Pergamon 0305-0548(94)00094-8

Computers Ops Res. Vol. 23, No. 1, pp. 93–102, 1996

0305-0548/96 $9.50 + 0.00

ON THE COMPLEXITY OF MANPOWER SHIFT SCHEDULING

Hoong Chuin Lau†

Department of Computer Science, Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro-ku, Tokyo 152, Japan

(*Received September 1993; in revised form November 1994*)

Scope and Purpose The scheduling of manpower resources is a critical concern in large service industries where employees are rostered on shifts in order to provide services round the clock. Researchers in the operations research and artificial intelligence communities have proposed various methods to solve manpower scheduling problems. This paper looks at the manpower scheduling problem from its computational complexity point of view. We show the computational intractibility of a restricted version of the manpower scheduling problem, namely, that of scheduling manpower subject to shift change constraints. Changing of shifts is a critical constraint especially for scheduling ground crews in airports and sea ports. We propose an efficient algorithm to solve a special case of this problem. We also show how our algorithm may be extended to handle more complex versions of the problem.

Abstract—We consider the shift assignment problem in manpower scheduling, and show that a restricted version of it is NP-hard by a reduction from 3SAT. We then present polynomial algorithms to solve special cases of the problem and show how they can be deployed to solve more complex versions of the shift assignment problem. Our work formally defines the computational intractibility of manpower shift scheduling and thus justifies existing works in developing manpower scheduling systems using combinatorial and heuristic techniques.

1. INTRODUCTION

Manpower scheduling (or rostering) is concerned with the scheduling of manpower resources to meet temporal operational requirements in ways that satisfy the goals and policies imposed by the management, labour union and the government. It is a critical management activity in service organizations which operate round-the-clock where workers are scheduled to work on multiple shifts. Examples include the scheduling of nurses in hospitals, ground crews in airports, and operators in telephone companies. Glover and McMillan [1] gives a good survey of the common manpower scheduling problems (MSP) faced by industry today.

MSPs are often modelled as optimization problems for constructing cost-optimal schedules that do not violate given scheduling constraints. As MSPs are highly complex, OR researchers have been investigating both sequential and combined approaches. For combined approaches, one could conceive formulating MSP as a comprehensive integer linear program. Unfortunately, the size and complexity of the resulting formulation often makes this approach rather impractical and difficult to maintain. Recently, Tien and Kamiyama [2] proposed a framework for solving MSP in three main stages‡, namely, *allocation*, *offday scheduling* and *shift assignment*:

1. *Allocation* computes the demands, i.e. the number of workers needed for each shift in each day so that the temporal requirements can be met. Allocation also determines the level of employment, i.e. computes the minimum number of workers needed to fulfill demands over the entire planning period.
2. *Offday scheduling* is concerned with assigning offdays on the schedule subject to offday

†Hoong Chuin Lau obtained his B.Sc. and M.Sc. in Computer Science from the University of Minnesota, Minneapolis in 1987 and 1988 respectively. Currently, he is a doctorate candidate at the Tokyo Institute of Technology. His research interests include resource scheduling, computational complexity and algorithm design.

‡Actually, their framework comprises 5 stages, but we have combined their stages 1 and 2, and stages 3 and 4, for simplicity of discussion.

and workstretch constraints. Typical offday constraints include the *A out of every B weekends off* and case-specific offday constraints to ensure a fair distribution of offdays among workers. Typical workstretch constraints impose upper and lower bounds on the lengths of workstretches shift assignment constraints to ensure workers have neither too short nor too long periods of work.

3. *Shift assignment* completes the schedule by assigning shifts to the schedule subject to demands and the shift assignment constraints. Common shift assignment constraints include the *shift change constraints* and *consecutive same shift constraints*. Shift change constraints permit specific shift changes (e.g. morning shift to afternoon shift) and forbid others (e.g. night shift to morning shift) so that workers are not unduly affected by irregular rest periods. Consecutive same shift constraints impose upper bounds on the number of consecutive days a worker is allowed to be on the same shift. Since some shifts (e.g. night shifts) may be more undesirable than others, this restriction will prevent any worker from having too many undesirable shifts.

The offday scheduling problem has been tackled by various researchers, e.g. [2, 3, 4, 5, 6], some of which addressed the combined offday scheduling and shift assignment problem. As for the shift assignment problem (SAP), many interesting variations of it have been studied. For example, Carraresi and Gallo [7] considered the problem of finding an even balance of shifts over the planning period; Martello and Toth [8] gave a heuristic approach to solve the bus-driver scheduling problem which assigns duties to drivers such that the total time spent driving and the spread time over the planning period is bounded; Balakrishnan and Wong [9] applied network optimization techniques to solve SAP subject to a host of constraints; and Koop [10] also utilized a network model to provide lower bounds on workforce size under various common shift change constraints.

