MISTA 2019

The General Task-based Shift Generation Problem: Formulation and Benchmarks

Nico Kyngäs · Kimmo Nurmi · Dries Goossens

Abstract In workforce scheduling, shift generation is the process of determining the shift structure, along with the tasks to be carried out in particular shifts and the competences required for different shifts. Application areas of staff rostering and shift generation include hospitals, retail stores, call centers, cleaning, home care, guarding, manufacturing and delivery of goods. The General Task-based Shift Generation Problem (GTSGP) is to create anonymous shifts and assign tasks to these shifts so that employees can be assigned to the shifts. The targeted tasks must be completed within given time windows. Tasks may have shift-local precedence constraints and transition times between tasks are considered. The goal is to maximize the total number of shifts the employees are able to execute. This paper presents a mathematical formulation and some benchmark instances for the GTSGP. We also develop some preliminary metrics to assess the difficulty of GTSGP instances. We briefly describe the PEAST algorithm, which is used to solve the benchmark instances, and report on the computational results.

1 Introduction to workforce scheduling and shift generation

Workforce scheduling is an extremely challenging and time-consuming process that every organization that has employees working on shifts or on irregular working days must manage. These organizations and their domains include hospitals, retail stores, call centers, cleaning, home care, guarding, manufacturing and delivery of goods. Good overviews of workforce scheduling are published by Ernst et al. [27], Musliu et al. [28],

Nico Kyngäs
Satakunta University of Applied Sciences
E-mail: nico.kyngas@samk.fi

Kimmo Nurmi
Satakunta University of Applied Sciences
E-mail: cimmo.nurmi@samk.fi

Dries Goossens
Department of Business Informatics and Operations Management, Ghent University
E-mail: dries.goossens@ugent.be

Di Gaspero et al. [29] and Vanden Berghe et al. [30]. A very interesting framework for the general employee scheduling problem was proposed by Kletzander and Musliu [31].

The workforce scheduling process has five main components (see Figure 1). First, *workload prediction* is the phase of determining the workload. The workload can be based on both known and predicted events. For example, the arrivals of customers can be predicted using models based on queueing theory, simulation and statistics, while known events may be gathered from current sales contracts. The events are transformed into either tasks or staff demand, depending mostly on the duration and spatial diversity of the events.

The process continues with *shift generation* that produces shifts from the given workload. The most important target is to match the shifts to the workload as accurately as possible. The generation of shifts is based on either the number of employees required to work at given timeslots or the tasks that the shifts must cover.

The generated shifts form an input for the *staff rostering* phase, where employees are assigned to the shifts. The most important issues to be considered are employees' competences and preferences, along with working and resting times laid down by either collective labor agreements or labor contracts. Staff rostering also includes the scheduling of days-off and vacations. The staffing requirements must be carefully considered in the *resource and holiday planning* phase. Holidays, training sessions and other short-term absences as well as long-term sick-leaves and forthcoming retirements have a major impact on staff rostering.

The generated staff rosters often need to be changed. *Daily rescheduling* deals with ad hoc changes that are necessary due to sick leaves and other no-shows. The organization should have an appropriate mechanism to choose suitable substitutes considering the qualifications, employment contracts, legal limitations and salaries.

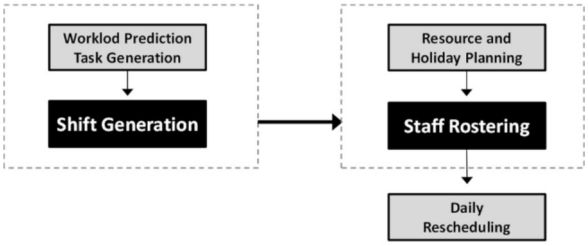

Fig. 1 The main components of the workforce scheduling process.

Academic workforce scheduling studies have mainly focused on staff rostering. In recent years, the generation of shifts has gained increasing interest. Significant benefit in financial efficiency and customer satisfaction can be achieved by optimizing the shift structure. In theory, the best results can be achieved when shift generation and staff rostering are processed and solved at the same time. However, even simplified variations of both problems and their sub-problems are known to be NP-hard and NP-complete

[18]-[22]. Nonetheless, some interesting implementations exist, see e.g. [23]-[26]. Due to the complexity issues in large-scale practical applications, shift generation and staff rostering are mainly solved separately. Our approach is to first generate the shifts and then roster the staff as described in Section 2.

The generation of shifts is based on either the varying number of required employees working during the planning horizon or the tasks that the shifts must cover. We call these employee-based and task-based shift generation problems. The first major contribution for the *employee-based shift generation* problem was the study by Musliu et al. [28]. They introduced a problem, in which the workforce requirements for a certain period of time were given, along with constraints about the possible start times and the length of shifts, and an upper limit for the average number of duties per week per employee. Di Gaspero et al. [32] proposed an employee-based problem in which the most important issue was to minimize the number of different kinds of shifts used. The problem statement also includes a collection of acceptable shift types, each of them characterized by the earliest/latest start time and minimum/maximum length of its shifts.

Bhulai et al. [33] presented a generalized model for multi-skill shift design in call centers. Their method generated a rough match between the predicted workload and labor capacity, taking the stochastic nature of the call arrival process into account. Kyngäs et al. [34] introduced the unlimited shift generation problem in which the most important goal is to minimize understaffing and overstaffing. They define a strict version of the problem, in the sense that each timeslot should be exactly covered by the correct number of employees. In [31]-[33], the shifts are limited to a number of types, for which the lengths and the start times of the shifts have to be within certain ranges. In [34], the lengths and the start times of the shifts are not strictly limited. In the person-based multitask shift generation problem with breaks presented in [35], employees can have their personal shift length constraints and competences. Even if the goal is to construct a set of shifts and not to assign them to employees, they ensure that the employees have the ability to execute the shifts.

