvers are combined using the solutions delivered by GrMCMF as initial states for LS trials. In order to maintain diversification, we exploit the non-determinism of GrMCMF to generate many different solutions. The initial state of each trial of LS is randomly selected among those states.

4.3 Computational results

The first experiment evaluates the running times needed to reach the best known solution. We ran the solvers on data Set 1 for 10 trials until they could reach the best known solution, and we recorded the running times for each trial.

Table 5 shows the average times and their standard deviations (in parentheses) expressed in seconds, needed by our solvers to reach the best known solution. The first two columns show the instance number and the best known cost for that instance. The third column reports the cost of the best solution found by *OPA* (Musliu et al., 2004). Bold numbers in the second column indicate that the best known solution for this instance was not found by *OPA*. Dash symbols denote that the best known solution could not be found in any of the 10 trials for those instances.

First, note that all three solvers in general produce better results than the commercial tool. In fact, LS always finds the best solution, GrMCMF in 20 cases and GrMCMF+LS in 29 cases out of 30 instances. OPA, instead, could find the best solution only for 17 instances. However, looking at the time performance on the whole set of instances, it is clear that LS is roughly 30 times slower than GrMCMF and 1.5 times slower than the hybrid heuristic. GrMCMF+LS is significantly outperformed by LS only for some few instances for which GrMCMF could not find the best known solution, thus biasing the local search part of the heuristic away from search space near the best known solution.

As a general remark, the LS algorithm proceeds by relatively sudden improvements, especially in the early phases of the search, while the behavior of the GrMCMF+LS is much smoother (we omit graphs showing this for brevity).

Starting local search from the solution provided by GrMCMF has also an additional benefit in terms of the increase of robustness, roughly measured by the standard deviations of the running times. In fact, for this set of instances, while the standard deviation for LS is about 50% of the average running time, this value decreases to 35% for GrMCMF+LS. The behavior of the GrMCMF solver is similar to the one of the hybrid heuristic and the standard deviation is about 35% of the average running time.

Instance	Best	OPA Musliu et al. (2004)	GrMCMF		LS		GrMCMF+LS	
1	480	480	0.07	(0.00)	5.87	(4.93)	1.06	(0.03)
2	**300**	390	—	(—)	16.41	(9.03)	40.22	(27.93)
3	600	600	0.11	(0.01)	8.96	(5.44)	1.64	(0.05)
4	**450**	1,170	—	(—)	305.37	(397.71)	108.29	(75.32)
5	480	480	0.20	(0.16)	5.03	(2.44)	1.75	(1.43)
6	420	420	0.06	(0.01)	2.62	(0.99)	0.62	(0.02)
7	**270**	570	1.13	(1.10)	10.25	(5.77)	6.95	(2.88)
8	**150**	180	—	(—)	18.98	(15.70)	10.64	(0.56)
9	**150**	225	3.53	(2.63)	11.85	(2.28)	8.85	(1.56)
10	**330**	450	—	(—)	66.05	(41.27)	84.11	(99.85)
11	30	30	0.21	(0.00)	1.79	(0.37)	0.85	(0.02)
12	90	90	0.25	(0.00)	6.10	(1.50)	3.84	(0.10)
13	105	105	0.35	(0.13)	7.20	(2.30)	3.82	(0.09)
14	**195**	390	—	(—)	561.99	(404.33)	60.97	(51.08)
15	180	180	0.04	(0.00)	0.89	(0.11)	0.40	(0.01)
16	**225**	375	—	(—)	198.50	(117.84)	151.78	(125.88)
17	**540**	1,110	—	(—)	380.72	(467.64)	288.42	(27.58)
18	720	720	1.71	(1.30)	7.72	(2.89)	7.32	(3.79)
19	**180**	195	—	(—)	38.33	(20.72)	31.12	(17.99)
20	540	540	0.11	(0.01)	15.24	(6.18)	1.69	(0.07)
21	120	120	0.28	(0.00)	6.19	(1.32)	2.18	(0.11)
22	75	75	0.65	(0.45)	3.67	(0.80)	3.80	(0.86)
23	**150**	540	6.19	(2.91)	19.16	(10.92)	22.15	(15.34)
24	480	480	0.11	(0.04)	2.85	(0.38)	1.44	(0.72)
25	**480**	690	—	(—)	503.40	(136.17)	—	(—)
26	600	600	1.50	(1.14)	9.59	(6.80)	9.20	(6.20)
27	480	480	0.07	(0.00)	4.02	(0.71)	2.34	(0.06)
28	270	270	2.24	(0.94)	9.25	(7.67)	3.81	(0.62)
29	**360**	390	—	(—)	20.59	(17.20)	10.00	(4.92)
30	75	75	0.26	(0.00)	2.78	(0.30)	1.95	(0.01)

