A mixed AI-OR heuristic for the minimum shift design problem

a Di Gaspero[*] and **Johannes Gärtner**[†] and **Guy Kortsarz**[‡] and **Nysret Musliu**[§] and **Andrea Schaerf**[¶] and **Wolfgan**[‖]

Abstract

We study the minimum shift design problem (MSD) that arose in a commercial shift scheduling software project: Given a collection of shifts and workforce requirements for a certain time interval, we look for a minimum cardinality subset of the shifts together with an optimal assignment of workers to this subset of shifts such that the deviation from the requirements is minimum. 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 $MECF$. We give a logarithmic hardness of approximation lower bound. In the second part of the paper, we present practical heuristics for MSD. First, we describe a local search procedure based on interleaving different neighborhood definitions. Second, we describe a new greedy heuristic that uses a min-cost max-flow ($MCMF$) subroutine, inspired by the relation between the MSD and $MECF$ problems. The third heuristic consists of a serial combination of the other two. An experimental analysis on structured random instances shows that our new heuristics clearly outperform an existing commercial implementation and highlights the respective merits of the heuristics for different performance parameters.

[*] digasper@dimi.uniud.it, University of Udine, Italy

[†] gaertner@ximes.com, Ximes Inc, Austria

[‡] guyk@camden.rutgers.edu, Rutgers University, USA

[§] musliu@dbai.tuwien.ac.at, TU Wien, Austria

[¶] schaerf@uniud.it, University of Udine, Italy

[‖] wsi@dbai.tuwien.ac.at, TU Wien, Austria

1 Introduction

The minimum shift design problem (MSD) concerns selecting which work shifts to use, and how many people to assign to each shift, in order to meet prespecified staffing requirements.

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, producing a software end-product called OPA (short for 'OPerating hours Assistant'). **HIDE? [** *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. 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.*,].

The staffing requirements are given for h days, which usually span a small multiple of a week, and are valid for a certain amount of time ranging from a week up to a year, typically consisting of several months (in the present paper, we disregard the problem of connecting several such periods, though this is handled in OPA). Each day j is split into n equal-size smaller intervals, called *timeslots*, which can last from a few minutes up to several hours. The staffing requirement for the ith timeslot ($i = 0, \ldots, n-1$) on day $j \in \{0, \ldots, h-1\}$ starting at t_i, namely $[t_i, t_{i+1})$, is fixed. **HIDE? [***As usual, t_{n+i} is equal to t_i of the following day.* **]** For every i and j we are given an integer value $b_{i,j}$ representing the number of persons needed at work from time t_i until time t_{i+1} on day j, with cyclic repetions after h days. Table 1 shows an example of workforce requirements with $h = 7$, in which, for conciseness, timeslots with same

Start	End	Mon	Tue	Wen	Thu	Fri	Sat	Sun
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	06:00	5	5	5	5	5	5	5

Table 1: Sample workforce requirements.

Shift type	Possible start times	Possible length
M (morning)	06:00 – 08:00	7h – 9h
D (day)	09:00 – 11:00	7h – 9h
A (afternoon)	13:00 – 15:00	7h – 9h
N (night)	22:00 – 24:00	7h – 9h

Table 2: Typical set of shift types.

requirements are grouped together (adapted from a real call-center).

When designing shifts, not all starting times are feasible, neither is any length allowed. The input thus also includes a collection of *shift types*. A shift type has minimum and maximum start times, and mimimum and maximum length. Table 2 shows a typical example of the set of shift types. Each shift $I_{s,l}$ with starting time t_s with $s \in \{0, \ldots, n-1\}$ and length l, belongs to a type, i.e., its length and starting times must necessarily be inside the intervals defined by one type. The shift types determine the m available shifts. Assuming a timeslot of length 15 minutes, there are $m = 324$ different shifts belonging to the types of Table 2. The type of shift I is denoted by $T(I)$.

The goal is to decide how many persons $x_j(I_{s,l})$ are going to work in each shift $I_{s,l}$ each day j so that $b_{i,j}$ people will be present at time $[t_i, t_{i+1})$ for all i and j. Many of the shifts are never used, hence for an unused shift I, $x_j(I) = 0$ for all j.

Let $\mathcal{I}_{t_i}$ be the collection of shifts that include t_i. A feasible solution gives h numbers $x_j(I)$ to each shift $I = I_{s,l}$ so that $p_{i,j} \stackrel{\text{def}}{=} \sum_{I \in \mathcal{I}_{t_i}} x_j(I) = b_{i,j}$, namely, the number of workers present at time t_i for all values of $i \in \{0, \ldots, n-1\}$ for all days $j \in \{0, \ldots, h-1\}$ meets the staffing requirements. This constraint is usually relaxed such that small deviations are allowed.

HIDE? [*Note that a better fit of the requirements might sometimes be achieved by looking for solutions covering more than one cycle of h days. Since this could easily be handled by extending the proposed heuristics or, even simpler, by repeating the requirements for a corresponding number of times, additionally is only very seldomly considered in practice, and theoretically adds nothing to the problem, we do not consider it in this paper.* **]**

We now discuss the quality of solutions, i.e. the *objective function* to minimize. When we allow small deviations to the requirements, there are three main objective components. The first and second are, naturally, the staffing excess and shortage, namely, the sums $ex \stackrel{\text{def}}{=} \sum_{i,j}(\max(0, p_{i,j} - b_{i,j}))$ and $sh \stackrel{\text{def}}{=} \sum_{i,j}(\max(0, b_{i,j} - p_{i,j}))$. The third component is the number of shifts selected. 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 **HIDE? [** *that have a number of advantages , e.g., if one tries to keep teams of persons together. Such teambuilding may be necessary due to managerial or qualification reasons. While teams are of importance in many but not all schedules, there are further advantages of fewer shifts. With fewer shifts, schedules are easier to design (with or without software support, see [Musliu* et al., *2002]). Fewer shifts also make such schedules* **]** easier to read, check, manage and administer; each of these activities being a burden in itself. **HIDE? [***In practice, a number of further optimization criteria clutters the problem, e.g., the average number of working days per week = duties per week. This number is an extremly 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, e.g., setting 4.81 as the maximum. Fortunately, this and most further criteria can easily be handled by straigthforward extensions of the heuristics described in this paper and add nothing to the complexity of MSD. We therefore concentrate on the three main criteria mentioned at the beginning of this paragraph.***]**

Start	Length	Mon	Tue	Wen	Thu	Fri	Sat	Sun
06:00	8h	2	2	2	6	2		
08:00	8h	3	3	3	3	3	3	3
09:00	8h	2	2	2	4	2	2	2
14:00	8h	5	4	2	2	5		
22:00	8h	5	5	5	5	5	5	5

Table 3: A solution for the problem of Table 1.

