Manpower Scheduling with Shift Change Constraints

LAU HOONG-CHUIN

1. Introduction

In service organizations which operate round-theclock, workers are often scheduled to work on multiple shifts. Examples are nurses in hospitals, ground crews in airports, and operators in telephone companies. In these organizations, the scheduling of manpower resources is a critical management function. Manpower scheduling problems (MSP) are concerned with the construction of schedules for workers or teams of workers in order to meet the time-varying workloads and to satisfy a set of constraints imposed by the management, the labour union and the government. Glover and McMillan\(^5\) gives a good survey of the common manpower scheduling problems.

Manpower scheduling (also referred to as rostering) is a main research topic in operations research. Recently, Tien and Kamiyama\(^2\) proposed an integer programming framework for solving MSP in general. In their framework, MSP is decomposed into three sub-problems—allocation, offday assignment and shift assignment. Allocation computes the demands, i.e. the number of workers needed for each shift in each day so that the time-varying workloads can be met. Offday assignment assigns offdays on the schedule in order that workers get enough rest between work and that demands can be met. Shift assignment completes the schedule by assigning shifts to non-offday slots subject to demands and the given shift assignment constraints.

Common approaches for solving shift assignment problems include heuristics\(^{9,33}\), network optimization\(^{11,37}\) and integer programming\(^{32,37}\). One common type of shift assignment constraints is the shift change constraints which govern the permissible patterns of shift changes that a worker can be given from one day to the next so that he can maintain a healthy biological clock. For instance, it is permissible to change from a morning shift to an afternoon shift, but not from an evening shift to a morning shift since there is not enough rest hours in between work. In this paper, we consider the shift assignment problem subject to shift change constraints. We call it the Changing Shift Assignment Problem or CSAP.

In our research\(^9,11\), we investigate the complexity of CSAP with different kinds of demands and shift change constraints. Motivated by the result that CSAP is NP-hard even in very restricted domains, we seek to find sub-problems that have real-world implications which are polynomial-time solvable.

2. Preliminaries

We explain the terms of reference. The scheduling period is the number of days for which manpower scheduling is performed. Shifts are numbered 1, 2, ... and 0 denotes an offday. A schedule is a matrix where rows represent workers and columns represent days of the scheduling 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. 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.
A demand matrix gives the number of workers required in each shift on each day of the scheduling 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 feasible schedule is a schedule with all slots assigned which (1) satisfies the demand matrix; (2) for any 2 adjacent slots in the schedule, the shift change is satisfied. Figure 1 gives an example of terms explained.

As in most literature, we assume that schedules are cyclic. In other words, for any worker, if he follows row w of the schedule in the current scheduling period, then he will follow row w+1 in the next scheduling period, and the rows wraparound. This 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 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 $W$ = number of workers, $I$ = number of days of the scheduling period, $J$ = number of shifts and $K$ = derived number of workstretches. We assume that $K$ is of the same order of magnitude as $W$. Let $D = I \times J$ demand matrix, where $D_{ij}$ is the number of workers required to work shift $j$ on day $i$. Let $S = W \times J$ show-up schedule matrix. Let $\delta = J \times J$ shift change matrix, where $\delta_{ij}$ may change to shift $j_i$ or 0 otherwise. Let $\sigma$ = feasible schedule. Then, CSAP is an NP search problem whose input is the tuple $(W, I, J, D, S, \delta)$ and output is $\sigma$ or fail.

Definition 2.1. A shift change matrix is monotonic if it is upper-triangular consisting of all ones.

The above definition has the following real world motivation. In industry, a worker usually works shifts which start no earlier than the day before so that he gets enough rest in between. Thus, if shifts are ordered by their start times,

![Fig. 1](image-url) (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.
a feasible schedule must contain workstretches which are monotonically non-decreasing sequences. Such schedule can be constructed using a monotonic shift change matrix.

**Definition 2.2.** A demand matrix $D$ is slack with respect to a show-up schedule $S$ if there exists a day $i$ such that the sum of demanded workers $\sum_{j=1}^{n}D_{i,j}$ is less than the supply of workers on day $i$ in $S$. The difference indicates the number of spare workers on day $i$. If there are zero spares on all days, then $D$ is said to be exact.

In industry, if demands are slack, the spare workers are assigned either extra offdays or some meaningful shifts so that shift change constraints are preserved. To give the user such flexibility, we introduce a special shift type called the $*$-shift (wildcard shift) and add the appropriate number of them into the demand matrix so that the demand becomes exact. A $*$-shift may be assigned to any slot, provided that shift change between its left and right adjacent slots is satisfied.

