Hindawi
Advances in Operations Research
Volume 2021, Article ID 8876990, 12 pages
https://doi.org/10.1155/2021/8876990

Research Article

A Successful Three-Phase Metaheuristic for the Shift Minimization Personal Task Scheduling Problem

Kimmo Nurmi and Nico Kyngäs

Satakunta University of Applied Sciences, Satakunnankatu 23, 28130 Pori, Finland

Correspondence should be addressed to Kimmo Nurmi; cimmo.nurmi@samk.fi

Received 16 September 2020; Revised 11 January 2021; Accepted 16 January 2021; Published 29 January 2021

Academic Editor: Panagiotis P. Repoussis

Workforce scheduling process consists of three major phases: workload prediction, shift generation, and staff rostering. Shift generation is the process of transforming the determined workload into shifts as accurately as possible. The Shift Minimization Personnel Task Scheduling Problem (SMPTSP) is a problem in which a set of tasks with fixed start and finish times must be allocated to a heterogeneous workforce. We show that the presented three-phase metaheuristic can successfully solve the most challenging SMPTSP benchmark instances. The metaheuristic was able to solve 44 of the 47 instances to optimality. The metaheuristic produced the best overall results compared to the previously published methods. The results were generated as a by-product when solving a more complicated General Task-based Shift Generation Problem. The metaheuristic generated comparable results to the methods using commercial MILP solvers as part of the solution process. The presented method is suitable for application in large real-world scenarios. Application areas include cleaning, home care, guarding, manufacturing, and delivery of goods.

1. Introduction

Workforce scheduling process consists of three major phases: workload prediction, shift generation, and staff rostering (see, e.g., [1]). Workload prediction is the phase of determining the workload based on known and predicted events. Shift generation is the process of transforming the determined workload into shifts as accurately as possible. The generated shifts form an input for the staff rostering phase, where employees are assigned to the shifts.

From the practical point of view, the workforce scheduling process relies on both optimization resources and human resources. It links the organization together, optimizing processes and streamlining decision-making. A shift scheduler should consider legal as well as human aspects of shift work. From an employee perspective, shift work is associated with pressure on social and family life, health issues, motivation, and loyalty. When the input data for the shift generation and for the staff rostering are well validated, significant benefits in financial efficiency and employee satisfaction can be achieved by applying relevant optimization methods. A good literature review of workforce scheduling can be found in [2].

The shift generation phase creates a shift structure including the tasks to be carried out in the shifts, the timing of the tasks and breaks, and the skills required in the shifts. 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 main contributions of this paper are the following:

(i) The presented three-phase metaheuristic can successfully solve the most challenging SMPTSP benchmark instances

(ii) The metaheuristic was able to solve 44 of the 47 instances to optimality

(iii) The metaheuristic produced the best overall results compared to the previously published methods

(iv) The metaheuristic generates comparable results to the methods using commercial MILP solvers as part of the solution process

(v) The results can be reached when solving a more complicated General Task-based Shift Generation Problem

The paper is organized as follows: first, we describe the Shift Minimization Personnel Task Scheduling Problem and the General Task-based Shift Generation Problem. Then, we introduce the set of most challenging benchmark instances for SMPTSP. We describe the three-phase metaheuristic that we have used to solve a wide variety of combinatorial problems. Lastly, we present our computational results and compare them to the best-known results.

1.1. Related Work. The first major contribution for the employee-based shift generation problem was the study by Musliu et al. [3]. They introduced a problem in which the workforce requirements for a certain period 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. [4] proposed an employee-based problem in which the most important issue was to minimize the number of different kinds of shifts used. Application areas of employee-based shift generation include hospitals, retail stores, and call centers, where the number of employees required at certain timeslots can be predicted from the arrival times of customers.

In the task-based shift generation problem, the goal is to create shifts and assign tasks to these shifts so that the employees can be assigned to the shifts. Application areas include cleaning, home care, guarding, manufacturing, and delivery of goods. The first major contribution of the task-based problem was the study by Dowling et al. [5]. 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. Valls et al. [6] presented a model where they minimized the number of workers required to perform a machine load plan. Later, Krishnamoorthy and Ernst [7] introduced a similar group of problems, which they called Personnel Task Scheduling Problems (PTSPs). Given the staff that are rostered on a particular day, the PTSP is to allocate each individual task, with specified start and end times, to available staff who have skills to perform the task.

Subsequently, Krishnamoorthy et al. [8] 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. Nurmi et al. [9] defined the General Task-based Shift Generation Problem (GTSGP) in which the task is to create anonymous shifts and assign tasks to these shifts so that employees can be assigned to the shifts.

2. Materials and Methods

2.1. Problem Description. The Shift Minimization Personnel Task Scheduling Problem (SMPTSP) consists of assigning a set of tasks with specific start and end times to employees who have specific skill sets and availability intervals. The objective is to find a feasible assignment of all the tasks that minimizes the number of employees used. The objective is motivated by situations where a large pool of casual employees is available and management would like to minimize the pool usage.

SMPTSP can be defined formally as follows. A set of tasks $J = \{t_1, \ldots, t_n\}$ needs to be allocated to a set of heterogeneous employees $E = \{e_1, \ldots, e_m\}$ over a specified planning horizon. The processing time interval at which a task t has to be carried out is determined by a timetable with fixed start time s_t and finish time f_t. Each employee e has a set of tasks $J_e \subseteq J$ that e can carry out. Each task t has a set of employees $E_t \subseteq E$ that can carry t. All sets J_e and E_t are defined based on skills of employees/skill requirements of tasks and availability of employees/time windows of tasks. The objective is to minimize the number of employees required to carry out the given set of tasks. The mathematical formulation of the problem was first given in [8].