In this paper, we consider another variation of the SAP, which we termed CSAP (*Changing Shift Assignment Problem*). CSAP is concerned with finding a satisfying assignment of shifts to workers subject to demands and shift change constraints. The problem arises in the scheduling of ground crews in airports to service inbound flights for their next journey. Here, shift types are often associated with flights because different flights require ground crews of different skill profiles to service. Hence, the number of shift types is large and the shift change constraints become fairly complicated in order to generate cost-effective schedules to meet the fluctuating flight requirements, even more so considering that full-time and part-time workers do not work the same set of shifts. In such environments, shift types are usually grouped into clusters where shift changes within the clusters are permitted, while shift changes across clusters are either monotonic, partially-ordered (i.e. they form a precedence graph) or even cyclic sometimes, as our experience indicate [11, 12]. Hence, we recognise CSAP to be a fundamental problem of SAP, particularly in service industries such as airports and seaports.

This paper proceeds as follows. We will show that CSAP is NP-hard, even under restricted assumptions, thereby allowing us to conclude that other more-general variants of shift scheduling problems are also NP-hard. Many years ago, Even *et al.* [13] presented an NP-hardness proof of a similar problem, the *Time Tabling Problem*. However, the proof was felt to be tedious; which motivated us to present a more intuitive but non-trivial proof for CSAP. Our proof can be slightly amended to prove the NP-hardness of the Time Tabling Problem. Next, we consider special cases of CSAP which are polynomially solvable. A trivial case is found when all shift changes are permitted. Here, an algorithm which matches demands to assignable slots would suffice. This case is, however, meaningless as far as manpower scheduling is concerned. A more realistic case would be when shift changes are *monotonic*. In manpower scheduling, it is a common practice that, as the days progress, a worker should be turned up for work no earlier than the day before so that he maintains a healthy biological clock. Thus, if we arrange shifts in order of their start times, the shift numbers assigned to each worker should be monotonically non-decreasing. The non-decreasing sequence may be broken by an offday because workers are expected to get enough rest during the offday to be turned up for an earlier shift the next day. We show that the running time for generating monotonic schedules is polynomial for both non-cyclic and cyclic schedules. Finally, we discuss how our algorithm may be extended to solve the more complex versions of SAPs, such as those involving consecutive same shift constraints, spare supply of workers, and non-monotonic shift changes.

2. PRELIMINARIES

We explain the terms of reference. The *planning period* is the number of days for which manpower scheduling is performed. *Shifts* are numbered 1, 2, ..., etc and 0 denotes an *offday*. A *schedule* is a matrix where rows represent workers and columns represent days of the planning period. Each matrix element is known as a *slot*, which will be assigned either a shift or an offday. A slot *precedes* another slot if it is on its adjacent left position. The *domain* of a slot is the set of shifts which are assignable to the slot. A schedule in which offdays have been assigned is known as a *show-up schedule*. In a show-up schedule, the number of workers not having offdays on a given day is the *supply* of workers on that day and those slots are termed *working slots*. A *demand matrix* gives the number of workers required in each shift on each day of the planning period. A *shift change matrix* is a boolean square matrix which defines the shift change permission from one shift to another. We assume that an offday may precede or follow any shift. A shift change matrix is *monotonic* if it is upper-triangular consisting of all ones. A *feasible schedule* is a schedule with all slots assigned which: (1) satisfies the demand matrix: and (2) for any 2 adjacent slots in the schedule, the shift change is satisfied. Figures 1(a) to (d) give an example of terms explained.

A schedule is said to be *cyclic* if the schedule is rotated row-wise from one planning period to the next. Cyclicity allows the schedule to be used indefinitely and also guarantees fairness of shift distribution among workers over time. Hence, a cyclic schedule can be seen as a contiguous sequence of slots from the upper-left corner to the lower-right corner of the schedule. We will represent a cyclic schedule by a list of *workstretches*, as shown in Fig. 1(e). A workstretch is a contiguous sequence of slots delimited by offdays, and the number of slots is its *length*. One may verify that transformation of schedule from the matrix to the workstretch representation and vice versa may be done in polynomial time.

