# Algorithms for Large Scale Shift Minimisation Personnel Task Scheduling Problems

M. Krishnamoorthy[a,1], A. T. Ernst[b,2], D. Baatar[c,3]

[a] *IITB-Monash Research Academy, IIT Bombay, Powai, Mumbai 400076, India,*
*and*
*Department of Mechanical Engineering, Monash University, Clayton, VIC 3800, Australia*

[b] *CSIRO Mathematics, Informatics and Statistics, Clayton, VIC 3168, Australia.*

[c] *Department of Mechanical Engineering, Monash University, Clayton, VIC 3800, Australia.*

**Abstract**

In this paper we introduce the Personnel Task Scheduling Problem (PTSP) and provide solution algorithms for a variant of this problem known as the Shift Minimisation Personnel Task Scheduling Problem (SMPTSP). The PTSP is a problem in which a set of tasks with fixed start and finish times have to be allocated to a heterogeneous workforce. Personnel work in shifts with fixed start and end times and have skills that enable them to perform some, but not all tasks. In other words, some personnel are qualified to only perform a subset of all tasks. The objective is to minimise the overall cost of personnel required to perform the given set of tasks. In this paper we introduce a special case in which the only cost incurred is due to the number of personnel (shifts) that are used. This variant of the PTSP is referred to as the Shift Minimisation Personnel Task Scheduling Problem (SMPTSP). While our motivation is a real-life personnel task scheduling problem, the formulation may also be applied to machine shop scheduling. We review the existing literature, provide mathematical formulations, and develop a heuristic approach for the SMPTSP.

*Keywords:* scheduling, lagrangean relaxation, task scheduling, machine scheduling, personnel scheduling

## 1. Introduction to the PTSP

In this paper, we provide an approach for solving a variant of the personnel task scheduling problem (PTSP) called the shift minimization personnel task scheduling problem (SMPTSP).

The PTSP is a common problem that occurs in many practical instances. Production managers in a production facility face this problem while scheduling plant operations. In situations where specific tasks have already been assigned to machines over a specified longer planning horizon, decisions have to be made on a day-to-day basis on the number of machines that are required and the specific machine to which each task is assigned. This problem can be formulated as an instance of the PTSP. This problem is also faced in roster implementation situations in operations that are known as day-of-operations. This problem is faced by shift supervisors in large service operations.

Consider a situation where a rosterer knows the start and end times of personnel rostered on a particular day. These start-and-end times are developed by the planning roster. In deriving these start and end times, the planning roster accumulates and aggregates all tasks and makes assumptions on shift

[1] mohank@iitbmonash.org (Corresponding author)
[2] Andreas.Ernst@cmis.csiro.au
[3] davaatseren.baatar@eng.monash.edu.au

start/end times as well as qualifications so as to simplify the complexity of the planning problem, which is quite complex already. In an operational day-to-day scenario, the personnel task scheduling problem is one of optimally allocating a specified set of tasks to the available/rostered personnel. The rosterer (or the shift supervisor) needs to allocate each individual task, with specified start and end times, to available staff who have the *qualifications* or *skills* to perform the task. Staff availability is indicated by personnel shifts (with specific start and end times, including overtime registrations).

This problem was originally derived from an airline application. A rostering solution was developed for ground staff at a major international airport (see Dowling *et. al.* [4]). In [4], a human resource planning and scheduling system is described. This is used to derive rosters for about 500 staff over a monthly planning horizon. Two recent reviews of the rostering literature are in Ernst *et. al.* [5] and Ernst *et. al.* [6].

In the first phase, a master-roster (or, a planning-roster) is created that satisfies all the union work constraints/regulations, enterprise bargaining agreements and workforce requirements. In the second phase, we performed micro-planning on a day-to-day basis.

Given the staff that are rostered on a particular day, the PTSP is to assign to each of staff the specific tasks that they will carry out (each with a specific start and end time and each with specific skill requirements).

We consider below the solution to an example problem with 10 shifts, each starting and finishing at different times. This is the same example that was also depicted in Krishnamoorthy *et. al.* [14]. The shifts are packed with tasks or jobs, each with a specific start time, end time and skill requirements. A possible solution to the PTSP may be represented as shown in Figure 1 below.

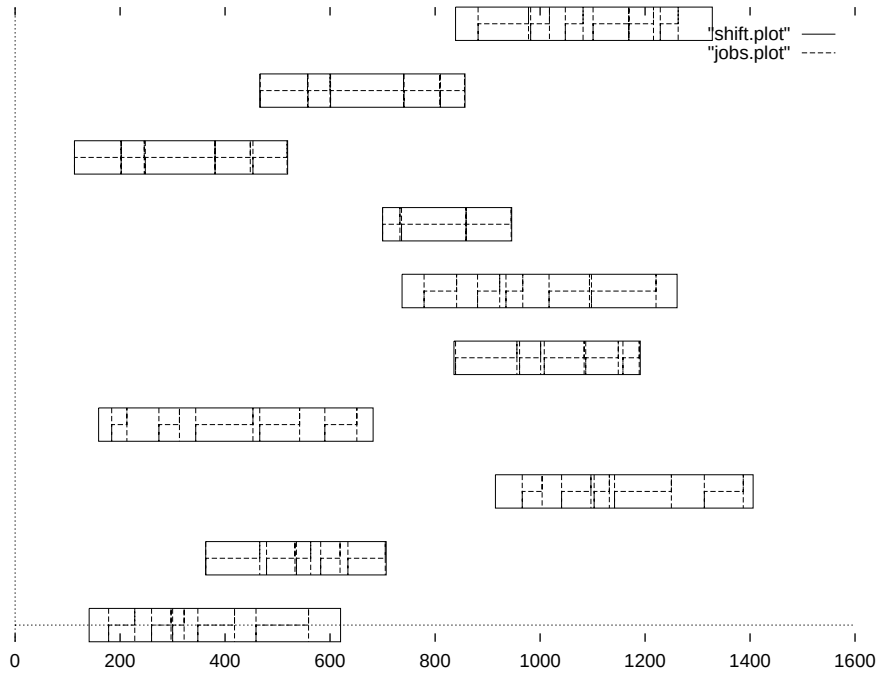


Figure 1: Example PTSP solution representation

The PTSP is a day-to-day operational problem. In Krishnamoorthy *et. al.* [14], we provide a detailed description of the PTSP and include in it a literature review, classification, taxonomy and an introduction to many variants of the basic PTSP.

In this paper we will concentrate mainly on a variant of the PTSP in which the number of personnel (shifts) required is to be minimised. In other words given a set of jobs to be performed, each with a start- and end-time, and a set of staff/machine/shift types with corresponding qualifications, we are interested in finding a solution that minimises the total number of staff, with no restrictions on the number from

each category to be used. We refer to this variant of the PTSP as the *Shift Minimisation PTSP* (SMPTSP). This variant of the PTSP is important both as a strategic planning tool to determine the number and mix of staff that an organisation should have. It is also useful as a day-to-day operational management tool.

We first define the problem of interest to this paper in Section 2. In this section, we provide some basic notations, describe some of the principal characteristics of the problem and also provide a formulation of SMPTSP. We describe some properties of the SMPTSP in Section 2.4. We describe our heuristic approach for solving the SMPTSP in Section 2. Our approach involves the development of a Volume Algorithm, which is described in 3.1, followed by the Wedelin Algorithm, described in 3.2. We test our approach on two sets of randomly generated data. These data, our detailed computational analysis and discussions on the computational performance of our approaches are provided in Section 4.

## 2. The Shift Minimisation Personnel Task Scheduling Problem (SMPTSP)

There are many variants of the PTSP depending on the specific model and objective function that is chosen. We may choose to minimise overtime costs only. We may choose to minimise the total number of staff that are used or the fixed cost incurred by the number of staff that are used.

In the shift minimisation personnel task scheduling problem (SMPTSP) our objective is to minimise the total number of personnel that are used. This variant of the PTSP is particularly useful in situations where there is a large pool of casual staff that are available and management would like to minimise the number of staff that are needed from this pool on a particular day in such a way that all tasks for that day are performed adequately. Staff that are available can only work between a known start time and a known end time, as determined by the *master roster* or the *planning roster* or through information contained in the the casual staff register.

### *2.1. Notations*

We repeat here the notations that have been provided already in Krishnamoorthy *et. al.* [14]. Let us define the following parameters:

| | |
|---|---|
| $J$ | The set of tasks that need to be assigned, $J = \{1, \ldots, n\}$. |
| $W$ | The set of workers that can perform tasks, $W = \{1, \ldots, m\}$. |
| $\bar{W}$ | A subset of $W$ that represents staff who are willing to work overtime (note that this also includes casual staff who are deemed to be completely on overtime). |
| $c_{jw}$ | Cost if task $j \in J$ is performed by worker $w \in W$. |
| $b_w$ | The fixed cost of using worker $w \in W$ |
| $s_j$ | The start time of task $j \in J$. |
| $f_j$ | The finish time of task $j \in J$ ($f_j > s_j$). |
| $S_w$ | The start time of shift $w \in W$. |
| $F_w$ | The finish time of shift $w \in W$ ($F_w > S_w$). |
| $\bar{S}_w$ | The start time of overtime for shift $w \in \bar{W}$. |
| $\bar{F}_w$ | The finish time of overtime for shift $w \in W$. |
| $P_j$ | The set of personnel that can perform task $j \in J$ ($P_j \subseteq W$). |
| $T_w$ | The set of tasks that worker $w \in W$ can perform ($T_w \subseteq J$). |

Please note also that the above list of all notations for PTSPs are included here for completeness, in the sense that although they are used for the PTSP, not all of them are used in describing the SMPTSP variant that we are about to describ in greater detail in this paper.

Shifts in the above definitions, may also represent machines or processors. Note that either $\bar{S}_w = S_w$ or $\bar{F}_w = F_w$. Also for casual staff $\bar{S}_w = S_w$ *and* $\bar{F}_w = F_w$. Note that $c_{jw}$ will, in general, be zero or

comparatively small for $w \in W \setminus \bar{W}$. This is because the full time staff, who are rostered to be present on a particular day, will be paid irrespective of what they do. However a cost $c_{jw} > 0$ is incurred to bring in casual or overtime staff to perform task $j$. In practical situations $\bar{W} \subset W$ could represent the set of casual staff (or staff on overtime) that are used on an as-required basis to complete the excess work on a given day.

As indicated in Krishnamoorthy *et. al.* [14], the sets $P_j$ and $T_w$ are defined through both, time window constraints as well as skill considerations. For example $w \in P_j$ if $[s_j, f_j] \subseteq [S_w, F_w]$ and if worker $w$ is qualified to perform task $j$. The sets $T_w$ can be derived from the sets $P_j$ through the relation $T_w = \{j \mid w \in P_j\}$. Note that in the above description we do not take into account lunch breaks with flexible start and finish times. We can use $b_w$, the fixed cost incurred for using worker $w$, to specify that some of the workers are more expensive than others.

Let us now define an interval graph, $G = (J, A)$, where $J$ is the node set, one for each task in $J$ and $A$ is the arc set. In this interval graph, we join two nodes $j, k \in J$ by an arc if the intervals $[s_j, f_j]$ and $[s_k, f_k]$ overlap with each other. In other words, both tasks are on at the same time and in that sense, *conflict* with each other. Let $C$ be the set of *maximal cliques* in this conflict graph. That is $C = \{K_1, \ldots, K_p\}$ consists of sets $K_t \subseteq J$ such that any two tasks in $K_t$ overlap for some interval of time and $K_t$ is maximal. No task in $J \setminus K_t$ will overlap with any one of the tasks in $K_t$. Finding all maximal cliques in an interval graph is easy. We have devised a polynomial time algorithm for identifying all maximum cliques. See Krishnamoorthy *et. al.* [14] for more details, including an example of how this might be used to define a possible approach for solving PTSPs. We provide the maximal clique algorithm here below for completeness.