Figure 1 shows a simplified instance of SMPTSP. The instance and the presented solution have the following characteristics:

(i) The planning period is divided into 18 timeslots

(ii) The number of tasks is fourteen and the number of employees is seven (indicated by letters from A to G)

(iii) The duration of the tasks is given by the length of the corresponding rectangles

(iv) The employees able to carry out a task are indicated in the rectangles

(v) The colors indicate which tasks belong to the same shift

(vi) The number of employees used to carry out the shifts is six

(vii) An employee carrying out a task is denoted by parentheses (employee G has no tasks)

The following basic assumptions for SMPTSP hold:

(A1) Preemption of tasks is not allowed

(A2) There are no precedence constraints among the tasks

(A3) Each task is processed exactly once without interruption

(A4) Each employee can execute at most one task at a time

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. Instead of minimizing the number of employees required to carry out the given set of tasks, the objective 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. The mathematical formulation of the problem was first given in [10].

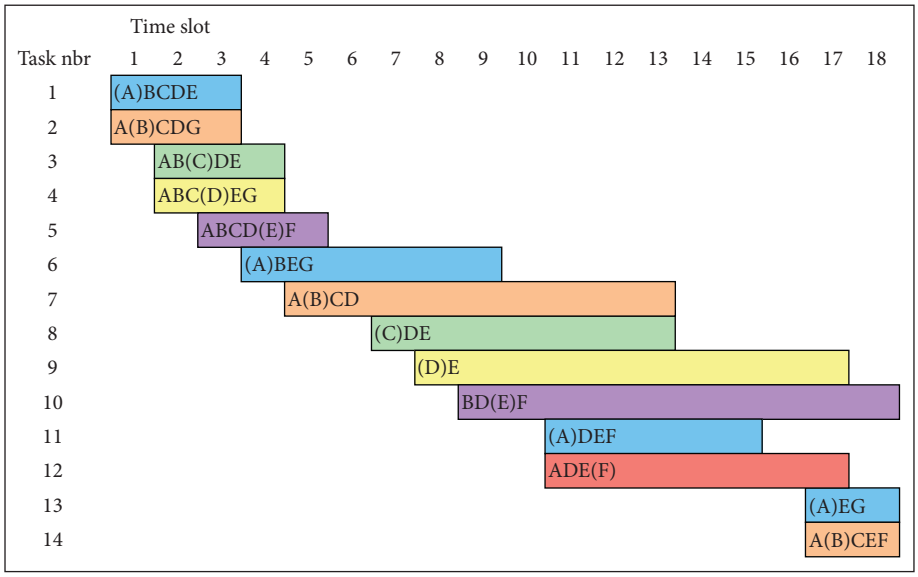

Figure 1: A simplified instance of SMPTSP and a feasible solution to the instance. The letters indicate employees able to carry out a task. An employee assigned to carry out a task is denoted by parentheses (employee G has no tasks). The colors indicate generated shifts.

The motivation for GTSGP derives from the workforce scheduling process. In the shift generation phase, we should create as versatile shifts as possible to ensure that the rostering of the staff can be completed. We should ensure that the resulting set of shifts can be carried out by the employees; that is, each shift can be assigned to an employee s.t. all shifts are assigned to someone and no employee is assigned to multiple shifts. In practical applications of GTSGP, the full-time permanent and temporary employees are expected to cover 100% of the total workload. This is opposite to the idea behind SMPTSP.

SMPTSP is a considerably simplified version of the GTSGP. Nonetheless, as Kroon et al. showed in [11], SMPTSP is NP-hard in the strong sense, even if preemption of tasks is allowed (A1). GTSGP has the same assumptions as SMPTSP besides (A2). GTSGP differs from SMPTSP in several important ways:

(B1) Tasks are not explicitly assigned to employees

(B2) Tasks are not fixed in time

(B3) Tasks may have shift-local precedence constraints

(B4) Transition times between tasks are considered

(B5) Employees have total working time restrictions

(B6) Employees have availability restrictions

To the best of our knowledge, eight significant and notable solution methods have been designed to tackle the problem. Krishnamoorthy et al. [8] used a Lagrangian approach to solve large instances of SMPSTP. They relaxed the task assignment constraints and used the deviations in the objective function with Lagrange coefficients, then solved each worker's problem independently. Lin and Ying [12] developed a three-phase heuristic for SMPTSP. They obtained 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. The experimental results underlined the superiority of the proposed algorithm over the Krishnamoorthy et al. algorithm.

Smet et al. [13] used a 2-phase constructive metaheuristic approach. In phase one, they used three constructive heuristic methods, and in phase two, they used a hybrid search and optimization method. Their method was the first one to solve all the instances of the first SMPTSP benchmark set (see the next subsection) to optimality. The authors stated that their novel algorithm holds the state of the art for the SMPTSP. Fages and Lapegue [14] used constraint programming with both a top-down and a bottom-up approach. Their extensive experiments showed that their contribution significantly improved a straightforward SMPTSP model, so that it competed with the best-known approaches.

Baatar et al. [15] developed a branch and bound scheme, which was used in conjunction with two column generation-based approaches and a heuristic algorithm to create an efficient solution procedure. The authors demonstrated that their approach performs better than just using a standard commercial MILP solver (CPLEX). Hojati [16] proposed a novel greedy heuristic for SMPTSP. The results showed that the heuristic performs well relative to the current solution methods and has the added advantage of being able to solve very large instances fast. The author noted that his method requires no commercial MILP solver as the other methods described earlier. Niraj Ramesh et al. [17] solved the problem by decomposing it into separate subproblems and developed several exact and heuristic methods to solve the resulting subproblems. The authors stated that even though they proved interesting theoretical results, their methods fulfilled the expectations practically by easily surpassing other comparable exact methods.

Chirayil Chandrasekharan et al. [18] improved the metaheuristic introduced in [12] and presented a decomposition-based method where subproblems are solved to optimality using exact techniques. The optimal solutions of subproblems are subsequently utilized to construct a feasible solution for the entire problem. The new features not only improved solution quality, but they also played a pivotal role in improving the scalability of the algorithm. The authors state that the method is suitable for application in large real-world scenarios. Their method was the first one to solve all the instances of the second SMPTSP benchmark set (see the next subsection) to optimality. Furthermore, they were able to produce optimal solutions to almost all the instances from the third SMPTSP benchmark set (see the next subsection).

Currently, only one solution method has been published for the General Task-based Shift Generation Problem. The problem was only recently introduced in Nurmi et al. [9]. The authors used the PEAST metaheuristic to solve the GTSGP test instances introduced in [10]. We will give insight into the metaheuristic in The Three-Phase Metaheuristic section.