In summary, we look for an assignment $x_j(I)$ to all the possible shifts that minimizes an objective function composed by a weighted sum of ex, sh and the number of used shifts, in which the weights depend on the instance. **HIDE? [***Note that, in practice, we use a more general weighted linear combination to take care of the three main as well as the other criteria, where weights can interactively be adjusted by the user.***]**

A typical solution for the problem from Table 1 that uses 5 shifts is given in Table 3. Note that there is a shortage of 2 workers every day from 10h–11h that cannot be compensated without having more shortage or excess. Also note that using less than 5 shifts leads to more shortage or excess.

In Section 2 we show a relation of MSD to the minimum edge-cost flow ($MECF$) problem (listed as [ND32] in [Garey and Johnson, 1979]). In this problem the edges in the flow network have a capacity $c(e)$ and a fixed usage cost $p(e)$. The goal is to find a maximum flow function

f (obeying the capacity and flow conservation laws) so that the cost $\sum_{e:f(e)>0} p(e)$ of edges carrying non-zero flow is minimized.

This problem is one of the more fundamental flow variants with many applications. A sample of these applications include optimization of synchronous networks **HIDE? [***(minimizing the number of connections with registers)* **]** (see [Leiserson and Saxe, 1991]), source-location (see [Arata *et al.*, 2000]), transportation (see **HIDE? [***[Equi* et al.*, 1997;* **?***; Goethe-Lundgren and Larsson, 1994; Magnanti and Wong, 1984]), scheduling (for***]** [Equi *et al.*, 1997; Goethe-Lundgren and Larsson, 1994; Magnanti and Wong, 1984]), scheduling (for example, trucks or manpower, see [Equi *et al.*, 1997; Lau, 1996]), routing (see [Hochbaum and Segev, 1989]), and designing networks (for example, communication networks with fixed cost per link used, e.g., leased communication lines, see **HIDE? [***[***?***; ***?***; ***?***; Hochbaum and Segev, 1989;* **?***; Kim and Pardalos, 1999;* **?***; Magnanti and Wong, 1984]).***]** [Hochbaum and Segev, 1989; Kim and Pardalos, 1999]).

HIDE? [*We prove that a restricted version of MSD is equivalent to a restricted variant of $MECF$ defined as follows.***]** The $UDIF$ (infinite capacities flow on a DAG) problem restricts the $MECF$ problem as follows:

1. Every edge not touching the sink or the source has infinite capacity. We call an edge *proper* if it does not touch the source of the sink. Non-proper edges, namely edges touching the source or the sink, have no restriction. Namely, they have arbitrary capacities.
2. The costs of proper edges is 1. The cost of edges touching the source or sink is zero.
3. The underlying flow network is a DAG (directed acyclic graph).
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, minimize the *number* of proper edges carrying nonzero flow (namely, minimizing $|\{e : f(e) > 0,\ e \text{ is proper}\}|$).

HIDE? [*We prove that a special case of MSD is equivalent to $UDIF$. Thus, a hardness of approximation result for $UDIF$ carries over to MSD. Indeed, we prove a logarithmic lower bound on the approximation of $UDIF$ and thus of MSD.***]**

HIDE? [

1.1 Results and organization of this paper

We show that a special case of MSD is equivalent to $UDIF$. We give a logarithmic hardness of approximation lower bound for the $UDIF$ problem (and thus for MSD) under the assumption $P \neq NP$.

[] *ws* **[experimental analysis of algorithms]]**

Related work

Flow-related work: It is well known that finding a maximum flow minimizing $\sum_e p(e)f(e)$ is a polynomial problem, namely, the well known min-cost max-flow problem (see, e.g., [Papadimitriou and Steiglitz, 1982]).

Krumke *et al* [Krumke *et al.*, 1998] studied the approximability of $MECF$. They show that, unless $NP \subseteq DTIME(n^{O(\log\log n)})$, for any $\epsilon > 0$ there can be no approximation algorithm on bipartite graphs with a performance guarantee of $(1-\epsilon)\ln F$, and also provide an $F-$ratio approximation algorithm for the problem on general graphs, where F is the flow value. [Carr *et al.*, 2000] point out a $\beta(G)+1+\epsilon$ approximation algorithm for the same problem where $\beta(G)$ is the cardinality of the maximum size bond of G, a bond being a minimal cardinality set of edges whose removal disconnects a pair of vertices with positive demand.

A large body of work is devoted to hard variants of the maximum flow problem. For example, the non-approximability of flows with priorities was studied in [Bellare, 1993]. In [Garg *et al.*, 1997] a $2-$ratio approximation is given for the NP-hard problem of multicommodity flow in trees. The same authors [Garg *et al.*, 1996] study the related problem of multicuts in general graphs. The special case of multicommodity flow, namely finding many disjoint paths (or path maximizing some profit function) is studied in [Guruswami *et al.*, 1999] and [Srinivasan, 1997]. For more such results see the compendium [Crescenzi and Kann, 1994].

Heuristic work on fixed charge network flow problems was done in **HIDE? [***[***?***; ***?***; ***?***; ***?***; ***?***; ***?***; Goethe-Lundgren and Larsson, 1994; Holmberg and Hellstrand, 1998; Hochbaum and Segev, 1989;* **?***; Khang and Fujiwara, 1991; Kim and Pardalos, 1999;* **?***; ***?***].***]** [Holmberg and Hellstrand, 1998; Khang and Fujiwara, 1991; Kim and Pardalos, 1999].