Compared to the employee-based shift generation problem, far fewer models and algorithms have been developed for the *task-based shift generation* problem in which a number of different tasks must be carried out. The problem is to create shifts and assign tasks to these shifts so that employees can be assigned to the shifts. The first major contribution of the task-based problem was the study by Dowling et al. [24]. They created a master roster, a collection of working shifts and off shifts, and then allocated a set of tasks to personnel with the requisite skills who are available for work on that day. Krishnamoorthy and Ernst introduced a group of problems called Personnel Task Scheduling Problems (PTSP) in [36]. Given the staff that are rostered on a particular day, the PTSP is to allocate each individual task, with specified start and end time, to available staff who have skills to perform the task. Later, Krishnamoorthy et al. [37] introduced a special case referred to as Shift Minimization Personnel Task Scheduling Problem (SMPTSP) in which the only cost incurred is due to the number of personnel (shifts) that are used. This variant was motivated by situations where a large pool of casual staff is available and management would like to minimize pool usage. A similar model was earlier presented in [38] where the authors minimized the number of workers required to perform a machine load plan.

Lin and Ying [39] developed a three-phase heuristic for the SMPTSP. They obtain an initial solution using a simple but very effective construction heuristic, which is then improved using an iterated greedy heuristic, which in turn is used as an initial upper bound while solving the MIP model of the problem. Lapegue et al. [40] introduced an equity objective to the SMPTSP. The idea is to find a solution where employees have approximately the same amount of work, thus generating more fair schedules.

The next section defines the General Task-based Shift Generation Problem (GTSGP) in which the tasks are not fixed in time. Furthermore, the tasks are not explicitly assigned to employees. The main contributions of this paper are a mathematical formulation, a benchmark instance generator and computational results with PEAST, CPLEX and Gurobi for the GTSGP.

The article is organized as follows. In the next section we define and formulate the GTSGP. Section 3 introduces the first set of benchmark instances for the GTSGP. Section 4 describes the solution methods used to tackle the problem. In Section 5, we present our computational results.

2 Problem definition and formulation

In this section, we define and formulate the General Task-based Shift Generation Problem (GTSGP). To the best of our knowledge, the problem was first introduced in [41]. The motivation for the GTSGP derives from the demand-oriented workforce scheduling process presented in Section 1. As stated earlier, the complexity issues in large-scale practical applications are extremely challenging, which is why shift generation and staff rostering are mainly solved separately. Unchecked separate solving may lead to infeasibility problems in staff rostering, which we attempt to alleviate by generating shifts that can be completed by as many employees as possible, leading to increased flexibility in the rotering phase.

Given the tasks that should be rostered on a particular day, *the GTSGP is to create anonymous shifts and assign tasks to these shifts so that employees can be assigned to the shifts.* The targeted tasks must be completed within a given time window. For example, shelving in retail stores is often carried out in the forenoon. Some tasks are so-called back-office tasks. For example, in a contact center answering emails might require a given number of working hours per day dedicated to the activity but these tasks can be done any time of the day.

Recall, that we roster the staff after the shifts have been generated for each day. We consider the shift structure separately for each day, so there is no connection between the shifts of different days. We should strive to ensure that the resulting set of shifts can be carried out by the employees, i.e. each shift can be assigned to an employee while all shifts are assigned to someone and no employee is assigned to multiple shifts. For each day, the goal is to maximize the number of feasible (shift, employee) pairs. A pair (s, e) is considered feasible if employee e can carry out shift s. It is assumed that the full-time permanent and temporary employees cover 100% of the total workload.

2.1 An example problem

Figure 2 shows a simplified instance of the GTSGP with the following characteristics:

1. The planning period is divided into 19 timeslots.
2. The number of tasks is 18 and the number of employees is 8.
3. The availability of the employees during the planning period is denoted by x.
4. The durations of the tasks are given by the length of the corresponding rectangle.
5. Eleven of the tasks have time windows denoted by black lines.
6. Eight of the tasks have strict time windows, i.e. they must be carried out exactly at their given times.
7. The last task is a back-office task, i.e. its time window is the entire planning horizon.
8. The employees skilled to carry out a task are indicated in the rectangle.

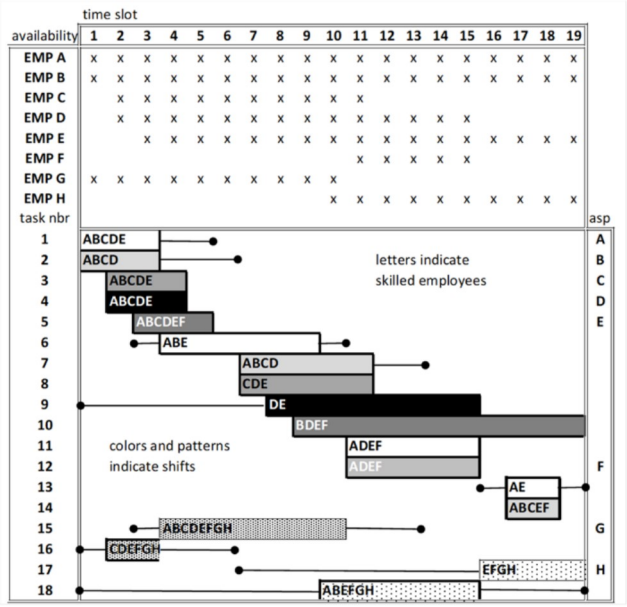

Fig. 2 A simplified instance of the GTSGP and a feasible solution to the instance. The ASP solution column shows the first task of each employee's shift.