2.2. SMPTSP Benchmark Instances. To the best of our knowledge, no real-world benchmark instances for SMPTSP have been published. Thus far, three sets of artificial benchmark instances have been published. Krishnamoorthy et al. [8] presented the first data set of 137 instances for SMPTSP. The data set is referred to as KEB instances. The instances were generated by five characteristics: number of employees, number of tasks, lengths of the tasks, tightness level, and multiskilling level. The tightness level is defined as the total length of all tasks as a percentage of the total availability of all employees. When this level is 100%, the tasks could cover exactly the timeslots available for the employees. The multiskilling level is defined as the average percentage of the total number of tasks each employee is qualified for. When this level is 100%, each employee can carry out all tasks. From now on, we will call the multiskilling level as the task skill level. The number of employees in the KEB data set ranges from 22 to 422, while the number of tasks ranges from 40 to 2105. The shift lengths are fixed to 1440. The lengths of the tasks vary between 50 and 400. The tightness level is fixed to about 90%, and the skill level is either 33% or 66%.

Smet et al. [13] generated the second data set of ten instances, because they were able to solve all the KEB instances to optimality. The data set is referred to as SWMB instances. The number of employees ranges from 44 to 153, while the number of tasks ranges from 258 to 1577. The shift lengths are fixed to 1440. The length of the tasks is either 120 or 280. The tightness level is fixed to about 90%, and the skill level is either 20% or 30%. Fages and Lapègue [14] generated the third data set of 100 instances, since the KEB and SWMB instances are trivial with respect to finding good-quality lower bounds. This data set is referred to as FL instances. The number of employees ranges from 62 to 948, while the number of tasks ranges from 71 to 1583. The shift lengths are fixed to 1560. The length of the task is either 120 or 280. The tightness level is fixed to about 90%, and the skill level is fixed to about 25%. A good summary of the three data sets can be found in [16].

In the interest of brevity, we will not solve all the 247 instances from the three data sets. Some of the instances are easy to solve, and most of the eight methods described earlier in this section perform well on most of the instances. We next determine 47 of the most challenging instances of the three data sets as follows:

KEB: the VAWA heuristic of Krishnamoorthy et al. [8] was not able to find the optimum solution, and the solution obtained by the constructive heuristic of Lin and Ying [12] was at least 5% inferior to the optimum solution totaling 14 of the 137 instances.

SWMB: all the 10 instances.

FL: the greedy heuristic of Hojati [16] was not able to find the optimum solution, or the solution obtained by the constraint programming approach of Fages and Lapègue [14] was at least 3% inferior to the optimum solution totaling 23 of the 100 instances.

Tables 1–3 show the characteristics of the selected instances. In addition to the tightness and skill levels, we define five other hardness indicators. The @AVG measure indicates the estimated average number of tasks per nonempty shift. The number of tasks is divided by a lower bound for the minimum number of shifts. For the lower bounds, we use the values published in [16] using the general lower bounding procedure by Solyali [19]. In addition to the task skill level, we define the shift skill level which describes how qualified an average employee is to carry out all the tasks of an average shift, i.e., $tsl * n/lb$, where tsl = task skill level, n = number of tasks, and lb = Solyali lower bound. The overlap level is the probability of two tasks to overlap. To calculate the probability, we need to iterate all task pairs once.

The %ICH measure indicates the percentage of the tasks feasibly assigned by the iterated constructive heuristic described in The Three-Phase Metaheuristic section. The heuristic maximizes the number of feasible (shift and employee) pairs, i.e., solving the GTSGP problem, as described earlier. The @PAIRS measure indicates an estimation of the average number of feasible pairs per nonempty shift. The number of feasible pairs generated by the iterated constructive heuristic is divided by the Solyali lower bound. Note that the GTSGP solution generated by the heuristic is quite far from the optimum. The values shown for %ICH and @PAIRS are the best of ten runs.

Recall that we have selected to solve the most challenging instances. The following observations can be drawn from the characteristics of the instances:

(i) FL instances have fewer tasks per shift than in most of the other instances. Partly due to this, also the shift skill levels of FL instances are higher. These should make them easier to solve.

(ii) The tightness levels of FL instances are lower than in the other instances. This should make them easier to solve.

Table 1: The characteristics of the 14 KEB instances.

KEB#	Opt	#Emps	#Tasks	@AVG	Tightness level	Task skill level	Shift skill level	Overlap level	%ICH	@PAIRS
9	40	49	104	2.6	89.9	35.0	6.52	57.3	**98.1**	4.15
11	20	24	119	6.0	90.0	36.2	0.24	26.0	**96.6**	0.90
45	60	67	420	7.0	90.0	33.9	0.05	24.1	**97.9**	0.92
59	59	70	525	8.9	91.4	34.4	**0.01**	19.4	**96.8**	**0.83**
75	60	72	665	11.1	90.0	34.2	**0.001**	15.4	**98.2**	**0.88**
77	160	180	688	4.3	90.0	33.4	0.87	36.8	99.0	2.77
79	80	94	689	8.6	90.1	33.6	**0.01**	19.8	99.4	0.99
80	99	112	691	7.0	90.9	33.8	0.05	24.4	99.4	1.03
89	70	88	788	11.3	90.1	34.0	**0.001**	15.3	**98.6**	**0.86**
94	80	93	881	11.0	90.0	33.8	**0.001**	15.6	98.9	**0.89**
98	80	91	896	11.2	90.0	34.2	**0.001**	15.3	99.2	0.94
106	100	121	1096	11.0	90.0	33.3	**0.001**	15.6	99.7	0.96
107	100	114	1112	11.1	90.0	33.7	**0.001**	15.4	99.8	0.96
108	128	162	1115	8.7	91.4	33.6	**0.008**	19.9	99.5	1.00

Three bold values on a row indicate a hard instance.

Table 2: The characteristics of the 10 SWMB instances.