This paper proceeds as follows. First, we prove that the decision problem associated with CSAP is NP-complete in general. Next, we show that CSAP with a monotonic shift change matrix and exact demand matrix (or MC-CSAP) can be solved by a greedy method which is nearly optimal. Finally, we consider CSAP with monotonic shift change and slack demand matrices (or MS-CSAP) under two realistic offday distribution patterns: (1) where workers work fixed-length stretches between offdays, which is a common phenomenon; and (2) where at least one of the days in the scheduling period have zero demands, typifying a day where the organization closes its operations. We show that both problems are polynomial-time solvable.

### 3. NP-Completeness of CSAP

In this section, we consider the decision problem of CSAP, CSAP (D), which asks whether a feasible schedule exists given the input $(W, I, J, D, S, \theta)$. Clearly, CSAP (D) is in NP. We will show the hardness of CSAP (D) by a polynomial many-one reduction from 3SAT.

Let $U=\{U_1, U_2, \ldots, U_n\}$ be a set of Boolean variables. A truth assignment for $U$ is a function $t: U \rightarrow \{\text{True}, \text{False}\}$. A clause $C$ over $U$ is a disjunction of literals over $U$, 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 $U$ is represented by a set of clauses $F=\{C_1, C_2, \ldots, C_m\}$ over $U$ with three literals per clause. We say that $F$ is satisfiable if there exists a truth assignment for $U$ that simultaneously satisfies all clauses in $F$. 3SAT is defined as follows:

**INSTANCE:** A set $U$ of boolean variables $(U_1, U_2, \ldots, U_n)$ and a 3CNF represented by $F=\{C_1, C_2, \ldots, C_m\}$ over $U$.

**QUESTION:** Is there a truth assignment for $U$ such that $F$ is satisfiable?

**Lemma 3.1.** For any 3CNF $F$ of $n$ variables $m$ clauses, there exists a 3CNF $F'$ of $n$ variables and $m'$ ($m' \leq 4m$) clauses such that,

1. $F'$ is satisfiable iff $F$ is satisfiable,
2. for all $i=1, \ldots, n$, $p_i=\bar{p}_i$, where $p_i$ (resp. $\bar{p}_i$) is the number of occurrences of $U_i$ (resp. $\bar{U}_i$) in $F'$, and
3. $m' \leq P$, where $P=\sum_{i=1}^{n}p_i$.

We call such 3CNF a normalized 3CNF.

**Proof.** Let $q_i$ (resp. $\bar{q}_i$) denote the number of occurrences of $U_i$ (resp. $\bar{U}_i$) in $F$. If $q_i \geq \bar{q}_i$, we add $q_i - \bar{q}_i$ number of clauses of the form $(U_i \lor U_j \lor \bar{U}_i)$, otherwise we add clauses $\bar{q}_i - q_i$ number of clauses of the form $(U_i \lor U_j \lor U_i)$. The desired $F'$ is $F$ plus at most $3m$ clauses since there are exactly $3m$ literals in $F$. Clearly, these additional clauses do not affect the satisfiability of $F$, thus $F'$ is satisfiable iff $F$ is satisfiable. Furthermore, since each clause has three literals and each variable $U_i$ occurs $2p_i$ times ($p_i$ times for each of $U_i$ and $\bar{U}_i$) in $F'$, $3m' = 2P$; hence $m' \leq P$.

**Theorem 3.1.** CSAP (D) is NP-complete, even for fixed $I \geq 5$, non-cyclic schedule with fixed workstretch lengths, exact demand matrix and upper-triangular shift change matrix.

**Proof.** Let $F$ be an instance of 3SAT which has been normalized. Suppose $F$ has $n$ variables and $m$ clauses. Then the corresponding instance of CSAP (D) is constructed as follows (see Fig. 2 (a)). To simplify writing, the ranges of $i, j$ and $k$ are respectively $\{1, \ldots, n\}, \{1, \ldots, p_i\}$ and $\{1, \ldots, m\}$.

2. Define the following shifts:
   - shifts $x_{ij}, \overline{x_{ij}}, y_{ij}, w_{ij}$, for all pairs of $i$ and $j$;
   - shifts $c_k$, for all $k$; and
   - shifts $u, v, w, z, \theta$.