The GTSGP differs from the SMPTSP described in Section 1 in several ways:

(G1) Tasks are not explicitly assigned to employees.

(G2) Tasks are not fixed in time.

(G3) Tasks may have shift-local precedence constraints, e.g. in case shifts of certain types are assigned to the same shift, one must be carried out before the other.

(G4) Transition times between tasks, e.g. travelling from the location of one task to the location of the next using some transportation, are considered.

(G5) The number of feasible (shift, employee) pairs is maximized.

```
Shift 1: Task #1(1)    Task #6(4)    Task #11(11)    Task #13(17)
Shift 2: Task #2(1)    Task #7(7)    Task #14(17)
Shift 3: Task #3(2)    Task #8(7)
Shift 4: Task #4(2)    Task #9(8)
Shift 5: Task #5(3)    Task #10(9)
Shift 6: Task #12(11)
Shift 7: Task #16(2)   Task #15(4)
Shift 8: Task #18(10)  Task #17(16)
```

Fig. 3 The feasible solution of the instance shown in Figure 2.

```
          Shift #1 #2 #3 #4 #5 #6 #7 #8
Employee #1  1* 1  0  0  0  1  0  0
Employee #2  0  1* 0  0  0  0  0  0
Employee #3  0  0  1* 0  0  0  0  0
Employee #4  0  0  1  1* 0  1  0  0
Employee #5  0  0  0  0  1* 1  0  0
Employee #6  0  0  0  0  0  1* 0  0
Employee #7  0  0  0  0  0  0  1* 1
Employee #8  0  0  0  0  0  0  0  1*
```

Fig. 4 The feasible pairs of the instance shown in Figure 2.

As an example, consider the instance shown in Figure 2. The colors and patterns of the tasks indicate which tasks are assigned to which shifts. A text version of the feasible solution shown in the figure is given in Figure 3. The starting times of the tasks given in parentheses correspond to the pairing between employees and shifts given in Figure 4, i.e. each shift could be carried out by at least the corresponding employee if the parenthesized task starting times were used. Starting times in the problem are actually employee-dependent and are not considered a part of the shifts.

Figure 4 shows the feasible (shift, employee) pairs of the instance. The number of feasible pairs is 14. For example, employee #1 can carry out the shifts 1, 2 and 6. Likewise, shift #3 can be carried out by employees 3 and 4. Furthermore, each shift can be assigned to an employee while all shifts are assigned to someone and no employee is assigned to multiple shifts. The asterisks denote the one and only such assignment.

2.2 Notation

S	The resulting set of shifts.
TS	The planning horizon, i.e. the contiguous set of timeslots.
T	The set of tasks.
E	The set of employees.
L	The set of locations.
C	The set of skills.
R	The set of transport types.
G	The set of task types or task groups.
M	The travel times of possible transitions. For any given transport type, the travel times must adhere to the triangle inequality, yet multiple modes of transport may be used.
P	The set of shift-local precedence constraints.
c_e	The cost of using employee e.
elb_e	Lower bound (incl.) on total working time of employee $e \in E$.
eub_e	Upper bound (incl.) on total working time of employee $e \in E$.
l_e^s	Starting location of employee $e \in E$.
l_e^e	Ending location of employee $e \in E$.
r_e	Transport type of employee $e \in E$.
a_e^s	Earliest timeslot (incl.) of availability of employee $e \in E$.
l_e^s	Latest timeslot (incl.) of availability of employee $e \in E$.
d_t	Duration of task $t \in T$.
tlb_t	Earliest timeslot (incl.) for starting time of task $t \in T$.
tub_t	Latest timeslot (incl.) for ending time of task $t \in T$.
g_t	Task type of task $t \in T$.
l_t	Location of task $t \in T$.
ttc_r	Task transition working time coefficient of transport type $r \in R$ ($ttc_r \in [0, 1]$). Transition time between two tasks is considered working time with ttc_r as coefficient, i.e. working time = transition time * ttc_r.
etc_r	Employee transition working time coefficient of transport type $r \in R$ ($etc_r \in [0, 1]$). Transition time to/from the first/last task of a shift is considered working time with etc_r as coefficient, i.e. working time = transition time * etc_r.
eta_r	Employee transition availability requirement of transport type $r \in R$ ($eta_r \in \{0, 1\}$), i.e. must the first/last transition of a shift be considered with respect to the assigned employee's availability?
$m_{rll'}$	Travel time from location $l \in L$ to location $l' \in L$ using transport type $r \in R$ ($m_{rll'} \in M$).
$p_{g,g'}$	Shift-local precedence constraint that dictates tasks of task type $g \in G$ cannot be succeeded by tasks of task type $g' \in G$.
T_e	The set of tasks doable by employee $e \in E$ ($T_e \subseteq T$).
E_t	The set of employees capable of carrying out task $t \in T$ ($E_t \subseteq E$).

2.3 Problem description

An employee e can be assigned to the shift s if and only if the following criteria hold:

(C1) He/she possesses all the skills indicated by the task types of the tasks assigned to the shift (*skill*).

(C2) The total working time of the tasks in the shift is within $[elb_e, eub_e]$ (*working time*).

(C3) All the timeslots of the tasks in the shift are within $[a_e^s, l_e^s]$ (*availability*).

(C4) He/she has enough transition time to move between the tasks in the shifts (*transition time*).

Note that these criteria implicitly define the shift structure from the skills, working times, availabilities and transition times of the employees. The GTSGP has four basic assumptions:

(B1) Every task will be assigned to a shift.