SWMB#	Opt	#Emps	#Tasks	@AVG	Tightness level	Task skill level	Shift skill level	Overlap level	%ICH	@PAIRS
1	40	50	258	6.5	89.6	19.5	0.003	25.6	**91.9**	**0.63**
2	40	44	510	12.4	87.6	19.6	**0.000**	13.3	**93.5**	**0.58**
3	77	102	525	6.8	93.5	30.0	0.027	25.4	96.8	0.92
4	98	113	647	6.6	91.7	20.0	0.002	25.7	96.4	0.86
5	59	77	777	13.2	91.5	29.7	**0.000**	13.2	96.4	0.85
6	116	135	777	6.7	92.9	19.9	0.002	25.8	96.3	1.68
7	59	70	781	12.8	88.5	19.9	**0.000**	13.1	**95.0**	**0.64**
8	79	88	1022	12.8	90.0	19.9	**0.000**	12.8	96.0	**0.65**
9	98	125	1308	13.2	90.9	19.8	**0.000**	13.2	96.5	**0.75**
10	116	153	1577	13.6	93.1	19.9	**0.000**	13.6	**95.8**	**0.66**

Three bold values on a row indicate a hard instance.

Table 3: The characteristics of the 23 FL instances.

FL#	Opt (LB*)	#Emps	#Tasks	@AVG	Tightness level	Task skill level	Shift skill level	Overlap level	%ICH	@PAIRS
5	30*	81	110	3.7	17.8	26.4	0.76	10.5	**98.2**	87.0
28	105	262	402	3.8	19.1	26.0	0.58	11.0	99.5	**31.7**
29	95	248	355	3.7	18.5	28.3	0.90	11.6	100	**39.7**
31	116	290	488	4.2	21.0	25.7	**0.33**	10.4	**97.7**	**32.0**
33	132	338	534	4.0	20.3	25.7	0.41	10.8	99.4	**35.1**
35	118	308	469	4.0	19.8	26.6	0.52	11.1	99.4	**37.3**
39	108	284	446	4.1	19.8	25.7	0.36	10.6	**98.4**	**29.9**
45	144	376	586	4.1	20.5	27.0	0.49	11.4	99.7	50.8
46	157	409	635	4.0	20.2	26.6	0.47	11.2	99.4	**46.1**
54	190	498	850	4.5	22.5	25.8	**0.23**	10.7	**96.8**	**48.0**
60	173	443	783	4.5	21.9	27.0	**0.27**	10.7	**98.6**	**46.3**
61	222	551	891	4.0	20.0	26.3	0.47	11.0	**98.2**	66.2
62	262	610	1096	4.2	20.8	25.5	**0.33**	10.2	99.0	64.8
63	203	524	905	4.5	21.9	26.5	**0.27**	11.1	**98.1**	56.4
64	140	366	570	4.1	20.2	26.2	0.43	11.0	99.1	**43.8**
68	219	561	958	4.4	21.4	27.3	0.35	11.0	99.3	66.0
69	211	550	891	4.2	21.1	26.1	0.34	10.8	99.3	62.8
77	248	648	1123	4.5	22.1	26.6	**0.25**	10.7	**98.4**	69.7
79	246	638	1052	4.3	21.0	26.3	**0.33**	10.8	99.4	74.1
80	222	578	885	4.0	19.6	27.0	0.54	11.2	99.5	77.5
84	247	644	1121	4.5	21.9	26.1	**0.22**	10.4	99.2	68.4
89	319*	790	1371	4.3	21.0	25.6	**0.29**	10.5	99.6	82.3
94	313	812	1394	4.5	21.9	25.7	**0.24**	10.6	**98.3**	78.1

Three bold values on a row indicate a hard instance.

(iii) About half of KEB and SWMB instances have high overlap levels, which should make them easier to solve.

(iv) The percentage of the tasks feasibly assigned by the iterated constructive heuristic (%ICH) is lower for SWMB instances, which could make them harder to solve. KEB and FL instances seem to be equally easy (or hard) to solve in this sense.

(v) FL instances are very similar to each other. This should make them equally hard (or easy) to solve.

According to the @PAIRS value, barely one employee is skilled to carry out each shift in KEB and SWMB instances. For FL instances, the generated solution will be such that quite many employees may carry out several shifts. This could make FL instances easier to solve.

2.3. The Three-Phase Metaheuristic. We have created the PEAST metaheuristic (see, e.g., [20]) to solve real-world scheduling problems. The metaheuristic has been in commercial use for several years, for example, in staff rostering and in professional sports league scheduling. Furthermore, we have used it to solve more academic problems, such as balanced incomplete block design, single round robin tournaments with balanced home-away assignments and preassignments, days-off scheduling, and constraint minimum break problems. However, it should be clear that even though a metaheuristic can be powerful for several problem types, it is not guaranteed to work well for other problem types. The No Free Lunch Theorem [21] implies that there cannot exist a superior optimization method.

Recently, we have started to work with a new real-world problem, which is an application of GTSGP. As part of this process, we have created a solution method suitable for commercial use. The method is based on the PEAST metaheuristic. In all the previous adoptions of PEAST, we have used random initial solutions. 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. However, due to the running time requirements, PEAST alone is too slow for the largest practical instances of GTSGP. We need a fast heuristic to generate very good initial solutions. Note that not reaching the best possible (academic) solution is not an issue in practical applications of GTSGP. After the optimization process, new tasks may arise and some of the tasks may need to be changed or removed.