In [Even *et al.*, 2002] the hardness result for the Minimum Edge Cost Flow Problem ($MECF$) is improved. This paper proves that $MECF$ does not admit a $2^{\log^{1-\epsilon} n}$-ratio approximation, for every constant $\epsilon > 0$, unless $NP \subseteq DTIME(n^{\text{polylog}n})$. The same paper also presents a bi-criteria approximation algorithm for $UDIF$, essentially giving an n^{ϵ} approximation for the problem for every ϵ.

Work on shift scheduling: There is a large body on shift scheduling problems (see [Laporte, 1999] for a recent survey). The larger body of the work is devoted to the case where the shifts are already chosen and what is needed is to allocate the resources to shifts, for which network flow techniques have, among others, been applied (see [Bartholdi *et al.*, 1980; Balakrishnan and Wong, 1990; Lau, 1996]).

Heuristics for the selection of shifts that bear some similarity to MSD have been studied by [Thompson, 1996]. [Bartholdi *et al.*, 1980] note that a problem similar to MSD where the requirement to minimize the number of selected shift is dropped and there are linear costs for understaffing and overstaffing can be transformed

into a min-cost max-flow problem and thus efficiently solved.

The relation between consecutive ones in *rows* matrices and flow, and, moreover, the relation of these matrices 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].

The only paper that, to our knowledge, deals exactly with MSD is [Musliu *et al.*,]. In Section 4, we will compare our heuristics in detail to the commercial OPA implementation described in [Musliu *et al.*,] by applying them to the benchmark instances used in that paper.

2 Theoretical results

To simplify the theoretical analysis of MSD, we restrict MSD instances in this section to instances where $h = 0$, that is, workforce requirements are given for a single day only, and no shifts in the collection of possible shifts span over two days, that is, 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 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 and 1 matrices with the *consecutive ones* property. We say that a matrix A obeys the consecutive ones (c1) property if all entries in the matrix are either 0 or 1 and all the 1 in each column appear consecutively.

A column *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 i starts (respectively, ends) is denoted by $b(i)$ (respectively $e(i)$).

We give a formal description of MSD via c1 matrices as follows. The columns of the matrix correspond to shifts. We are given a system of inequalities: $A \cdot x \geq b$ with $x \in \mathcal{Z}^n,\ x \geq 0$, where A is an $n \times m$, c1 matrix, and b is a vector of length n of positive integers. Only x vectors meeting the above constraints are feasible. The optimization criteria is represented as follows. Let A_i be the ith row in A. Let $|x|_1$ denote the L_1 norm of x.
Input: A, b where A has the c1 property (in the columns) and the b_i are all positive.
Output: A vector $x > 0$ with the following properties.

1. The vector x minimizes $|Ax - b|_1$
2. Among all vectors minimizing $|Ax-b|_1$, x has minimum number of non-zero entries.

Claim 1 *The restricted noncyclic variant of MSD where a zero deviation solution exists (namely, $Ax^* = b$ admits a solution), $h = 1$ and all shifts start and finish on the same day, is equivalent to the $UDIF$ problem.*

The proof can be found in Appendix A, 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 to find the minimum (weigthed) 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 3.2.

We next prove that unless $P = NP$, there is some constant $c < 1$ such that approximating $UDIF$ within $c \ln n-$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.*

We use a reduction from Set-Cover. The detailed proof is given in Appendix B.

3 Practical heuristics

We present two practical heuristics. First, we describe a local search procedure based on interleaving different neighborhood definitions. Second, we describe a new greedy heuristic that uses a min-cost max-flow ($MCMF$) subroutine, inspired by the relation between the MSD and $MECF$ problems. **HIDE? [** *The third heuristic consists of a serial combination of the other two.* **]**

3.1 Local Search Heuristic Solver

Our first solver is fully based on the local search paradigm [Aarts and Lenstra, 1997]. In order to describe it, we first define the search space and the strategy for generating an initial solution. Afterwards, we describe a set of neighborhood relations for the exploration of the search space and the search strategies.

Search space and initial solution
We consider as a state S for MSD a set of shifts $\{I_1, I_2, \ldots\}$ with their staff assigned. The shifts of a state are split into two kinds:

- *Active* shifts: non-zero staff is assigned to it, that is, at least one employee is assigned to the shift at some day.
- *Inactive* shifts: they have no employees for all days in the week. This kind of shifts does not contribute to the solution and to the objective function, and its role is explained in Section 3.1.

The initial solution is built in a random way. For each shift type, we create a fixed number of random distinct active and inactive shifts. For the active ones, we assign for each day a random number of employees. In details, 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 below, we build a solution with 4 active and 2 inactive shifts per type, with 1 to 3 employees per day per shift for the active shifts. If the shift type has less than 6 shifts, we reduce the shifts accordingly, starting from inactive ones.

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 S is the set of solutions which are obtained applying a set of local perturbations, called *moves*, on S.

In this work we consider three different neighborhood relations. The way these relations are employed during the search is thoroughly explained in Section 3.1. 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, possibly, some rules for handling special cases.

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

- **Attributes:** $\langle I, j, a\rangle$, where $I \in S$, $j \in \{1..7\}$ is a day, $a \in \{\uparrow, \downarrow\}$.
- **Preconditions:** If $a = \downarrow$ then $x_j(I) > 0$.
- **Effects:** if $a = \uparrow$ then $x_j(I) = x_j(I) + 1$, else $x_j(I) = x_j(I) - 1$
- **Special cases:** if I is an inactive shift (and $a = \uparrow$, by precondition), I becomes active and a new random distinct inactive shift (if a distinct shift exists) is inserted for the type $T(I)$.

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

- **Attributes:** $\langle I_1, I_2, j\rangle$, where $I_1, I_2 \in S$, and $j \in \{1..7\}$.
- **Preconditions:** $x_j(I_1) > 0$, $T(I_1) = T(I_2)$.
- **Effects:** $x_j(I_1) = x_j(I_1) - 1$ and $x_j(I_2) = x_j(I_2) + 1$.
- **Special cases:** If I_2 is an inactive shift, I_2 becomes active and a new random distinct inactive shift (if a distinct shift exists) is inserted for the type $T(I_1)$ (equal to $T(I_2)$). If the move makes I_1 inactive, then I_1 is removed from the current state.