Table 5: Times to reach the best known solution for Set 1. Data are averages and standard deviations (in parentheses) for 10 trials.

4.4 Time-limited experiments

Moving to the time-limited experiments, we perform two experiments on the different sets of instances. The first experiment with limited running times aims at showing how the solver scales up with respect to the optimum number of shifts. For this purpose we recorded the cost values of 100 runs of our solvers with the different settings on the instances of Sets 1 and 2, for which the value of a good solution is known. The runs were performed by granting to each trial a time-limit of 10 seconds.

The results are grouped on the basis of the instance size and are shown in Figure 4. The X axis of the figure shows the number of shifts in the best known solution where results on instances with the same number of shifts are clustered together. The Y axis shows normalized costs, obtained by dividing the difference between the average cost and the best cost by the latter value. In other words, each cost y obtained on instance i, for which the best known cost is $best_i$, is transformed by means of the function $f_i(y) := \frac{y - best_i}{best_i}$.

Figure 4 presents the data as box-and-whiskers plots, i.e., it shows the range of variation (the interval $[f_i(min\text{-}cost_i), f_i(max\text{-}cost_i)]$), denoted by the dashed vertical line, and the frequency distribution of the solutions. The latter measure is expressed by means of a boxed area showing the range between the 1st and the 3rd quartile of the distribution (accounting for 50% of the frequency). The horizontal

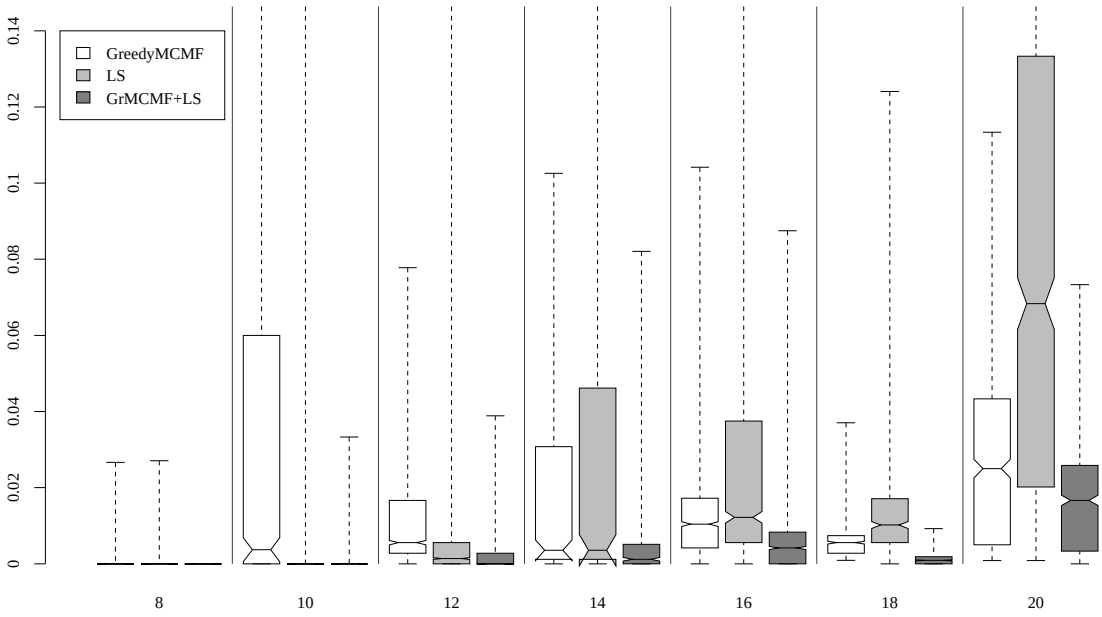


Figure 4: Aggregated normalized costs for 10s time-limit on data Sets 1 and 2.

line within the box denotes the median of the distribution and the notches around the median indicate the range for which the difference of medians is significant at a probability level of $p < 0.05$.

