1996

Combinatorial approaches for hard problems in manpower scheduling

Hoong Chuin LAU
Singapore Management University, hclau@smu.edu.sg

Journal of the Operations Research
Society of Japan
Vol. 39, No. 1, March 1996

COMBINATORIAL APPROACHES FOR HARD PROBLEMS IN MANPOWER SCHEDULING

Hoong Chuin Lau
Tokyo Institute of Technology

(Received March 24, 1994; Revised January 30, 1995)

Abstract Manpower scheduling is concerned with the construction of a workers' schedule which meets demands while satisfying given constraints. We consider a manpower scheduling problem, called the Change Shift Assignment Problem(CSAP). In previous work, we proved that CSAP is NP-hard and presented greedy methods to solve some restricted versions. In this paper, we present combinatorial algorithms to solve more general and realistic versions of CSAP which are unlikely solvable by greedy methods. First, we model CSAP as a fixed-charge network and show that a feasible schedule can be obtained by finding disjoint paths in the network, which can be derived from a minimum-cost flow. Next, we show that if the schedule is tableau-shaped, then such disjoint paths can be derived from an optimal path cover, which can be found by a polynomial-time algorithm. Finally, we show that if all constraints are *monotonic*, then CSAP may be solved by a pseudo-polynomial backtracking algorithm which has a good run-time performance for random CSAP instances.

1 Introduction

Manpower scheduling problems (MSP) arise in large manufacturing or service organizations which operate in multiple shifts. Examples of applications are scheduling nurses in hospitals, ground crews in airports, and operators in telephone companies. Manpower scheduling is concerned with the generation of a workers' schedule consisting of shift and offday assignments to meet the time-varying manpower demands while satisfying a set of constraints imposed by the management, labour union and the government.

Manpower scheduling is a complex problem in general and thus an active topic in operations research (e.g. [1, 2, 4, 7, 8]). Recently, Tien and Kamiyama [11] proposed an integer programming framework for solving the MSP in general. In their framework, MSP is decomposed into a pipeline of three sub-problems – allocation, offday scheduling and shift assignment. Allocation inputs the temporal manpower demands and outputs the manpower demands in terms of the number of workers required during each shift in each day. Offday scheduling assigns offdays to the schedule subject to manpower demands and certain offday scheduling constraints. Shift assignment then completes the schedule by assigning shifts to the working days subject to manpower demands and shift assignment constraints.

We are mainly concerned with the shift assignment problem (SAP). Many interesting variations of the shift assignment problem have been studied. For example, Carraresi and Gallo [5] considered the problem of finding an even balance of shifts over the planning period; Martello and Toth [10] 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 [1] applied network optimization coupled with Lagrangian relaxation to solve SAP subject to a host of scheduling constraints.

In this paper, we consider yet 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 manpower demands and the *shift change constraints.* The shift change constraints define the permissible shift changes from one day to the next so that workers 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. As an application, CSAP arises in the scheduling of airport ground crews to service in-bound flights for their next journey. Here, shift types corresponds to 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 are fairly complicated in order to generate cost-effective schedules to meet the fluctuating flight requirements. This is even more so considering the fact that full-time and part-time workers do not work the same set of shifts.

In [9], we proved that CSAP is NP-hard, even in very restricted domains. In that paper, we also presented cases of CSAP which can be solved by greedy methods. This paper is a sequel which presents the more difficult cases, for which greedy methods are unlikely to be successful. The paper proceeds as follows. Section 2 gives the preliminaries and formally defines CSAP. Section 3 introduces the fixed-cost network flow problem and show that CSAP can be reduced to that problem. This enables us to solve CSAP by known techniques for the fixed-cost network flow problem. Section 4 shows that if the schedule is tableau-shaped, then CSAP can be solved by a polynomial-time algorithm for finding an optimal path cover. Section 5 presents a pseudo-polynomial backtracking algorithm to solve CSAP with monotonic shift change constraints. Section 6 gives the conclusion.