**Algorithm 1** (Finding maximal cliques)**.**

---

**Sort jobs***:* lexicographically *in ascending order of* $s_j$ *and* $f_j$

**Initialize***:* $p = 0$, $C = \emptyset$,
$s = \min_{j \in J} s_j$, ***stop****=0;*

**while** *stop=false* **do**
$f = \min_{j: s \leq f_j} f_j$
$p = p + 1$
$K_p = \{j \in J \mid s_j \leq f \leq f_j\}$
$C = C \bigcup \{K_p\}$
**if** $\exists s_j > f$ **then** $s = \min_{s_j > f} s_j$
**else** ***stop****=true*

**end while**

---

Further, for each $w \in W$, let $C^w$ be the set of maximal cliques in the subgraph of $G$ induced by $T_w$. The elements of $C^w$ will be denoted by $K_t^w$.

We illustrate these sets with a small example below, again noting that this has also been provided in Krishnamoorthy *et. al.* [14].

---

**Example 1.** *Assume a set of tasks with starting and ending times as shown in Figure 2.*

The resulting interval graph is as shown in Figure 3.

The cliques are: $C = \{\{1, 2, 3\}\ \{2, 3, 4\}\ \{8, 9\}\ \{4, 5, 6, 8\}\ \{5, 6, 7, 8\}\}$. In other words, it is not possible for a single worker to perform tasks 1, 2 and 3 simultaneously. Similarly, it is not possible for a single

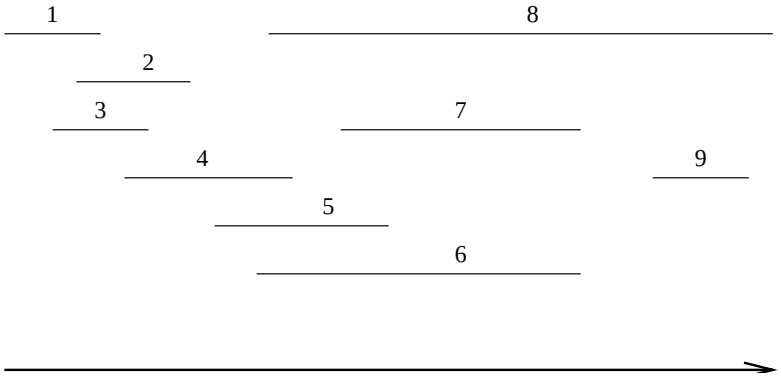


Figure 2: Set of tasks (See also [14]

worker to perform tasks 5, 6, 7, and 8 simultaneously. Let us assume that, for a particular worker $w$, we have $T_w = \{2, 4, 6, 8\}$. In other words, worker $w$ in our example, can only perform tasks 2, 4, 6 and 8. Then, we can define $C^w = \{\{2, 4\}\ \{4, 6, 8\}\}$.

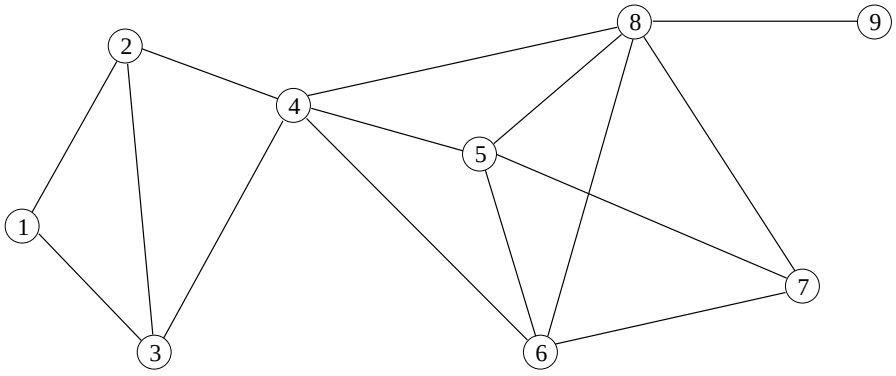


Figure 3: Interval graph (See also Krishnamoorthy *et. al.* [14])

---

*2.2. Related Problems*

A similar problem is referred to in the literature as the *fixed job schedule problem* or a *heterogenous workforce scheduling problem*. The SMPTSP follows the model in Valls *et. al.* [17], where a heterogenous workforce assignment problem is studied. They minimise the number of workers required to perform a machine load plan.

Different variants of SMPTSP have been given in the literature. If all personnel/machines are identical and all tasks require the same type of machines (or if all workers have the same skills and all tasks require the same skills from the workers) and if we were to minimise the number of machines that are used, the resulting problem is a *fixed job schedule problem* (FJSP). This problem has been studied by Gertsbakh *et. al.* [10] and by Fischetti *et. al.* [9]. The FJSP has been solved in $O(n \log n)$ time by Gupta *et. al.* [12] and also by Nakajima *et. al.* [16]. Arkin *et. al.* [1] provide an $O(n^2 \log n)$ algorithm for a problem similar to the fixed job schedule problem except that they use a cost minimisation objective with some value attached to each job. In other words they study a problem in which each job has a fixed start and end time and a value. They then minimise the value of the jobs left uncompleted on identical machines. Arkin *et. al.* [1] show that their version of the fixed job schedule problem is NP-complete if each job has, associated with it, a subset of machines on which it can be processed.

Fischetti *et. al.* [7] solve the FJSP with spread time constraints. In this version of the problem, we have $n$ tasks each with a start and end time $(s_j,\ f_j)$. We are also given an unlimited number of identical processors that are always available ($S_w = -\infty$ and $F_w = \infty$). The objective is to minimise the number of machines used (as in the SMPTSP). The constraint is that difference between the end of the last job

and the start of the first job on every machine does not exceed a *max spread time*. Any approach for SMPTSP will solve this variant of the fixed job schedule problem, but with a modification to the way we process the input data. Fischetti *et. al.* [7] show that this problem is NP-hard.

Fischetti *et. al.* [8] also introduce the fixed job schedule problem with *working time constraints*, where we have $n$ tasks with start and end times and an unlimited number of identical processors ($S_w = -\infty$ and $F_w = \infty$). The objective is to complete all tasks but with the minimum number of processors, such that no processor works, in total, for more than a given working time $T$. In other words, it is necessary to spread the work load. If we let $t_j = f_j - s_j$, then we can use SMPTSP and an additional working time constraint for each identical shift to derive the above problem. Since $S_w$ and $F_w$ are finite in our formulation, the working time for every processor/person is automatically constrained. However, through the use of a working time constraint we may be able to achieve work load balancing in applications where there are relatively few jobs.

Kroon *et. al.* [15] introduced a problem which is called the tactical fixed interval scheduling problem (TFISP) which is perhaps the closest related problem to SMPTSP. They determine the minimum number of parallel nonidentical machines that can process a set of jobs (each belonging to a job class and each with a fixed start and finish time) in a non-preemptive manner. Each machine can only perform tasks from a prespecified subset of job classes and can process only one job at a time. Problem TFISP is almost identical to SMPTSP, except that in SMPTSP the machines also have an availability restriction. However, we can model SMPTSP as a TFISP by assuming that all personnel work a fixed duration shift with $S_w$ being equal to the start of the first job that needs to be processed ($\forall w \in W$) and $F_w$ being equal to the end of the last job that needs to be processed ($\forall w \in W$). Further, we can preprocess the input data to exclude from $T_w$ all jobs that have start and/or end times that are outside the *actual* availability time window for worker $w$. Similarly, we can preprocess the input data to exclude from $P_j$ all personnel who have their *actual* shift times outside of the job's completion time window.

A similar problem has been introduced in Gondran *et. al.* [11] (page 476, Exercise 4). This interesting variant exercise is one of identifying the minimum number of aircraft required to fly all the flights in a given planning horizon. We have to assume that all planes are available at all points in time in the planning horizon. In other words, none of the planes require any maintenance or repair and that the turnaround time between successive flights by the same plane is small enough to be ignored. We also ignore geographicap connectivity constraints. Each flight (which is akin to a task in our case) has a start and end time and requires planes (staff, in our example) of a particular type (skill, in our case). Given that the planes (shifts) are available for specific durations, we have to minimise the number of planes (staff). Thus, this problem similar to the SMPTSP we are considering in this paper. This example has been cited in Gondran *et. al.* [11] (page 204–205, Example 23).

Kroon *et. al.* [15] discuss some practical applications of TFISP. They also analyse the computational complexity of TFISP and provide exact and approximate algorithms for its solution.

Kolen *et. al.* [13] provide a comprehensive review of *interval scheduling* problems in which the major distinguishing feature is that apart from the job duration, the starting time of the job has also got to be honoured. This excellent paper provides proofs of results, complexity results and approximability results for variants of interval scheduling problems. The paper also provides some intersting applications of interval scheduling problems including cottage rentals and telecommunications bandwidth allocation. The paper also provides some algorithms for two variants of interval scheduling which are similar to the SMPTSP. In the first, each machine (shift in our example) has a given known interval during which it is available. The second variant is one in which although all machines are continuously available, each task/job has a maximal (or ideal) machine on which it can be processed. The SMPTSP is actually a combination of the two variants that have been considered and moreover, it considers a minimisation of the total number of machines/shifts to be used.

*2.3. Formulation of the SMPTSP*

We define the following decision variables. Let,

$$x_{jw} = \begin{cases} 1 & \text{if task } j \in J \text{ is assigned to worker } w \in W \\ 0 & \text{otherwise} \end{cases}$$

Also, let $y_w = 1$ if worker $w \in W$ is used and zero otherwise. Note that in SMPTSP (and TFISP), the objective is to minimise the number of workers that are required to carry out all the jobs, where preemption is not permitted. We can then define Problem-SMPTSP by the following mathematical model.

**Problem-SMPTSP**

$$Z_{SMPTSP} = \min \quad \textstyle\sum_{w \in W} b_w y_w$$

$$\text{s.t.} \quad \sum_{w \in P_j} x_{jw} = 1, \qquad \forall\ j \in J \tag{1}$$

$$\sum_{j \in K_t^w} x_{jw} \leq y_w, \qquad \forall\ w \in W,\ K_t^w \in C^w \tag{2}$$

$$0 \leq y_w \leq 1 \qquad \forall w \in W \tag{3}$$

$$x_{jw} \in \{0,1\} \qquad \forall\ j \in J,\ w \in W. \tag{4}$$

In the above formulation of SMPTSP, the objective function minimises the number of workers that are used to complete all jobs. Although $b_w$ can be some meaningful number, in our computational experiments, we set $b_w = 1$ for all $w \in W$, thereby merely minimising the number of workers that are used. Constraint (1) ensure that each job is performed by exactly one worker. Constraint (2) ensure that only one of the jobs in each set $K_t^w \in C^w$ is performed by worker each $w \in W$. Moreover, this constraint ensure that a job is performed by $w$ only if that worker is chosen. Finally for TFISP problems, the upper bound in (3) is not present so that $y_w$ may assume arbitrary integer values.