The figure shows that, for short runs, the hybrid solver is superior to GrMCMF and LS alone, both in terms of solution quality and robustness: the ranges of variation are shorter and the frequency boxes are tinier.

Looking at these results from another point of view, it is worth noting that GrMCMF+LS is able to find more low-cost (and even min-cost) solutions that are significantly better than those found by LS and GrMCMF. Furthermore, it is apparent that the hybrid heuristic scales better than its components, since the deterioration in the solution quality with respect to the number of shifts grows very slowly and always remains under an acceptable level (7% on the worst case, and about 2% for 75% of the runs).

The second time-limited experiment aims at investigating the behavior of the solver when provided with a very short running time on 'unknown' instances (we use here the term unknown by contrast with the sets of instances constructed around the seed solution). We performed this experiment on the Set 3 and recorded the cost values found by our solver over 100 trials. Each trial was granted 1 second of running time, in order to simulate a practical situation in which the user needs a fast feedback from the solver. As already mentioned in the introduction, for this problem, speed is extremely important, to allow for real-time refinement of requirements during a meeting with customers.

In Table 6 we report the average and the standard deviation (in parentheses) of the cost values found by each heuristic. The best algorithm on each instance is highlighted in boldface. The symbol † denotes the cases for which the difference between the distribution of solutions was not statistically significant (Mann-Whitney test with $p < 0.01$) and therefore there was no "clear winner".

In this case the hybrid heuristic performs better than LS on all instances, and it shows a better behavior in terms of algorithm robustness (in fact, the standard deviation of GrMCMF+LS is usually more than an order of magnitude smaller than the one of LS). Moreover, even the GrMCMF achieves better results than the LS heuristic, due to the running time performance of the local search procedure.

Instance	GrMCMF		LS		GrMCMF+LS	
1	2,445.00	(0.00)	9,916.35	(3,216.35)	**2,386.80**	(9.60)
2	**7,672.59**	(34.92)	9,582.00	(1,564.39)	7,691.40	(50.95)
3	**9,582.14**	(32.97)	12,367.50	(1,576.56)	9,597.00	(27.14)
4	6,634.40	(39.67)†	8,956.50	(2,091.76)	6,681.60	(119.70)†
5	10,053.75	(41.58)	10,311.60	(198.54)	**9,996.00**	(127.35)
6	2,082.17	(17.31)†	4,712.25	(1,614.34)	2,076.75	(7.50)†
7	**6,075.00**	(0.00)	12,251.70	(1,553.22)	6,087.00	(16.92)
8	9,023.46	(45.78)	10,512.60	(1,658.02)	**8,860.50**	(55.98)
9	6,039.18	(23.36)†	11,640.60	(2,264.50)	6,036.90	(28.80)†
10	**2,968.95**	(16.88)	(4,067.10)	1,226.20	3,002.40	(41.49)
11	5,511.43	(26.11)	7,888.20	(2,116.61)	**5,490.90**	(69.86)
12	4,231.96	(51.83)	11,410.05	(1,510.78)	**4,171.20**	(22.23)
13	4,669.50	(56.42)†	10,427.55	(1,593.59)	4,662.00	(39.77)†
14	**9,616.55**	36.24	10,130.40	(290.29)	9,660.60	(50.81)
15	11,448.90	(92.21)†	13,563.60	(1,415.06)	11,445.00	(110.66)†
16	10,785.00	(75.98)	11,180.40	(298.02)	**10,734.00**	(62.38)
17	4,746.56	(37.72)	11,735.40	(2,253.94)	**4,729.05**	(38.13)
18	6,769.41	(84.94)	9,516.60	(1,929.90)	**6,692.40**	(54.55)
19	5,183.16	(57.26)	10,825.20	(2,254.98)	**5,157.45**	(51.22)
20	**9,153.90**	(80.98)	12,481.80	(1,834.54)	9,174.90	(62.97)
21	6,072.86	(44.37)	14,102.55	(1,194.54)	**6,053.55**	(32.28)
22	12,932.31	(82.87)	16,418.70	(1,729.19)	**12,870.30**	(63.89)
23	**8,384.24**	(122.90)	9,788.40	(886.01)	8,390.40	(72.22)
24	10,545.00	(26.11)	11,413.20	(769.58)	**10,417.80**	(79.84)
25	**13,204.80**	(18.45)	14,038.80	(701.17)	13,252.20	(100.42)
26	13,152.73	(97.27)†	17,326.50	(2,421.45)	13,117.80	(114.15)†
27	10,084.94	(24.86)†	10,866.60	(651.65)	10,081.20	(45.91)†
28	10,641.21	(100.81)	11,543.40	(675.60)	**10,603.80**	(86.49)
29	6,799.41	71.78	12,075.30	2,710.45	**6,690.00**	(62.81)
30	13,770.68	92.32	14,808.60	692.80	**13,723.80**	(72.79)