Shift $x_{ij}$ (resp. $\overline{x_{ij}}$) corresponds to the $j$th
triangular matrix (see Fig. 2(b)):  
- for all pairs of \( i \) and \( j \),  
  \[ u \rightarrow v_{ij}, u_{ij} \rightarrow x_{ij}, u_{ij} \rightarrow \bar{x}_{ij}, x_{ij} \rightarrow y, \bar{x}_{ij} \rightarrow \bar{y}, \]
  \( x_{ij} \rightarrow 0, \bar{x}_{ij} \rightarrow \theta, \]
  \( w_{ij} \rightarrow x_{ij}, w_{ij} \rightarrow \bar{x}_{ij} \), where \( j' = j - 1 \) (if \( 2 \leq j \leq p \)) and \( j' = p \) (if \( j = 1 \));  
- for all \( i, j, k \),  
  \( x_{ij} \rightarrow c_k \) (resp. \( \bar{x}_{ij} \rightarrow c_k \)) if \( U_i \) (resp. \( \bar{U}_i \)) occurs the \( j \)th time in clause \( k \);  
- \( y \rightarrow z \); and  
- all other shift changes are set to 0.

The above definition of \( \delta \) forces the following conditions needed for the reduction:  
1. All shift \( v \)'s are in TR and \( w \)'s are in FR.  
2. All shift \( c \)'s and \( \theta \)'s are in \( 1K \) and \( y \)'s are in FR.  
3. For all pairs of \( i \) and \( j \), shift \( x_{ij} \) is in TR if \( \bar{x}_{ij} \) is in FR. Consider any pair \( i, j \). If \( x_{ij} \) is in TR, then it has to be preceded by \( u_{ij} \). Thus, \( u_{ij} \) cannot precede \( \bar{x}_{ij} \). Thus, \( \bar{x}_{ij} \) must be in FR. The reverse is obvious.  
4. For all \( i \), if there exists a \( j \) such that shift \( x_{ij} \) is in TR, then all other \( x_{i,j'} \)'s \((j' = 1, \ldots, p)\) are in TR. Consider any \( i \) and w.l.o.g suppose \( x_{ij} \) is in TR. From the previous condition, \( x_{ij} \) is in FR and thus preceded by \( w_{ij} \); since \( x_{ij} \) can be preceded only by \( v_{ij} \) or \( w_{ij} \), \( x_{ij} \) must be in the TR. This in turn implies that \( \bar{x}_{ij} \) is in FR, and the argument repeats.  
5. Shift \( x_{ij} \) (resp. \( \bar{x}_{ij} \)) precedes \( c_k \) only if \( U_i \) (resp. \( \bar{U}_i \)) occurs in clause \( C_k \) and sets \( C_k \) to \( True \).  

Clearly the reduction can be done in polynomial time. We claim that \( F \) has a satisfying assignment if and only if the constructed CSAP (D) instance has a feasible schedule. Assume \( F \) has a satisfying truth assignment \( t \). Define the feasible schedule as follows. For each \( i \) such that \( t(U_i) = True \), assign all shift \( x_{ij} \)'s to column \( 3 \) of TR, and all shift \( \bar{x}_{ij} \)'s to column \( 3 \) of FR. The reverse occurs for each \( t(U_i) = False \). For each clause \( C_k \), assign shift \( c_k \) to be adjacent to any \( x_{ij} \) whose corresponding literal occurs in \( C_k \). This is always possible since every clause has at least one true literal. As there are more true literals than clauses, shift \( \theta \)'s are used to absorb the remaining true literals. Conversely, given a feasible schedule, we assign \( U_i \) to \( True \) if one (and thus all) \( x_{ij} \) shifts are in TR, and assign \( U_i \) to \( False \) otherwise. For every shift \( c_k \) in column \( 4 \), suppose it is preceded by a shift \( x_{ij} \) in column \( 3 \). Then, we know that the literal \( U_i \) is \( True \) and sets the clause \( C_k \) to \( True \). Thus, \( F \) is satisfiable. □

We have deliberately used many shifts to

occurrence of the literal \( U_i \) (resp. \( \bar{U}_i \)) in \( F \). Shift \( c_k \) corresponds to the \( k \)th clause in \( F \). The rest are filler shifts.  
3. Define a 2P×5 show-up schedule matrix \( S \). Let the top and bottom half number of rows be called the True Region (TR) and False Region (FR) respectively. The regions are named so because column 3 of them will contain shifts corresponding to the true and false literals respectively. Let column 1 of FR and column 5 of TR be assigned offdays.  
4. Define the exact demand matrix \( D \) as shown in Fig. 2 (b) so that all the shift \( u \)'s will be assigned to column 1, \( v \)'s and \( w \)'s to column 2, \( x \)'s and \( \bar{x} \)'s to column 3, \( c \)'s, \( \theta \)'s and \( y \)'s to column 4, and \( z \)'s to column 5 of the schedule respectively.  
5. To improve readability, we use the notation \( a \rightarrow b \) to mean \( \delta_{ab} = 1 \). Define the shift change matrix \( \delta \) as the following upper-