We generate an initial solution using a simple ruin and recreate heuristic (RRH) similar to that described in [22]. The pseudocode is given in Figure 2. The ruin operator removes strings of adjacent tasks of random length from the solution. All tasks assigned to the incumbent solution are processed in random order. For each task t, a random string that contains t is removed unless the shift containing t has already been removed. When the total number of removed tasks exceeds the given parameter, the ruin operator quits. The recreate operator adds free tasks one by one to their respective best positions in the incumbent solution. First, all free tasks are sorted in random order. For each task t, all feasible addition positions in the incumbent solution are evaluated. Note that the concept of a position depends on the exact problem.

```
round ← 0, bestSol ← null
while round < f do
    storedSol ← currentSol
    seed ← random task from T
    sm ← maximum number of shifts to ruin
    tm ← maximum number of tasks to ruin per shift
    tasks ← list of all tasks in random order
    T' = ∅
    for t ∈ tasks
      if t ∉ T'
            l ← U (1, min(|S (t)|, tm))
            Remove a random string of l tasks from S(t)
            Update ruin quota
            T' ← T' ∪ t
        end if
        if ruin quota is full
            break
        end if
    end for
    tasks ← list of all unassigned tasks in random order
    for t ∈ tasks
        P ← all spots in all current shifts where adding t is
          feasible in worsening order of objective
        for p ∈ P
            if U (1, 100) ≥ lowLevelSkipChance
                Add task t to spot p
                break
            end if
        end for
    end for
    Update bestSol if necessary
    if U (1, 100) ≤ highLevelSkipChance
        currentSol ← storedSol
    end if
    round ← round + 1
end while
```

Figure 2: The pseudocode of the ruin and recreate heuristic.

For GTSGP, a position is determined by an immediate predecessor, e.g., a task or an employee. Note that the tasks can have wide time windows in GTSGP; hence, the order of tasks within a shift is not predetermined. In SMPTSP, there are no time windows, which fixes the order of tasks within a shift. Task t is then added to the position that leads to the best objective function value, with a small chance to skip over to the next best position. Consecutive skipping is not constrained in any way, so t might not get assigned even if it has feasible addition positions. When all free tasks have been processed, the recreate operator quits.

We further speed up the overall running time by generating an initial solution to the ruin and recreate heuristic. Initial solutions are generated by using a very fast iterated constructive heuristic (ICH) based on the ideas presented in [12]. The pseudocode is given in Figures 3(a) and 3(b). This is the heuristic referred in The Three-Phase Metaheuristic

STEP 0
Set T = set of tasks t_{ij}, where i denotes the task index and j runs through all the possible start times of the task i
Set r_{ij} = false, for each task in T (restricted tasks)
Set n = number of tasks (task indexes)
Set a_i = true, for each task i (assignable task indexes)
Set number of restricted iterations $r_{\max}$ = number of employees
Set d_i = duration of task i
Set number of assigned tasks $m = 0$
Set maximum number of assigned tasks $m_{\max} = 0$

STEP 1
Set TaskPairwiseConflict (t_{ij}, t_{kl}) = true, if tasks t_{ij} and t_{kl} overlap in time; otherwise set to false
Set TaskConflict (t_{ij}, s) = true, if task t_{ij} has a pairwise conflict with at least one of the tasks currently in the shift s; otherwise set to false
Set EmployeeConflict (t_{ij}, s) = true, if the employee s is not skilled for task t_{ij}; otherwise set to false

STEP 2
Sort tasks in T by ascending start time, time window and task index

STEP 3
For each task t_{ij} in T, where a_i = true
InsertToBestShift (t_{ij})

STEP 4
If $m = n$, STOP
If cycle occurred, i.e. the same tasks and start times have been assigned to the shifts, Goto STEP 7

STEP 5
For each task t_{ij} in T , where a_i = true and r_{ij} = false
MoveToAnotherShift (t_{ij})
If task t_{ij} has been processed at this step more than $r_{\max}$ times, set r_{ij} = true

STEP 6
Set r_{ij} = false, for each task in T, if at least one of the tasks has been restricted more than $r_{\max}$ iterations
Goto STEP 3

STEP 7
Set left over tasks to random shifts and random (feasible) start times

(a)

InsertToBestShift (t_{ij})

Process the shifts in random order
Choose the shift s for which
EmployeeConflict(t_{ij}, s) = false and
TaskConflict(t_{ij}, s) = false and
Σd_k = max, where task $k \in s$
If such shift can be found
Insert task t_{ij} to shift s
Set a_i = false
Set $m = m + 1$
If $m > m_{\max}$, save the current solution and update $m_{\max}$

MoveToAnotherShift (t_{ij})

Process the shifts in random order
Choose the shift s for which
EmployeeConflict (t_{ij}, s) = false and
Σd_k = min, where task $k \in s$ and TaskPairwiseConflict(t_{ij}, t_{k^*}) = true
If such shift can be found
For each task t_{kl} for which TaskPairwiseConflict(t_{ij}, t_{kl}) = true
Remove the task k from shift s and set a_k = true
Set $m = m - 1$
Insert task t_{ij} to shift s
Set a_i = false
Set $m = m + 1$

(b)

Figure 3: (a), (b) The pseudocode of the iterated construction heuristic.

section when calculating the %ICH measure. The heuristic assigns the unassigned tasks one by one to an assignable shift until all tasks have been considered (STEP 3). When trying to assign a task, we choose the shift with no employee and task conflicts and with the largest total cumulative processing time. A reassignment procedure (STEP 5) is repeatedly applied to allocate each unassigned task of to a shift until all tasks have been assigned or cycling exists. When trying to assign task t, we choose shift s with no employee conflicts and with the least total cumulative processing time with the tasks that are currently in conflict with task t. To reassign task t to shift s, we remove all tasks in shift s that conflict with task t.

The PEAST metaheuristic tries to improve the solution generated by the ruin and recreate heuristic (and the iterated constructive heuristic). In practical applications of GTSGP, the PEAST metaheuristic

(i) generates as versatile shifts as possible to

(ii) ensure that the rostering of the staff can be completed, so that

(iii) the computation time is still acceptable considering the release time of the rosters.

Therefore, we do not seek the fastest possible solution method. For our purposes, it is advantageous to use more computation time in order to achieve shifts that are more versatile.

The pseudocode of the PEAST metaheuristic is given in Figure 4. We have created PEASTP (see, e.g., [20]) by combining features from six well-known metaheuristics: MH1, genetic algorithm; MH2, ejection chain method; MH3, tabu search; MH4, simulated annealing; MH5, variable neighbourhood search; MH6, ruin and recreate method.

The performance of these metaheuristics has been scientifically justified, and a significant number of experimental studies have been run at the algorithmic level. They have indisputably introduced true novelties to the repertoire of optimization methods. By combining and carefully tuning the most efficient operators of the scientifically valid metaheuristics, an experienced heuristic designer can solve real-life optimization problems efficiently. As stated at the beginning of the section, there is evidence that PEAST can successfully solve different problem domains. To the best of our knowledge, no other method combines metaheuristic features such as in PEAST.