2 Preliminaries

An instance of CSAP is defined by 6 parameters, the number of workers W, the *scheduling period* I, the number of shifts J, the $J \times I$ *demand matrix* D, the $W \times I$ *show-up schedule* U and the $J \times J$ *shift change matrix* δ. Shifts are numbered from 1 to J and shift 0 denotes an offday. The demand matrix gives the manpower requirement, i.e. $D_{j,i}$ is the number of workers required to work shift j on day i. The show-up schedule is a blank schedule with offdays already assigned. The unassigned entries are known as *slots* and the number of slots on each day gives the *supply* of workers for that day. Let DS_i and DD_i denote the total supply of and demand for workers on day i respectively. We assume that for all i, $DD_i \leq DS_i$ (since otherwise, there is no feasible schedule). The shift change matrix is a boolean square matrix which defines the shift change permission, i.e. $\delta_{j_1,j_2} = 1$ if shift j_1 may be followed by shift j_2 and 0 otherwise. We assume that an offday may precede or follow any shift. The output of CSAP is a *feasible schedule* σ in which all slots are assigned to shift numbers 1 to J such that (1) all demands are met; and (2) the shift change is permitted between any 2 adjacent slots. If there is no feasible schedule, the primitive **fail** must be returned. To simplify presentation of algorithms, we assume that $\sigma = U$ initially and thus U is not explicitly used. A *partial schedule* is one which has not been fully assigned. As in most literature, we assume that a schedule is used in a *cyclic* fashion. In other words, if a certain worker is on row w of the schedule in the current scheduling period, then he will be on row $w+1$ in the next scheduling period, and the rows wrap around. This allows the schedule to be used indefinitely and also guarantees fairness of shift distribution among workers over time. Cyclicity may take on other forms, but in this paper, we will consider only the above case. Hence, we can represent a schedule as a list of *workstretches* where each workstretch is a contiguous sequence of slots delimited by offdays. Let K denote the number of workstretches of the given show-up schedule, σ_k denote the kth workstretch; $\sigma_{k,i}$ denote a slot on day (position) i (note that there may be more than one such slot); $\sigma_{k,i-1}$ and $\sigma_{k,i+1}$ denote the slots to the left and right of $\sigma_{k,i}$ respectively, considering wraparound. We further assume that K is of the same order of magnitude as W. One can verify that the matrix representation can be converted to the workstretch representation and vice versa in polynomial time.

Fig. 1 gives an illustration of the terms explained. In the figure, (a) is a demand matrix; (b) is a show-up schedule; (c) is a shift change matrix; (d) is a feasible schedule derived from (b) which satisfies the demand matrix subject to the shift change matrix; and (e) is the workstretch representation of (d). The numbers in brackets represent the *start days* of the respective workstretches.

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: CSAP Input and Output

A shift change matrix is said to be *monotonic* if it is upper-triangular consisting of all ones. Monotonic shift changes has the following real-world motivation. In industry, a worker usually starts work at time no earlier than that of the day before so that he gets enough rest in between. Thus, if we order the shifts by their start times, a feasible schedule is composed of workstretches which are monotonically non-decreasing sequences. This explains the significance of the monotonic shift change matrix.

A demand matrix D is said to be *slack* with respect to the show-up schedule if there exists a day i such that $DD_i < DS_i$. Let $DSL_i = DS_i - DD_i$ be the number of slack units on day i. Let $TS = \sum_{i=1}^{I} DS_i$, $TD = \sum_{i=1}^{I} DD_i$, and $S = TS - TD$ be the total supply, total demand and total slack respectively. If S=0 then demand is said to be *exact*. In industry, if demands are slack, the spare workers are assigned either extra offdays or shifts such that shift change constraints are not violated. So in the same way, we introduce a special shift type called the $*$-shift (wildcard shift) and add the appropriate number of them into the schedule. In other words, we create a demand for DSL_i units of $*$-shifts on day i, for all i. Note that A $*$-shift can be assigned to any slot.

The following complexity result about CSAP is known:

Theorem 2.1. *([9]) CSAP is NP-hard, even for $I = 5$, if D is exact and δ is upper-triangular.*

3 CSAP with Slack Demand and Arbitrary Shift Change

In this section, we will solve the most general version of CSAP, i.e. where the demand matrix D is slack and shift change matrix δ is any arbitrary matrix. This problem is NP-hard by Theorem 2.1. We convert CSAP into a fixed-charge network and apply any known algorithm to find a minimum-cost flow. We then prove that any minimum-cost flow induces a feasible schedule iff one exists.