occurrence of the literal \( U_i \) (resp. \( \bar{U}_i \)) in \( F \). Shift \( c_k \) corresponds to the \( k \)th clause in \( F \). The rest are filler shifts.  
3. Define a 2P×5 show-up schedule matrix \( S \). Let the top and bottom half number of rows be called the True Region (TR) and False Region (FR) respectively. The regions are named so because column 3 of them will contain shifts corresponding to the true and false literals respectively. Let column 1 of FR and column 5 of TR be assigned offdays.  
4. Define the exact demand matrix \( D \) as shown in Fig. 2 (b) so that all the shift \( u \)'s will be assigned to column 1, \( v \)'s and \( w \)'s to column 2, \( x \)'s and \( \bar{x} \)'s to column 3, \( c \)'s, \( \theta \)'s and \( y \)'s to column 4, and \( z \)'s to column 5 of the schedule respectively.  
5. To improve readability, we use the notation \( a \rightarrow b \) to mean \( \delta_{ab} = 1 \). Define the shift change matrix \( \delta \) as the following upper-

occurrence of the literal \( U_i \) (resp. \( \bar{U}_i \)) in \( F \). Shift \( c_k \) corresponds to the \( k \)th clause in \( F \). The rest are filler shifts.  
3. Define a 2P×5 show-up schedule matrix \( S \). Let the top and bottom half number of rows be called the True Region (TR) and False Region (FR) respectively. The regions are named so because column 3 of them will contain shifts corresponding to the true and false literals respectively. Let column 1 of FR and column 5 of TR be assigned offdays.  
4. Define the exact demand matrix \( D \) as shown in Fig. 2 (b) so that all the shift \( u \)'s will be assigned to column 1, \( v \)'s and \( w \)'s to column 2, \( x \)'s and \( \bar{x} \)'s to column 3, \( c \)'s, \( \theta \)'s and \( y \)'s to column 4, and \( z \)'s to column 5 of the schedule respectively.  
5. To improve readability, we use the notation \( a \rightarrow b \) to mean \( \delta_{ab} = 1 \). Define the shift change matrix \( \delta \) as the following upper-

occurrence of the literal \( U_i \) (resp. \( \bar{U}_i \)) in \( F \). Shift \( c_k \) corresponds to the \( k \)th clause in \( F \). The rest are filler shifts.  
3. Define a 2P×5 show-up schedule matrix \( S \). Let the top and bottom half number of rows be called the True Region (TR) and False Region (FR) respectively. The regions are named so because column 3 of them will contain shifts corresponding to the true and false literals respectively. Let column 1 of FR and column 5 of TR be assigned offdays.  
4. Define the exact demand matrix \( D \) as shown in Fig. 2 (b) so that all the shift \( u \)'s will be assigned to column 1, \( v \)'s and \( w \)'s to column 2, \( x \)'s and \( \bar{x} \)'s to column 3, \( c \)'s, \( \theta \)'s and \( y \)'s to column 4, and \( z \)'s to column 5 of the schedule respectively.  
5. To improve readability, we use the notation \( a \rightarrow b \) to mean \( \delta_{ab} = 1 \). Define the shift change matrix \( \delta \) as the following upper-
ease discussion. One could replace $y$ and $x$ by $u, v$, $w$ by $x_0, w_0$, thereby reducing the number of shifts needed to $2P + m + 2 = 4m + 2$.

4. Greedy Algorithm for ME-CSAP

We describe a greedy algorithm $G$ to solve ME-CSAP, i.e. CSAP with monotonic shift change matrix and exact demand matrix, regardless of offday distribution. Essentially, $G$ assigns shifts in increasing shift numbers. All workstretches are assigned contiguously from left to right. The leftmost (resp. rightmost) slot of a workstretch refers to its first (resp. last) unassigned slot, and the tail of a workstretch refers to the sequence of slots from its leftmost slot to its rightmost slot. For each assignment, we greedily pick the workstretch with the longest tail (ties broken arbitrarily).

Henceforth, let $\sigma_k$ denote the $k$th workstretch of $\sigma$; $\sigma_{k,i}$ denote a slot in workstretch $k$ at day (position) $i$; $\sigma_{k,i-1}$ and $\sigma_{k,i+1}$ denote the slots to the left and right of $\sigma_{k,i}$ respectively, considering wraparound. Let $l(k)$ denote the position of the leftmost unassigned slot of $k$.