PEASTP uses a population of solutions in each iteration (MH1). The reproduction operation is, to a certain extent, based on steady-state reproduction: the new solution replaces the old one if it has a better or equal objective function value. Furthermore, the worst solution is replaced with the best one at given intervals, i.e., elitism is used. The ejection chain search is the heart of PEASTP. It explores promising areas in the search space. The ejection chain search (MH2) extends a basic hill-climbing step to generate a sequence of moves in one step, leading from one solution candidate to another. The ejection operator actually implements several simple local search operators to work on a single solution (MH1).

The ejection chain search is improved by introducing a tabu list, which prevents reverse order moves in the same sequence of moves (MH3). A simulated annealing refinement is used to decide whether to commit to a sequence of moves in the ejection chain search (MH4). This refinement is different from the standard simulated annealing. It is used in a three-fold manner. The simulated annealing refinement and tabu search are used to avoid staying stuck in promising search areas too long. Shuffling operators assist in escaping from local optima. They are used to perturb a solution into a potentially worse solution in order to escape from local optima (MH5). A shuffling followed by several ejection chain searches obtains better solutions using the same idea as the ruin and recreate method (MH6).

PEAST uses a *population of solutions* in each iteration. A new solution instantly replaces an old one if it has a better or equal objective function value. Furthermore, the worst solution is replaced with the best one at given intervals, i.e., elitism is used. PEAST uses a traditional penalty method, which assigns positive weights (penalties) 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 a unique ADAGEN method described in [20].

Each of the five metaheuristic components in PEAST is crucial to produce good-quality solutions. This was verified for three different problem domains in [20]. Recently, the results in solving GTSGP instances [10] showed that when any one of the components was removed, the results were clearly worse. The same held even if PEAST was given twice as much time to run without one of the components.

The implementation of PEAST has changed so distinctly that we call the new version PEASTP. The data structures have been recoded, and the calculation of the cost function in ejection chain search has been renewed. The modifications have enabled us to parallelize PEAST (hence PEASTP), which in turn enables us to solve problem instances far larger than before. Nonetheless, we cannot use random initial solutions when solving practical GTSGP instances.

3. Results and Discussion

This section presents our computational results for SMPTSP benchmark instances introduced in the previous section. The results are juxtaposed against the results of all other recent SMPTSP solution methods known to us. Recall that we solve the instances as GTSGP instances and that SMPTSP is a considerably simplified version of GTSGP. As a result, we are not only optimizing the minimum number of shifts, but also at the same time generating as versatile shifts as possible. That is to say, we generate the SMPTSP results as a by-product when solving a more complicated General Task-based Shift Generation Problem. Due to the goal of this paper, we only present our SMPTSP results here, and not the actual GTSGP results. We used the same version of PEAST that was used when solving the GTSGP instances in [10]. We used no domain-specific knowledge to generate better solutions, nor did we do any parameter fine-tuning. In fact, we used the same version that is in commercial use for staff rostering and sports scheduling. The parameter values have been verified in several earlier implementations and applications of PEAST, see, e.g., the very detailed experiments in [20].

Table 4 shows the summary of the published results of the eight solution methods described in the previous section. Three of the solution methods are purely heuristic methods: K10 [8], LY14 [12], and H18 [16]. Our results are shown in column NK20. The five other methods use commercial MILP solvers as part of the solution process: S14 [19], FL13 [14], B15 [15], R18 [17], and C20 [18]. Table 5 shows the detailed results.

Chirayil Chandrasekharan et al. [18] noted that FL instances were unavailable when S14 method was published and that a summary of the S14 results was obtained upon request. The authors provided us the instance-specific results of the S14 method for FL instances.

```
Set iteration limit t, population size n,
    elitism interval e, shuffling interval s and ADAGEN update interval a
Generate a random initial population of solutions Si for 1 <= i <= n
Set best_solution = null, iteration = 1
WHILE iteration ≤ t
    pop = 1
    WHILE pop <= n
        Apply ejection chain search to solution S_pop to get a new solution
        IF Cost (S_pop) < Cost (best_solution) THEN Set best_solution = 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 solution with the best one
    Set iteration = iteration + 1
END WHILE
Output best_solution
```

Figure 4: The pseudocode of the PEASTP metaheuristic.

Table 4: The summary of the nine solution methods.

	#	K10	LY14	S14	FL13	B15	H18	R18	C20	NK20
KEB	14	14	5	0	0	2	0	1	1	0
SWMB	10	–	–	5	9	4	10	5	0	1
FL	23	–	–	2*	16	–	15	–	4	2
Total solved		0/14	5/14	40/47	22/47	16/24	22/47	18/24	42/47	44/47

*The authors provided the detailed results. The values on the colored cells denote the number of instances that were not solved to optimality (or to the lower bound value if applicable).

As was explained earlier, we run our metaheuristic using as much computation time as practicably possible. For practical GTSGP applications, this could be several hours depending on the length of the planning period and on the number of processors and cores available for the computation. We ran our metaheuristic for eight hours for each of the benchmark instances. The calculation was not interrupted when the optimum SMPTSP value was reached since we were solving the GTSGP problem. To achieve the best possible GTSGP solution, we needed to use the entire calculation time for each instance. The test runs were carried out on a standard laptop, Intel Core i7-8550 at 1.8 GHz with 8 GB RAM running Windows 10.

The results show that our three-phase metaheuristic can successfully solve the most challenging SMPTSP benchmark instances. Another observation is that our metaheuristic generates comparable results to the methods using commercial MILP solvers as part of the solution process. We can also state that our metaheuristic produced the best overall results compared to the other methods. The metaheuristic was able to solve 44 of the 47 instances to optimality. For two instances, our solutions might be optimal. Only one instance was such that our solution was inferior.

Recall that we have selected to solve the most challenging instances. It is obvious that the number of tasks and the number of employees 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 for an employee increases. Accordingly, Lin and Ying [12] stated that instances with longer task lengths should be relatively easy to solve. Smet et al. [13] noted that shorter tasks and lower task skill levels make an instance harder to solve.