*2.4. Properties of the SMPTSP*

As shown by Kroon *et. al.* [15], SMPTSP is NP-Hard problem in the strong sense, even if preemption of tasks is allowed.

If we take away the qualification constraints in SMPTSP, the problem is easy and can be solved by a polynomial time algorithm. All we need is a forward pass maximal clique algorithm on an interval graph which can be solved by Algorithm 2 in polynomial time.

**Algorithm 2** (Forward Pass Maximal Cliques Algorithm)**.**

---

**Sort jobs:** *in ascending order of* $s_j$
**Initialize:** *the set of shifts used in the solution*
$\mathcal{S} = \emptyset$
**for all** $j \in N$ *do*
**if** *job* $j$ *can be performed by some shift* $w \in \mathcal{S}$
**then** *Allocate* $j$ *to shift* $w$
**else** *Add a new shift* $w = \{j\}$ *to* $\mathcal{S}$
**next** $j$

---

Note that it is easy to check if a job can be performed by a shift $w$ if we keep track of the latest completion time of any of the jobs in $w$ whenever a new job is added. This means that the main loop in Algorithm 2 requires $\mathcal{O}(n)$ operations. The most computationally expensive part of the algorithm is sorting the jobs. The above algorithm automatically calculates a lower bound on SMPTSP. Given that the qualification constraints are relaxed, this lower bound may not be very useful. However we use the algorithm in our heuristic.

### *2.5. Packing Jobs into Shifts*

Suppose we are given a shift $w \in W$ and a profit $p_j$ for each job $j$ that could be performed by $w$. An interesting subproblem of SMPTSP is one of identifying the set of jobs that should be packed into $w$ so as to maximise profit. This subproblem arises in several of the algorithms descibed in this paper. We refer to this subproblem as the shift packing problem and notate it as $Pack(w, \pi)$.

$$\begin{aligned} \max \quad & \sum_{j \in J} \pi_j x_{jw} \\ \text{s.t.} \quad & \sum_{j \in K_t^w} x_{jw} \leq 1, \qquad \forall\ K_t^w \in C^w \\ & x_{jw} \in \{0,1\} \qquad\qquad \forall\ j \in J. \end{aligned}$$

This problem can be solved in polynomial time using a simple algorithm based on shortest paths.

---

**Example 2.** *Consider the worker $w$ from the previous example, we have $T_w = \{2, 4, 6, 8\}$.*

*The set of tasks can be performed by the worker is depicted along the time horizon in the Figure 4. The*

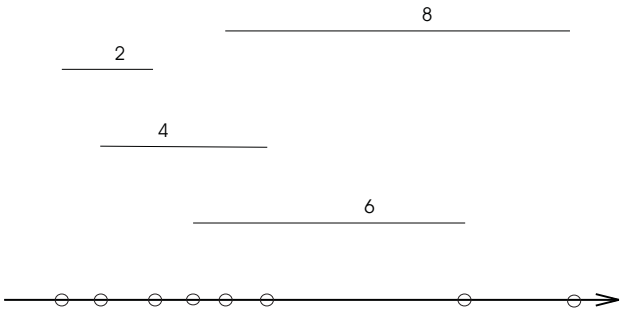


Figure 4: Tasks

Pack$(w, \pi)$ *problem for this worker can be formulated as a shortest path problem in a directed graph as shown in Figure 5, where the first and last nodes represent the source and sink nodes, respectively; and the remaining nodes are in one–to–one correspondence with the start and end times, in ascending order, of the tasks $j \in T^w$. Two kinds of arcs that are defined in this graph.*

- *All nodes are connected with it's neigbours by directed arc in the direction of the time horizon. In the Figure 5, those arcs are depicted by the dashed lines and they all have associated zero cost.*
- *The remaining directed arcs are defined according to the tasks in $T^w$, which start and end at the nodes corresponding to the start and finish time of a task $j \in T^w$, respectively. Those arcs have associated cost $-\pi_j$.*

*The graph is acyclic so we can formulate* Pack$(w, \pi)$ *as a shortest path problem from the source to the sink node.*

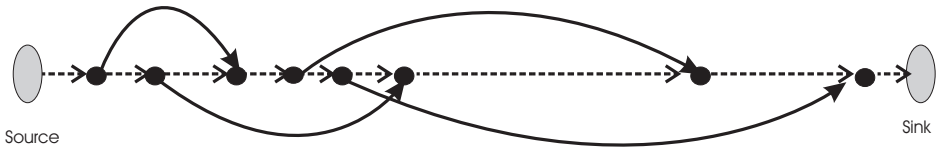


Figure 5: Network Flow Representation

*Note that in the following algorithm we will assume that the cliques $K_t^w \in C^w$ are numbered in chronological order, by the time at which the overlap of the jobs in $K_t^w$ occurs.*

**Algorithm 3** (Pack$(w, \pi)$)**.**

**Initialize:** $\delta_t = 0$ *for each clique* $K_t^w \in C^w$
*The optimal set of jobs* $\mathcal{J} = \emptyset$
**for** $t = 1$ **to** $|C^w|$ **do**
**for all** $j \in K_t^w$ *not already examined* **do**
*Let* $\bar{t}$ *be the last clique* $K_t$ *s.t.* $j \in K_{\bar{t}}^w$
**if** $\delta_t + \pi_j > \delta_{\bar{t}}$ **then**
$\delta_{\bar{t}} = \delta_t + \pi_j$
$pred_{\bar{t}} = t - 1$
$path_{\bar{t}} = j$
**next** $j$
**if** $\delta_t > \delta_{t+1}$ **then**
$\delta_{t+1} = \delta_t$
$pred_{t+1} = t$
$path_{t+1} = \emptyset$
**next** $t$
**while** $t > 0$ **do**
$\mathcal{J} = \mathcal{J} \cup path_t$
$t = pred_t$
**while end**
**Output** *The optimal set of jobs* $\mathcal{J}$ *with maximum profit* $\delta_{|C^w|}$

Algorithm 3 can be thought of as a shortest path algorithm in a network consisting of $|C^w| + 1$ nodes. Appart from the trivial arcs with zero cost from each node $t$ to node $t+1$, there is one arc for each job $j$ going from node $\underline{t} = \min\{t : j \in K_t\} - 1$, to node $\bar{t} = \max\{t : t \in K_t\}$.

## 3. A Lagrangean Approach for Solving the SMPTSP

In this section, we describe a new Lagrangean approach used for solving the SMPTSP.

We first employ the Volume Algorithm (VA) on the LP relaxation of SMPTSP in which constraint (1) is also relaxed. This gives us a lower bound on the LP relaxation of SMPTSP that is, in most cases, quite close to the LP-relaxation solution of the SMPTSP. Of course, this solution is not feasible with respect to the SMPTSP. We use the VA because it produces an improved lower bound, particularly for large-sized problems, in a reasonable amount of computing time. We then use the solution produced by the VA as the starting point for the application of the Wedelin's Algorithm (WA) to derive an upper bound and a feasible solution to the SMPTSP.

We adopt this approach because real-life instances of the SMPTSP are very large and involve many workers/shifts and many tasks/jobs. In such large-scale real-life instances even the LP relaxation does not solve within a half an hour of CPU time. Thus it becomes imperative for us to devise an approach that is close enough to the optimal solution but is derived in a reasonable amount of computing time.

Before we developed a special-purpose heuristic approach, which we will discuss in the rest of this section, we did try and implement a general purpose simulated annealing approach. However, we found that the overalll quality of solutions found was generally poor.

In the following part we first describe the VA for the SMPTSP and then the WA approach.

### *3.1. Volume Algorithm for the SMPTSP*

Real-life instances of SMPTSP are very large and involve many workers and shifts and also many tasks and jobs. For such instances solving the LP relaxation of SMPTSP is quite time consuming. Therefore, in order to find approximate solution we use Volume Algorithm (VA) developed by Brahona *et. al.* [2]. In this work, they modified the traditional subgradient algorithm to produce approximate primal and dual solutions while maintaining low computational costs. This method is based on a new duality theorem developed by the authors. As the authors suggested this method can be seen as a fast way of an approximation of Dantzig Wolfe decomposition. Let us consider a linear program

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & Dx = e \\ & x \geq 0 \end{aligned}$$

and its Lagrangean relaxaion (LR)

$$\begin{aligned} z = \min \quad & c^T x - \pi^T (Ax - b) \\ \text{s.t.} \quad & Dx = e \\ & x \geq 0 \end{aligned}$$

Then the Volume algorithm works approximately in the following way. Note that there are some implementational details that have not included here for simplicity of presentation. The exact algorithm used is the implementation publicly available in the Coin-OR library[4], parameter choices are discussed in Section 4.2.1.

**Algorithm 4** (Volume Algorithm)**.**

| | |
|---|---|
| **Initialize** | *Lagrangean multipliers* $\bar{\pi}$ |
| | *upper bound UB* |
| | *iteration number* $t = 1$ |
| | $\beta \in (0, 2)$ |
| | $\alpha \in (0, 1)$ |
| **Step 0** | **Solve LR**, *obtain* $\bar{x}$ *and* $\bar{z}$ |
| | set $x^0 = \bar{x}$ *and* $z^0 = \bar{z}$ |
| **Step 1** | *compute subgradient* $v^t = b - A\bar{x}$ |
| | *update step length* $s = \beta \frac{UB - \bar{z}}{\|v^t\|^2}$ |
| | *update* $\pi^t = \bar{\pi} + sv^t$ |
| **Step 2** | **Solve LR** *find* $x^t$ *and* $z^t$ |
| | *update* $\bar{x} = \alpha x^t + (1 - \alpha)\bar{x}$ |
| **Step 3** | **if** $z^t > \bar{z}$ **then** |
| | $\bar{\pi} = \pi^t$ *and* $\bar{z} = z^t$ |
| | **endif** |
| | $t = t + 1$ |
| | **Go to** *Step 1* |

[4] http://www.coin-or.org

The algorithm terminates if both $\|v\|$ and $|c\bar{x} - \bar{z}|$ are below a certain threshold. Here the main differences from the traditional subgradient method are that only uphill moves are accepted and the update of $\bar{x}$ in Step 2. The convex combination with the previous value of the violation (step direction) serves to stabilize the subgradient method. The approximate solution $\bar{x}$ can be considered as a convex combination of the vectors produced by solving the LR in each iteration of the above algorithm. The approximation is based on the duality theorem mentioned before.

In this method, the main factor that affects low computation time is solving the Lagrangean Relaxation (LR) efficiently. For the SMPTSP, by dualizing the constraints (1) we get the LR problem which can be solved efficiently by solving independent shift packing problems.

### *3.2. Wedelin's Algorithm*

Here we describe the use of the generalized Wedelin algorithm [3], which was originally developed by Wedelin [18] for a class of large scale (0-1) programming problems, where the constraint matrix consists of zero and one entries.

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \in \{0,1\}^n \end{aligned}$$

The advantage of this approach is that it does not depend on the solution of the linear programming relaxation. The basic idea is to consider a Langrangean Relaxation(LR) of the problem where all constraints are moved into the objective function.

$$\begin{aligned} \min \quad & c^T x - \pi^T (Ax - b) \\ \text{s.t.} \quad & x \in \{0,1\}^n \end{aligned}$$

For this unconstrained problem the Lagrangean coefficients are adjusted using a co-orindate ascent method. That is for a violated constraint the associated lagrange multiplier is chosen such that the new problem has an integer solution where the violated constraint is satisfied. For the unconstrained problem it is trival to find the integer solution since the optimal value of the binary variables are defined by the sign of the coefficient of the variable in the objective function. That makes the algorithm extremely fast. However even for a non-optimal value of the lagrangean multiplier there may be no ascent direction along any of the co-ordinates. Furthermore even for the optimal value of the lagrange multipliers the solution to the subproblems may not be feasible with respect to the relaxed constraints. Therefore, Wedelin [18] suggested the use of an approximation algorithm where the basic idea is to apply a *bias* $p_{ij}$ to the Lagrangean multipliers for each variable $i$ in constraint $j$, i.e. $\pi_{ij} = \pi_j + p_{ij}$.