Suppose $j$ is the current shift to be assigned. Let $B$ be the set of days where at least one shift $j$ has not yet been assigned in the schedule. We say that a workstretch is potentially assignable at position $i$ to shift $j$ if its leftmost slot position is $i$ and $i$ is in $B$. Let $A$ be the set of workstretches that are potentially assignable at their respective leftmost slot positions to shift $j$. Algorithm $G$ is given as follows:

```
procedure G:
Step 1. for $j = 1$ to $J$ do
Step 2. $B \leftarrow \{1 \leq i \leq l/day \ i \ has \ some \ unassigned \ shift \ j\}$;
Step 3. $A \leftarrow \{1 \leq k \leq K | \sigma_k \ is \ potentially \ assignable \ at \ position \ l(k) \ to \ shift \ j\}$;
Step 4. while $B \neq \emptyset$ do
Step 5. if $A = \emptyset$ return fail;
Step 6. choose $k \in A$ such that $\sigma_k$ has the longest tail;
Step 7. assign $j$ to $\sigma_{k,l(k)}$;
Step 8. update $A$ and $B$;
Step 9. endwhile
Step 10. endfor
Step 11. return $\sigma$;
```

To prove that greedy works, we introduce the notion of dominance.

**Definition 4.1.** Given a partial schedule $\sigma$ such that workstretches $k$ and $k'$ are potentially assignable at position $i$ to an unassigned shift $j$, $k$ is said to dominate $k'$ with respect to position $i$ and shift $j$ if assigning $j$ to $\sigma_{k,i}$ always leads to a feasible solution whenever assigning $j$ to $\sigma_{k',i}$ instead does.

**Lemma 4.1.** (Swapping Lemma) Let $j(1 \leq j \leq J)$ be the current shift. Let $\sigma$ be the partial schedule constructed by $G$ so far. Let $k$ be any workstretch in $A$ that has the longest tail. Then, $k$ dominates all workstretches with respect to $l(k)$ and $j$.

**Proof.** For simplicity, let $i = l(k)$. We prove by an adversary argument as follows. Suppose $\sigma^*$ is a hypothetical feasible schedule extended from $\sigma$ such that shift $j$ resides in $\sigma^*_{k,i}$ instead of $\sigma_{k,i}$. We show how to construct another feasible schedule extended from $\sigma$ such that shift $j$ resides in $\sigma^*_{k,i}$ instead of $\sigma^*_{k,i}$. First, observe that:

1. Since we schedule in non-decreasing shift numbers, $\sigma^*_{k,i} > j$;
2. By monotonicity, $\sigma^*_{k,i-1} \leq j$ and $\sigma^*_{k,i+1} \leq j$; and
3. Workstretch $k$ is at least as long as $k'$ from position $i$ onwards. Reason:

   a. If $\sigma_{k,i}$ is also a leftmost slot, then the fact that $G$ did not pick $k'$ means that $\sigma_{k,i}$ is at least as long as $\sigma_{k,i}$.

   b. Otherwise, by monotonicity, slots $\sigma^*_{k,i} > j$ have to be assigned $j$ also. If $\sigma_{k'}$ were longer than $\sigma_{k}$ from position $i$ onwards, then $k'$ would have a longer tail than $k$ and hence would have been chosen by $G$ instead of $k$, a contradiction.

Thus, we have the scenario as shown in Fig. 3(a). We swap the contents of $\sigma^*_{k,i}$ and $\sigma_{k,i}$. Surely, the content of $\sigma^*_{k,i}$ (i.e. $j$) may be moved to $\sigma^*_{k,i}$. The reverse is possible if $\sigma^*_{k,i} > \sigma_{k,i}$ or $\sigma^*_{k,i}$ is a rightmost slot; otherwise, it means $\sigma^*_{k,i} > \sigma_{k,i}$, and we apply swapping recursively. By doing so, $k'$ will eventually reach its rightmost slot before $k$ does, since the latter is at least as long. Since the swapping does not violate monotonicity, the lemma holds.

**Theorem 4.1.** Let $x$ be an instance of ME-CSAP. $G$ returns a feasible schedule for $x$ if and only if $x$ has a feasible schedule.

