A hybrid network flow tabu search heuristic for the minimum shift design problem

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

*Department of Mathematics and Computer Science, University of Udine
I-33100 Udine, Italy
digasper@dimi.uniud.it

†Ximes Inc
Schwedenplatz 2/26, A - 1010 Wien, Austria
gaertner@ximes.com

‡Computer Science Department, Rutgers University
Camden, NJ 08102, USA
guyk@crab.rutgers.edu

§Institute of Information Systems, Vienna University of Technology
A-1040 Wien, Austria
musliu@dbai.tuwien.ac.at

⋆DIEGM, University of Udine
I-33100 Udine, Italy
schaerf@uniud.it

¶Institute for Software Technology, Graz University of Technology
A-8010 Graz, Austria
wsi@ist.tugraz.at

The minimum shift design problem (*MSD*) is a scheduling problem that commonly arises in workforce management activities. Given a collection of shifts and workforce requirements, the problem consists in finding a minimum cardinality set of work shifts, and the number of workers to assign to each shift, in order to meet (or minimize the deviation from) prespecified staff requirements. This problem reduces to a special case of the minimum edge-cost flow problem (*MECF*) for which a logarithmic hardness of approximation lower bound can be shown. We propose a two-stage heuristic for *MSD*. The first stage, inspired by the relation between the *MSD* and *MECF* problems, is based on a greedy heuristic that relies on a min-cost max-flow (*MCMF*) subroutine. The second stage is a composite tabu search procedure based on the interleaving of different neighborhood definitions. We compare our heuristic with

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

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 1: Sample workforce requirements (left side), typical set of shift types (right side).

previous results on publicly available instances. The outcome of the comparison shows that our heuristic significantly outperforms the previous approach.

1 Introduction

The requirements in an *MSD* problem 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. Each day j is split into n equal-size smaller intervals $[t_i, t_{i+1})$, called *timeslots*, which can last from a few minutes up to several hours. The staffing requirement for the ith timeslot (identified by its starting time t_i, $i = 0, \ldots, n-1$), on day $j \in \{0, \ldots, h-1\}$ is fixed. 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.

An example of workforce requirements with $h = 7$ is shown on the left side of Table 1, the first day being labeled 'Mon' etc. In the table, for conciseness, timeslots with same requirements are grouped together (the example is adapted from a real call-center). When designing shifts, not all starting times are feasible, nor are all lengths allowed. For this reason, the input also includes a collection of *shift types*, each of them characterized by a minimum and maximum starting time, a mimimum and maximum length, and time granularity equal to the length of the timeslots. The right side of Table 1 shows an example of a set of shift types. Each shift $I_{s,l}$ with starting time t_s ($s \in \{0, \ldots, n-1\}$) and length l, belongs to a type, i.e., its length and starting times must necessarily lie inside the intervals defined by one type. In the following, the type of shift I is denoted by $T(I)$.

The goal of the *MSD* problem is to decide how many persons $x_j(I_{s,l})$ are going to work in each shift $I_{s,l}$ for each day j, so that $b_{i,j}$ people will be present at time t_i for all i and j. Let $\mathcal{I}_{t_i}$ be the collection of shifts that include the timeslot t_i. A feasible solution consists of h numbers $x_j(I)$ assigned to each shift $I = I_{s,l}$ so that $p_{i,j} := \sum_{I \in \mathcal{I}_{t_i}} x_j(I) = b_{i,j}$. That is, 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. However, in practical cases this constraint is relaxed, so that small deviations are allowed.

To this aim, each solution of the *MSD* problem is evaluated by means of an *objective function* to be minimized. The objective function is a weighted sum of three components,

in which the weights depend on the instance at hand. The first and second components are, naturally, the staffing excess and shortage, namely, the sums $\sum_{i,j}(\max(0, p_{i,j} - b_{i,j}))$ and $\sum_{i,j}(\max(0, b_{i,j} - p_{i,j}))$. The third component of the objective function is the number of shifts selected. In fact, it is important to have only few shifts as they lead to schedules easier to read, check, manage, and administer.

There is a large body of literature on shift scheduling problems (see [2] for a recent survey). Heuristics for the selection of shifts that bear some similarity to *MSD* have been studied by [4]. However, the only paper that, to our knowledge, deals exactly with *MSD* is [3]. In Section 2, the problem is defined and the equivalence between *MSD* and a restricted version of *MECF* is shown. In Section 3, a heuristic method for *MSD* is introduced. In Section 4, we compare our heuristic with the results presented in the cited paper by applying it to a set of benchmark instances.

2 Theory

A restricted version of *MSD* is equivalent to the *UDIF* problem (the infinite capacities flow problem on a direct acyclic graph, or DAG), which, in turn, is a restricted variant of *MECF*. The *UDIF* problem is defined as follows: (1) Every edge not touching the sink or the source, called *proper* edge, has infinite capacity. Non-proper edges, namely edges touching the source or the sink, have arbitrary capacities. (2) The costs of proper edges is 1. The cost of edges touching the source or the sink is zero. (3) The underlying flow network is a 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, minimize the *number* of proper edges carrying nonzero flow (namely, minimizing $|\{e : f(e) > 0,\ e \text{ is proper}\}|$).

Thanks to the equivalence of *MSD* and *UDIF*, a hardness of approximation result for *UDIF* carries over to *MSD*: there is a constant $c < 1$ so that approximating the *UDIF* problem within $c \ln n$ is NP-hard (all proofs are omitted for brevity).