The following notations will be used throughout the paper. Let K = number of workers,

Shift\Day

	M	T	W	H	F	S	U
1	1	0	0	1	0	0	0
2	2	3	2	2	1	2	3
3	0	1	1	0	1	1	0
4	1	0	1	1	2	2	1
5	1	1	0	1	1	1	1
6	1	2	3	2	0	0	1
7	0	0	1	0	0	0	0
8	2	1	1	1	3	1	0

(a)

Worker\Day

	M	T	W	H	F	S	U
1	-	-	-	-	-	0	0
2	-	-	-	-	-	-	0
3	0	-	-	-	-	-	-
4	0	0	-	-	-	-	-
5	-	0	0	-	-	-	-
6	-	-	-	0	0	-	-
7	-	-	-	-	0	0	-
8	-	-	-	-	-	0	0
9	-	-	-	-	-	-	0
10	-	-	-	0	-	-	-

(b)

Shift\Shift

	1	2	3	4	5	6	7	8
1	1	0	1	0	1	0	1	0
2	0	1	0	1	0	1	0	1
3	0	0	1	0	1	0	1	0
4	0	0	0	1	0	1	0	1
5	0	0	0	0	1	0	1	0
6	0	0	0	0	0	1	0	1
7	0	0	0	0	0	0	1	0
8	0	0	0	0	0	0	0	1

(c)

Worker\Day

	M	T	W	H	F	S	U
1	4	6	6	8	8	0	0
2	1	3	3	5	5	5	0
3	0	2	2	2	4	4	4
4	0	0	2	2	4	4	6
5	8	0	0	1	3	3	5
6	5	5	7	0	0	2	2
7	6	6	6	6	0	0	2
8	2	2	6	6	8	0	0
9	2	2	4	4	8	8	0
10	8	8	8	0	2	2	2

(d)

Workstretch\Day

1	(F)	2	2	2	4	6	6	8	8
2	(M)	1	3	3	5	5	5		
3	(T)	2	2	2	4	4	4		
4	(W)	2	2	4	4	6	8		
5	(H)	1	3	3	5	5	5	7	
6	(S)	2	2	6	6	6	6		
7	(U)	2	2	2	6	6	8		
8	(M)	2	2	4	4	8	8		
9	(M)	8	8	8					

(e)

Fig. 1. (a) Demand matrix; (b) show-up schedule; (c) shift change matrix; (d) feasible schedule of (b) which satisfies demand matrix (a) and shift change matrix (c); and (e) workstretch representation of (d), where the letters in brackets represent the start days of the respective workstretches.

I = number of days of the planning period, and J = number of shifts. Let $D = I \times J$ demand matrix, where $D_{i,j}$ is the number of workers required to work shift j on day i. Let $S = K \times I$ show-up schedule matrix. Let $V = K \times I$ matrix containing the domains of respective slots. Let $\delta = J \times J$ shift change matrix, where $\delta_{j_1, j_2} = 1$ if the change of shift from j_1 to j_2 is permitted, and 0 otherwise. Let $\sigma = K \times I$ feasible schedule. Then, CSAP is an NP search problem whose input is the tuple $(K, I, J, D, S, V, \delta)$ and output is σ or fail.

3. PROBLEM COMPLEXITY

In this section, we consider the decision problem of CSAP, CSAP(D), which asks whether a feasible schedule exists given the input $(K, I, J, D, S, V, \delta)$. Clearly, CSAP(D) is in the class NP. We will show the completeness of CSAP(D) by a polynomial many-one reduction from 3SAT. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of *Boolean variables*. A *truth assignment* for X is a function $t : X \to \{\text{True}, \text{False}\}$. If x is a variable in X, then the symbols x and $\bar{x}$ are called a *positive* and *negative literal* respectively. A *clause* C over X is a disjunction of literals over X, and we say that C is *satisfied* by a truth assignment if at least one of its literals is *True* under that assignment. A 3CNF over X is represented by a set of clauses $F = \{C_1, C_2, \ldots, C_m\}$ over X with three literals per clause. We say that F is *satisfiable* if there exists a truth assignment for X that simultaneously satisfies all clauses in F. 3SAT is defined as follows [14]:

INSTANCE: A set X of boolean variables $\{x_1, x_2, \ldots, x_n\}$ and a 3CNF represented by $F = \{C_1, C_2, \ldots, C_m\}$ over X.
QUESTION: Is there a truth assignment for X such that F is satisfiable?

Theorem 3.1. CSAP(D) is NP-complete, even for fixed $J \geqslant 6$, fixed shift change matrix and non-cyclic schedule with no offdays.