**Proof.** If $G$ exits successfully, then all slots in $\sigma$ would have been assigned shifts in a monotonic fashion. Thus, $\sigma$ is a feasible schedule. Conversely, if a feasible solution exists, it suffices to show that after $j(0 \leq j \leq J)$ for-
Fig. 3 Examples of dominance. Shaded slots represent the leftmost slots. Suppose workstretches $k$ and $k'$ are both potentially assignable at position $i$ to shift $j$. In (a), $k$ dominates $k'$ because $k$ has a longer tail. In (b), $k$ dominates $k'$ because $k$ has a longer tail and a shorter head.

iterations, the algorithm produces a partial monotonic schedule which meets demands for shifts 1 to $j$. This claim may be proved by induction on the loop index $j$ as follows. For $j = 0$, the partial schedule is the show-up schedule, and the claim obviously holds. Assume that after iteration $j-1(j-1<J)$, the claim holds. In iteration $j$, a leftmost slot always exists by the induction hypothesis. By the Swapping Lemma, $G$ always picks a dominating workstretch for assignment and thus maintains the feasibility of the partial schedule.

We briefly comment on time complexity. Note that the lower-bound worst-case time complexity of any algorithm for solving ME-

CSAP is $\Omega(KI)$, since every slot has to been assigned once. The complexity of algorithm $G$ is given as follows. The matrix sum of $D$ is at most $W \times I = O(KI)$. Since each while-iteration consumes one unit of demand, the total number of while-iterations $= O(KI)$. Each while-iteration is $O(K)$. Hence, the worst-case complexity of $G$ is $O(\max(K^2I, II))$. However, by careful implementation using ordinary heaps, we can achieve an amortized complexity of $O(\max(K^2I, KI \log K, II))$. Assuming that $I$ and $J$ are small compared with $K$, the dominating amortized complexity is $O(KI \log K)$, which is a factor $O(\log K)$ from the optimal.

5. MS-CSAP with Fixed-Length Workstretches

We switch our attention to CSAP with slack demands or MS-CSAP. MS-CSAP is intuitively more difficult than ME-CSAP because we cannot associate a fixed value to the *-shifts.

In this section, we consider MS-CSAP where workstretches have equal lengths. An example of such schedule is given in Fig. 4. Such a schedule is geometrically simple in that it belongs to the class of schedules where none of the workstretches is longer than another at both ends. We term such class of schedules doubly-jagged schedules. It turns out that MS-CSAP with doubly-jagged schedules can be solved greedily by extending the definition of potential assignability. Consequently, MS-CSAP with fixed-length workstretches can be solved greedily.

Again, consider assigning workstretches from left to right in increasing shift numbers. Let $j$ denote the current shift number and suppose there is an unassigned shift $j$ on day $i$. Redefine potential assignability as follows:

**Definition 5.1.** A workstretch $k$ is said to be potentially assignable at position $i$ to shift $j$ if:
1. $\sigma_{k,i}$ is unassigned;
2. there is no unassigned shift $j$ on days $l(k), \ldots, i-1$; and
3. there is at least one unassigned *-shift for each day $l(k), \ldots, i-1$.

One may verify that, at any one time, a workstretch is potentially assignable to shift $j$ at at most one position. Let $p(k)$ denote the potentially-assignable position of workstretch $k$. The head of workstretch $k$ is defined as the sequence of slots from $\sigma_{k,p(k)}$ to $\sigma_{k,p(k)-1}$ and the tail is the sequence of slots from $\sigma_{k,p(k)}$ to its rightmost slot. Algorithm AG (augmented greedy) is same as G, except the following step:

Step 7. Assign *-shift (s) to the head of $k$ and assign $j$ to $\sigma_{k,p(k)}$.

This greedy approach still works, due to the following corollary of the Swapping Lemma:

**Corollary 5.1.** Given two workstretches $k$ and $k'$ potentially assignable at some position $i$ to current shift $j$, $k$ dominates $k'$ with respect to $i$ and $j$ if (see Fig. 3(b)):
1. $k'$'s tail is longer than or equal to $k$'s; and
2. $k'$'s head is shorter than or equal to $k$'s.

**Proof.** Use an adversary argument similar to the proof of the Swapping Lemma. Suppose $j$ has been assigned to $\sigma_{k,i}^*$ instead of $\sigma_{k,i}$. Then, the head slots of $\sigma_{k'}^*$ are all assigned *-shifts, while the head slots of $\sigma_{k}^*$ and slot $\sigma_{k,i}^*$ must be assigned shifts greater than $j$ or *-shifts. By swapping the contents of $\sigma_{k}^*$ and $\sigma_{k'}^*$ from position $l(k)$ onwards, we can derive a feasible schedule which has $j$ assigned to $\sigma_{k,i}^*$ instead of $\sigma_{k,i}$, and the corollary holds.