3.1 Fixed-Charge Network Flow Problem

Let V be a set of nodes and E be a set of directed arcs on V. Each arc $(u,v) \in E$ has: (1) a variable cost c_{uv} = cost per unit flow from u to v, (2) a fixed cost h_{uv} = cost of using the arc, (3) a lower bound lb_{uv} = minimum amount that must flow on the arc, and (4) a capacity ub_{uv} = maximum amount that can flow on the arc. Let $b(u)$ represent the supply/demand quantity of node u. If $b(u) > 0$, then u is a supply node; if $b(u) < 0$, it is a demand node. We assume that $\sum_{u \in V} b(u) = 0$. Let x_{uv} be the amount of flow on arc (u,v) and y_{uv} be a 0/1 variable that takes the value 1 if the flow on arc (u,v) is positive and 0 otherwise.

The fixed-cost network flow problem (FCNP) is an optimization problem whose input is a fixed-cost network $N = (V,E,c,h,lb,ub,b)$ and output is a minimum-cost flow together with its cost z. It may be formulated by the following mathematical program:

$$\begin{array}{lll} \text{minimize} & z = \sum_{(u,v)\in E} c_{uv}x_{uv} + \sum_{(u,v)\in E} h_{uv}y_{uv} & & (1) \\ \text{subject to} & \sum_{(u,v)\in E} x_{uv} - \sum_{(v,u)\in E} x_{vu} = b(u), & \text{for } u \in V & (2) \\ & lb_{uv} \le x_{uv} \le ub_{uv}y_{uv}, & \text{for } (u,v) \in E & (3) \\ & y_{uv} \in \{0,1\};\ x_{uv} \ge 0, & \text{for } (u,v) \in E & (4) \end{array}$$

The cost function (1) consists of two terms: the first term accounts for the total flow's variable cost, and the second term accounts for the fixed cost. Constraints (2) represent the flow-balance constraints; constraints (3) ensure that if $y_{u,v}$=0, no flow will occur between nodes u and v.

It is known that FCNP is NP-hard, even when the arc costs are constant and the underlying graph is bipartite [6]. However, due to its wide applications in production planning, transportation and communication network design, facilities location, VLSI design, etc, efficient algorithms have been developed, particularly for layered networks. Discussions on the computational efficiency of such algorithms may be found in [6] and its references. In this paper, we assume one such efficient algorithm is available and use it as a subroutine.

3.2 Algorithm F

To solve CSAP, we propose the following algorithm:

```
procedure F($I,J,K,D,U,\delta$):
1.  compute $DS,DD,DSL,TS,TD,S$;
2.  construct the fixed-cost network $N = (V,E,c,h,lb,ub,b)$;
3.  obtain a minimum-cost flow by solving FCNP with input $N$;
    let $f_{min}(N)$ and $z_{min}$ denote the flow and its cost respectively;
4.  if $z_{min} > TS$, return fail;
5.  obtain a feasible schedule $\sigma$ from $f_{min}(N)$ (see Lemma 3.3 below);
6.  return $\sigma$.
```

Fig. 2 gives an illustration of algorithm F. In this example, $K=3$, $I=3$, $J=2$ and shift change is monotonic. 2(a) shows a demand matrix and (b) shows a show-up schedule. From (a) and (b), we derive $DD=[1,3,2]$, $DS=[2,3,2]$ and $DSL=[1,0,0]$. The network constructed is as shown in (c). Numbers in nodes represent the supply/demand quantities. To simplify drawing, the edges from layer 3 to 1 are not drawn. 2(d) shows a minimum-cost flow of (c). Numbers above edges represent flow values. 2(e) shows a feasible schedule which is derived from (d).

Shift\Day	1	2	3
1	0	1	1
2	1	2	1

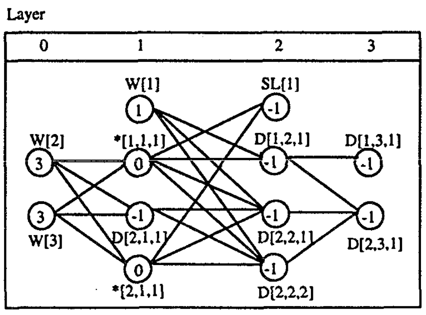
(a)

Workstretch\Day	1	2	3
1	0	-	0
2	-	-	-
3	-	-	-

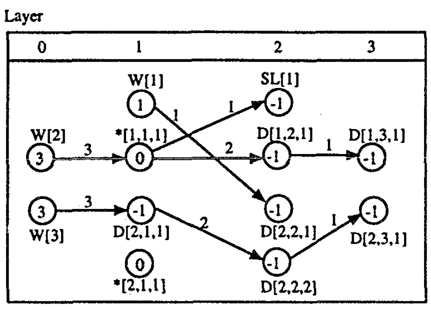
(b)

Workstretch\Day	1	2	3
1	0	2	0
2	*	1	1
3	2	2	2

(e)



(c)



(d)

Fig. 2: Illustration of algorithm F

We will discuss how the fixed-cost network N is constructed (Step 2) and then prove the correctness of our algorithm by showing that any minimum-cost flow induces a feasible schedule provided that the cost does not exceed the total supply TS.