Proof. Let F be an instance of 3SAT with n variables and m clauses. We construct an instance of CSAP as follows (see Fig. 2):

1. Let $K = n$ and $I = 2m - 1$. Define a $K \times I$ show-up schedule with no offdays.
2. Let $J = 6$. We rename the 6 shifts as $0, 1, \ldots, 5$, where 0 is a shift instead of an offday.
3. Define the demand matrix D as follows

$$D_{i,j} = \begin{cases} 1, & \text{if } i \text{ is odd and } j = 1 \\ 0, & \text{otherwise.} \end{cases}$$

We explain the motivation of this construction. The main idea is to let each row of the schedule represent one variable while each *odd* column represents one clause. The *even* columns will be used to transmit information which will be explained later. Initialize the domains of odd columns as follows

$$V_{k,2i-1} = \begin{cases} \{0, 1\}, & \text{if } x_k \text{ or } \overline{x_k} \text{ occurs in } C_i \\ \{2, 3, 4, 5\}, & \text{otherwise.} \end{cases}$$

In other words, when a variable occurs in a clause, the corresponding slot has a $\{0, 1\}$ domain (henceforth called an *occupied-slot*); otherwise, it has a $\{2, 3, 4, 5\}$ domain (*vacant-slot*). The *True*/*False* assignment of variables thus corresponds to the assignment of shifts 1/0 to respective occupied slots. An occupied slot is said to have a *positive sign* if it corresponds to a positive literal, and *negative sign* otherwise.

To handle the condition that each clause must contain at least one literal set to *True*, we set the demands for shift 1 to be one on the odd rows, as shown in the demand matrix definition above.

It remains to show how to ensure that a variable is uniquely set across all clauses. That is, if an occupied-slot, say $S_{k,i}$, is set to 0 (resp. 1), then we must ensure that all other occupied slots $S_{k,j}$ $(j \neq i)$ whose signs are equal to the sign of $S_{k,i}$ must be set to 0 (resp. 1); while those occupied slots whose signs are different must be set to 1 (resp. 0). We introduce *mediums*, the even columns, to transmit information across two odd columns. To ensure that information is transmitted properly, define the shift change matrix as follows.

Worker \ Day	1	2	3
1	{0,1}	{4,5}	{0,1}
2	{2,3,4,5}	{2,3,4,5}	{0,1}
3	{0,1}	{2,3,4,5}	{2,3,4,5}
4	{0,1}	{2,3}	{0,1}

Worker \ Day	1	2	3
1	0	4	1
2	2	2	0
3	1	3	3
4	0	2	0

Shift \ Shift	0	1	2	3	4	5
0	0	0	1	0	1	0
1	0	0	0	1	0	1
2	1	0	1	0	0	0
3	0	1	0	1	0	0
4	0	1	0	0	1	0
5	1	0	0	0	0	1

Fig. 2. The first schedule is the instance of CSAP(D) corresponding to the 3CNF $F = (x_1 \vee \overline{x_3} \vee \overline{x_4}) \wedge (\overline{x_1} \vee x_2 \vee \overline{x_4})$. The second schedule is a feasible schedule of the first and corresponds to the truth assignment $t(x_1) = False$, $t(x_2) = False$, $t(x_3) = False$ and $t(x_4) = True$. The third matrix gives the structure of the fixed shift change matrix used in reduction.

Consider any variable x_k. First, suppose x_k occurs in clause i followed by clause $i+1$. To force the two consecutive occupied slots $S_{k,2i-1}$ and $S_{k,2i+1}$ to be identically assigned, we set the domain of the medium slot $S_{k,2i}$ (i.e. $V_{k,2i}$) to $\{2,3\}$ and the shift change matrix as follows

$$\delta_{0,2} = \delta_{2,0} = \delta_{1,3} = \delta_{3,1} = 1.$$

In other words, shift 2 is used to transmit a 0 while shift 3 is used to transmit a 1. Likewise, if x_k occurs in one clause and $\overline{x_k}$ in the next, we set the domain of the medium slot to $\{4,5\}$ and set the shift change matrix elements

$$\delta_{0,4} = \delta_{1,5} = \delta_{4,1} = \delta_{5,0} = 1.$$

In other words, shift 4 receives a 0 and sends a 1; while shift 5 receives a 1 and sends a 0. Finally, we have the case where there are vacant-slots between two consecutive occupied slots. To make the vacant slots transmit the necessary information, we set the shift change

$$\delta_{i,i} = 1, \quad \text{for } i = 2, \ldots, 5.$$

All even-column slots not initialized so far have domains $\{2,3,4,5\}$ and all unset shift change matrix elements are set to 0.