(B2) Preemption of tasks is not allowed.

(B3) Each task is processed only once without interruption.

(B4) Each employee can execute only one task at a time.

A solution is feasible, if the following hard constraints have no violations:

(H1) The tasks in the shift do not overlap in time (*overlap*).

(H2) Some tasks may have shift-local precedence constraints, that is, a task may not be executed after some other task if they are assigned to the same shift (*precedence*).

(H3) Each shift can be executed (C1-C4 hold) by one or more employees (*shift*).

(H4) Each shift can be assigned to an employee s.t. all shifts are assigned to someone and no employee is assigned to multiple shifts (*combination*).

Note that the criterion H4 actually implies H3.

The GTSGP can now be stated as follows:

1. Satisfy the hard constraints H1-H4.
2. – Maximize the sum of number of feasible (shift, employee) pairs over all pairs.
   – Minimize employee costs.
   – Minimize travelling time.

2.4 Problem formulation

In this section, we propose an integer programming formulation for the general task-based shift generation problem. In order to ensure that hard constraint (H4) is satisfied without having to solve a separate assignment problem, we opt to have a dedicated (possibly empty) shift $s(e)$ for each employee e. This enforces that, if we can feasibly assign all tasks to shifts, it will be possible to assign a shift to each employee such that all shifts are assigned to someone in the staff rostering phase (although employee e need not necessarily receive shift $s(e)$ at that point).

The following decision variables are used in the model:

$$x^s_{et} = \begin{cases} 1 & \text{if for shift } s(e) \text{ the first task is } t, \\ 0 & \text{otherwise.} \end{cases}$$

$$x_{ett'} = \begin{cases} 1 & \text{if task } t \text{ is immediately followed by task } t' \text{ in shift } s(e) \text{ with } t \neq t' \\ 0 & \text{otherwise.} \end{cases}$$

$$x^f_{et} = \begin{cases} 1 & \text{if for shift } s(e) \text{ the last task is } t, \\ 0 & \text{otherwise.} \end{cases}$$

$$b_{et} = \begin{cases} 1 & \text{if task } t \text{ belongs to shift } s(e) \\ 0 & \text{otherwise.} \end{cases}$$

$$sx_e = \begin{cases} 1 & \text{if shift } s(e) \text{ is non-empty} \\ 0 & \text{otherwise.} \end{cases}$$

y_{et} = starting time (i.e. first slot) of task $t \in T$ assuming the shift it belongs to is assigned to employee $e \in E$.

$wt_{ee'}$ = working time for employee $e \in E$ assuming they carry out the shift of employee $e' \in E$.

$$ve_{e'e} = \begin{cases} 1 & \text{if employee } e \text{ may not carry out shift } s(e') \\ 0 & \text{otherwise.} \end{cases}$$

$$vs_{e'e} = \begin{cases} \geq 1 & \text{if employee } e \text{ does not have the skills for shift } s(e') \\ 0 & \text{otherwise.} \end{cases}$$

$$vwu_{e'e} = \begin{cases} \geq 1 & \text{if shift } s(e') \text{ has too much working time for employee } e \\ 0 & \text{otherwise.} \end{cases}$$

$$vwl_{e'e} = \begin{cases} \geq 1 & \text{if shift } s(e') \text{ has too little working time for employee } e \\ 0 & \text{otherwise.} \end{cases}$$

$$vtu_{e'et} = \begin{cases} \geq 1 & \text{if employee } e \text{ would be late at task } t \text{ if carrying out shift } s(e') \\ 0 & \text{otherwise.} \end{cases}$$

$$vteu_{e'et} = \begin{cases} \geq 1 & \text{if employee } e \text{ would be late at their end location due to} \\ & \text{the last task } t \text{ of shift } s(e') \text{ if carrying out shift } s(e') \\ 0 & \text{otherwise.} \end{cases}$$

Problem-GTSGP

$$Z_{GTSGP} = min\Bigg(\alpha * \sum_e sx_e c_e \tag{1}$$

$$- \beta * \sum_{e,e'} (1 - ve_{e'e}) \tag{2}$$

$$+ \gamma * \left(\sum_e wt_{ee} - \sum_t d_t \right) \Bigg) \tag{3}$$

$$\text{s.t.} \quad \sum_t x^s_{et} \leq 1 \ \ \forall e \in E \tag{4}$$

$$\sum_t x^f_{et} \leq 1 \ \ \forall e \in E \tag{5}$$

$$\sum_e b_{et} = 1 \ \ \forall t \in T \tag{6}$$

$$b_{et} = x^s_{et} + \sum_{t'} x_{et't} \ \ \forall e \in E, t \in T \tag{7}$$

$$b_{et} = \sum_{t'} x_{ett'} + x^f_{et} \ \ \forall e \in E, t \in T \tag{8}$$

$$sx_e \leq \sum_t b_{et} \ \ \forall e \in E \tag{9}$$

$$sx_e \geq b_{et} \ \ \forall e \in E, t \in T \tag{10}$$

$$y_{et} \geq tlb_t \ \ \forall e \in E, t \in T \tag{11}$$

$$y_{et} \geq a^s_e + eta_{r_e} m_{r_e l^s_e l_t} \ \ \forall e \in E, t \in T \tag{12}$$

$$y_{et'} + d_{t'} + m_{r_e l_{t'} l_t} \leq y_{et} + M_1(1 - \sum_{e'} x_{e't't}) \quad \forall e \in E, t, t' \in T \tag{13}$$