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:** $\langle I, l, p\rangle$, where $I \in S$, $l \in \{\uparrow, \downarrow\}$, and $p \in \{\leftarrow, \rightarrow\}$.
- **Preconditions:** The shift obtained from I by the application of the move must belong to $T(I)$.
- **Effects:** If $l = \uparrow$ the shift I is enlarged by 1 timeslot, if $l = \downarrow$ it is shrunk by 1 timeslot. If $p = \leftarrow$ the action identified by p is performed on the left-hand side of I, if $p = \rightarrow$ it takes place to the right-hand side.

In a previous work, Musliu *et al.* [Musliu *et al.*,] 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.

First, 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.

Secondly, even though ES is not a basic move type, we employ it because it turned out to be very effective for the search, especially in joint action with the concept of inactive shift. In fact, the move that passes one employee from a shift to a similar one makes a very small change to the current state, allowing thus for fine grain adjustments that could not be found by the other move types.

Inactive shifts allow us to insert new shifts and to move staff between shifts in a uniform way. This approach limits the creation of new shifts only to the current inactive ones, rather than considering all possible shifts belonging to the shift types (which are many more). The possibility of creating any legal shift is rescued if we insert as many (distinct) inactive shifts as compatible with the shift type. Experimental results, though, show that there is a trade-off between computational cost and search quality which seems to have its best compromise in having 2 inactive shifts per type.

Search strategies

We experimented with three different meta-heuristics, namely 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.* [Musliu *et al.*,], that use tabu search as well, we use the three neighborhood relations selectively in various phases of the search, rather than exploring the overall neighborhood at each iteration.

In details, we 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 ES alone, the RS alone, and the union of the two neighborhoods CS and RS, respectively.

HIDE? [*the following neighborhoods:*

- *the ES alone*
- *the RS alone*
- *the 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 an 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 related to the saving of computational time, which could be obtained in other ways, for example by 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, although the CS move could lead to a more promising region of the search space. This intuition is supported by the experimental analysis that shows the our results are much better than those in [Musliu *et al.*,].

HIDE? [*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 by the runner 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.*

]

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 are selected experimentally. **HIDE? [** *The ranges are roughly suggested by the cardinality of the different neighborhoods, in the sense that a larger neighborhood deserves a longer tabu tenure.***]** If a move is in the tabu list, its *inverse* is excluded from the neighborhood exploration. The inverse of a move is the move that applied in the state obtained from the application of the first one in S leads back to S. According to the standard aspiration criterium 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 found.

HIDE? [*As already mentioned, each runner stops when it has performed a fixed number of iterations without any improvement (called* idle iterations*).* **]** Tabu lengths and idle iterations are selected once for all, and the same values are used for all instances. The selection turned out to be robust enough for all tested instances, and it is shown in Table 4.

HIDE? [*The first set on experiments show the time needed for reaching the best solution that is known. The time necessary to run one trial of the algorithm varies between 1 and 30 seconds, depending on the instance and on the single run. In this first test, the solver is ran several times with new initial states, until it gets to the best solution that is known.* **]**

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

Table 4: Tabu search parameter settings

3.2 GreedyMCMF

Based on the equivalence of the (non-cyclic) MSD problem to $UDIF$, a special case of the $MECF$ problem for which no efficient algorithm is known (see Section 2), and the relationship of the latter with the $MCMF$ problem for which efficient algorithms are known, we propose a new greedy heuristic GreedyMCMF() that uses a polynomial min-cost max-flow subroutine MCMF(), as shown in pseudocode in Table 5. It is based on the observation that the $MCMF$ subroutine can easily compute the optimal staffing with minimum (weighted) deviation when slack edges have associated costs corresponding, respectively, to the weights of shortage and excess. Note that it is not able to simultaneously minimize the number of shifts that are used.

However, as the MCMF() subroutine cannot consider cyclicity, we must first perform a preprocessing step that determines a good split-off time where the cycle of h days should be broken. This is done by calling MCMF() with different starting times chosen between 5:00 and 8:00 on the first day of the cycle. All possibilities in this interval are tried while eliminating all shifts that span the chosen starting point when translating from MSD to the network flow instances. The number of possibilities depends on the length of the timeslots of the instance, e.g., when the timeslots last 30 minutes, MCMF() will be called with starting times 5:00, 5:30, 6:00, 6:30, 7:00, 7:30, and 8:00 in the morning of the first day. 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 shown to provide adequate results in practice, which can be explained by the observation that there is usually a complete exchange of workforce between 5 and 8 a.m. on Monday mornings.

The greedy heuristic then removes all shifts that did not contribute to the MSD 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 will not be considered for removal again.

Finally, when no shifts can be removed anymore with-

```
GreedyMCMF(SetOfAllAllowedShifts,
  WorkforceRequirements):

/* Preprocessing step: where to break cyclicity? */
SplitOffTime =
  FindBestSplitOffTime(SetOfAllAllowedShifts,
    WorkforceRequirements)

/* Greedy part with MCMF subroutine */
FlowInstance =
  MSD2Flow(SetOfAllAllowedShifts,
    WorkforceRequirements,
    SplitOffTime)
BestFlowSoFar = MCMF(FlowInstance)
MSD_Solution =
  ShiftsAndWorkforceIn(BestFlowSoFar)
MinCostSoFar = MSD_Eval(MSD_Solution)
Shifts = ShiftsInUseIn(MSD_Solution)
TriedShifts = {}

REPEAT
  ShiftToBeTried =
    UniformlyChooseAShiftFrom(Shifts - TriedShifts)
  ShiftsMinus1 = Shifts - {ShiftToBeTried}
  FlowInstance =
    MSD2Flow(ShiftsMinus1,
      WorkforceRequirements,
      SplitOffTime)
  CurrentFlow = MCMF(FlowInstance)
  MSD_Solution =
    ShiftsAndWorkforceIn(CurrentFlow)
  CurrentCost = MSD_Eval(MSD_Solution)
  IF CurrentCost < MinCostSoFar THEN
    MinCostSoFar = CurrentCost
    BestFlowSoFar = CurrentFlow
    Shifts = ShiftsInUseIn(MSD_Solution)
  ENDIF
  TriedShifts = TriedShifts ∪ {ShiftToBeTried}
UNTIL Shifts - TriedShifts = {}

/* Postprocessing step to recover cyclicity */
MSD_Solution =
  ShiftsAndWorkforceIn(BestFlowSoFar)

REPEAT
  MSD_Solution1 = MSD_Solution
  MSD_Solution =
    BestOfExchangeStaffNeighborhood(MSD_Solution1)
UNTIL
  MSD_Eval(MSD_Solution) ≥
  MSD_Eval(MSD_Solution1)

RETURN MSD_Solution
```