Table 6: Cost values on Set 3 within 1s time-limit. Data are averages and standard deviations (in parentheses) over 100 trials.

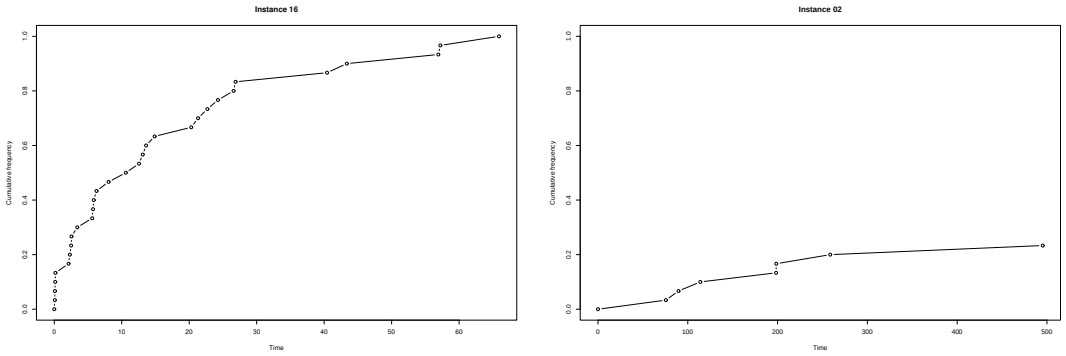
However, differently from the results of the previous experiment, the hybrid heuristic does not dominate the GrMCMF on all instances. In fact, it is possible to see that GrMCMF+LS finds better results on 15 instances, whereas GrMCMF prevails in 8 cases. On 7 instances there is no clear winner among the two heuristics, and these cases are indicated by the symbol † in the table.

The reason of this behavior is related to the amount of running time needed by the local search procedure in comparison to the short time granted (1 sec). Indeed, in another experiment (omitted for brevity) with a higher time-limit, the behavior of the three heuristics tends to be similar to the one observed in the previous experiment, indicating the absence of bias in the construction of sets 1 and 2 of instances.

4.5 Instance analysis

Our last set of experiments aims at shedding light on the characteristics of the instances employed in these experimentation. Our attempt is to understand which features make a given instance hard to solve by our heuristics.

For this purpose, we run 30 trials of 500 seconds of the GrMCMF+LS solver for each instance of Set 3, and we plot the so-called *cumulative relative frequency*. The plots show the average times necessary to reach the best-known solution with a given fraction of trials, and represent an estimation



(a) *Easy* instance (b) *Hard* instance

Figure 5: Distribution of the running times over 30 trials of GrMCMF+LS up to the best-known solution on two instances of set 3.

of the probability of finding the best solution within a given time limit. This frequency for two instances is shown in Figure 5. For example, the point (10,0.43) in a plot means that in 10 seconds, 43% of the runs reached the best-known solution.

As for other local search solvers for NP-hard problems (Hoos and Stützle, 1999), the probability of finding a solution over time has the shape of an exponential distribution (i.e., $P(X \leq x) = 1 - e^{-\lambda x}$).