$$y_{et'} + d_{t'} + m_{r_e l_{t'} l_t} \leq y_{et} + (2 - b_{et} - b_{et'}) * M_1 \quad \forall e \in E, t, t' \in T : p_{g_t, g_{t'}} \in P \tag{14}$$

$$wt_{ee'} = etc_{r_e} \sum_t \left(m_{r_e l^s_e l_t} x^s_{e't} + m_{r_e l_t l^e_e} x^f_{e't} \right) + \sum_t \left(ttc_{r_e} \sum_{t'} m_{r_e l_t l_{t'}} x_{e'tt'} \right) + \sum_t d_t b_{e't} \ \ \forall e \in E, e' \in E \tag{15}$$

$$ve_{ee} = 0 \ \ \forall e \in E \tag{16}$$

$$vs_{e'e} \geq b_{e't}(1 - q_{et}) \ \ \forall t \in T, \forall e, e' \in E \tag{17}$$

$$vwu_{e'e} \geq wt_{ee'} - eub_e \ \ \forall e, e' \in E \tag{18}$$

$$vwl_{e'e} \geq elb_e - wt_{ee'} \ \ \forall e, e' \in E \tag{19}$$

$$vtu_{e'et} \geq y_{et} - tub_t + d_t - 1 + M_1 * (b_{e't} - 1) \quad \forall t \in T, \forall e, e' \in E \tag{20}$$

$$vteu_{e'et} \geq y_{et} + d_t + eta_{r_e} m_{r_e l_t l^e_e} - a^e_e - 1 - M_1 * (1 - x^f_{e't}) \quad \forall t \in T, \forall e, e' \in E \tag{21}$$

$$M_2 * ve_{e'e} \geq vs_{e'e} + vwu_{e'e} + vwl_{e'e} + \sum_t (vtu_{e'et} + vteu_{e'et}) \quad \forall e, e' \in E \tag{22}$$

$$x^s_{et} = x^f_{et} = 0 \ \ \forall e \in E, t \in T : t \notin T_e \tag{23}$$

$$x_{ett'} = 0 \ \ \forall e \in E, t, t' \in T : e \notin E_t \cap E_{t'} \tag{24}$$

$$x^s_{et}, x_{ett'}, x^f_{et} \in \{0, 1\} \ \ \forall e, e' \in E, t \in T \tag{25}$$

$$b_{et}, sx_e, ve_{e'e} \in \{0, 1\} \ \ \forall e, e' \in E, t \in T \tag{26}$$

The objective function consists of the weighted sum of minimizing the cost of used employees (1) and traveling time considered working time (3) and maximing so-called able pairs (2), i.e. the number of (shift, employee) pairs where the employee may carry out the shift.

The sequential structure of individual shifts is considered in equations (4) - (8). Each nonempty shift has a first task, a last task, and a possibly empty chain of tasks between them. A single task belongs to exactly one shift. Equations (9) and (10) ensure sx_e indicates nonemptiness of the shift corresponding to employee $e \in E$. Equation (9) is actually redundant from the viewpoint of the model, but it is included for the sake of clarity.

Equations (11) - (13) enforce lower bounds on starting times of tasks. The respected bounds are the earliest starting time of the task, the earliest availability of the employee and preceding task in the shift.

Equation (14) ensures shift-local precedence constraints are respected whenever constrained tasks are assigned to the same shift.

Equation (15) calculates working times for (shift, employee) pairs.

Equation (16) ensures that each employee may carry out their corresponding shift.

Equations (17) - (21) calculate the violations in individual constraints between all (shift, employee) pairs. Equation (22) composes these into indicator variables to signal whether individual pairs are compatible.

All parameters affecting y_{et} have integral values, hence y_{et} will always be integral.

The following constants were used in the model:

$M_1 = |TS| + \max_t d_t + \max_{r,l,l'} m_{rll'} + 1,$

$M_2 = 2M_1 \left(1 + \max\{n \in N : \sum_{i=1}^{n} d_{t_P(i)} \leq l_e^s - a_e^s + 1 \ \forall e \in E \text{ or } n = 0\}\right)$ where $P(i)$ is a permutation on task indices that maps tasks to ascending order by duration,

$$q_{et} = \begin{cases} 1 & \text{if employee } e \text{ has the right skills to do task } t, \\ 0 & \text{otherwise.} \end{cases}$$

To decrease model size and improve performance, especially in depot-based instances, it is possible to define an equivalence relation between employees that are identical with respect to all constraints pertaining time, i.e. availability, locations and transport type. This can be used to limit time calculations on the equivalence classes instead of all employees. We have omitted this representation for the sake of clarity of the model.

3 Benchmark instances

A number of benchmark instances have been published for workforce scheduling problems. The best-known benchmark data set for shift scheduling (or staff rostering) was generated by Curtois and Burke [1]. Nurse rostering benchmark instances have been created by Smet et al. [2]. Data sets for the first and second international nurse rostering competitions can be found in [3] and [4]. Nurmi [5] has introduced benchmark instances

Table 1 Seventeen parameters of the test instance generator for the GTSGP.

Set	Parameter	Set	Parameter
Volume	Nbr of timeslots	Shift	Avg nbr of employees fit for a shift
	Nbr of employees		Avg nbr of tasks in a shift
	Nbr of task types		Avg difference of shift durations (%)
	Nbr of skills		
	Min/max nbr of skills in task types	Emp	Avg difference of working times (%)
Task	Pct of fixed tasks	Trans	Nbr of transports
	Pct of back-office tasks		Prob of a location change between tasks in the same shift
	Avg task window deviation (%)		Max transition time required between tasks in the same shift
	Whether or not to use task type precedence constraints		

for the core staff rostering problem. Three sets of test instances for the shift minimization personnel task scheduling problem can be found in [6]. Benchmark instances for the workforce scheduling and routing problem can be found in [7]. This section presents the first benchmark instances for the general task-based shift generation problem.