**Theorem 5.1.** Let $x$ be an instance of MS-CSAP with a doubly-jagged show-up schedule. AG returns a feasible schedule for $x$ if and only if $x$ has a feasible schedule.

**Proof.** Consider any point in the execution of AG. Suppose the unassigned portion of the
schedule is doubly-jagged. Let $j$ be the current shift. Then, for any 2 workstretches which are potentially assignable at some position $i$ to shift $j$, one must dominate the other with respect to $i$ and $j$ by Corollary 5.1. Moreover, it is clear that a workstretch which is not potentially assignable at $i$ to $j$ is always dominated by one which is. Thus, any workstretch in $A$ with the longest tail dominates all workstretches. Since AG picks a dominating workstretch for assignment, the schedule remains feasible, and since that workstretch is the longest, the unassigned portion of the schedule remains doubly-jagged. We can prove inductively that the theorem holds. □

**Corollary 5.2.** Let $x$ be an instance of MS-CSAP with fixed-length workstretches. AG returns a feasible schedule for $x$ if and only if $x$ has a feasible schedule.

6. **MS-CSAP with Fixed Offdays**

We now consider MS-CSAP where there exists a day in which all slots have been assigned offdays. We rotate the columns of the schedule so that that day becomes day 0 of the scheduling period. The resulting schedule becomes non-cyclic since workstretches do not wrap-around. Non-cyclicity coupled with double-jagged workstretches allow MS-CSAP to be solved with a matching-based algorithm M which has a lower worst-case time complexity than G or AG.

Basically, M assigns shifts from day 1 to $I-1$ (i.e., column by column). The tail of a workstretch now refers to the sequence of slots from the current day to its rightmost slot. For each column, arrange workstretches in non-increasing order of tail length and assign shifts in non-increasing order upwards, assigning *-shifts whenever monotonicity is violated. We will show that, in this way, M always matches the *-shifts to the longest possible workstretches, which is a sufficient condition for constructing a feasible schedule if one exists. M is given as follows and **Fig. 5** gives an example of its execution.

**procedure M:**

1. **Step 1.** for $i = 1$ to $I-1$ do
2. **Step 2.** for workstretch $k$ in increasing order of tail length do
3. **Step 3.** let $j$ be the largest unassigned shift;
4. **Step 4.** if $j \geq \sigma_{k,i}$ then assign $j$ to $\sigma_{k,i}$;
Fig. 6 (a) shows a feasible schedule in which column \( i-1 \) is sorted but column \( i \) is not; (b) shows the schedule after sorting column \( i \); (c) shows the minimal canonical schedule.

Step 5. \textbf{else if} there is an unassigned \( * \)-shift \textbf{then} assign \( * \) to \( \sigma_{h,i} \);

Step 6. \textbf{else return fail ;}

Step 7. \textbf{endfor}

Step 8. \textbf{endfor}

Step 9. \textbf{return} \( \sigma \);

To show the correctness of \( M \), we show how to convert from an arbitrary feasible schedule into the schedule constructed by \( M \) without violating feasibility. For every assigned \( * \)-shift, define its \textit{shift number} to be the shift number of its preceding slot. If the preceding slot is an offday, then its shift number is defined to be \( 1 \). A column is said to be \textit{sorted} if shifts are assigned in that column in order of tail lengths (i.e. for any 2 workstretches, the longer workstretch always has the smaller shift number).

\textbf{Definition 6.1.} A \textit{canonical schedule} is a schedule such that every column is sorted.

\textbf{Lemma 6.1.} Let \( x \) be an instance of MS-CSAP with fixed offdays. If there exists a feasible schedule for \( x \), then there exists a feasible canonical schedule for \( x \).

\textbf{Proof.} By induction on the columns of the schedule. Consider a feasible schedule \( \sigma^* \) such that columns 1 to \( i-1 \) \((1 \leq i \leq I)\) are sorted. We show that there exists a feasible schedule such that columns 1 to \( i \) are sorted. At column \( i \), the workstretches are either \textit{surviving} workstretches (i.e. those whose starting day is before day \( i \)) or \textit{new} workstretches (i.e. those whose starting day is at day \( i \)). Since all workstretches are doubly-jagged, new workstretches are always longer than surviving workstretches. And since the preceding slots of new workstretches are all offdays (shift 0), column \( i-1 \) remains sorted even considering those new workstretches. Now pick the longest workstretch \( k \). Suppose the slot \( \sigma_{k,i} \) does not contain the smallest shift number, and the smallest shift number is assigned to workstretch \( k'(i.e. \sigma_{k',i}) \). Then, we can swap the corresponding slots between \( k \) and \( k' \) from position \( i \) onwards (as shown in Fig. 3 (a)) without violating the feasibility of the schedule. Perform the process iteratively on the next longest workstretch until all workstretches have been examined. It is clear that column \( i \) of \( \sigma^* \) is now sorted (see Fig. 6(a) and (b)).