This completes the definition of the reduction process. Clearly the above reduction process takes polynomial time. We claim that F has a satisfying assignment if and only if the constructed CSAP problem instance has a feasible schedule.

First, assume F has a truth assignment. For each variable x_k, if x_k is set to *True*, the occupied-slots of row k will be set to 1 or 0 if their signs are positive and negative respectively. The reverse holds for the case of *False*. Since each clause has one *True* literal, the number of shift 1's in each odd column is at least 1, which satisfies the demand constraints. The shift change constraints are not violated since each variable has a single truth value.

The converse also holds. Given a feasible schedule, we simply read off the shift assigned to the leftmost *occupied* slot of each row k to get the truth value of the corresponding variable x_k in F (i.e. *True* for 1 and *False* for 0). If the slot has a negative sign (i.e. it represents a negative literal), then the truth value is negated. Now since each column contains at least one shift 1 to meet the demands, each clause in F has at least one literal set to *True*, and thus F has a satisfying assignment. □

4. MONOTONIC CSAP

In this and the next sections, we solve CSAP with monotonic shift change constraints. Monotonic CSAP is technically interesting because a combinatorial calculation shows that the total number of monotonic sequences of size n and maximum value m is given by

$$\binom{n+m-1}{m-1}, \text{ which is } O(n^m).$$

Hence, a naive scheduling algorithm would have exponential time complexity in the worst case.

Khoong [15] recently proposed a simple heuristic cum iterative improvement scheme for generating monotonic cyclic schedules, but could not guarantee that a feasible solution would be found if one exists. Our algorithm guarantees that a solution be found if one exists, for both non-cyclic as well as cyclic schedules.

We assume that all slots are of the domain $\{1, 2, \ldots, J\}$. We also assume that for each day i, the sum of demands for all shifts equals the supply of workers on day i.

4.1. Algorithm G

We propose a greedy algorithm G to solve monotonic CSAP with non-cyclic schedules. Essentially, G is a greedy algorithm which assigns shifts in increasing shift numbers and for each shift, it schedules by columns. The *leftmost* (resp. *rightmost*) slot of a row refers to its first (resp. last) unassigned slot, and the *tail* of a row refers to the sequence of slots from its leftmost slot to its rightmost slot. We use the following notations. Let σ_k denote the kth row of σ; $\sigma_{k,i}$ denote a slot in row k column i; $\sigma_{k,i-1}$ and $\sigma_{k,i+1}$ denote the slots to the left and right of $\sigma_{k,i}$.

procedure G:
0. initialize $\sigma = S$;
1. **for** $j = 1$ **to** J **do**
2. **for** $i = 1$ **to** I **do**
3. **until** $D_{i,j} = 0$ **do**
4. $A \leftarrow \{1 \leqslant K \leqslant K \mid k$'s leftmost slot is at column $i\}$;
5. **if** $A = \emptyset$ **then fail**;
6. pick k from A such that row k has the longest tail;
7. $\sigma_{k,i} \leftarrow j$; $D_{i,j} \leftarrow D_{i,j} - 1$;
8. **enduntil**
9. **endfor**
10. **endfor**

4.2. Proof of correctness

Theorem 4.1. G returns a feasible schedule if and only if there exists a feasible schedule.

Proof. If G exits successfully, then all slots would be assigned shift values in monotonic fashion. Thus, σ is a feasible schedule. To prove the converse, it suffices to show that if a feasible solution exists, then every assignment made by G maintains the feasibility of the partial schedule. Consider any arbitrary iteration with current shift j and column i. Let σ be a partial schedule constructed by G just before this iteration. Let σ' be σ plus one new assignment $\sigma_{k,i} \leftarrow j$, where k is picked by G in Step 6. Then, Lemma 4.1 (see below) tells us that σ' is also feasible. Thus, by induction on the loop indices, it is clear that the converse holds. □

Lemma 4.1. σ' admits a feasible schedule if σ admits a feasible schedule.