3.3 Network Construction

A fixed-cost network is constructed as follows (see Appendix A for the precise algorithmic description):

1. For each workstretch k, create a *W-node* $W[k]$ whose supply quantity is $|\sigma_k|$.
2. For $1 \leq j \leq J$ and $1 \leq i \leq I$, create $D_{j,i}$ number of *D-nodes* $D[j,i,1], D[j,i,2], \cdots, D[j,i,p]$ (where $p = D_{j,i}$). Each D-node has demand quantity 1.
3. For $1 \leq j \leq J$ and $1 \leq i \leq I$, create DSL_i number of *$*$-nodes* $*[j,i,1], *[j,i,2], \cdots, *[j,i,q]$ (where $q = DSL_i$). Each $*$-node has demand quantity 0.
4. For $1 \leq i \leq I$, create one *slack node* $SL[i]$ whose demand quantity is DSL_i. Slack nodes act as sinks which absorb the flows out of $*$-nodes.

The nodes are arranged into a cyclic layered network, where layer i $(0 \leq i \leq I)$ contains:

1. W-nodes representing workstretches that start on day $i+1$;
2. D-nodes and $*$-nodes associated with day i (i.e. nodes whose second index is i); and
3. the slack node associated with day $i-1$, i.e. $SL[i-1]$.

Edges represent permissible shift changes. Since the first (i.e. leftmost) slot of a workstretch can be assigned to any shift, each W-node in layer i is connected to all D-nodes and $*$-nodes in layer $i+1$. D-nodes and $*$-nodes in adjacent layers are connected if shift changes are permitted. All $*$-nodes in layer i are connected to the slack node in layer $i+1$. All edges are assigned unit fixed costs, except the edges from $*$-nodes to slack nodes, which are assigned zero fixed costs. All variable costs are set to zeros.

3.4 Proof of Correctness

Let $f(N)$ denote a flow of the given network N. A node in N is said to be *in-active* if at least one of its incoming edges has a positive flow value. Similarly, a node is said to be *out-active* if at least one of its outgoing edges has a positive flow value. If neither condition occurs, then a node is said to be *dead*. Ignoring all dead nodes, it is clear that $f(N)$ is a directed layered graph such that,

1. all W-nodes, D-nodes and slack nodes are in $f(N)$;
2. the number of $*$-nodes in $f(N)$ is at least S, since the sum of demand quantities of slack nodes is S and each $*$-node can supply at most one unit of flow to its slack node; and
3. z (the cost of $f(N)$) equals the total *number* of edges in $f(N)$ which have fixed costs.

In a flow, we say that a *fork* occurs at a node u if u has more than one outgoing edges. Similarly, a *join* occurs at a node u if u has more than one incoming edges. A path is *disjoint* if none of the nodes along the path has a fork or a join.

Definition 3.1. *A disjoint path flow is a flow such that:*

1. *the edges of the flow can be partitioned two sets of edges: (a) disjoint paths connecting the W-nodes, D-nodes and $*$-nodes and (b) edges from $*$-nodes to slack nodes; and*
2. *every $*$-node supplies exactly one unit to its slack node.*

Definition 3.2. *A column of a schedule in matrix representation is said to be proper if every slot in that column has been assigned a shift and the column assignment satisfies the demand for that day. Likewise, a partial schedule is said to be i-proper if the columns 1 to i of the schedule are proper. Hence, a schedule is feasible if and only if it is I-proper.*

Let $\lambda = (I, J, K, D, U, \delta)$ be an instance of CSAP. Let N denote the network constructed by Step 2 of algorithm F. Then, the following lemma holds:

Lemma 3.3. *Every disjoint path flow of N induces a feasible schedule for λ iff λ has a feasible schedule.*