Given a feasible canonical schedule \( \sigma \), we can construct the \textit{minimal} feasible canonical schedule by 'bubbling' up the \( * \)-shifts to the longer workstretches as much as possible (see Fig. 6 (c)):

\textbf{procedure} bubble-up: \textbf{for} \( i=2 \) \textbf{to} \( I-1 \) \textbf{do}

Step 2. arrange workstretches in decreasing order of tail lengths;

Step 3. \textbf{for} all adjacent workstretch pairs \( k \) and \( k' \) \textbf{do}

Step 4. \textbf{if} \( \sigma_{k,i} = * \) \textbf{and} \( \sigma_{k,i} \geq \sigma_{k',i-1} \) then swap the contents of \( \sigma_{k,i} \) and \( \sigma_{k',i} \);

Step 5. \textbf{endfor}

Step 6. \textbf{endfor}

\textbf{Lemma 6.2.} Every iteration of \( M \) constructs a sorted column with the \( * \)-shifts assigned to the longest possible workstretches.

\textbf{Proof.} Consider any arbitrary iteration \( i \). Clearly, \( M \) constructs a sorted column \( i \). Now suppose that column \( i \) is not minimal. Then, there exists a \( * \)-shift which can be bubbled up. That is, there is a step in \( M \) in which a non-\( * \)-shift could have been assigned but a \( * \)-shift was assigned instead. But this is a contradiction since \( M \) only assigns a \( * \)-shift if it fails to assign a non-\( * \)-shift.

\textbf{Theorem 6.1.} Let \( x \) be an instance of MS-CSAP with fixed offdays. Then \( M \) returns a feasible schedule for \( x \) if and only if \( x \) has a feasible schedule.

\textbf{Proof.} If \( x \) has a feasible schedule, then by \textbf{Lemma 6.1}, there exists a canonical schedule which is feasible and this schedule can be minimized by applying the bubbling operation described above. By \textbf{Lemma 6.2}, we know that \( M \) always produces that minimal canonical schedule.

We comment briefly on the time complexity of \( M \). In order to assign in increasing order of tail length (Step 2), we add a preprocessing step between Steps 1 and 2 which sorts the workstretches in tail length order. This operation takes \( O(K \log K) \) time. Steps 3 to 6 take constant time. Thus, the worst-case time complexity of \( M \) is \( O(KI \log K) \).
7. Conclusion

We considered CSAP, the manpower shift scheduling problem with shift change constraints. We showed that CSAP is NP-hard even when the scheduling period is fixed, schedule is non-cyclic, demand is exact and the shift change matrix is upper-triangular. We also presented polynomial algorithms to solve three sub-problems of CSAP. These sub-problems have applications in the real world, as our experience indicates.

Several open problems arise from this research. Firstly, we ask if there exists a time-optimal algorithm for solving CSAP with monotonic shift change and exact demands. In Ref. 10, we presented a pseudo-polynomial time algorithm based on branch-and-bound to solve CSAP with monotonic shift change and slack demands. Its worst-case time complexity is exponential in the number of \( s \)-shifts. We ask whether a polynomial time algorithm can be designed. Finally, we believe that there exists other shift change constraints more general than monotonicity for which CSAP is polynomial.

Shift assignment problems with constraints apart from shift change constraints are also interesting to consider. For instance, problems where workers are allowed to state their preferences for working different shifts on different days. We hope that more research will be conducted for manpower scheduling problems for they are theoretically interesting and promise high economic values.

Acknowledgments. I wish to thank Osamu Watanabe for many suggestions of improvement to the paper.

References


(Received September 7, 1994)
(accepted January 12, 1995)

Lau Hoong-Chuin obtained his BSc and MSc in Computer Science from the University of Minnesota at Minneapolis, USA in 1987 and 1988 respectively. Currently, he is a doctoral candidate at the Dept. of Computer Science, Tokyo Institute of Technology, Japan. His current research interests include design and analysis of algorithms, particularly for scheduling problems in operations research and artificial intelligence.