Researchers quite often only solve some special artificial cases or one real-world case. The strength of randomly generated instances is the ability to produce instances with a wide variety of different properties. Still, they should be sufficiently simple to use in different test environments. The strength of practical cases is self-explanatory when the ultimate goal is to solve real-world problems and to integrate our optimization algorithms to scheduling software packages. However, real-world instances are often confidential and sometimes too complex to represent in academic papers. The test instances presented here were randomly generated allowing us to construct arbitrarily difficult cases. The data for the instances is available online [8].

We have generated 15 random test instances in a structured way to ensure similarities to practical cases. An algorithm performing well on certain artificial instances may not perform well on another set of artificial instances, which is why we created an elaborate random test instance generator for the GTSGP. The generator is guided by five parameter sets having a total number of seventeen parameters (see Table 1).

The generator creates instances where the number of shifts equals the number of employees. The basic idea is to construct an instance along with a feasible solution. We next describe the main elements of the generator.

First, a 0-1 matrix representing a solution to the corresponding assignment problem (ASP) is generated. This ASP solution is copied to the PAIRS solution, which consists of feasible (shift, employee) pairs. Next, the collection of skills related to each task type is generated. The transport types for the employees are generated so that each transport mode is used at least once. The global transition matrix M is initialized to zero.

For each shift the tasks are generated as follows. First, the number of tasks, the shift duration and the shift start and end times are generated. Then, for each task, the start and end times and the location are generated. Note that the total number of tasks is known only after the generator stops. For the location, the probability of a location change after the previous task is considered. If the location changes, the transition time

Table 2 Characteristics of the 16 test instances.

	EX	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
#timeslots	19	10	10	20	30	30	15	100	48	20	20	30	64	64	32	64
#emps	8	5	10	10	10	10	15	15	15	20	20	30	30	30	45	60
%time	51.4	77.8	91.7	66.6	83.5	92.9	100	65.7	81.8	75.8	73.8	40.7	100	100	79.6	83.5
#tasks	18	18	48	29	88	72	43	47	139	113	100	98	285	223	239	425
@tasks	2.3	3.6	4.8	2.9	8.8	7.2	2.9	3.1	9.3	5.7	5	3.3	9.5	7.4	5.3	7.1
%fixed	44.4	38.9	70.8	3.4	14.8	19.4	100	29.8	53.2	52.2	28	1	23.5	100	28.5	52
%backoff	5.6	0	0	0	0	0	0	10.6	5	0	0	6.1	0.7	0	2.1	0.2
%overlap	21.3	22.1	15.6	31.6	6.8	12.3	51.7	31.3	6.7	10.5	11.8	17.4	6.6	22.3	8.2	12.5
%maybe	60.1	52.3	47.1	74.1	29.6	30.6	51.7	74.7	53.7	53	54.1	85.4	37	22.3	69.2	43.4
@margin	65.4	80.1	79.9	104.3	112.9	118	0	175.3	591.8	214.2	140	231.3	380.8	0	326.3	239.3
#task types	11	3	10	5	10	10	15	5	20	10	10	10	30	5	20	20
@restrict	0	1.4	13.7	1.5	3.2	5.1	0	3.6	17.9	7.2	8	21.2	6.9	0	10.9	16.1
#skills	13	4	30	20	10	10	30	15	20	30	30	10	30	20	30	60
%skill	21	81.4	65.1	57.1	12.4	7.5	56.6	56.3	23.3	11.7	27.6	19.7	16.7	57.8	9.9	1.4
#locations	1	13	1	1	1	19	1	1	19	15	1	1	21	16	10	1
#transports	0	2	0	0	0	2	0	0	3	10	0	0	5	5	3	0
@pseudo	2.5	1	0.8	-0.2	0	1.4	0	0.7	0.6	3.6	2.8	0.3	0.7	1.1	0.8	0.7
ˆpseudo	-1.2	0.1	0	-0.4	-0.1	-0.3	0	-0.2	-1.6	-1.3	-0.8	-0.7	-1.6	-0.5	-3.9	-3.3
%greedy	94.4	72.2	93.8	89.7	93.2	77.8	100	93.6	88.5	85	94	88.8	92.6	95.5	89.5	93.7

%time = average availability of the employees to carry out the tasks, @tasks = average number of tasks in a shift, %fixed = percentage of fixed tasks, %backoff = percentage of back-office tasks , %overlap = probability of two tasks overlapping with random start times, %maybe = percentage of the tasks that can overlap, @margin = average percentage margin of task time windows, @restrict = the total number of tasks that cannot be executed after some other task in the same shift divided by the total number of tasks, %skill = average percentage of the total number of tasks each employee is qualified for, @pseudo = estimation of the number of "average employees" needed to complete the tasks, ˆpseudo = estimation of the number of "busy employees" needed to complete the tasks, %greedy = percentage of the tasks feasibly assigned by the greedy method.

is assigned. For the non-zero transition time, the transition matrix is updated. The time window of the task is changed in case of a fixed task or a back-office task. Next, the task type is assigned. The skills required to carry out the shift is updated according to the assigned task type. Furthermore, the task types that follow each other in the shift are updated. Finally, the shortest path algorithm is applied to the transition matrix of the selected transport type to ensure the triangle inequality holds.