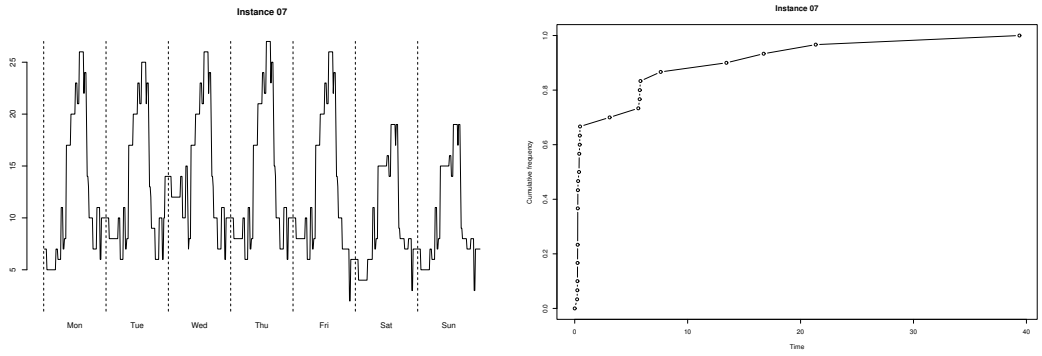
For some instances the algorithm could not always reach the best known solution within the time limit of 500 seconds. Therefore, we have only an approximate picture for those instances and we consider them as *hard* instances. An example of the behavior of the algorithm on this kind of instances is provided in Figure 5(b). By contrast, we classify the other instances, such the one shown in Figure 5(a), as *easy* instances.

Given this classification of hard and easy instances, we attempt to find out which features of a given instance make it belong to one or the other class. For this purpose, since all the other parameters remain unchanged among the whole set of instances, we look at the requirements profile.

Our first hypothesis is that the presence of peaked requirements (i.e., profiles with frequent differences within adjacent timeslots) might be a source of hardness for our heuristic. However, after a comprehensive analysis (not presented for brevity) of all the graphs we are able to refute this hypothesis. A counterexample is provided with the graphs of Figure 6. In fact, the two instances presented in the figure have a similar peaked shape, but the GrMCMF+LS solver exhibits quite a different behavior on the two cases (the first instance is easy and the second one hard).

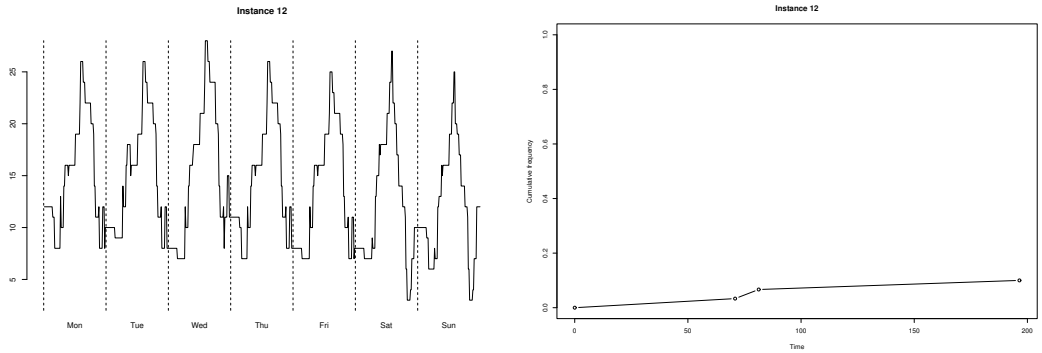
Another reasonable hypothesis for characterizing the behavior of the heuristic is that the source of hardness might be related to the variability of the requirements among different days. Unfortunately, also this hypothesis is rejected by our complete analysis. An example is shown in Figure 7. The figure reports the running-time distribution for two instances with similar variabilities but the behavior of the GrMCMF+LS is quite different in the two cases.

It is worth noting that if we relax our success criterion by including also those runs that reach a solution whose cost is slightly above the best known value, then most of the hard instances become easy. This is shown by the plots in Figure 8, which report the behavior of the heuristic up to the best-known solution (Fig. 8(a)) against the behavior of the heuristic to a reference solution whose cost is 2% above the cost of the best-known solution (Fig. 8(b)). This phenomenon suggests that the main source of hardness is the presence, in the cost landscape, of one or more "nearly optimal" solution with a large basin of attraction for local search.