Table 5: Greedy heuristic with min-cost max-flow subroutine.

out increasing the deviation, a final postprocessing step is made to restore cyclicity. It consists of a simple repair step made by a fast hill-climbing runner that uses the ES neighborhood relation (see Section 3.1). 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[1], an efficient implementation of a scaling push-relabel algorithm [Goldberg, 1997], slightly edited to be callable as a library.

4 Computational results

In this section, we first describe the instances used for our experimental analysis, then we illustrate the performance parameters that we want to highlight, and finally we show the results.

4.1 Instances description

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

Set 1 contains the 30 instances that where investigated and described in [Musliu *et al.*,]. 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.*,] for the commercial OPA implementation. These instances were basically generated by constructing feasible solutions with some random elements as they usually appear in real instances, and then taking the resulting staffing numbers as workforce requirements. This implies that a very good solution with zero deviation from workforce requirements is known. Note that our heuristics could find even better solutions for several of the instances, so these constructed solutions may be suboptimal. Nevertheless, we refer in the following to the best solutions we could come up with for these instances as 'best known' solutions for them.

Set 2 contains similar instances 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 towards 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 remaining set is therefore composed of instances where with high likelihood solutions without deviations do not exist:

Set 3 contains instances without 'best known' solutions. They 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,

[1] © 1995 – 2001 IG Systems, Inc., `http://www.avglab.com/andrew/soft.html`

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.

Set 4 contains four groups of 3 instances each, where the first instances in each of the four groups do correspond to different basic real or randomly generated instances. The second instances in each group are almost equivalent to the first, the difference being that the length of their timeslots are halfed. The third instance in each group also is almost equivalent to the first, the difference being that the workforce requirements are doubled. Group 1 corresponds a rather complicated real instance provided for comparison purposes to the randomly generated ones. The first instances in Groups 2, 3, and 4 are equal to instances 5, 20, and 22, respectively, of Set 3, and thus roughly correspond to increasingly difficult instances. The choice of these instances from Set 3 was done randomly from each of the three kind of parameter sets (12, 16, and 20 valid and invalid shifts) of Set 3. **HIDE? [***We believe that other instances of Set 3 will yield similar results to the ones we report below and that these results are thus more or less representative for the different paramter sets.***]**

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.*,].

4.2 Experimental setting

We made two types of experiments, aiming at two different performance parameters:

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

Our experiments have been run on different machines. The running times have been normalized according to the DIMACS netflow benchmark[2] to the times of a PC equipped with a 1.5GHz AMD Athlon processor with 384 MB ram running Linux Red Hat 7.1 and gcc version 2.96 (calibration timings on that machine for above benchmark: t1.wm: user 0.030 sec t2.wm: user 0.360 sec). Because of the normalization from another machine running MS Windows NT and using MS Visual Basic, the reported running times should be taken as indicatory only.

We experiment with the following three heuristic solvers:

H1 The local search procedure repeated several times starting from different (random) initial solutions. The procedure is stopped when the time granted in elapsed or the best solution is reached.

H2 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).

H3 The two solvers are combined using the solutions delivered by H2 as initial states for H1 trials. In order to maintain diversification, we exploit the nondeterminism of H2 to generate many different solutions. The initial state of each trial of H1 is randomly selected among those states. IS THIS TRUE AT THE MOMENT?

4.3 Computational results

Median time necessary to reach the best known solution

Table 6 shows the median times (in seconds) needed by our heuristics to reach the best known solution out of 10 trials for data Set 1. The first two columns show the instance number and its best known cost, the third column shows the cost of the best solution found in [Musliu *et al.*,]. The dash symbol denotes that the best known solution could not be found.

First notice that our solvers produce results much better than the solver of OPA. In fact, H1 always finds the best solution, H2 in 21 cases, and H3 in 29 cases, whereas OPA finds the best solution only in 17 instances. The table also shows that H1, although it finds the best solution, is always much slower than H2, and generally slower than H3 as well.

To show how heuristics scale up, we show the performances for our solvers within 10 seconds time for Sets 1 and 2, grouped based on their size. The X axis of Figure 1 reports the number of shifts in the best known solution, and the results on instances of equal number are averaged. The Y axis shows the difference of the average cost to the best cost divided by the best cost. This figure shows that for short runs H1 is clearly inferior to H2 and H3, which are comparable.

The above experiments show that H1 is superior in reaching the best known solution, but it requires more time that H2.

Results on Set 3 and further tests on real life examples confirm these trends**HIDE? [** *and are omitted for brevity***]**.