In other words, we have a matrix of Lagrangian multipliers, instead of a vector. This is adjusted in each iteration such that solution of the unconstrained problem satisfies the violated constraints.

For the SMPTSP we move only constraints (1) into the objective function and using the *bias* $p_{jw}$, we get the following relaxation of the SMPTSP:

$$\begin{aligned} \min \quad & \sum_{w \in W} b_w y_w - \sum_{j \in J} \sum_{w \in P_j} \pi_{jw} x_{jw} \\ \text{s.t.} \quad & \sum_{j \in K_t^w} x_{jw} \leq y_w, \qquad \forall\ w \in W,\ K_t^w \in C^w \\ & 0 \leq y_w \leq 1 \qquad \forall w \in W \\ & x_{jw} \in \{0,1\} \qquad \forall\ j \in J,\ w \in W. \end{aligned}$$

and the resulting problem can be decomposed into $|W|$ independent subproblems

$$\begin{aligned} \min \quad & b_w y_w - \sum_{j \in J} \pi_{jw} x_{jw} \\ \text{s.t.} \quad & \sum_{j \in K_t^w} x_{jw} \leq y_w, \qquad \forall\ w \in W,\ K_t^w \in C^w \\ & 0 \leq y_w \leq 1 \\ & x_{jw} \in \{0,1\} \qquad\qquad \forall\ j \in J,\ w \in T_w. \end{aligned}$$

The optimal solution to the subproblems can be determined by solving the corresponding packing problem $Pack(w, \pi_w)$, which can be solved in polynomial time as we mentioned before. Thus, Wedelin's method for the SMPTSP can be interpretated as adjusting workers preference to tasks '*artificially*', such that the desicions made by individual workers for their own benefit does not have conflict with others interest and produces a feasible solution to the original problem. The details of how the $\pi_{jw}$ values are updated are given in Section 4.2.2.

Depending on the value of the $\pi_{jw}$ matrix the relaxation can be the same as the standard Lagrangian relaxation (if $\pi_{jw} = \pi_j\ \forall\ w$). On the other hand clearly any integer feasible solution $\bar{x}$ can be found by solving subproblems with $\pi_{jw} = M\, x_{jw}$ for some sufficiently large constant $M$.

As we mentioned above, Wedelin's method does not depend on any feasible solution to the LP relaxation of the original problem. However, from our experiments, when we use an optimal solution to the LP relaxation as the starting Lagrangean multipliers, we achieve convergence to a feasible solution faster than when we used exact Lagrangean multipliers or employ a trivial start with $\pi_{jw} = 1$.

This can be interpreted as follows: In the optimal solution the fractional values can be thought of as the probability that the jobs are assigned to the shifts, or initial shift preferences.

## 4. Computational Results

In this section we give detailed information about how SMPTSP instances are generated. We then provide implementation details of the heuristic algorithm. We then compare the results produced by CPLEX 11.2 and the heuristic we developed.

### *4.1. Data Sets*

Since this research is motivated by problems in personnel scheduling, we designed a new dataset with characteristics which are motivated by this application. First shifts are created with fixed start and end times.

For the purposes of this study, for any instance of the problem we have assumed that all shifts/workers are available at all times in the planning period. In other words, we have assumed that *all* shifts/workers have the same shift length as the planning period. Thus, all shifts/workers are with a duration of 1440 minutes (or 24 hours). We can do that without loss of generality and without comporomising the resemblance to real-world problems. A 24-hour shift can be seen as equivalent to two 12-hour shifts (or three 8-hour shifts) by workers of the same type/skill.

Once the shifts are created, these are then filled with jobs to a given tightenss $T\%$. The tightness here is defined as the total length of all tasks, as a percentage of the total duration of all shifts. The job lengths are created using a triangular distribution $\text{Tri}(a, b, c)$ parameterised by the minimum $a$, the median $b$, and the maximum $c$. This distribution was chosen because in most application there would be many jobs that take about the same duration with fewer tasks that have extreme durations. The tasks are then uniformly distributed throughout the shifts. In this way a feasible solution is first constructed. Apart from the shift to which it is originally assigned, we assume that each job can also be carried out

by other staff members with a probability $M$ which depends on the degree of multi-skilling of the staff members. Finally, we increased the number of shifts up to 30%, by including randomly generated shifts with the same multi–skilling level.

Since there are a few parameters that determine the method for generating data sets, we have limited ourselves to investigating the following values in this paper:

Number of shifts $|W|$: This is the main parameter of interest as to see how different methods scale as the problem size increases. For real-world problems we would like to solve problems with about 500 staff. However, in this paper we limit ourselves to 22 to 420 workers/shifts.

Tightness: In this paper we only use a tightness level of 90%. This is partly due to the fact that problems with a much lower level of tightness are generally easier to solve and hence less interesting. Furthermore when solving SMPTSP's as part of a general rostering system which generates rosters, the master planner will try to produce shifts with as little idle time as possible in order to optimise the overall roster. Hence SMPTSP sub-problems with a high tightness level occur quite naturally in our experience.

Job lengths: Three different job length distributions are used in this paper:

**L** Long jobs with an even distribution Tri$(200, 300, 400)$.
This leads to relatively easy problems as there are fewer jobs.

**M** Medium length jobs generated using a Tri$(50, 200, 250)$ distribution.
This leads to a collection with a bias for slightly longer jobs but with some small jobs.

**S** Short tasks with only a few longer ones, generated using distribution Tri$(50, 100, 200)$.

Multi-skilling: Just two levels are used in this paper. In the first the each staff member can perform about 2/3 of the jobs on average, in the second each staff member can perform about 1/3.

We randomly generated problem instances gradually increasing the complexity until both exact and heuristic approaches were not able to generate feasible solutions. In this way 137 problem instances were generated with number of workers and jobs in the range $22 \leq |W| \leq 420$ and $40 \leq |J| \leq 2105$; and with tightness 90%. For each problem instance the multi–skilling level and job length distributions are chosen randomly.

The data files that we have generated are numbered/ordered lexicographically by the number of jobs and workers (See Figure 6).

The name of the data file presents the file number, number of workers, number of jobs and the multi–skilling level. For example, Data_10_51_111_66 is the 10th data file which presents a problem instance with 51 workers, 111 jobs and with a multi–skilling level of 66%.

### *4.2. Implementation details*

In this section we present detailed implementation details of the algorithms that we presented in this paper. A heuristic algorithm can be embedded in the Volume Algorithm and run every certain number of iterations to find a feasible solution. Using Wedelin's algorithm in that way provides different starting preferences for the algorithm. However, in our initial tests carried out on a few instances, this appoach worked better for small-sized problems but for large problems it increased the computational time drastically without much improvement in the solution. Therefore, in our computations we did not use any heuristic algorithm inside the Volume Algorithm until we got the lower bound and the corresponding approximate solution.

As we mentioned before in Section 2.4, SMPTSP is $NP - hard$. Thus, there is no guarantee of finding an exact solution – or even a good solution – for large-sized problems without spending a significant

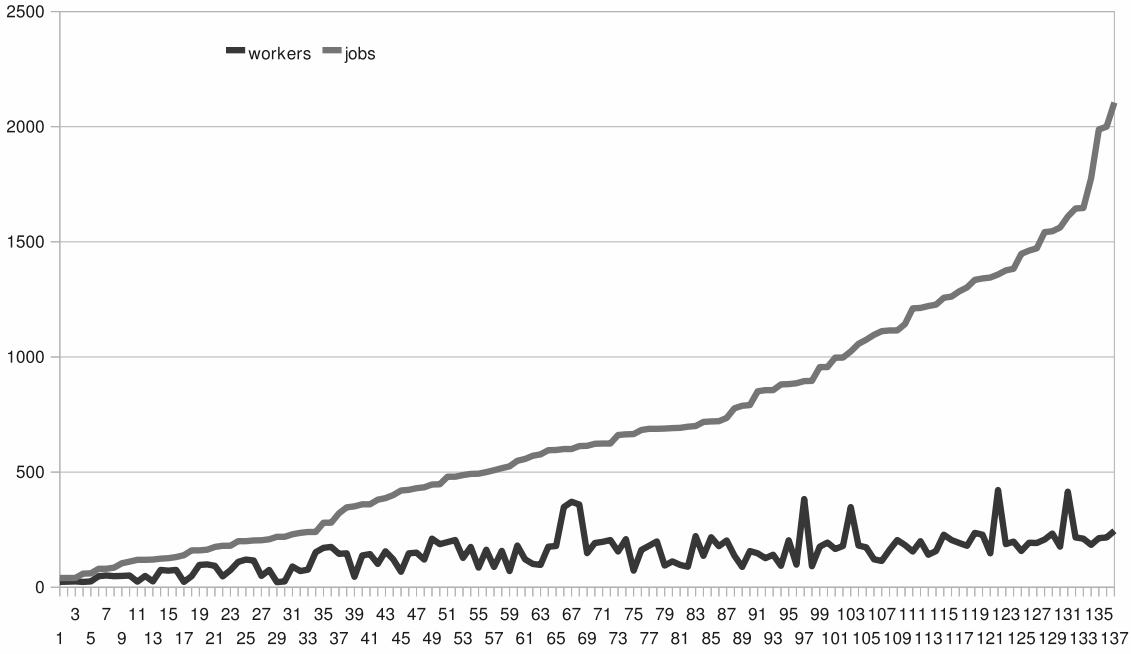


Figure 6: Data set: number of workers and jobs

amount of computational time. Therefore, for our heuristic algorithm, we limited the total computation time to 1800 seconds; and this includes the computational time for the Volume Algorithm and Wedelins Algorithm. For the Volume Algorithm, we did not directly set any time limit. However, we restricted the number of iterations to 6000. In 11 instances the Volume Algorithm was terminated within this limit. The computation time for VA and WA are depicted in Figure 7 for the different data sets.