(a) Requirements

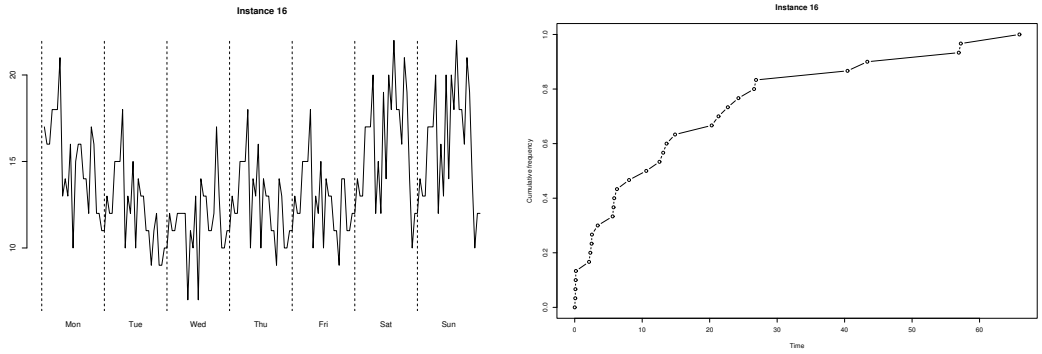
(b) Running-time distribution



(c) Requirements

(d) Running-time distribution

Figure 6: Behavior of the GrMCMF+LS on instances with peaked requirements.



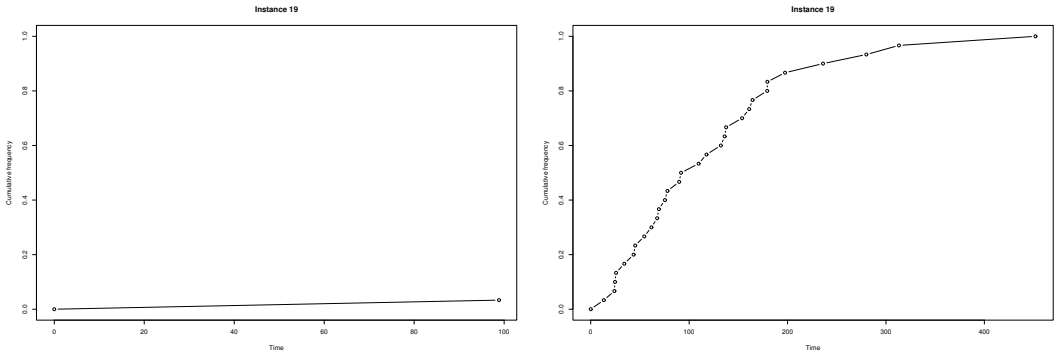
(a) Requirements

(b) Running-time distribution

(c) Requirements

(d) Running-time distribution

Figure 7: Behavior of the GrMCMF+LS on instances with variable requirements among different days.



(a) Running-time distribution to the best-known solution

(b) Running-time distribution to a reference solution whose cost is 2% above the cost of the best-known solution

Figure 8: Strict and relaxed behavior of the GrMCMF+LS.

In conclusions, a characterization of the instances in term of their features is not a simple issue for MSD and it needs additional analysis on the instances in order to obtain a precise picture.

5 Related work

Though there is a wide literature on shift scheduling problems (see, e.g., Laporte, 1999, for a recent survey), the larger body of work is devoted to the problem of allocation of resources to shifts, for which network flow techniques have, among others, been applied (e.g., Balakrishnan and Wong, 1990; Bartholdi et al., 1980).

Heuristics for a shift selection problem that has some similarity with *min*-SHIFT DESIGN have been studied by Thompson (Thompson, 1996). Bartholdi et al. (1980) noted that a problem similar to *min*-SHIFT DESIGN can be translated into a *min*-COST *max*-FLOW problem (which is a polynomial problem, see, e.g., (Papadimitriou and Steiglitz, 1982)) and thus efficiently solved. The translation can be applied under the hypothesis that the requirement of minimizing the number of selected shifts is neglected and the costs for assignments that do not fulfill the requirements are linear.

The relation between consecutive ones in *rows* matrices and flow, and, moreover, the relation of these matrices with shortest and longest path problems on DAGs were first given in Veinott and Wagner (1962). In (Hochbaum, 2000) optimization problems on c1 matrices (on columns) are studied. For problems on circular ones matrices and further references see (Bartholdi et al., 1980) and (Hochbaum, 2000).