Proof. Without loss of generality, let σ^+ represent one such feasible schedule derived hypothetically from σ. If $\sigma^*_{k,i} = \sigma^+_{k,i} = j$, then σ^+ obviously admits a feasible schedule. Otherwise, we can conclude that $\sigma^+_{k,i} > j$ because we schedule in non-decreasing shift numbers. Without loss of generality, suppose that shift j is at row k' in σ^*, i.e. $\sigma^*_{k',i} = j$. By monotonicity, $\sigma^*_{k',i-1} \leqslant j$, which means that row k' also has leftmost slot at column i in σ^+. The fact that G did not pick k' means that the tail of k is at least as long as k' (by Step 6). The contents of rows k and k' of the feasible schedule σ^+ are as shown in Fig. 3(a). By swapping certain slots on rows k and k' of σ^+, we can obtain *another* feasible schedule whose slot assignments match exactly those of σ', thereby proving the lemma. That resulting feasible schedule is shown in Fig. 3(b). Swapping is carried out as follows. Surely, $\sigma^+_{k',i}$ may be moved to row k. The reverse is possible if $\sigma^*_{k',i+1} \geqslant \sigma^*_{k,i}$ or $\sigma^*_{k,i}$ is a rightmost slot. If not, we may conclude $\sigma^+_{k',i-1} < \sigma^*_{k,i+1}$ and thus may swap elements $\sigma^+_{k,i+1}$ and $\sigma^*_{k',i+1}$. Continuing inductively this way, row k' will eventually hit its rightmost slot before k does because the latter is at least as long. □

4.3. Complexity analysis

Step 4 takes $O(K)$ time since it has to scan a column of $O(K)$ workers, and checking for leftmost

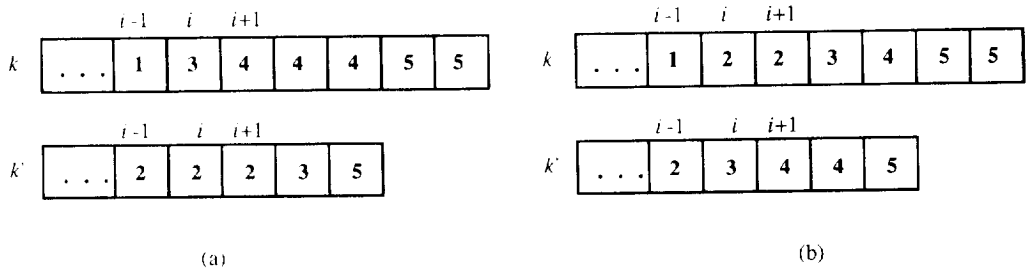

Fig. 3. Rows k and k' of σ^* with current shift $j = 2$. (a) shows the contents before the swap and (b) shows the contents after the swap.

slots takes constant time. Step 6 is also $O(K)$ time in the worst case because A contains at most K elements, and checking the tail length takes constant time. Steps 5 and 7 take constant time. Hence, the worst case time complexity is $O(K^2IJ)$, since $D_{i,j} \leqslant K$ for all i, j. However, observe that the total number of iterations is determined by the matrix sum of the demand matrix which is bounded above by the total number of working slots in the schedule. The number of working slots is at most $K \times I$ and hence the worst-case time complexity is $O(K^2I)$, which is independent of J.

5. EXTENSION TO CYCLIC SCHEDULES

We now consider an extension of algorithm G to manage cyclic schedules. The implication of cyclic schedules is that shift changes across rows have to be dealt with if the last slot of a row and the first slot of its next row are both working slots. Note that a workstretch may span across more than 1 week, as shown in first workstretch of Fig. 1(e). This problem remains NP-hard under the same restrictions because it is a general case of the non-cyclic CSAP by appending a dummy column consisting of all offdays after the last column. However, the good news is when shift changes are monotonic, the problem remains polynomially solvable with no worse time complexity.

We briefly discuss how our algorithm can be extended. Since days may wrap around, we now view the schedule by its workstretch representation. Thus, instead of rows and columns, we now deal with workstretches and days. As before, we schedule in increasing shift numbers. For each current shift j, instead of assigning j column by column, we maintain another data structure B which contains the set of days which have unassigned shift j's. A now contains the set of workstretches whose leftmost slots are at any day contained in B. Each time, we again pick a workstretch from A with the longest tail. The proof of correctness is direct extension of Theorem 4.1 and will not be elaborated here. As far as the time complexity is concerned, it is clear that each iteration decrements the demand matrix sum by 1 unit. Hence, the worst-case time complexity can be verified to be $O(K^2I)$.

6. EXTENSION TO GENERAL SAPS

Finally, we consider extensions to the more complex variants of SAPs.

6.1. Handling consecutive same shift constraints

Our algorithm can be easily extended to incorporate the consecutive same shift constraints. In Step 4 of algorithm G, to determine if a workstretch is qualified for assignment, we add the additional check that it has not been assigned the current shift consecutively for a maximum allowable number of times. More precisely, replace Step 4 with

4'. $A \leftarrow \{1 \leqslant k \leqslant K$ k's leftmost slot is at position i *and* k has not been assigned a maximum number of shift j's consecutively$\}$.