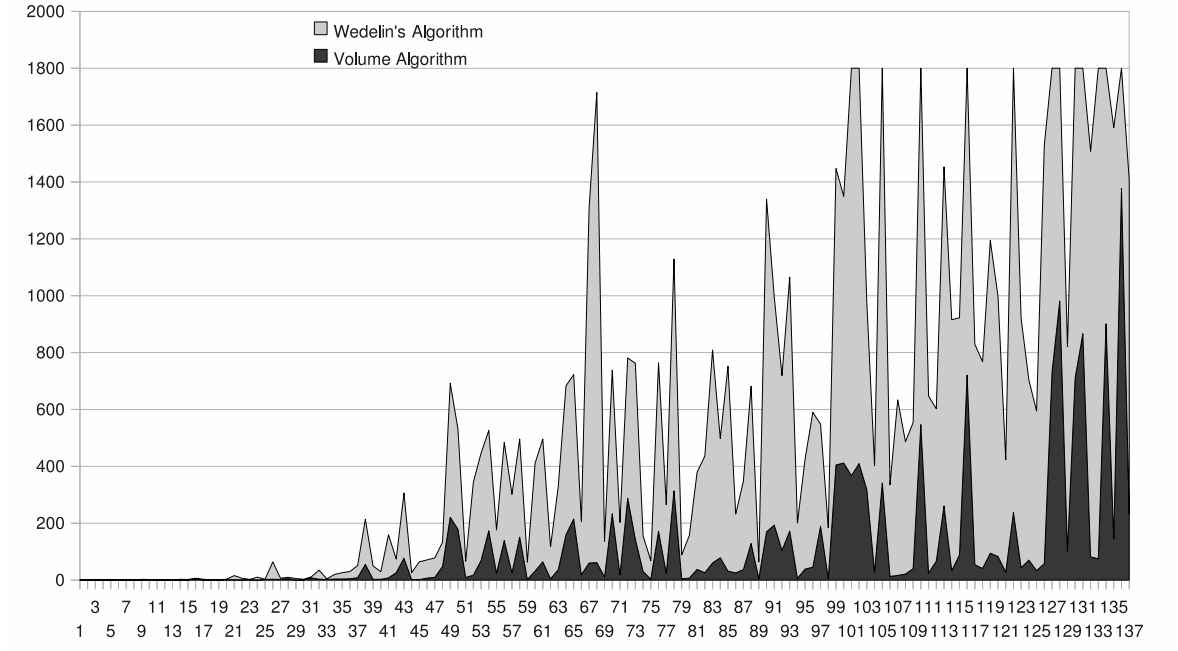


Figure 7: CPU time for Volume and Wedelin's Algorithms

*4.2.1. Volume Algorithm*

We used open source C++ codes from Coin-OR project[5] and implemented the Volume algorithm(see Algorithm 4) for the SMPTSP. In the algorithm the step size $s$ along subgradient direction is controlled by two parameters: the upper bound UB and the agility parameter $\beta$. The algorithm starts with a small value of UB, and UB is increased to $1.05\bar{z}$ on every occasions where the best lower bound $\bar{z}$ is greater than 0.95UB. The agility parameter $\beta$ is updated based on how many times iterations of the the following types are encountered:

- *Red:* there is no improvement i.e. $z^t \leq \bar{z}$, it means a smaller step size is required.
- *Yellow:* there is an improvement, i.e. $z^t > \bar{z}$, and $v^t(b - Ax^t) < 0$, longer step in the subgradient direction would have given a smaller value of $z^t$
- *Green:* there is an improvement, i.e. $z^t > \bar{z}$, and $v^t(b - Ax^t) \geq 0$, it suggests a longer step length

The agility parameter $\beta$ is updated as follows:

- if the iteration is green we set $\beta$ to $\min\{1.1\beta,\ 2\}$
- after 4 consecutive yellow iterations we set $\beta$ to $\min\{1.1\beta,\ 2\}$
- after 10 consecutive red iterations we set $\beta$ to $\max\{0.66\beta,\ 0.005\}$

As we mentioned before the approximate solution is a convex combination of vectors produced by solving the Lagrangian relaxation. The initial value of convex combination coefficient $\alpha$ is 0.1 and it is updated to $\max\{\frac{\alpha}{2},\ 10^{-4}\}$ in every 50 iterations if there is no improvement at least by 1%.
We have not set time limits to the Volume algorithm, however we set restrictions on the total number of iterations. The algorithm terminates if one of the following three conditions is met:

1. *Iteration Limit:* if the number of iterations reaches 6000
2. *GAP absolute precision:* if $\|v\|_\infty < 0.02$ and $|\bar{z}| < 0.0001$ and $|\bar{z} - c\bar{x}| < 0$.
3. *GAP relative precision:* if $\|v\|_\infty < 0.02$ and $|\bar{z}| \geq 0.0001$ and $\frac{|\bar{z} - c\bar{x}|}{|\bar{z}|} < 0.01$

*4.2.2. Wedelin's algorithm*

As proposed, Wedelin's algorithm starts with the approximation solution to the linear program of the PTSP as the initial preference of the workers to the jobs. In each step of Wedelin's algorithm, a constraint is chosen randomly among all violated constraints, say it correponds to job $\bar{j}$. Then it finds the smallest and the second smallest reduced costs $r^-$ and $r^+$, respectively, for all workers who are qualified for the job $\bar{j}$ . If a worker, say $\bar{w}$, has the smallest reduced cost $r^-$ then the preferences are updated as follows

$$\begin{aligned} \pi_{\bar{j}\bar{w}} &= \frac{r^- + r^+}{2} + \delta \\ \pi_{\bar{j}w} &= \max\{\frac{r^- + r^+}{2} - \delta,\ 0\} \quad \forall w \in P_{\bar{j}}, w \neq \bar{w}. \end{aligned}$$

In that way, we can ensure that the job $\bar{j}$ will be assigned to the worker $\bar{w}$ with the adjusted preferences. Where $\delta$ ensures that the job is assigned to only one worker in a case if $r^- = r^+$, i.e. there are several

[5]https://projects.coin-or.org/Vol

workers with the same smallest reduced cost $r^-$. However, in order to reduce the effect of the adjusted preferences to the current solution, we have to make $\delta$ as small as possible. Moreover, the convergance rate of the algorithm depends on choice of the parameters. Wedelin suggested to choose the parameter as

$$\delta = \frac{k}{1-k}\frac{(r^- + r^+)}{2} + \ell$$

where $k \in (0,1)$ and $\ell > 0$. In our implementation of the algorithm we used $k = 0.0006$ and $\ell = 0.001$. As we mentioned before, we choose the violated constraint randomly. If there is no improvement in last $\max\{mn/5, 20000\}$ iterations we reset the preferences to the current best and try another violated constraint choosing it randomly. Whenever the algorithm found a better feasible solution we updated the best solution and removed a worker from the list, who has the highest number of jobs that are assigned to other workers as well. Then algorithm proceeds to find a better solution with the current list of workers. There are two stopping critetia:

1. CPU time reaches 1800 seconds
2. Current best solution is feasible but no imrovement in last $\theta$ iterations, where $\theta = 20000$

### *4.3. Discussion of Results*

We compare our heuristic algorithm against the MIP model developed in Section 2 and implemented using CPLEX 11.2 embedded in C++. The total computation times are depicted in Figure 8. Figure 8

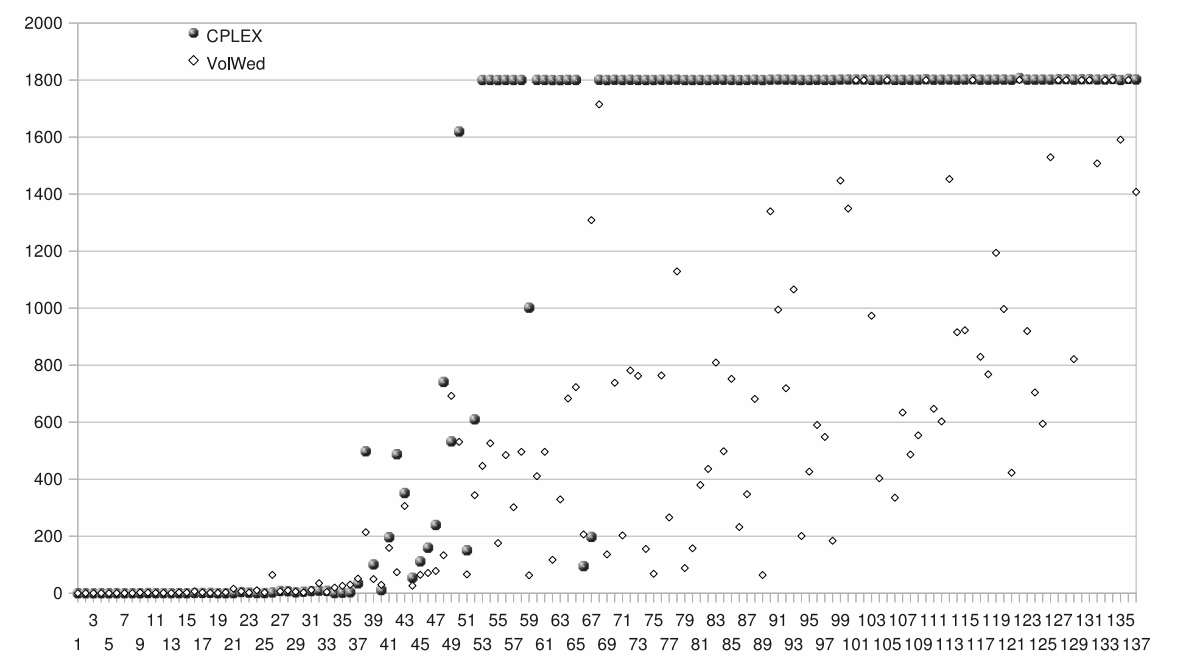


Figure 8: CPU time

shows that in the first 52 problem instances CPLEX was able to solve all problems to optimality within the given maximum time limit.

It is worth mentioning once again that the data files are ordered lexicographically by the number of jobs and workers.

Let us now divide the data set into 3 groups for which CPLEX was able to find

1. an exact optimal solution

2. a feasible solution

3. no feasible solution

CPLEX was able to find an exact optimal solution in 55 problem instances (See Table 1). It is interesting to observe that many of these problems that were solved optimally by CPLEX are in the first third of problems when we sort the data set lexicographically, as we have (by number of jobs and workers).

This is not completely divergent to our expectations – in other words, as expected and as our collective wisdom and intuition suggests, as problem size increases, for the SMPTSP, we will be more and more dependent on heuristic methods for obtaining solutions.

However, for these problems that CPLEX was able to solve to optimality, we can see that the average *Upper GAP* (or, the difference between the upper bound and the optimal solution, expressed as a percentage of the optimal solution) is not that high.

In actual fact, out of these 55 (relatively simpler) problem instances, our heuristic found the exact solutions in 46 instances and produced a feasible solution in remaining 9 instances with an average GAP of 4.5%. This figure is distorted by the GAPs for two problems for which our heuristic produced upper GAPs of 11.67% and 15.25% (the maximum GAP over all problems).

If we ignore these two problem instances, the average GAP over the remaining problems that were not solved to optimality is 1.94%. Interestingly, the computational times for the heuristic are, by and large, smaller than CPLEX times in most instances.

The GAPs between exact and heuristic solution is shown in Figure 9.

This comparison and analysis, albeit on smaller problems, gives us confidence that we can apply the heuristic to the full data set. Both CPLEX and heuristic algorithms produced the solutions within the given time limit.