To the authors knowledge, the only paper that deals explicitly with the formulation of the *min*-SHIFT DESIGN problem considered in this work is a previous paper of Musliu et al. (2004), which presents the OPA software. In Section 4, we have compared our heuristic solver with the implementation described in (Musliu et al., 2004) by applying it to the set of benchmark instances used in that paper.

6 Conclusions

The *min*-SHIFT DESIGN problem is an important shift scheduling problem that arises in many industrial contexts. We provided complexity results for it and designed a hybrid heuristic algorithm com-

posed of a constructive heuristic (suggested by the complexity analysis) and a multi-neighborhood tabu search procedure.

This problem appears to be quite difficult in practice, even for small instances, which is also supported by the theoretical results. An important source of hardness is related to the variability in the size of the solution, since dropping this requirement makes the problem solvable in polynomial time.

In the experimental part, the hybrid heuristic and its underlying components have been evaluated both in terms of ability to reach good solutions and in quality of solutions reached in fast runs. The outcomes of the comparison show that the hybrid heuristic combines the good features of its components. Indeed, it obtained the best performances in terms of solution quality (thanks to the increased thoroughness allowed by tabu search) but with a lower impact on the overall running time. Furthermore, we compare our heuristics with the results obtained with a commercial software as reported in (Musliu et al., 2004). Our hybrid heuristic clearly outperforms this commercial implementation, and thus can be considered as the best general-purpose solver among the other heuristics that were compared to it.

In practice, a number of further optimization criteria clutters the problem, e.g., the average number of working days per week. This number is an extremely good indicator with respect to how difficult it will be to develop a schedule and what quality that schedule will have. The average number of duties thereby becomes the key criterion for working conditions and is sometimes even part of collective agreements. Fortunately, this and most further criteria can easily be handled by straightforward extensions of the heuristics described in this paper and add nothing to the complexity of MSD. We therefore concentrate on the three main criteria described in this paper.

7 Further ideas

The GreedyMCMF heuristic could be made even more efficient by noting that usually only very few edges change from one call to the next call of the MCMF() subroutine. We currently call the MCMF() subroutine each time from scratch. However, CS2 (Goldberg, 1997) supports a variant that recomputes an optimal flow more efficiently after a change in costs. It might thus prove worthwhile to track changes in the flow instance and recompute only those parts that are necessary, thus speeding up the MCMF() calls in the heuristics.

An idea for a promising heuristic might also be to integrate the MCMF() subroutine more directly in the local search procedure instead of just calling them serially one after the other as in the third variant of our heuristics.

Another simple heuristic that suggests itself might be to combine the MCMF() subroutine (together with the postprocessing step described in Section 3.1 that reestablishes cyclicity) with a genetic algorithm type of optimization heuristic. Indeed, the genetic code could consist merely of a bitvector of all possible shifts, possibly ordered by their starting times. The phenotype would then consist of the shifts and number of staff as computed by the MCMF() subroutine followed by the postprocessing step, applied only to the subset of shifts that have their bits set to 1. Optimization could then be done with the usual crossover and mutation operators on populations of solution candidates, with selection being based probabilistically on the scores of the phenotype solution candidates. Initial populations could contain random bitvectors as well as shifts selected by single runs of the heuristics described in this paper.

We also tried to apply PPRN[3], a library for nonlinar network flow problems described in (Castro and Nabona, 1996) to our instances instead of calling $MCMF$ but got only unsatisfactory results as

[3] http://www-eio.upc.es/~jcastro/pprn.html

this package cannot correctly deal with fixed charge style nonlinearities. Other software specializing on $MECF$ type problems or aiming at more general integer constraint problems might yield better results.

Acknowledgments: This work was supported by Austrian Science Fund Project No. **Z29-N04** and by the Italian Ministry of Education, University and Research (MIUR) under the project PRIN 2003 "Design and implementation of a solver based on local search for the execution of declarative specifications for combinatorial problems".

References

Aarts, E. and Lenstra, J. K., editors (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons, New York (USA).

Ahuja, R., Magnanti, T., and Orlin, J. (1993). *Network Flows*. Prentice Hall, Englewood Cliffs (NJ, USA).

Balakrishnan, N. and Wong, R. (1990). A network model for the rotating workforce scheduling problem. *Networks*, 20:25–42.