SWMB instances were generated based on these observations. Krishnamoorthy and Ernst [7] stated that the tightness of an instance should be close to 90% to make it challenging. FL instances were generated so that the maximal number of overlapping tasks does not provide a good lower bound for an instance [14]. Furthermore, the instances have an average more available employees per task than KEB instances, but they use a significantly smaller percentage of these employees in optimal solutions. This should make an instance more challenging. The SWMB instances were the only instance set where our method did not yield the best results out of all compared methods.

The above assumptions are in line with our reasoning given in the previous section. Based on our test runs, we can quite safely state that a low shift skill level, a low %ICH value, and a low @PAIRS value indicate a hard instance. Thus, the bold values in Tables 1–3 indicate a hard instance. Note that the bold values have been selected with respect to the other instances in the same data set.

In general, we argue that FL instances should be quite easy to solve, because quite many employees may carry out several shifts. For our metaheuristic, FL instances are quite easy to solve, excluding the instances FL#5 and FL#89. Our

Table 5: The detailed results of nine solution methods for KEB, SWMB, and FL instances.

KEB	Opt	K10	LY14	S 14	FL13	B15	H18	R18	C20	NK20
9	40	1							1	
11	20	1								
45	60	7								
59	59	9								
75	60	11								
77	160	2								
79	80	13	1							
80	99	8								
89	70	16								
94	80	11								
98	80	10	2							
106	100	13	4			2				
107	100	12	3			3				
108	128	17	4					–		
SWMB	Opt	K10	LY14	S14	FL13	B15	H18	R18	C20	NK20
1	40				7	1	2			
2	40			1	4		4			
3	77				11		1			
4	98				6		2			
5	59						2	–		
6	116				9	1	3			
7	59			2	6		5	–		1
8	79			1	7		5	–		
9	98			1	7	8	3	–		
10	116			2	10	10	4	–		
FL	Opt	K10	LY14	S14	FL13	B15	H18	R18	C20	NK20
5	30			1	1		1		1	1
28	105						1			
29	95				5		1			
31	116				5					
33	132						1			
35	118						1			
39	108						1			
45	144				5					
46	157				5					
54	190				6					
60	173				6					
61	222						1			
62	262						1			
63	203				8		2			
64	140						1			
68	219				8		1			
69	211				9					
77	248				8		1		1	
79	246				3		1			
80	222				2		1			
84	247				10					
89	319			1	6		1		1	1
94	313				8				6	

*The authors provided the detailed results. Green color indicates that the optimum value was reached. Red color denotes the difference to the optimum value (or to the lower bound). White color indicates that results have not been published.

solution for these instances is one above the lower bound value. These solutions were reached very easily. Therefore, we speculate that these two instances might be such that the lower bound value is not the optimum value. Note that the instances were also unsolved by the Chirayil Chandrasekharan et al. method in [18].

The KEB instances are easy to solve. The instances have the highest task skill levels. Furthermore, as is the case for FL

instances, KEB instances have high percentage of the tasks feasibly assigned by the iterated constructive heuristic.

We note that SWMB#7 instance is extremely hard for our metaheuristic. We have no conclusions for this at this time. Table 2 shows that SWMB#2 instance should be at least as hard to solve. However, it is not that hard for our metaheuristic. For SWMB#7 instance, in addition to the standard setup for our metaheuristic, we tried many different setups for iterated construction heuristic, ruin and recreate heuristic, and PEASTP. We even increased the running time. Nonetheless, our run-time measures showed that we are not even close enough to be able to solve the instance. As a final experiment, we tried to use Gurobi to improve our best solution. Unfortunately, we could not find a better solution or prove the solution optimal with seven days of computation time. Table 5 shows that as many as two solution methods have successfully solved the instance. Furthermore, the published results showed that the solutions were generated fairly fast.

Finally, we registered the running time elapsed to reach the first solution to SMPTSP while we were continuing to reach the best possible GTSGP solution within the given computation time. For the KEB instances, the median time to reach the SMPTSP solution was 0.9 minutes. The minimum time was 0.001 minutes, and the maximum time was 9 minutes. For the SWMB instances, the corresponding times were 86, 15, and 207 minutes, and for the FL instances 0.7, 0.1, and 19 minutes. This verifies our earlier discussions that SWMB instances should be the most challenging ones.

4. Conclusions

We presented a three-phase metaheuristic to Shift Minimization Personnel Task Scheduling Problem (SMPTSP). The metaheuristic actually solves a more complicated General Task-based Shift Generation Problem (GTSGP). The results for SMPTSP were generated as a by-product when solving GTSGP.

In the first phase of the method, we generated a population of initial solutions by using a very fast iterated constructive heuristic. In the second phase, a ruin and recreate heuristic was used to improve the solutions. Finally, the parallelized PEAST metaheuristic used the population of solutions to generate a final solution. In all the previous adoptions of PEAST, we have used random initial solutions. However, PEAST alone is too slow for the largest practical instances of GTSGP. The size of these instances is equal to the size of the largest SMPTSP instances solved in this paper.

The computational complexity of GTSGP and SMPTSP instances depends mainly on the number of tasks, the number of employees, and especially the average number of tasks per shift. These values set the limit for practical use of the PEAST method. Recall that we have only a couple of hours to generate solutions. Our test runs have shown that we cannot use PEAST when we have more than thousand tasks, especially when we have several hundred of employees or the average number of tasks per shift approaches ten. In these cases, we should only use ICH and RRH heuristics. However, the solutions could still be acceptable for practical applications, since after the shift generation and staff rostering have completed, new tasks will most certainly arise and some of the tasks need to be changed or removed.

We solved each instance using eight hours computation time on a standard laptop. Our test runs have shown that for practical instances on a high-computing environment, this corresponds to two hours. This requires using high-performance computers with high number of processors and cores and with very fast memory. Two hours is acceptable considering the entire workforce scheduling process and the release time of the final rosters. However, the running time is significantly higher than the running times of comparable methods.