Median value obtained within a time bound

[INSERT GRAPHICS FROM XLS FILE (GNUPLOT?). Examples=Instances, Solv=H.] [] *ws*

Tables **??-??** show the results of the median values from 10 independent runs obtained by each heuristic **(including the commercial OPA software)]** on the in- [] *ws*

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

Inst.	Opt	[Musliu *et al.*,]	H1	H2	H3
1	480	480	3.67	0.07	1.05
2	300	390	16.78	—	31.47
3	600	600	7.39	0.12	1.63
4	450	1170	124.00	—	86.89
5	480	480	4.59	0.15	1.06
6	420	420	2.54	0.06	0.62
7	270	570	9.15	1.30	6.24
8	150	180	42.50	—	13.26
9	150	225	12.15	4.09	8.08
10	330	450	98.00	4.70	131.85
11	30	30	1.64	0.21	0.85
12	90	90	6.18	0.26	3.85
13	105	105	6.56	0.30	3.79
14	195	390	470.94	—	98.75
15	180	180	0.86	0.04	0.40
16	225	375	174.00	—	340.23
17	540	1110	544.00	—	218.25
18	720	720	6.79	1.86	6.44
19	180	195	31.22	—	39.11
20	540	540	12.14	0.11	1.70
21	120	120	6.23	0.29	2.17
22	75	75	3.56	0.38	3.46
23	150	540	16.41	3.45	9.05
24	480	480	2.74	0.11	1.22
25	480	690	770.89	—	—
26	600	600	7.63	1.52	6.47
27	480	480	4.17	0.07	2.33
28	270	270	5.36	1.41	3.60
29	360	390	35.50	—	9.19
30	75	75	2.67	0.27	1.95

Table 6: Times to best for Set 1

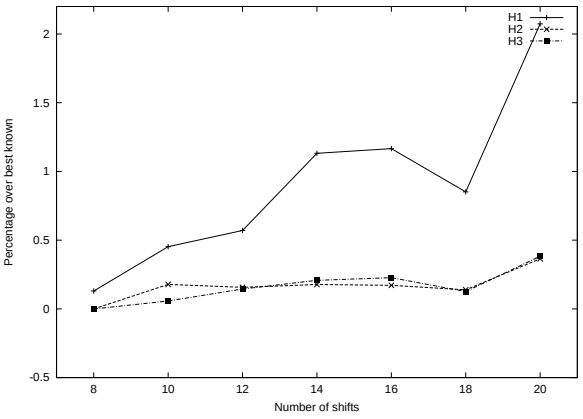

Figure 1: Results for 10 seconds bound

stances of Set 4 within a time bound of 100 seconds.

[For the moment, I do not discuss instances 1-3 as there seem to be problems with the data for H2, and the results in the graph for H3 for instances 1-3 are for sure incorrect due to a calculation error on our side that will be corrected soon.] [] *ws*

[I also do not discuss H3 for the other graphs, see my comments in the accompanying mail as of 2003/1/6.] [] *ws*

One thing to notice is that H2 is clearly superior to H1. **[I hope that we will also be able to report that The solver H3 has the good qualities of both, and therefore it can be considered the best general-purpose solver.]** [] *ws*

[There are a number of more detailed comments (halfed intervals vs double workforce vs original instances; scaling for 12, 16, and 20 valid and invalid shifts, plus examples 1-3; random instances vs real instances) that can be made, but I want to wait for the data of the new H3 and the corrected values for instances 1-3]. [] *ws*

[Add some comment on time to best known versus time limited with optimum not known w.r.t. Johnson's remarks on this topic.] [] *ws*

[Add Luca's experimental results from 2003/1/6:

I've just finished performing the analysis of the data and I've found out some surprising behavior. First of all I fully answer to your initial question: "I could find an adapted better solution with the IntervalDivided instance only in the third random example in the data set by employing Solver 3". This result could be found in 7 cases out of 30 runs.

However, by employing a Mann-Withney test, we cannot reject the hypothesis that the two samples of results are equal, i.e., the algorithm run on the IntervalDivided insance is in the same slot of results as the algorithm run on the original one. For this reason I've started some additional experiments to look more in detail if the optimal solution can be reached also on the original instance.

This behavior is typical of Solver 3. In fact, also in the RealLife example, Solver 3 on the IntervalDivided solution can found the best-known solution in 4 out of 30 cases, while on the plain instance this solution cannot be found at all.

My feeling is that Solver 3 on the IntervalDivided instances has more freedom to move (i.e., it can make smaller steps toward a good solution), but I do not have any evidence of this feeling at present.

Finally, both the solvers run on the DoubleDuties instances perform poorly, and in most cases the best solutions found cannot be adapted to the original instance.] [] *Luca*

[Maybe we could try to prove that no better solution than in the original instance can exist in the instances with intervals halfed and the ones with doubled workforce requirements.] [] *

[Add OPA timings (with and without GUI) with values in the figures. Add some sentences (above?) on these results compared to the Heuristics.] [] *Nysret*

[Note any other anomalies that need explanation, try to explain them.] [] *

5 Conclusions

The *MSD* is an important scheduling problem that needs to be solved in many industrial contexts. We provided complexity results for it and designed a set of heuristics based both on these theoretical results and on local search procedures.

The heuristics have being compared both in terms of ability to reach good solutions and in quality reached in fast runs. For this problem, speed is of crucial importance to allow for immediate discussion in working groups and refinement of requirements. Without quick answers, understanding of requirements and consensus building would be much more difficult.

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 extremly 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 straigthforward 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.

References

[Aarts and Lenstra, 1997] Emile Aarts and Jan Karl Lenstra, editors. *Local Search in Combinatorial Optimization*. Wiley, 1997.

[Arata *et al.*, 2000] K. Arata, S. Iwata, K. Makino, and S. Fujishige. Source location: Locating sources to meet flow demands in undirected networks. In *SWAT*, 2000.

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

[Bartholdi *et al.*, 1980] J.J. Bartholdi, J.B. Orlin, and H.D. Ratliff. Cyclic scheduling via integer programs with circular ones. *Operations Research*, 28:110–118, 1980.

[Bellare, 1993] M. Bellare. Interactive proofs and approximation: reduction from two provers in one round. In *The second Israeli Symposium on the Theory of Computing*, pages 266–274, 1993.

[Carr *et al.*, 2000] R.D. Carr, L.K. Fleischer, V.J. Leung, and C.A. Phillips. Strengthening integrality gaps for capacitated network design and covering problems. In *Proc. of the 11th ACM/SIAM Symposium on Discrete Algorithms*, 2000.

[Crescenzi and Kann, 1994] P. Crescenzi and V. Kann. A compendium of NP optimization problems. `http://www.nada.kth.se/~viggo/problemlist/compendium.html`, 1994.