The proof of correctness involves a simple extension of lemma 4.1. We will illustrate it by an example. Suppose we have the constraint that there should be no three consecutive shift 3's in a row. Suppose current shift $j = 2$ and after swapping σ^*k, i and σ^*k', i, workstretch k' violates that

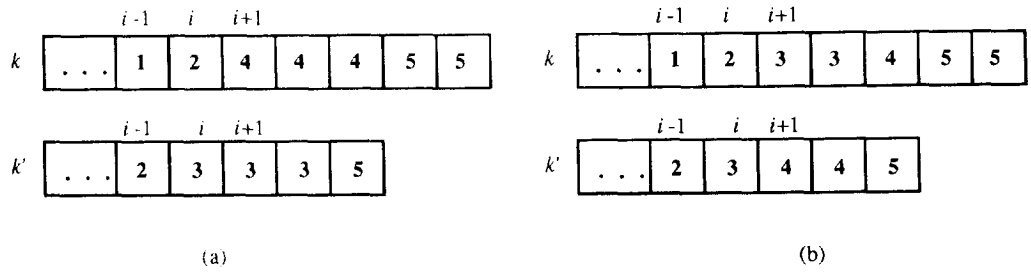

Fig. 4. How swapping can remove violation of the consecutive same shift constraint.

Table 1. Performance of branch and bound ($I = 7$ and $J = 3$)

K	ϵ	S	Max nodes expanded	Avg nodes expanded
10	0.1	7	555	45
10	0.2	15	2040	82
10	0.5	37	151	39
10	0.8	60	28	17
20	0.1	15	16444	232
20	0.2	30	26262	222
20	0.5	75	105	76
20	0.8	120	44	33
40	0.1	30	572187	2113
40	0.2	60	213590	600
40	0.5	150	194	152
40	0.8	240	79	62

constraint, as shown in Fig. 4(a). Then, by further swapping, that constraint will be satisfied under all possible circumstances, as shown in Fig. 4(b).

6.2. *Handling spare demands*

If there are spare supply of workers on some day(s), we will assign spare slots to some dummy shifts. The determination of which dummy shift numbers to assign such that monotonicity is preserved under all circumstances turns out to be a difficult problem under the greedy framework we proposed. Several variants of this problem have been discussed in another paper [16]. There, if we restrict the structure of workstretches, then the greedy method proposed can be extended so that, instead of considering only leftmost slots for assignments, we have to consider slots which could be assigned dummy shifts followed by a proper shift. It remains an open question whether monotonic CSAP with spare supply is polynomially solvable in general. To date, we can solve it by incorporating our greedy procedure into a branch-and-bound algorithm as a bounding procedure, the details of which is presented in [17]. Experiments on random instances of CSAP using this approach yield encouraging results, summarized as follows.

Recall that I is the planning period, J is the number of shifts, K is the number of workstretches. Let S denote the number of spare supply units. An instance of monotonic CSAP is generated as follows. Randomly generate a $K \times I$ cyclic show-up schedule and for each day, distribute the S spare units in proportion to the number of working slots†. The demands for all J shifts are then randomly generated based on the cyclic schedule and distribution of sparse units. In the first set of experiments, we fix $I = 7$, $J = 3$ and vary K and S. For each pair of K and S, we randomly generate 1000 input instances which include both feasible and infeasible instances. Table 1 shows the results of the experiment. The experiment indicates that our branch and bound algorithm works hardest when the ratio of spare units to working slots (denoted ϵ) is between 0.1 and 0.2. In those cases, the maximum number of nodes over all instances expanded grows exponentially with S, but the average number of nodes is still bounded by a lower-degree polynomial of S. In all other cases, both the maximum and average number of nodes expanded are polynomial relative to K, I, J and S.

†In the real world setting, this is often the case.

Table 2. Performance of branch and bound ($K = 20, I = 7$ and $\epsilon = 0.1$)

J	Max nodes expanded	Avg nodes expanded
1	188	139
3	2129	223
5	61874	737
10	91156	1551
20	242535	2444
40	247343	3412
100	285831	3617

We next investigate the effect of varying J while fixing $K = 20, \epsilon = 0.1$ and $I = 7$. Again for each J, we generate and test 1000 random instances. Table 2 shows the experimental results. Here, we learn that the maximum number of nodes generated by branch and bound grows quadratically with K, I, J and ϵ, while the average number of nodes grows linearly.