When the tasks have been generated for the shifts, we first check the task types that follow each other in the shifts and create the shift-local precedence constraints accordingly. Then we decide how many employees are skilled to carry out each shift. Next, we select the employees and update the PAIRS solution accordingly. Finally, for each employee we run through the generated shifts and assign working times and possessed skills using the information generated for the tasks in the shift. When the generator ends, the tasks in the shifts present a feasible solution to the generated instance. The PAIRS solution includes the set of feasible (shift, employee) pairs and the ASP solution is a subset of the PAIRS solution.

Table 2 shows the characteristics of the 16 test instances. The EX instance is the simplified instance shown in Figure 2. The rest of the instances are generator instances. The number of timeslots varies between 10 and 100, the number of employees between 5 and 60 and the number of tasks between 19 and 425. The transport types are used in seven instances and precedence constraints in eleven instances. Two instances have only fixed tasks, and two instances have back-office tasks.

Most of the characteristics are self-explanatory, but some need more insight. For example, the $\%time$ measure is 0.50, if employee#1 is always available to carry out 25% of the tasks, employee#2 50% and employee#3 75%. The calculation of the task overlapping probabilities ($\%overlap$ and $\%maybe$) is very time-consuming. We need to iterate all task pairs and for each pair check for both tasks all possible time slot positions

within the time windows. The ^*pseudo* estimator is calculated as the sum of the task durations divided by the sum of the maximum working times of the employees. The @*pseudo* estimator uses average working times and considers the average transition times for the employees.

In order to compute %greedy, the percentage of the tasks that can be feasibly assigned to shifts by a greedy method, we use an iterative constructive heuristic. The basic idea is similar to the heuristic for the SMPTSP presented in [39]. However, the complexity of the GTSGP compared to the SMPTSP requires more advanced features. Furthermore, the greedy method uses a sophisticated iteration mechanism. The implementation details of the method are out of the scope of this study.

We believe that some of these characteristics can help us to understand the hardness of the instances. We return to this question in Section 5, where we present our computational results. First, the next section describes the solution methods used to tackle the problem.

4 Solution methods

We have created the PEAST algorithm [17] to solve challenging scheduling problems. There is evidence that the algorithm can successfully solve different problem domains. We have used it to schedule sports leagues (see e.g. [42] and [11]) and staff rosters [12] and to generate shifts [13] and school timetables [14]. Furthermore, we have used the algorithm to solve academic problems, such as balanced incomplete block design [10], single round robin tournaments with balanced home-away assignments and pre-assignments [10], days-off scheduling [15] and constraint minimum break problems [16].

The PEAST algorithm is a population-based metaheuristic. The heart of the algorithm is an ejection chain search, which is used to explore promising areas in the search space. Another important feature of the algorithm is the use of shuffling operators. They assist in escaping from local optima. Furthermore, simulated annealing refinement and tabu search are used to avoid staying stuck in promising search areas too long. The acronym PEAST stems from the methods used: Population, Ejection, Annealing, Shuffling and Tabu. We will next discuss these and other important features of the algorithm briefly. For detailed discussion we refer to [17]. The pseudo-code of the algorithm is given in Figure 5.

The ejection chain search extends a basic hill-climbing step to generate a sequence of moves in one step, leading from one solution candidate to another. The search operator moves an object, o_1, from its old position, p_1, to a new position, p_2, and then moves another object, o_2, from position p_2 to a new position, p_3, and so on, ending up with a sequence of moves. In the GTGSP, the object is a task and the current position is the relative position of the task in the list of tasks in the current shift. We refer to [17] for the detailed discussion of the object selection and the stopping criteria.

We improve the search by introducing a short-term *tabu list* which prevents reverse order moves in the same sequence of moves. i.e. if we move an object o from position p_1 to position p_2, we do not allow o to be moved back to position p_1 before a new sequence of moves begins. We use the *simulated annealing refinement* to decide whether to commit to a sequence of moves in the ejection chain search. This refinement is different from the standard simulated annealing. It is used in a three-fold manner [17]:

```
Set iteration limit t, population size n,
    elitism interval e, shuffling interval s and ADAGEN update interval a
Generate a random initial population of schedules S_i for 1 <= i <= n
Set best_schedule = null, iteration = 1
WHILE iteration ≤ t
    pop = 1
    WHILE pop <= n
        Apply ejection chain search to schedule S_pop to get a new schedule
        IF Cost(S_pop) < Cost(best_schedule) THEN Set best_schedule = S_pop
        pop = pop + 1
    END WHILE
    Update simulated annealing framework
    IF round ≡ 0 (mod a) THEN Update the ADAGEN framework
    IF round ≡ 0 (mod s) THEN Apply shuffling operators
    IF round ≡ 0 (mod e) THEN Replace the worst schedule with the best one
    Set iteration = iteration + 1
END WHILE
Output best_schedule
```

Fig. 5 The pseudo-code of the PEAST algorithm.

1. when choosing an object to be moved,
2. when choosing the new position of the object, and
3. when the sequence of moves is cut short.

The *shuffling operators* are used to perturb a solution into a potentially worse solution in order to escape from local optima. For example, a number of random objects are moved from their current positions to new random positions. Two shuffling operators are used in the GTGSP. The first one moves m tasks to random positions in random shifts. The value m is a random number between $m_{min} = 1$ and $m_{max} =$ the number of employees divided by 10, where m_{min} and m_{max} are constants determined by algorithm tuning. The other one selects from a random shift all the tasks that no employee can carry out and moves each of them to a random position in a random shift.