Proof. Let $f(N)$ denote any disjoint path flow of N. Then, $f(N)$ contains exactly K disjoint paths, since there are K W-nodes. Each disjoint path in $f(N)$ must originate from a W-node and end in a D-node or $*$-node. Consider the disjoint path which begins with an arbitrary W-node $W[k]$,

$$(W[k], D[j_1, i, p_1], D[j_2, i+1, p_2], ..., D[j_r, i+r-1, p_r]),$$

where $r = b(W[k])$. This represents a permissible shift assignment of workstretch k, where shift j_t is assigned to slot $\sigma_{k,i+t-1}$ for $1 \le t \le r$. Similarly, given a feasible schedule for λ, it is easy to construct a disjoint path flow for N, by associating each workstretch assignment to one disjoint path. □

Let $f_{min}(N)$ and z_{min} denote a minimum-cost flow of N and its cost respectively. Then the following two lemmas hold.

Lemma 3.4. $z_{min} \ge TS$.

Proof. $f_{min}(N)$ contains TD D-nodes and at least S $*$-nodes. Every such node has at least one incoming edge, and each such edge has a unit fixed cost. Since $TS = TD + S$ by definition, $z_{min} \ge TS$. □

Lemma 3.5. $z_{min} = TS$ *iff $f_{min}(N)$ is a disjoint path flow.*

Proof. If $f_{min}(N)$ is a disjoint path flow, then clearly $z_{min} = TS$. Now, suppose $z_{min} = TS$. Then, $f_{min}(N)$ must have TD D-nodes and exactly S $*$-nodes, and every such node has exactly one incoming edge. Therefore,

1. every $*$-node sends one unit of flow to its respective slack node;
2. joins cannot exist in $f_{min}(N)$;
3. forks cannot exist in $f_{min}(N)$.
   This may be shown by contradiction. We scan $f_{min}(N)$ from layer 0 and suppose a fork first occurs between layers $i-1$ and i. Since layers $0,...,i-1$ are fork and join free, we can construct an $i-1$-proper schedule. Thus, number of out-active nodes in layer $i-1$ = number of in-active nodes in layer i, and so a join must occur between layer $i-1$ and i, giving rise to a contradiction. □

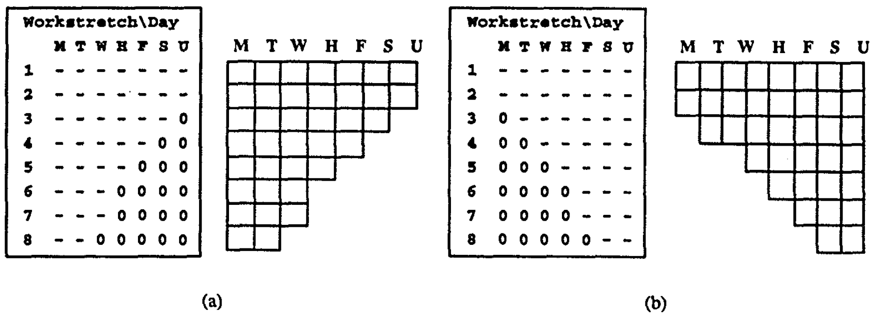
Theorem 3.6. *Algorithm F finds a feasible schedule iff one exists.*

Proof. If the algorithm returns a schedule, then the schedule is clearly feasible. Conversely, if a feasible schedule exists, by Lemma 3.3, there exists a disjoint path flow for the network constructed by Step 2 of F. By Lemma 3.4 and 3.5, any minimum-cost flow is a disjoint path flow. Hence Step 3 of F always returns a disjoint path flow, and Step 5 produces the corresponding feasible schedule. □

4 CSAP with Exact Demand and Arbitrary Shift Change

We now consider CSAP such that the demand matrix is exact. Further, we restrict workstretches to be non-cyclic and have equal start days or end days. This includes the case that all workers have the same offday(s). If we rearrange workstretches in non-decreasing order of their lengths, i.e., $|\sigma_k| \ge |\sigma_{k+1}|$, for $k = 1, ..., K-1$, then the schedule will be shaped like a tableau as shown in Fig. 3(a) or (b), with each row of the tableau representing one workstretch. We call this problem T-CSAP and show that it is polynomially solvable.