Bar-Ilan, J., Kortsarz, G., and Peleg, D. (2001). Generalized submodular cover problems and applications. *Theoretical Computer Science*, 250(1–2):179–200.

Bartholdi, J., Orlin, J., and H.Ratliff (1980). Cyclic scheduling via integer programs with circular ones. *Operations Research*, 28:110–118.

Burke, E. K., Causmaeker, P. D., Berghe, G. V., and Landeghem, H. V. (2004). The state of the art of nurse rostering. *Journal of Scheduling*, 7:441–499.

Castro, J. and Nabona, N. (1996). An implementation of linear and nonlinear multicommodity network flows. *European Journal of Operational Research*, 92:37–53.

Di Gaspero, L. and Schaerf, A. (2003a). Multi-neighbourhood local search with application to course timetabling. In Burke, E. and Causmaecker, P. D., editors, *Lecture Notes in Computer Science*, number 2740 in Lecture Notes in Computer Science, pages 263–278. Springer-Verlag, Berlin-Heidelberg (Germany).

Di Gaspero, L. and Schaerf, A. (2003b). EASYLOCAL++: An object-oriented framework for flexible design of local search algorithms. *Software Practice & Experience*, 33(8):733–765.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability—A guide to NP-completeness*. W.H. Freeman and Company, San Francisco.

Gärtner, J., Musliu, N., and Slany, W. (2001). Rota: a research project on algorithms for workforce scheduling and shift design optimization. *AI Communications: The European Journal on Artificial Intelligence*, 14(2):83–92.

Glover, F. and Laguna, M. (1997). *Tabu search*. Kluwer Academic Publishers, Dordrecht (the Netherlands).

Glover, F. and McMillan, C. (1986). The general employee scheduling problem: An integration of MS and AI. *Computers & Operations Research*, 13(5):563–573.

Goldberg, A. (1997). An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22:1–29.

Hochbaum, D. (2000). Optimization over consecutive 1's and circular 1's constraints. Unpublished manuscript.

Hoos, H. H. and Stützle, T. (1999). Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artificial Intelligence*, 112:213–232.

Hoos, H. H. and Stützle, T. (2005). *Stochastic Local Search: foundations and applications*. Morgan Kaufmann, San Francisco (CA), USA.

Jackson, W., Havens, W., and Dollard, H. (1997). Staff scheduling: A simple approach that worked. Technical Report CMPT97-23, Intelligent Systems Lab, Centre for Systems Science, Simon Fraser University. Available at `http://citeseer.nj.nec.com/101034.html`.

Johnson, D. S. (2002). A theoretician's guide to the experimental analysis of algorithms. In Goldwasser, M. H., Johnson, D. S., and McGeoch, C. C., editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, pages 215–250. American Mathematical Society.

Krumke, S., Noltemeier, H., Schwarz, S., Wirth, H.-C., and Ravi, R. (1998). Flow improvement and network flows with fixed costs. In *Proceedings of the International Conference on Operations Research (OR-98)*, Zürich (Switzerland).

Laporte, G. (1999). The art and science of designing rotating schedules. *Journal of the Operational Research Society*, 50:1011–1017.

Lau, H. (1996). On the complexity of manpower scheduling. *Computers & Operations Research*, 23(1):93–102.

Musliu, N., Gärtner, J., and Slany, W. (2002). Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1–2):85–98.

Musliu, N., Schaerf, A., and Slany, W. (2004). Local search for shift design. *European Journal of Operational Research*, 153(1):51–64.

Papadimitriou, C. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Englewood Cliffs (NJ, USA).

Raz, R. and Safra, S. (1997). A sub constant error probability low degree test, and a sub constant error probability PCP characterization of NP. In *Proceedings of the 29th ACM Symposium on Theory of Computing*, pages 475–484, El Paso (Texas, USA).

Thompson, G. (1996). A simulated-annealing heuristic for shift scheduling using non-continuously available employees. *Computers & Operations Research*, 23(3):275–278.

Tien, J. and Kamiyama, A. (1982). On manpower scheduling algorithms. *SIAM Review*, 24(3):275–287.

Veinott, A. and Wagner, H. (1962). Optimal capacity scheduling: Parts I and II. *Operation Research*, 10:518–547.