Due to the very large search space of the GTSGP, we have to consider two calculation issues. First, when generating a sequence of moves, we have to calculate the cost function many times during one ejection chain search. Furthermore, we have to calculate the cost function while rolling back the moves, often down to the starting point. Therefore we only recalculate those parts of the solution that are changed due to the single moves of the move sequence. We also have to solve an assignment problem to check the hard constraint H4, which can be solved in $O(n^3)$ time using appropriate data structures [43]. This is far too slow since we have to solve the problem in each single move in the ejection chain search. Fortunately, implementing the move sequence in such a way that we can apply the ideas presented in [44], we are able to calculate only the changes incurred to the initial assignment problem. In our implementation the calculation of changes requires $O(n^2)$ operations.

The PEAST algorithm uses a traditional penalty method, which assigns positive weights to the hard and soft constraints and sums the violation scores to get a single value to be optimized. The soft constraints are assigned fixed weights according to their significance. However, the hard constraints are assigned dynamic weights using the ADAGEN method described in [17].

The PEAST algorithm has been criticized of being nothing more than a collection of old ideas. To the best of our knowledge, at least the ejection chain search, simulated

annealing refinement and dynamic weights assigned to the hard constraints have been novel ideas. More importantly, each of the five components in the PEAST name are crucial to produce good-quality solutions. This was verified for three different problem domains in [17]. We will demonstrate this for the GTSGP in Section 5.

Most heuristic algorithms created to solve real-world cases use a fast greedy method to generate a sufficiently good initial solution. For the PEAST algorithm, we have found no evidence that a sophisticated initial solution improves results. On the contrary, random initial solutions seem to yield superior or at least as good results in all the real-world cases where we have applied the algorithm. We will demonstrate this for the GTSGP in Section 5.

5 Computational results

This section presents our computational results for the benchmark instances introduced in Section 3. We solved the instances using the PEAST algorithm described in Section 4. We used exactly the same version and the parameters of the algorithm that are in use in the real-world optimization in the workforce scheduling [12] and in the professional sports league scheduling [42]. We used no domain-specific knowledge in order to generate better solutions.

For the benchmark instances the number of shifts required was known a priori to be equal to the number of employees. Due to in part this, all our tests were done using weights $\alpha = 0, \beta = 1, \gamma = 0$ for the cost function. This choice leads to maximization of able pairs, which in general tends to maximize the number of shifts used.

The PEAST algorithm was able to find a feasible solution to all but one of the instances. Table 3 shows the results and the number of minutes required to reach the solution (best of ten runs). All test runs using the PEAST algorithm were carried out using an Acer Laptop on an Intel Core i7-8550 at 1.8GHz with 8GB RAM operating Windows 10. We suspect that better solutions for most of the instances exist. The data for the instances is available online [8].

Mathematical solvers were used for the sake of getting a baseline on problem difficulty. The aim was not comparing solvers against either PEAST or each other. The Gurobi runs were done with Gurobi 8.1 through the Java API on a 4GHz i7-4790K with 16GB of RAM using default settings. The CPLEX runs were done with Dell XPS 8930, with Intel Core i7-8700 @ 3.2 GHz. All solver runs were limited to 30 minutes. For instances 00, 04 and 05, GUROBI found provably optimal solutions.

We believe that some of the characteristics of the GTSGP instances presented in Section 3 could help us to understand the hardness of the instances. Of course, the number of tasks, the number of employees and the number of time slots have a direct influence on the hardness of the instance, since they enlarge the search space. To be more exact, the combinatorial search space explodes when the average number of tasks in a shift (@tasks) and the number of average employees needed to complete the tasks (@pseudo) increase. In our first test runs, the %time, @margin and %greedy measures seem to influence the hardness of the problem the most. However, we have not yet conducted enough experiments to determine what makes an instance easy or hard.

Table 3 The results for the sixteen benchmark instances.

	PEAST ofv	PEAST time (m)	GUR	GUR (b)	CPLEX	CPLEX (b)
EX	15	1				
00	6	1	6	6	x	x
01	53	1	51	56	49	57
02	25	1	25	33	22	36
03	15	1	-	42	-	50
04	22	85	22	22	x	x
05	146	1	146	146	146	161
06	100	3	-	163	-	176
07	110	21	-	-	x	x
08	-	-	-	-	x	x
09	60	1	-	264	-	352
10	205	171	-	-	-	778
11	371	65	-	-	x	x
12	441	30	-	-	x	x
13	585	59	-	-	x	x
14	615	202	-	-	x	x

PEAST ofv = best PEAST solution found, PEAST time (m) = time to find feasible solution by PEAST, GUR (b) = best value (bound) found by GUROBI, CPLEX (b) = best value (bound) found by CPLEX.
- = not found.
x = not run.

Table 4 The effect of individual PEAST components on solution quality.

Without	Solution value(*)	Without	Solution value(*)
Population	3	Elitism	11
Ejection chain search	11	ADAGEN	1
SA refinement	2	Random start(x)	1
Shuffling operators	17		
Tabu list	1		
PEAST	0		

(*) The solution value indicates the number of hard constraint violations. The value is the median of ten runs.
(x) Greedy initial solutions used instead of random initial solutions.

As stated in Section 4, each of the five components in the PEAST name are crucial to produce good-quality solutions. To demonstrate this for the GTSGP, we solved benchmark instance #8 without using each of these components. The results are shown in Table 4. The results are in line with our findings in other problem domains. The results also show that elitism and ADAGEN method produce better results. Furthermore, a greedy initial solution does not improve results. Instead, random initial solutions yield better results.

6 Conclusions and future steps

We presented the problem definition and the mathematical formulation of the General Task-based Shift Generation Problem (GTSGP). We justified the importance of the problem in the big picture of the workforce optimization and the real-world workforce scheduling process.

We introduced the first set of benchmark instances for the GTSGP, which has been made available online. We solved the instances to feasibility using the PEAST al-