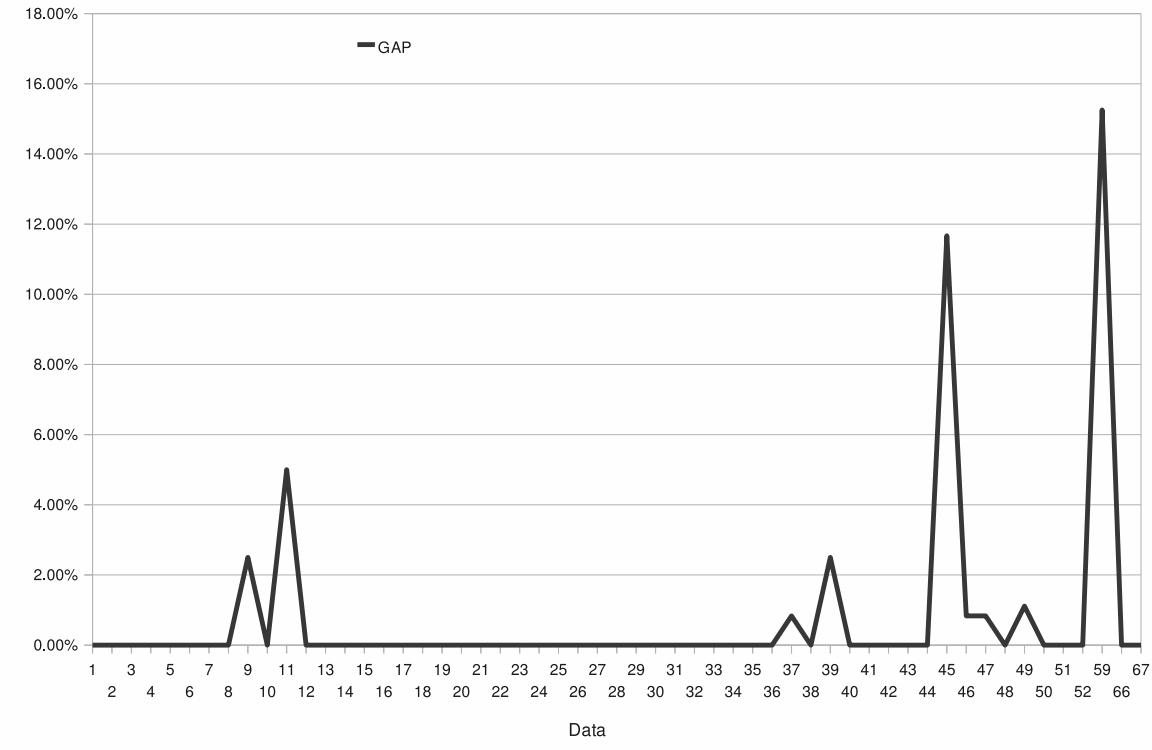


Figure 9: Heuristic solution gap for instances where CPLEX found exact optimal solutions

Still, we held off on making any detailed analysis till we analysed the results for those data where

| Data | CPLEX | | Heuristic | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Volume Algorithm | | | Wedelin's Algorithm | | | Total |
| | $W^*$ | CPU | #It | lb | CPU | $W^*$ | Gap | CPU | CPU |
| Data_1_23_40_66 | 20 | 0.01 | 411 | 19.80 | 0.05 | 20 | 0.00% | 0.03 | 0.08 |
| Data_2_24_40_33 | 20 | 0.01 | 402 | 19.94 | 0.02 | 20 | 0.00% | 0.00 | 0.02 |
| Data_3_25_40_66 | 20 | 0.00 | 1089 | 19.99 | 0.14 | 20 | 0.00% | 0.02 | 0.16 |
| Data_4_23_59_33 | 20 | 0.01 | 454 | 19.97 | 0.04 | 20 | 0.00% | 0.05 | 0.09 |
| Data_5_25_60_33 | 20 | 0.04 | 403 | 19.96 | 0.04 | 20 | 0.00% | 0.02 | 0.06 |
| Data_6_48_80_66 | 40 | 0.11 | 501 | 40.00 | 0.26 | 40 | 0.00% | 0.15 | 0.41 |
| Data_7_51_80_66 | 40 | 0.06 | 457 | 40.00 | 0.26 | 40 | 0.00% | 0.14 | 0.40 |
| Data_8_48_85_33 | 40 | 0.11 | 509 | 39.77 | 0.14 | 40 | 0.00% | 0.47 | 0.61 |
| Data_9_49_104_33 | 40 | 0.14 | 606 | 39.90 | 0.21 | 41 | 2.50% | 2.15 | 2.36 |
| Data_10_51_111_66 | 40 | 0.74 | 1474 | 39.66 | 1.31 | 40 | 0.00% | 0.73 | 2.04 |
| Data_11_24_119_33 | 20 | 0.34 | 372 | 19.97 | 0.06 | 21 | 5.00% | 0.93 | 0.99 |
| Data_12_49_119_33 | 40 | 0.35 | 406 | 39.83 | 0.16 | 40 | 0.00% | 0.72 | 0.88 |
| Data_13_25_120_33 | 20 | 0.26 | 376 | 19.96 | 0.07 | 20 | 0.00% | 0.52 | 0.59 |
| Data_14_75_124_33 | 60 | 0.24 | 748 | 59.39 | 0.45 | 60 | 0.00% | 2.16 | 2.61 |
| Data_15_72_126_33 | 60 | 0.08 | 607 | 59.48 | 0.36 | 60 | 0.00% | 1.57 | 1.93 |
| Data_16_75_131_66 | 60 | 0.85 | 2465 | 59.46 | 4.07 | 60 | 0.00% | 2.48 | 6.55 |
| Data_17_23_139_66 | 20 | 1.05 | 754 | 19.95 | 0.31 | 20 | 0.00% | 2.53 | 2.84 |
| Data_18_48_160_66 | 40 | 1.40 | 501 | 40.00 | 0.56 | 40 | 0.00% | 0.61 | 1.17 |
| Data_19_97_160_33 | 80 | 0.09 | 485 | 79.52 | 0.50 | 80 | 0.00% | 0.54 | 1.04 |
| Data_20_99_163_33 | 80 | 0.12 | 1226 | 79.10 | 1.36 | 80 | 0.00% | 1.71 | 3.07 |
| Data_21_93_175_33 | 80 | 0.28 | 1085 | 78.95 | 1.26 | 80 | 0.00% | 14.45 | 15.71 |
| Data_22_47_180_66 | 40 | 4.24 | 831 | 39.59 | 1.11 | 40 | 0.00% | 5.32 | 6.43 |
| Data_23_74_180_66 | 60 | 2.00 | 506 | 59.71 | 1.07 | 60 | 0.00% | 1.11 | 2.18 |
| Data_24_110_200_33 | 100 | 0.70 | 440 | 98.99 | 0.71 | 100 | 0.00% | 9.56 | 10.27 |
| Data_25_120_200_33 | 100 | 0.46 | 1085 | 99.01 | 1.93 | 100 | 0.00% | 0.82 | 2.75 |
| Data_26_116_203_66 | 100 | 2.74 | 571 | 97.35 | 2.85 | 100 | 0.00% | 61.29 | 64.14 |
| Data_27_49_204_66 | 40 | 8.06 | 1414 | 39.44 | 2.16 | 40 | 0.00% | 4.23 | 6.39 |
| Data_28_75_208_66 | 60 | 7.21 | 1696 | 59.24 | 4.93 | 60 | 0.00% | 4.78 | 9.71 |
| Data_29_22_219_66 | 20 | 3.09 | 556 | 19.82 | 0.33 | 20 | 0.00% | 5.11 | 5.44 |
| Data_30_25_219_66 | 20 | 5.23 | 664 | 19.90 | 0.47 | 20 | 0.00% | 2.47 | 2.94 |
| Data_31_90_230_66 | 80 | 7.58 | 1731 | 79.22 | 7.64 | 80 | 0.00% | 3.66 | 11.30 |
| Data_32_70_236_66 | 60 | 9.00 | 992 | 58.91 | 3.00 | 60 | 0.00% | 31.82 | 34.82 |
| Data_33_76_240_66 | 60 | 8.16 | 663 | 59.64 | 2.18 | 60 | 0.00% | 2.19 | 4.37 |
| Data_34_152_240_33 | 120 | 0.75 | 1312 | 118.80 | 3.79 | 120 | 0.00% | 15.87 | 19.66 |
| Data_35_171_280_33 | 140 | 1.68 | 917 | 138.61 | 3.70 | 140 | 0.00% | 22.10 | 25.80 |
| Data_36_175_280_33 | 140 | 3.27 | 1049 | 138.61 | 4.60 | 140 | 0.00% | 25.50 | 30.10 |
| Data_37_145_321_33 | 120 | 34.73 | 1904 | 118.73 | 8.09 | 121 | 0.83% | 42.91 | 51.00 |
| Data_38_147_347_66 | 120 | 497.85 | 3830 | 118.80 | 54.93 | 120 | 0.00% | 159.00 | 213.93 |
| Data_39_45_351_66 | 40 | 101.05 | 1092 | 39.57 | 2.72 | 41 | 2.50% | 46.76 | 49.48 |
| Data_40_138_360_33 | 120 | 11.16 | 608 | 118.80 | 2.60 | 120 | 0.00% | 26.81 | 29.41 |
| Data_41_144_360_66 | 120 | 196.70 | 563 | 118.43 | 7.67 | 120 | 0.00% | 151.35 | 159.02 |
| Data_42_101_380_66 | 80 | 487.69 | 2960 | 79.16 | 26.10 | 80 | 0.00% | 47.54 | 73.64 |
| Data_43_156_387_66 | 140 | 351.50 | 3972 | 138.56 | 76.58 | 140 | 0.00% | 229.23 | 305.81 |
| Data_44_121_400_33 | 100 | 54.22 | 504 | 98.99 | 1.97 | 100 | 0.00% | 24.38 | 26.35 |
| Data_45_67_420_33 | 60 | 111.66 | 803 | 59.08 | 1.62 | 67 | 11.67% | 62.72 | 64.34 |
| Data_46_147_423_33 | 120 | 159.70 | 1161 | 118.85 | 6.88 | 121 | 0.83% | 64.06 | 70.94 |
| Data_47_150_430_33 | 120 | 239.53 | 1535 | 118.52 | 9.34 | 121 | 0.83% | 68.62 | 77.96 |
| Data_48_120_434_66 | 100 | 741.34 | 3520 | 99.08 | 47.76 | 100 | 0.00% | 85.49 | 133.25 |
| Data_49_211_446_66 | 180 | 532.73 | 6001 | 175.74 | 220.76 | 182 | 1.11% | 471.47 | 692.23 |
| Data_50_187_447_66 | 160 | 1619.62 | 6001 | 158.02 | 178.46 | 160 | 0.00% | 352.31 | 530.77 |
| Data_51_196_480_33 | 160 | 150.77 | 992 | 157.63 | 9.18 | 160 | 0.00% | 56.91 | 66.09 |
| Data_52_205_480_66 | 160 | 610.03 | 556 | 156.80 | 17.17 | 160 | 0.00% | 326.70 | 343.87 |
| Data_59_70_525_33 | 59 | 1001.70 | 717 | 58.45 | 1.93 | 68 | 15.25% | 61.08 | 63.01 |
| Data_66_348_600_33 | 300 | 95.09 | 604 | 297.31 | 18.52 | 300 | 0.00% | 187.60 | 206.12 |
| Data_67_371_600_66 | 300 | 197.69 | 501 | 295.38 | 59.86 | 300 | 0.00% | 1248.74 | 1308.60 |

Table 1: Instances CPLEX solved to optimality

CPLEX was able to produce some feasible solutions, but not the optimal solution. We had to compare the performance of our heuristic against these problem instances.