6.3. *Handling non-monotonic shift changes*

In our experience, if the shift changes are *almost* monotonic, then algorithm G can be used as a fast heuristic to generate a first-cut schedule. This schedule is then subject to local improvement schemes such as 2-opt on the assignments for each day of the planning period to eliminate any shift change constraint violations. The resulting schedule is usually quite good. Otherwise, shift changes are likely to be far from being monotonic. Such problems have been shown to be NP-hard above and can be efficiently tackled either by constraint satisfaction [18] or by a minimum-cost network flow model with fixed charges [17].

7. CONCLUSION

We studied the manpower shift scheduling problem which is an interesting problem among the operations research community. Particularly, we considered a problem which arises in the scheduling of airport ground crews in servicing of flights. We proved that a restricted version of it is NP-hard. This observation allows us to justify techniques which have been applied in manpower scheduling systems such as implicit enumeration and heuristic search. We showed that the case of monotonic shift changes is polynomially solvable for both non-cyclic and cyclic schedules with the same computational time complexity. We showed how the algorithm developed could be extended to solve more complex variants of the manpower shift scheduling problem.

Our contribution marks a step in formally analyzing the complexity of scheduling problems which embody desirable constraints such as monotonicity of shift changes. There remains many open questions when shift changes are non-monotonic. Other future research topics include proving the computational complexities of and designing efficient algorithms for other variants of the manpower scheduling problem as well as related resource scheduling problems.

Acknowledgements—I wish to thank Osamu Watanabe for interesting discussions and ideas contributed. Much appreciation also to the anonymous referees for suggestions of improvement to the paper.

REFERENCES

1. F. Glover and C. McMillan, The general employee scheduling problem: An integration of MS and AI. *Comput. Ops Res.* **13,** 563–573 (1986).
2. J. Tien and A. Kamiyama, On manpower scheduling algorithms. *SIAM Review* **24,** 275–287 (1982).
3. K. R. Baker and M. J. Magazine, Workforce scheduling with cyclic demands and day-off constraints. *Mgmt Sci.* **24,** 161–167 (1977).
4. R. N. Burns and M. W. Carter, Work force size and single shift schedules with variable demands. *Mgmt Sci.* **31,** 599–607 (1985).
5. R. N. Burns and G. J. Koop, A modular approach to optimal multiple-shift manpower scheduling. *Ops Res.* **35,** 100–110 (1987).
6. J. G. Morris and M. J. Showalter, Simple approaches to shift, day off and tour scheduling problems. *Mgmt Sci.* **29,** 942–950 (1983).
7. P. Carraresi and G. Gallo, A multi-level bottleneck assignment approach to the bus drivers' rostering problem. *Eur. J. Ops Res.* **16,** 163–173 (1984).

8. S. Martello and P. Toth, A heuristic approach to the bus driver scheduling problem. *Eur. J. Ops Res.* **24,** 106–117 (1986).
9. N. Balakrishnan and R. T. Wong, A network model for rotating workforce scheduling problem. *Networks* **20,** 25–32 (1990).
10. G. Koop, Multiple shift workforce lower bounds. *Mgmt Sci.* **34,** 1221–1230 (1988).
11. C. M. Khoong and H. C. Lau, ROMAN: An integrated approach to manpower planning and scheduling. In *Computer Science and Operations Research: New Development in Their Interfaces* (Edited by O. Balci, R. Sharda and S. Zenios), pp. 383–396. Pergamon Press, Oxford (1992).
12. C. M. Khoong, H. C. Lau and L. W. Chew, Automated manpower rostering: Techniques and experience. *Int. Trans. Opl. Res.* **1,** 353–361 (1994).
13. S. Even, A. Itai and A. Shamir, On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.* **5,** 691–703 (1976).
14. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA (1979).
15. C. M. Khoong, A simple but effective heuristic for workshift assignment. *Omega* **21,** 393–395 (1993).
16. H. C. Lau, Manpower scheduling with shift change constraints. *Lecture Notes in Comput. Sci. (Proc. ISAAC'94)* **834,** 616–624 (1994).
17. H. C. Lau, Combinatorial approaches for hard problems in manpower scheduling. Submitted (1994).
18. H. C. Lau, Preference-based scheduling via constraint satisfaction. In *Optimization Techniques and Applications (Proc. ICOTA'92)* (edited by K. H. Phua *et al.*), pp. 546–554. World Scientific (1992).