[Equi *et al.*, 1997] L. Equi, G. Gallo, S. Marziale, and A. Weintraub. A combined transportation and scheduling problem. *European Journal of Operational Research*, 97(1):94–104, 1997.

[Even *et al.*, 2002] Guy Even, Guy Kortsarz, and Wolfgang Slany. On network design problems: Fixed cost flows and the covering steiner problem. In *8th Scandinavian Workshop on Algorithm Theory (SWAT)*, LNCS 2368, pages 318–329, 2002.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman and Co., 1979.

[Garg *et al.*, 1996] N. Garg, M. Yannakakis, and V.V. Vazirani. Approximating max-flow min-(multi)cut theorems and their applications. *Siam J. on Computing*, 25:235–251, 1996.

[Garg *et al.*, 1997] N. Garg, M. Yannakakis, and V.V. Vazirani. Primal-dual approximation algorithms for integral flow and multicuts in trees. *Algorithmica*, 18:3–20, 1997.

[Gärtner *et al.*, 2001] Johannes Gärtner, Nysret Musliu, and Wolfgang Slany. 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, 2001.

[Glover and Laguna, 1997] Fred Glover and Manuel Laguna. *Tabu search*. Kluwer Academic Publishers, 1997.

[Goethe-Lundgren and Larsson, 1994] M. Goethe-Lundgren and T. Larsson. A set covering reformulation of the pure fixed charge transportation problem. *Discrete Appl. Math.*, 48(3):245–259, 1994.

[Goldberg, 1997] Andrew V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *J. Algorithms*, 22:1–29, 1997.

[Guruswami *et al.*, 1999] V. Guruswami, S. Khanna, R. Rajaraman, B. Shepherd, and M. Yannakakis. Near-optimal hardness results and approximation algorithms for edge-disjoint paths and related problems. In *STOC*, 1999.

[Hochbaum and Segev, 1989] D.S. Hochbaum and A. Segev. Analysis of a flow problem with fixed charges. *Networks*, 19(3):291–312, 1989.

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

[Holmberg and Hellstrand, 1998] K. Holmberg and J. Hellstrand. Solving the uncapacitated network design problem by a lagrangean heuristic and branch-and-bound. *Operations Research*, 46:247–259, 1998.

[Johnson, 2002] David S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In *Proc. 5th and 6th DIMACS Implementation Challenges*. AMS, 2002. To appear.

[Khang and Fujiwara, 1991] D.B. Khang and O. Fujiwara. Approximate solutions of capacitated fixed-charge minimum cost network flow problems. *Networks*, 21(6):689–704, 1991.

[Kim and Pardalos, 1999] D. Kim and P.M. Pardalos. A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. *Oper. Res. Lett.*, 24(4):195–203, 1999.

[Krumke *et al.*, 1998] S.O. Krumke, H. Noltemeier, S. Schwarz, H.-C. Wirth, and R. Ravi. Flow improvement and network flows with fixed costs. In *OR-98*, Zürich, 1998.

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

[Lau, 1996] H.C. Lau. Combinatorial approaches for hard problems in manpower scheduling. *J. Oper. Res. Soc. Japan*, 39(1):88–98, 1996.

[Leiserson and Saxe, 1991] C.E. Leiserson and J.B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6(1):5–35, 1991.

[Magnanti and Wong, 1984] T.L. Magnanti and R.T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18:1–55, 1984.

[Musliu *et al.*,] Nysret Musliu, Andrea Schaerf, and Wolfgang Slany. Local search for shift design. European Journal of Operational Research (to appear). `http://www.dbai.tuwien.ac.at/proj/Rota/DBAI-TR-2001-45.ps`.

[Musliu *et al.*, 2002] Nysret Musliu, Johannes Gärtner, and Wolfgang Slany. Efficient generation of rotating workforce schedules. *Discrete Applied Mathematics*, 118(1-2):85–98, 2002.

[Papadimitriou and Steiglitz, 1982] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

[Srinivasan, 1997] A. Srinivasan. Improved approximation for edge-disjoint paths, unsplittable flow, and related routing problems. In *FOCS*, pages 416–425, 1997.

[Thompson, 1996] G.M. Thompson. A simulated-annealing heuristic for shift scheduling using non-continuously available employees. *Comput. Oper. Res.*, 23(3):275–288, 1996.

[Veinott and Wagner, 1962] A.F. Veinott and H.M. Wagner. Optimal capacity scheduling: Parts i and ii. *Operation Research*, 10:518–547, 1962.

A The relation between MSD and $UDIF$

We state in the following the proof of Claim 1, followed by an explanation of how shortage and excess can be handled by a small linear adaptation of the network flow problem.

Proof. We are following here a path similar to the one in \cite{H-2000} in order to get this equivalence. See also, e.g., \cite{AMO-93}.

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:

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

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

As $\mathcal{T}$ is regular the two sets of feasible vectors for $Ax = b$ and for $\mathcal{T} \cdot Ax = \mathcal{T}b$ are equal. The matrix $\mathcal{F} = \mathcal{T}A$ is a matrix with only (at most) two nonzero entries in each column: one being a 1 and the other being a -1. In fact, all columns i in A create a column in $\mathcal{F} = \mathcal{T}A$ with exactly one -1 entry and exactly one 1 entry except for columns i with 1 in the first row (namely, so that $b(i) = 1$). These columns leave one 1 entry in row $e(i)$, namely, in the row column i ends. Call these columns the *special* columns.

The matrix $\mathcal{F}$ can be interpreted as a flow matrix (see for example \cite{BKP-98}). Column j of the matrix is represented by an edge e_j. We assign a vertex v_i to each row i. Add an extra vertex v_0.

An edge e_j with $\mathcal{F}_{ij} = 1$ and $\mathcal{F}_{kj} = -1$ goes out of v_k into v_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.