In 28 problem instances CPLEX could not find the exact optimal solutions within the given time limit of 1800 CPU seconds but was able to produce feasible solutions (See Table 2).

| Data | CPLEX | | | | Heuristic | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | Volume Algorithm | | | Wedelin's Algorithm | | | Total |
| | lb | $W^*$ | Gap | CPU | #It | lb | CPU | $W^*$ | Gap | CPU | CPU |
| Data_53_127_487_66 | 100 | 116 | 13.79% | 1800.50 | 4120 | 98.94 | 69.01 | 101 | 1.00% | 377.44 | 446.45 |
| Data_54_175_492_66 | 140 | 141 | 0.71% | 1800.64 | 6001 | 138.21 | 173.17 | 140 | 0.00% | 352.83 | 526.00 |
| Data_55_85_493_66 | 70 | 78 | 10.26% | 1800.19 | 2631 | 69.28 | 23.68 | 72 | 2.86% | 152.22 | 175.90 |
| Data_56_163_500_66 | 140 | 142 | 1.41% | 1800.63 | 5047 | 138.48 | 139.04 | 140 | 0.00% | 345.57 | 484.61 |
| Data_57_88_508_66 | 70 | 79 | 11.39% | 1800.34 | 2677 | 69.17 | 25.51 | 72 | 2.86% | 276.19 | 301.70 |
| Data_58_158_517_66 | 140 | 156 | 10.26% | 1800.71 | 5559 | 138.62 | 151.23 | 140 | 0.00% | 344.89 | 496.12 |
| Data_60_181_549_66 | 139 | 155 | 10.32% | 1800.78 | 942 | 133.96 | 31.58 | 139 | 0.00% | 379.02 | 410.60 |
| Data_61_121_557_66 | 100 | 116 | 13.79% | 1800.57 | 3529 | 98.91 | 64.11 | 100 | 0.00% | 431.88 | 495.99 |
| Data_62_101_571_33 | 80 | 82 | 2.44% | 1800.18 | 939 | 78.42 | 4.36 | 90 | 12.50% | 112.70 | 117.06 |
| Data_63_97_577_66 | 80 | 96 | 16.67% | 1800.09 | 2762 | 79.18 | 36.26 | 82 | 2.50% | 292.70 | 328.96 |
| Data_65_179_596_66 | 159 | 174 | 8.62% | 1800.68 | 5291 | 157.36 | 214.82 | 159 | 0.00% | 508.09 | 722.91 |
| Data_68_359_613_66 | 300 | 303 | 0.99% | 1801.02 | 501 | 269.49 | 61.49 | 300 | 0.00% | 1653.13 | 1714.62 |
| Data_69_148_614_33 | 120 | 129 | 6.98% | 1800.31 | 1295 | 118.41 | 11.15 | 125 | 4.17% | 124.93 | 136.08 |
| Data_70_192_623_66 | 160 | 178 | 10.11% | 1801.06 | 5208 | 157.90 | 233.60 | 160 | 0.00% | 504.62 | 738.22 |
| Data_71_197_624_33 | 158 | 182 | 13.19% | 1800.51 | 1360 | 156.42 | 18.64 | 158 | 0.00% | 184.02 | 202.66 |
| Data_72_205_624_66 | 160 | 174 | 8.05% | 1801.06 | 6001 | 157.66 | 287.70 | 160 | 0.00% | 493.65 | 781.35 |
| Data_73_155_661_66 | 120 | 138 | 13.04% | 1800.69 | 4546 | 118.82 | 143.64 | 123 | 2.50% | 618.60 | 762.24 |
| Data_74_209_664_33 | 180 | 203 | 11.33% | 1800.37 | 1787 | 178.38 | 30.31 | 180 | 0.00% | 124.98 | 155.29 |
| Data_75_72_665_33 | 60 | 64 | 6.25% | 1800.52 | 802 | 59.08 | 2.78 | 71 | 18.33% | 65.37 | 68.15 |
| Data_76_162_683_66 | 140 | 159 | 11.95% | 1801.00 | 4357 | 138.26 | 171.51 | 140 | 0.00% | 592.39 | 763.90 |
| Data_83_222_700_66 | 180 | 200 | 10.00% | 1801.16 | 891 | 169.64 | 60.11 | 180 | 0.00% | 748.91 | 809.02 |
| Data_85_217_720_66 | 180 | 194 | 7.22% | 1800.94 | 501 | 176.74 | 31.41 | 180 | 0.00% | 720.59 | 752.00 |
| Data_86_178_721_33 | 140 | 166 | 15.66% | 1800.11 | 1871 | 138.41 | 25.13 | 146 | 4.29% | 206.86 | 231.99 |
| Data_87_203_735_33 | 170 | 192 | 11.46% | 1800.68 | 2107 | 168.34 | 36.78 | 174 | 2.35% | 310.69 | 347.47 |
| Data_89_88_788_33 | 70 | 83 | 15.66% | 1800.11 | 971 | 68.69 | 5.19 | 86 | 22.86% | 58.68 | 63.87 |
| Data_95_204_882_33 | 170 | 197 | 13.71% | 1800.68 | 1766 | 167.99 | 38.63 | 177 | 4.12% | 387.81 | 426.44 |
| Data_97_383_895_33 | 300 | 316 | 5.06% | 1800.86 | 3097 | 296.29 | 189.25 | 300 | 0.00% | 359.01 | 548.26 |
| Data_103_348_1024_33 | 300 | 333 | 9.91% | 1800.82 | 4654 | 296.76 | 317.78 | 303 | 1.00% | 655.52 | 973.30 |

Table 2: Instances CPLEX produced feasible solutions

On the contrary, for these instances our heuristic algorithm outperformed CPLEX in terms of producing reasonable feasible solutions. In 15 out of these 28 problem instances, our heuristic was able to find the exact optimal solution.

Furthermore, in 9 instances, the heuristic produced solutions with a max GAP of 4.17%. Overall for these problems, the heuristic algorithm produced an average GAP that is 2.2 times lower than the average GAP produced by CPLEX. This is a significant result.

Moreover, the CPU times for the heuristic delivering these feasible solutions is reasonable when compared with CPLEX. For all 28 instances the heuristic was able to find feasible solutions well within the given upper CPUtime limit and, across all problems in this set, the average CPU time of our heuristic was 500 seconds (or on average, 3 times) faster than CPLEX.

The integrality gaps for these problems are depicted in Figure 10.

In the remaining 54 problem instances CPLEX was not able to produce a feasible solution at all. Moreover, in only 10 instances it was able to solve the LP relaxation within the given time limit (See Table 3).

These are the large problem instances with an average of 178.9 workers and 1181.3 jobs. We see from this that as problem size increases, CPLEX is unable to cope with the increase in the complexity. The

| Data | CPLEX | | Heuristic | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Volume Algorithm | | | Wedelin's Algorithm | | | Total |
| | lb | CPU | #It | lb | CPU | $W^*$ | Gap | CPU | CPU |
| Data_64_176_595_66 | 160 | 1800.93 | 4049 | 158.03 | 159.18 | 160 | 0.63% | 523.89 | 683.07 |
| Data_77_180_688_33 | 160 | 1801.12 | 1749 | 158.41 | 24.39 | 162 | 1.89% | 241.36 | 265.75 |
| Data_78_199_688_66 | – | 1801.49 | 6001 | 156.62 | 313.69 | 160 | 1.91% | 815.19 | 1128.88 |
| Data_79_94_689_33 | 80 | 1800.53 | 887 | 78.43 | 4.61 | 93 | 17.72% | 83.63 | 88.24 |
| Data_80_112_691_33 | 99 | 1800.51 | 972 | 97.08 | 6.79 | 107 | 9.18% | 150.67 | 157.46 |
| Data_81_97_692_66 | 80 | 1800.70 | 2375 | 78.95 | 37.45 | 83 | 5.06% | 341.98 | 379.43 |
| Data_82_89_697_66 | 80 | 1800.45 | 1744 | 79.05 | 25.65 | 82 | 2.50% | 410.42 | 436.07 |
| Data_84_136_718_66 | 120 | 1801.00 | 2649 | 118.81 | 78.54 | 120 | 0.84% | 419.32 | 497.86 |
| Data_88_137_777_66 | – | 1801.27 | 3868 | 118.73 | 128.74 | 123 | 3.36% | 552.74 | 681.48 |
| Data_90_157_791_66 | – | 1801.49 | 3844 | 136.59 | 169.47 | 140 | 2.19% | 1170.12 | 1339.59 |
| Data_91_147_851_66 | – | 1801.97 | 4746 | 116.75 | 193.29 | 124 | 5.98% | 801.20 | 994.49 |
| Data_92_126_856_66 | – | 1801.45 | 3439 | 96.86 | 104.13 | 106 | 9.28% | 614.95 | 719.08 |
| Data_93_141_856_66 | – | 1801.31 | 4484 | 117.62 | 172.46 | 125 | 5.93% | 893.19 | 1065.65 |
| Data_94_93_881_33 | 80 | 1800.43 | 867 | 78.43 | 5.76 | 91 | 15.19% | 194.86 | 200.62 |
| Data_96_98_886_66 | – | 1801.05 | 2197 | 78.46 | 45.30 | 83 | 5.06% | 544.80 | 590.10 |
| Data_98_91_896_33 | 80 | 1800.70 | 860 | 78.12 | 5.79 | 90 | 13.92% | 178.23 | 184.02 |
| Data_99_176_956_66 | – | 1801.84 | 6001 | 157.67 | 404.77 | 160 | 1.27% | 1042.57 | 1447.34 |
| Data_100_194_956_66 | – | 1802.06 | 5377 | 158.37 | 411.75 | 160 | 0.63% | 937.58 | 1349.33 |
| Data_101_166_997_66 | – | 1802.05 | 6001 | 138.39 | 367.21 | 144 | 3.60% | 1432.81 | 1800.02 |
| Data_102_179_997_66 | – | 1802.13 | 6001 | 136.46 | 409.71 | 147 | 7.30% | 1390.31 | 1800.02 |
| Data_104_181_1057_33 | – | 1801.09 | 1263 | 143.53 | 28.13 | 165 | 14.58% | 375.13 | 403.26 |
| Data_105_173_1075_66 | – | 1802.51 | 4462 | 147.56 | 341.22 | 156 | 5.41% | 1458.80 | 1800.02 |
| Data_106_121_1096_33 | – | 1800.80 | 1086 | 98.14 | 12.83 | 113 | 14.14% | 322.21 | 335.04 |
| Data_107_114_1112_33 | – | 1800.77 | 1425 | 98.39 | 16.32 | 112 | 13.13% | 617.51 | 633.83 |
| Data_108_162_1115_33 | – | 1801.08 | 1104 | 125.38 | 21.13 | 145 | 15.08% | 465.39 | 486.52 |
| Data_109_205_1115_33 | – | 1801.31 | 1455 | 154.30 | 39.94 | 176 | 13.55% | 513.75 | 553.69 |
| Data_110_183_1143_66 | – | 1803.05 | 5957 | 153.35 | 546.72 | 167 | 8.44% | 1253.33 | 1800.05 |
| Data_111_155_1211_33 | – | 1801.36 | 1199 | 135.82 | 24.67 | 155 | 13.97% | 622.27 | 646.94 |
| Data_112_200_1213_33 | – | 1801.52 | 2193 | 167.06 | 66.06 | 194 | 15.48% | 536.45 | 602.51 |
| Data_113_141_1221_66 | – | 1801.88 | 4668 | 108.98 | 260.78 | 114 | 4.59% | 1192.05 | 1452.83 |
| Data_114_157_1227_33 | – | 1801.23 | 1583 | 136.42 | 33.54 | 157 | 14.60% | 882.04 | 915.58 |
| Data_115_228_1257_33 | – | 1801.88 | 2263 | 175.09 | 87.72 | 199 | 13.07% | 834.53 | 922.25 |
| Data_116_205_1262_66 | – | 1803.04 | 5620 | 173.06 | 720.77 | 190 | 9.20% | 1079.27 | 1800.04 |
| Data_117_192_1285_33 | – | 1801.31 | 1823 | 147.21 | 53.66 | 170 | 14.86% | 775.56 | 829.22 |
| Data_118_180_1302_33 | – | 1801.49 | 1481 | 144.65 | 40.92 | 165 | 13.79% | 726.99 | 767.91 |
| Data_119_236_1335_33 | – | 1801.96 | 2067 | 185.24 | 94.74 | 208 | 11.83% | 1099.28 | 1194.02 |
| Data_120_228_1341_33 | – | 1801.92 | 1899 | 184.29 | 82.73 | 208 | 12.43% | 914.09 | 996.82 |
| Data_121_147_1345_33 | – | 1801.04 | 1402 | 118.16 | 27.75 | 140 | 17.65% | 394.87 | 422.62 |
| Data_122_422_1358_66 | – | 1807.16 | 501 | 324.48 | 237.37 | 348 | 7.08% | 1562.81 | 1800.18 |
| Data_123_187_1376_33 | – | 1801.68 | 1365 | 156.01 | 43.40 | 178 | 13.38% | 876.18 | 919.58 |
| Data_124_198_1383_33 | – | 1801.64 | 2052 | 156.08 | 70.27 | 182 | 15.92% | 633.96 | 704.23 |
| Data_125_157_1448_33 | – | 1801.90 | 1362 | 127.61 | 33.07 | 152 | 18.75% | 561.23 | 594.30 |
| Data_126_193_1462_33 | – | 1801.68 | 1585 | 164.78 | 57.93 | 191 | 15.76% | 1471.45 | 1529.38 |
| Data_127_192_1472_66 | – | 1803.21 | 5283 | 167.83 | 731.43 | 185 | 10.12% | 1068.63 | 1800.06 |
| Data_128_207_1542_66 | – | 1803.42 | 6001 | 175.29 | 981.41 | 205 | 16.48% | 818.69 | 1800.10 |
| Data_129_233_1546_33 | – | 1801.78 | 1961 | 175.84 | 101.76 | 206 | 17.05% | 719.25 | 821.01 |
| Data_130_176_1562_66 | – | 1802.98 | 6001 | 138.35 | 711.29 | 145 | 4.32% | 1088.76 | 1800.05 |
| Data_131_415_1610_33 | – | 1803.44 | 5224 | 344.07 | 866.24 | 359 | 4.06% | 933.81 | 1800.05 |
| Data_132_216_1645_33 | – | 1801.97 | 1593 | 183.09 | 81.51 | 211 | 14.67% | 1426.39 | 1507.90 |
| Data_134_184_1776_66 | – | 1804.09 | 5833 | 157.56 | 901.19 | 179 | 13.29% | 898.89 | 1800.08 |
| Data_135_213_1988_33 | 180 | 1800.37 | 2198 | 177.20 | 144.48 | 206 | 15.73% | 1446.32 | 1590.80 |
| Data_137_245_2105_33 | – | 1802.07 | 2615 | 187.55 | 231.10 | 223 | 18.62% | 1176.44 | 1407.54 |