We showed that the presented three-phase metaheuristic can successfully solve the most challenging SMPTSP benchmark instances. The metaheuristic produced the best overall results compared to the previously published methods. Furthermore, the metaheuristic generated comparable results to the methods using commercial MILP solvers as part of the solution process. The comparison is not entirely fair, as our method takes more time but also solves a more complex problem.

The metaheuristic was able to solve 44 of the 47 instances to optimality. For two instances, our solutions might be optimal. Only one instance was such that our solution was inferior. We are still working on finding out whether the two remaining FL instances can be solved to the lower bound. We also continue to examine why one of the SWMB instances is extremely difficult for our metaheuristic. Determining the cause should help us improve the generality of our solution method. We might encounter similar results on some of the easy instances, e.g., the ones not included in our experiments.

In the near future, we will publish a fourth data set for SMPTSP. We will also present an extension of SMPTSP.

Data Availability

The data for the SMPTSP instances are available online. The three data sets can be found in [23] (T. Lapègue, "Personnel Task Scheduling Problem Library" (online)), available at https://sites.google.com/site/ptsplib/smptsp/instances (last access 15 January 2021).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] K. Nurmi and J. Kyngäs, "The core staff rostering problem," in *IAENG Transactions on Engineering Sciences*, pp. 390–403, World Scientific Publishing Company, Singapore, 2016.

[2] J. Van den Bergh, J. Beliën, P. De Bruecker, E. Demeulemeester, and L. De Boeck, "Personnel scheduling: a literature review," *European Journal of Operational Research*, vol. 226, no. 3, pp. 367–385, 2013.

[3] N. Musliu, A. Schaerf, and W. Slany, "Local search for shift design," *European Journal of Operational Research*, vol. 153, no. 1, pp. 51–64, 2004.

[4] L. Di Gaspero, J. Gärtner, G. Kortsarz, N. Musliu, A. Schaerf, and W. Slany, "The minimum shift design problem," *Annals of Operations Research*, vol. 155, no. 1, pp. 79–105, 2007.

[5] D. Dowling, M. Krishnamoorthy, H. Mackenzie, and D. Sier, "Staff rostering at a large international airport," *Annals of Operations Research*, vol. 72, pp. 125–147, 1997.

[6] V. Valls, A. Pérez, and S. Quintanilla, "A graph colouring model for assigning a heterogeneous workforce to a given schedule," *European Journal of Operational Research*, vol. 90, no. 2, pp. 285–302, 1996.

[7] M. Krishnamoorthy and A. T. Ernst, "The personnel task scheduling problem," in *Optimization Methods and Applications*, X. Yang, K. L. Teo, and L. Caccetta, Eds., vol. 52, pp. 343–368, Springer, Boston, MA, USA, 2001.

[8] M. Krishnamoorthy, A. T. Ernst, and D. Baatar, "Algorithms for large scale shift minimisation personnel task scheduling problems," *European Journal of Operational Research*, vol. 219, no. 1, pp. 34–48, 2012.

[9] K. Nurmi, N. Kyngäs, and J. Kyngäs, "Workforce optimization: the general task-based shift generation problem," *International Journal of Applied Mathematics*, vol. 49, no. 4, 2019.

[10] N. Kyngäs, K. Nurmi, and D. Goossens, "The general task-based shift generation problem: formulation and benchmarks," in *Proceedings of the 9th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA)*, Ningbo, China, December 2019.

[11] L. G. Kroon, M. Salomon, and L. N. Van Wassenhove, "Exact and approximation algorithms for the operational fixed interval scheduling problem," *European Journal of Operational Research*, vol. 82, no. 1, pp. 190–205, 1995.

[12] S.-W. Lin and K.-C. Ying, "Minimizing shifts for personnel task scheduling problems: a three-phase algorithm," *European Journal of Operational Research*, vol. 237, no. 1, pp. 323–334, 2014.

[13] P. Smet, T. Wauters, M. Mihaylov, and G. Vanden Berghe, "The shift minimisation personnel task scheduling problem: a new hybrid approach and computational insights," *Omega*, vol. 46, pp. 64–73, 2014.

[14] J.-G. Fages and T. Lapègue, "Filtering AtMostNValue with difference constraints: application to the shift minimisation personnel task scheduling problem," in *Principles and Practice of Constraint Programming*, C. Schulte, Ed., vol. 8124, pp. 63–79, Springer, Berlin, Germany, 2013.

[15] D. Baatar, M. Krishnamoorthy, and A. T. Ernst, "A triplet-based exact method for the shift minimisation personnel task scheduling problem," in *Algorithms—ESA*, N. Bansal and I. Finocchi, Eds., vol. 9294, pp. 59–70, Springer, Berlin, Germany, 2015.

[16] M. Hojati, "A greedy heuristic for shift minimization personnel task scheduling problem," *Computers & Operations Research*, vol. 100, pp. 66–76, 2018.

[17] D. Niraj Ramesh, M. Krishnamoorthy, and A. T. Ernst, "Efficient models, formulations and algorithms for some variants of fixed interval scheduling problems," in *Data and Decision Sciences in Action*, R. Sarker, H. A. Abbass, S. Dunstall et al., Eds., pp. 43–69, Springer International Publishing, Cham, Switzerland, 2018.

[18] R. Chirayil Chandrasekharan, P. Smet, and T. Wauters, "An automatic constructive matheuristic for the shift minimization personnel task scheduling problem," *Journal of Heuristics*, 2020.

[19] O. Solyali, "The shift minimization personnel task scheduling problem: an effective lower bounding procedure," *Hacettepe Üniversitesi İktisadi ve İdari Bilimler Fakültesi Dergisi*, vol. 34, no. 2, 2016.

[20] N. Kyngäs, K. Nurmi, and J. Kyngäs, "Crucial components of the PEAST algorithm in solving real-world scheduling problems," in *Lecture Notes on Software Engineering*, pp. 230–236, World Scientific, Singapore, 2013.

[21] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[22] J. Christiaens and G. Vanden Berghe, "Slack induction by string removals for vehicle routing problems," *Transportation Science*, vol. 914, 2019.

[23] T. Lapègue, "Personnel task scheduling problem library," 2021, https://sites.google.com/site/ptsplib/smptsp/instances.