A *node-disjoint path cover*, or simply *path cover*, of a directed graph is a collection of node-disjoint paths (possibly having zero length) which covers all nodes of the graph. Given a directed graph G, a path cover always exists. An optimal path cover is one which has the least number of paths, and its size is denoted



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

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

Fig. 3: Tableau-shaped schedules

by $\zeta(G)$. The problem of finding an optimal path cover for an n-node directed acyclic graph is solvable in $O(n^{2.5})$ time [3].

Using the notion of path cover, we propose the following algorithm P to solve T-CSAP:

```
procedure P($I, J, K, D, U, \delta$):
1.  construct the corresponding network $N$ using Step 2 of algorithm F;
2.  apply any known algorithm to find an optimal path cover $\mu$ for $N$;
3.  if $\zeta(N) > K$ return fail;
4.  obtain a feasible schedule $\sigma$ from $\mu$;
5.  return $\sigma$.
```

We prove that algorithm P is correct. This can be achieved by showing that an optimal path cover always contains K disjoint paths and it induces a feasible schedule.

Let $\lambda = (I, J, K, D, U, \delta)$ be an instance of T-CSAP and N be the layered network constructed by Step 2 of algorithm F. Since all workstretches have equal start days, we remove all W-nodes from N. Clearly, layer 1 of the network contains K nodes. Since demand is exact, there there are neither $*$-nodes nor slack nodes in N. The resulting network is as shown in Fig. 4. In this figure, (a) shows the given tableau-shaped schedule; (b) shows the network constructed from (a); and (c) shows an optimal path cover corresponding to (b).

Since a path in N consists of at most one node from each layer and there are K nodes in layer 1, we get:

Lemma 4.1. *$\zeta(N) \geq K$.*

Lemma 4.2. *Let $\mu = (\mu_1, \mu_2, ..., \mu_K)$ be a path cover of size K of N. Let $|\mu_k|$ denote the number of nodes on path k. W.l.o.g, suppose $|\mu_k| \geq |\mu_{k+1}|$ for $k = 1, ..., K-1$. Then, $|\mu_k| = |\sigma_k|$, for $k = 1, ..., K$.*

Proof. Consider the non-increasing sequences $\{|\mu_1|, |\mu_2|, ..., |\mu_K|\}$ and $\{|\sigma_1|, |\sigma_2|, ..., |\sigma_K|\}$. Suppose k is the smallest index such that $|\mu_k| \neq |\sigma_k|$. If $|\mu_k| > |\sigma_k|$, then we may conclude that $DD_i > DS_i$ for $i = |\sigma_k| + 1$ to $|\mu_k|$, which is a contradiction because demand is exact. Similarly, if $|\mu_k| < |\sigma_k|$, then $DD_i < DS_i$ for $i = |\mu_k| + 1$ to $|\sigma_k|$, a contradiction also. Thus, such k does not exist. □

Theorem 4.3. *λ has a feasible schedule iff $\zeta(G) = K$.*

Proof. Suppose $\zeta(G) = K$ and μ is an optimal path cover of G. We construct a feasible schedule σ as follows. For each $1 \leq k \leq K$, the path μ_k, composed of a sequence of nodes $(D[j_1, 1, p_1], D[j_2, 2, p_2], ..., D[j_r, r, p_{|\mu_k|}])$, represents a permissible shift assignment of workstretch k such that shift j_t is assigned to slot $\sigma_{k,t}$ for $1 \leq t \leq |\mu_k|$. The assignment is always possible by Lemma 4.2. Since μ covers all nodes in G and all workstretches are assigned, σ is feasible. Conversely, given a feasible schedule σ, we apply the reverse procedure to obtain a path cover of size K. This has to be optimal by Lemma 4.1. □

5 CSAP with Slack Demand and Monotonic Shift Change

We next consider the subproblem of CSAP such that the demand matrix D is slack; and shift change matrix δ is monotonic. We call this M-CSAP.

It has been shown in [9] that a polynomial-time greedy algorithm can be used to solve M-CSAP if the given demand matrix is *exact*. Thus, a naive method to solve M-CSAP would be to consider all possible substitutions of $*$-shifts with actual shift numbers and apply the greedy method exhaustively. We call this method the *exhaustive greedy* method or EG, which we employ as a yardstick algorithm for evaluating the performance of our backtracking algorithm.