Table 3: Instances CPLEX not solved

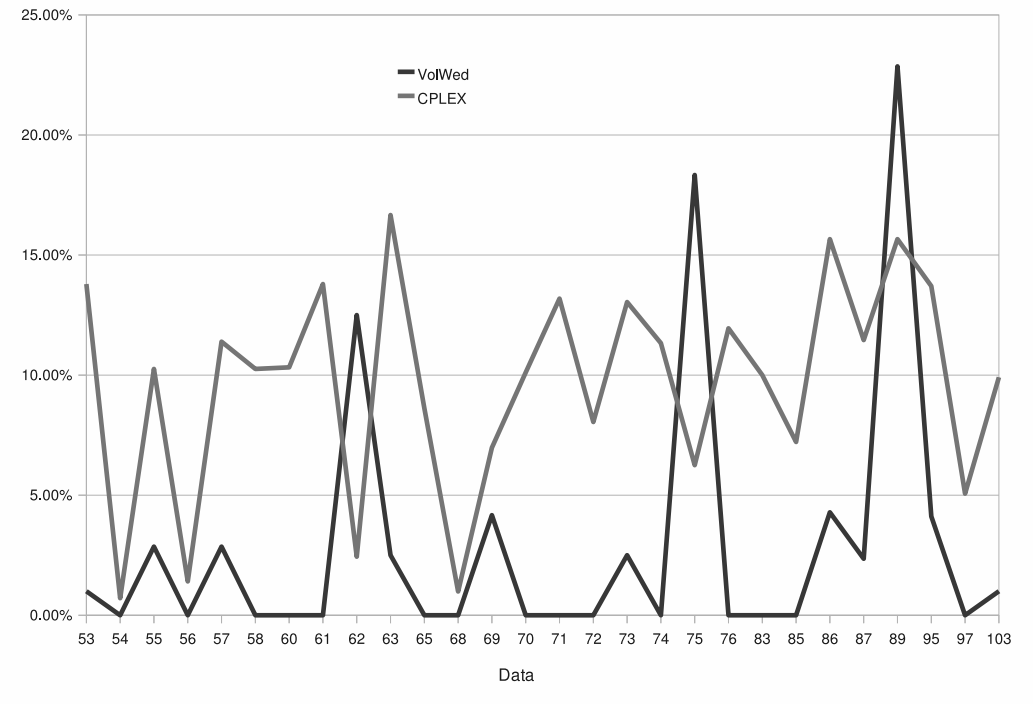


Figure 10: CPLEX vs Heuristic Gap

preceding analysis shows that our heuristic was able to withstand the increase in complexity as a result of increase in problem size.

For 52 of these instances our heuristic was able to produce feasible solutions with a *GAP-range* between 0.63% and 18.62% and with an average 10.27%. Here, we define GAP-range as the difference between the best upper bound and the highest lower bound expressed as a percentage of the best lower bound.

The average CPU time for the heuristic over all these difficult data sets was 1011.6 seconds.

In just two instances both CPLEX *and* our heuristic aproach did not produce any feasible solution (See Table 4). For these two problems, it was impossible to develop a heuristic result within the given time limit. It will remain a challenge for future research to identify solutions to these problem instances.

| Data | CPLEX | | Heuristic | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | Volume Algorithm | | | Wedelin's Algorithm | | | Total |
| | lb | CPU | #It | lb | CPU | $W^*$ | UC | CPU | CPU |
| Data_133_211_1647_33 | – | 1802.09 | 1429 | 181.66 | 74.77 | 211 | 19 | 1725.25 | 1800.02 |
| Data_136_216_2000_66 | – | 1804.26 | 6001 | 173.81 | 1377.14 | 216 | 256 | 422.98 | 1800.12 |

Table 4: Instances both CPLEX and heuristic not solved

Please note that all of the above mentioned datasets have been uploaded onto OR-Library[6] so as to enable other researchers to download these data and test their approaches on these.

## 5. Conclusions

In this paper, we address the shift minimisation personnel task scheduling problem (SMPTSP), which is NP–hard. We first defined the problem and discuss its properties. We then articulated similarities

[6]http://people.brunel.ac.uk/ mastjjb/jeb/info.html

between this problem and other problems in the literature and reviewed the literature, including a few survey papers on this problem and other similar problems.

We describe a mathematical formulation and develop a heuristic algorithm. The express purpose of the method developed in this paper is to enable us to solve large problem instances such as might be found in real-world applications. We thereby provide a mechanism whereby a rosterer can apply this method to develop solutions to day-to-day workforce task allocation instances. Many of these instances involve over 200 workers and 300 jobs/tasks. Empirical testing shows that typically our heuristic is able to solve these problem instances and provide reasonable upper bound (feasible solutions) for these problem instances.

The heuristic performs reasonably in small problem instances that CPLEX 11.2 is able to solve to optimality. The heuristic performs very well – both in terms of solution value as well as CPU time – over problems where CPLEX is able to find feasible but not provably optimal solutions (in a given max CPU time). For very large problems, where CPLEX even fails to deliver a lower bound in a reasonable amout of CPU time our heuristic is clearly superior.

Thus, we conclude that our algorithm is computationally efficient and can be used to tackle real–life problems or as a heuristic that can be embedded into other exact solution algorithms for solving the SMPTSP.

That said, there is still room for improvement. We still need to develop much more efficient methods for solving the SMPTSP (and other related problems). We feel that work in this area is still at a nascent stage. As we have shown there are some problems that even our heuristic cannot solve. Exact methods such as column generation may provide more efficient methods for solving larger problem instances.

## References


[1] Arkin, M., Silverberg, B., 1987. Scheduling jobs with fixed start and end times. Discrete Applied Mathematics 18, –.

[2] Barahona, F., Anbil, R., 2000. The volume algorithm: producing primal solutions with a subgradient method. Mathematical Programming 87, 385–399.

[3] Bastert, O., Hummel, B., de Vries, S., 2010. A generalized wedelin heuristic for integer programming. INFORMS Journal on Computing 22, 93–107.

[4] Dowling, D., Krishnamoorthy, M., Mackenzie, H., Sier, D., 1997. Staff rostering at a large international airport. The Annals of Operations Research 72, 125–147.

[5] Ernst, A. T., Jiang, H., Krishnamoorthy, M., Owens, B., Sier, D., 2004. An annotated bibliography of personnel scheduling and rostering. Annals OR 127 (1-4), 21–144.

[6] Ernst, A. T., Jiang, H., Krishnamoorthy, M., Sier, D., 2004. Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153 (1), 3–27.

[7] Fischetti, M., Martello, S., Toth, P., 1987. The fixed job schedule problem with spread-time constraints. Operations Research 35 (6), 849–858.

[8] Fischetti, M., Martello, S., Toth, P., 1989. The fixed job schedule problem with working-time constraints. Operations Research 37 (3), 395–403.

[9] Fischetti, M., Martello, S., Toth, P., 1992. Approximation algorithms for fixed job schedule problems. Operations Research 40 (S1), S96–S108.

[10] Gertsbakh, I., Stern, H. I., 1978. Minimal resources for fixed and variable job schedules. Operations Research 26, 68–85.

[11] Gondran, M., Minoux, M., 1984. Graphs and Algorithms. A Wiley–Interscience Publication. John Wiley, New York.

[12] Gupta, U. I., Lee, D. T., Leung, Y. T., 1979. An optimal solution for the channel assignment problem. IEEE Transactions on Computing C-28, 807–810.

[13] Kolen, A. W., Lenstra, J. K., Papadimitriou, C. H., Spieksma, F. C., 2007. Interval scheduling: A survey. Naval Research Logistics 54 (5), 530–543.

[14] Krishnamoorthy, M., Ernst, A. T., 2001. The personnel task scheduling problem. Optimisation Methods and Applications, Eds., X. Yang and K. L., Teo and L. Caccetta, Kluwer, 343–368.

[15] Kroon, L. G., Salomon, M., Wassenhowe, L. N. V., July-August 1997. Exact and approximation algorithms for the tactical fixed interval scheduling problem. Operations Research 45 (4), 624–638.

[16] Nakajima, K., Hakimi, S. L., Lenstra, J. K., 1982. Complexity results for scheduling tasks in fixed intervals on two types of machines. SIAM Journal on Computing 11, 512–520.

[17] Valls, V., Pérez, A., Quintanilla, S., 1996. A graph colouring model for assigning a heterogenous workforce to a given schedule. European Journal of Operations Research 90, 285–302.

[18] Wedelin, D., 1995. An algorithm for large scale 01 integer programming with application to airline crew scheduling. The Annals of Operations Research 57, 283–301.