3 A heuristic for *MSD*

Our heuristic is divided into two stages, namely a greedy construction for the initial solution and a tabu search procedure [1] that iteratively improves it, which are described in Section 3.1 and 3.2, respectively. We found empirically that the combination of the two heuristics for *MSD* was superior to the two individual heuristics taken by themselves.

3.1 GreedyMCMF

Based on the equivalence of the (non-cyclic) *MSD* problem to *UDIF* for which no efficient algorithm is known (see Section 2), and the relationship with the *MCMF* problem, we propose a new greedy heuristic GreedyMCMF() that uses a polynomial min-cost max-flow subroutine (MCMF()). The algorithm 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, however, the algorithm is not able to minimize the number of shifts that are used.

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 (in practice, there is usually a complete exchange of workforce between 5:00 and 8:00 on the mornings of the first day). 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 (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 shown to provide adequate results in practice.

In the main loop, 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 it will not be considered for removal again.

Once no shifts can be removed anymore without increasing the deviation, a postprocessing step restores cyclicity. It consists of a simple repair step performed by a fast hill-climbing runner that uses the ExchangeStaff (ES) (one employee in a given day is moved from one shift to another one of the same type) neighborhood relation (see Section 3.2). 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.

3.2 The Tabu Search Procedure

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. The active shifts of the initial solution are those built by the greedy algorithm, and the solution is completed with a set of random distinct inactive shifts. The number of inactive shifts has been selected experimentally, and it has been set to 2 inactive shifts per shift type.

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: ChangeStaff (CS): The staff of a shift is increased or decreased by one employee. ExchangeStaff (ES): This is the neighborhood relation mentioned above where one employee in a given day is moved from one shift to another one of the same type. 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.

We combine the different neighborhoods in a compound tabu search algorithm, that uses them selectively in different phases of the search. We have chosen tabu search as the driving meta-heuristic of our local search algorithm, since it has shown to give the better results in a preliminary experimentation phase.

In detail, our procedure interleaves three different tabu search *runners* using the ES alone, the RS alone, and the union of the two neighborhoods CS and RS, respectively. The runners are invoked circularly 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 interleaving pattern (i.e. CS → RS → ES+RS) has been obtained experimentally, by evaluating various different combinations on many instances.

The reason for using limited neighborhood relations is not directly 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 *diversification* in the search. In fact, some 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 worsens it by a smaller amount, although the CS move could lead to a more promising region of the search space.

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.

4 Computational results

For our analysis we consider the 30 instances that where described and investigated in [3] available at `www.dbai.tuwien.ac.at/proj/Rota/benchmarks.html`. These instances were generated by [3] by constructing feasible solutions and then taking the resulting staffing numbers as workforce requirements. This implies that a solution with zero deviation from workforce requirements is known which is referred to as the 'best known' solution.

Although our solver reaches good solutions, for some instances it shows running times that might be unacceptable in interactive sessions. For this reason, we analyzed how it performs within a given time bound of 10 seconds.

In addition, in order to study the behaviour of our solver for increasing sizes of the solution, we grouped the results on the instances based on the number of shifts in the 'best known' solution. In order to have a relevant number of instances for each size, we added 30 more instances constructed to feature a number of shifts that ranges from 8 to 20.

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 (min and max) cost to the best cost divided by the best cost.

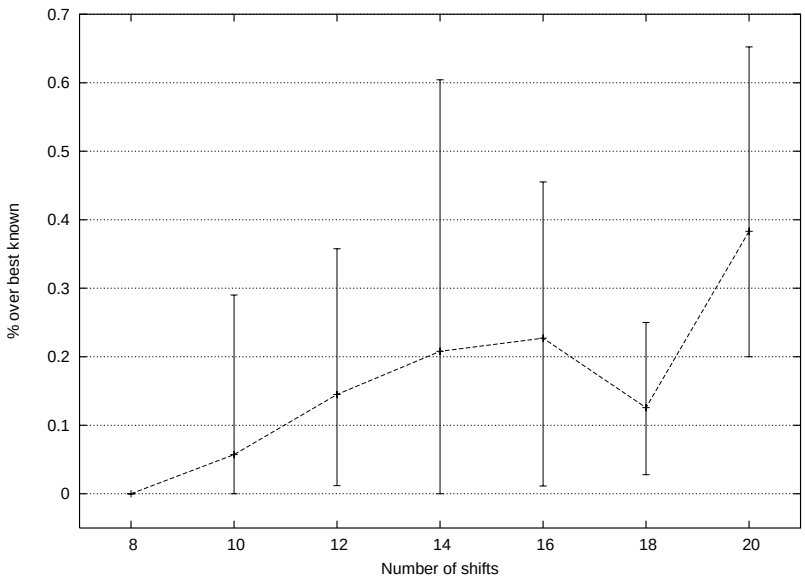

Figure 1: Results for 10 seconds bound

The above experiments show that the relative degradation of performance is very small (about 0.2%) and only moderately increases for larger instances.

Finally, in order to understand the relative importance of the two stages of the algorithm, we performed other tests (omitted for brevity) using them in isolation. That is, we evaluated the solution of GreedyMCMF and the one of tabu search starting from a random initial solution. Not surprisingly, GreedyMCMF is generally fast but it does not always find the best solution; on the other hand tabu search finds all the best solutions, but is it slower. The combined solver presented in this paper outperforms them both and thus can be thus considered, among them, the best general-purpose solver.

Acknowledgments: This work was supported by Austrian Science Fund Project No. **Z29-N04**.

References

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

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

[3] 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`.

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