In addition, for all special rows i ending at $e(i)$, we add an edge from v_0 into $v_{e(i)}$. Add an edge of capacity b_1 from s to v_0. Let $\bar{b} = \mathcal{T}b$. The $\bar{b}$ vector determines the way all vertices (except v_0) are joined to the sink t and source s. If $\bar{b}_i > 0$ then there is an edge from v_i to t with capacity $\bar{b}_i$. Otherwise, if $\bar{b}_i < 0$, there is an edge from s to v_i with capacity $-\bar{b}_i$. Vertices with $\bar{b}_i = 0$ are not joined to the source or sink. All edges not touching the source or sink have infinite capacity.

Note that the addition of the edge from s into v_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. A saturating flow is a flow saturating all the edges entering the sink. It is easy to see that if there exists a saturating flow, 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). Hence, the problem is transformed into the following question: Given G, 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. The network G is a DAG (directed acyclic graph). This clearly holds true as all edges go from v_i to v_j with $j > i$. In addition, all capacities on edges not touching the sink or source are infinite (see the above construction).

On the other hand, given a $UDIF$ instance with a saturating flow (namely, where one can find a flow function saturating all the edges entering the sink) 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 v_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 q, p with $q < p$, necessarily, $\mathcal{F}_{qj} = -1$ and $\mathcal{F}_{pj} = 1$ (if there is a column j that does not contain an $\mathcal{F}_{qj} = -1$, set $q = 0$). Hence, add to A the c1 column with 1 from rows $q + 1$ to p.

We note that the restriction of the existance of a flow saturating the flow along edges entering 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 $UDIF$. □

To understand how this can be used to also find solutions to MSD instances where 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 $A_i x - y_i = b_i$, namely, 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 0, 1 and -1 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. Its corresponding graph is the graph of $\mathcal{F}$ with the addition of an infinite capacity edge from v_i into v_{i-1} $(i = 1, \ldots, n)$. Call these edges the y edges. 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 may assume w.l.o.g. 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.

B A hardness result for $UDIF$ and MSD

We next prove Theorem 2.1:

Proof. We prove a hardness reduction for UDIF under the assumption $P \neq NP$. We use a reduction from Set-Cover. We need a somewhat different proof than \cite{KNSWR-98} 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 $\mathcal{B}(V_1, V_2, A)$ with edges only crossing between V_1 and V_2. We may assume that $|V_1| = |V_2| = n$. We look for a minimum sized set $S \subseteq V_1$ so that $N(S) = V_2$ (namely, every vertex in V_2 has a neighbor in S). If $N(S) = V_2$ we say that S covers V_2. We may assume that the given instance has a solution. The following is proven in \cite{RS-97}.

Theorem B.1 *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 $\mathcal{B}(V_1, V_2, \mathcal{E})$ be the instance of the set cover problem at hand so that $|V_1| = |V_2| = n$. Add a source s and a sink t. Connect s to all the vertices of V_2 with capacity one edges. Direct all the edges of $\mathcal{B}$ from V_2 to V_1. Now, create n^2 copies V_1^i of V_1 and for convenience denote $V_1 = V_1^0$. For each $i \in \{0, \ldots, n^2 - 1\}$, connect in a directed edge the copy $v_1^i \in V_1^i$ of each $v_1 \in V_1$ to the copy $v_1^{i+1} \in V_1^{i+1}$ of v_1 in V_1^{i+1}. Hence, a perfect matching is formed between contiguous V_1^i via the copies of the $v_1 \in V_1$ vertices. The vertices of $V_1^{n^2}$ are all connected to t via edges of capacity n. Note that by definition, all other edges (which are edges touching neither the source nor 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 S be the set of vertices $S \subseteq V_1$ so that for every vertex $v_2 \in V_2$ there exists a vertex $s \in S$ such that edge (v_2, s) carries positive flow.

Note that for every $v_2 \in V_2$ there must be such an edge for otherwise the flow is not optimal. Further note that the flow units entering S must be carried throughout the copies of S in all of the V_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 $n^2 \cdot |S| + n$. The n term comes from the n edges touching the vertices of V_2.

Further, note that S must be a set cover of V_2 in the original graph $\mathcal{B}$. Indeed, every vertex v_2 must have a neighbor in S. Finally, note that it is indeed possible to get a solution with $n^2 \cdot s^* + n$ edges where s^* is the size of the minimum set cover using an optimum set cover S^* 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 S is chosen. The following properties end the proof: The number of vertices n' in the new graph is $O(n^3)$. In addition, the additive term n is negligible for large enough n in comparison to $n^2 \cdot |S|$ where S is the chosen set cover. Hence, the result follows for $c < 1/3 < 1$. □

C 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 \cite{Goldberg97} 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.

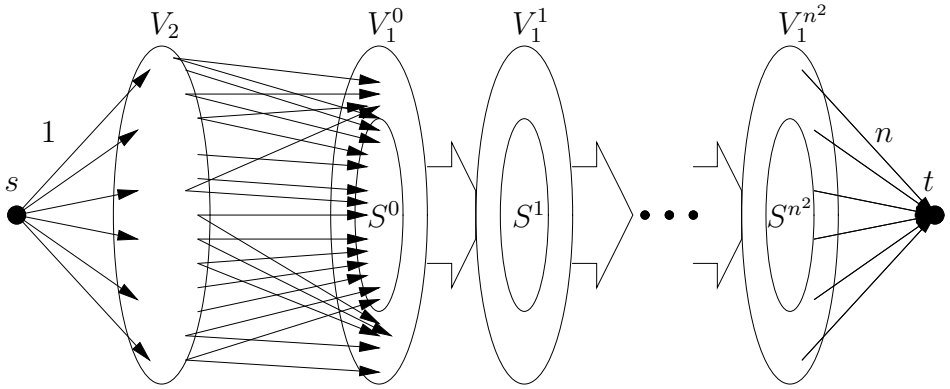

Figure 2: Schematic illustration of the reduction from the Set-Cover problem to the UDIF problem.

Another simple heuristic that very naturally (in the double sense of the word) suggests itself might be to combine the MCMF() subroutine (together with the postprocessing step described in Section 3.2 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 \cite{Castro96} to our instances instead of calling $MCMF$ but got only unsatisfactory results as 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.

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

Contents