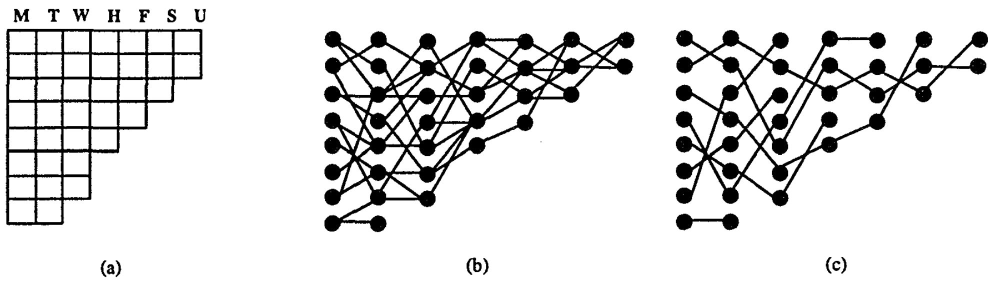


Fig. 4: Example of network constructed for tableau-shaped schedule

```
procedure EG($I,J,K,D,U$):
1.  for all possible exact demand $D'$ derived from $D$ do
2.    if there is a greedy feasible schedule $\sigma$ with demand $D'$ return $\sigma$;
3.  return fail.
```

Recall that S is the total number of slack units. The number of different ways to substitute $*$-shifts by shift numbers 1,..., J is J^S. Hence, the worst-case time complexity of EG is $O(J^S \cdot K^2 I)$ since the worst-case time complexity of the greedy algorithm has been shown to be $O(K^2 I)$ [9].

5.1 Algorithm BT

We propose a depth-first backtracking algorithm BT. BT assigns shifts in increasing order subject to two dominance criteria for pruning search space. All workstretches are assigned contiguously from left to right. Define the *leftmost* slot of a workstretch k, denoted $l(k)$, as its leftmost unassigned slot, and the *tail* as the sequence of slots from its leftmost slot to its rightmost slot.

Definition 5.1. *A workstretch is 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. it is possible to assign $*$-shift(s) to the slots $\sigma_{k,l(k)},\ldots,\sigma_{k,i-1}$, followed by assigning shift j to $\sigma_{k,i}$.

One can easily verify that a workstretch is potentially assignable at *at most* one position. Let $p(k)$ denote the potentially-assignable position of workstretch k. Define the *head* of workstretch k as the sequence of slots from $\sigma_{k,l(k)}$ to $\sigma_{k,p(k)-1}$.

Definition 5.2. *A workstretch k dominates another workstretch k' with respect to (position) i and (shift) j iff assigning j to $\sigma_{k,i}$ always leads to a feasible solution whenever assigning the same j to $\sigma_{k',i}$ does.*

We propose the two dominance criteria for the backtracking algorithm. Let j be the current shift (i.e. shifts 1 to $j-1$ have already been assigned). Let B be the set of days which have at least one unassigned shift j. For all $i \in B$, let C_i (cluster i) be the set of all workstretches k such that $p(k) = i$ and k is not dominated by other workstretches in C_i w.r.t. i and j. Let C be the set union of all such clusters.

Criteria 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. 5):*

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.

Criteria 2. *Suppose cluster C_i contains a workstretch k whose tail is longest among all workstretches in C. Then k dominates every workstretch in $C_{i'}$ with respect to i and j, for all $i' \in B$ such that $i' \neq i$.*

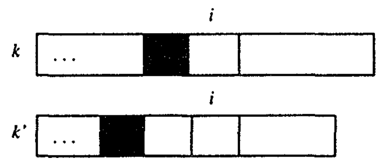


Fig. 5: Example of dominance. Black slots represent leftmost slots.

Based on these dominance criteria, we propose the following recursive algorithm BT. Assume that the input parameters I, J, K and D are global variables. BT has 2 input parameters j (current shift) and σ (current partial schedule), and 1 output parameter σ^* (output schedule).

```
procedure BT($j, \sigma, \sigma^*$):
1.   if $j > J$ then return 1;
2.   compute $B$;
3.   if $B = \emptyset$ then $done \leftarrow$ BT($j+1, \sigma, \sigma^*$);
4.   else
4.1.     compute clusters $C_i$ for all $i \in B$;
4.2.     select $i'$ such that $C_{i'}$ contains an element with longest-tail;
4.3.     for each $k$ in $C_{i'}$ until $done$
4.3.1.           assign *(s) to the head of $\sigma_k$ and assign $j$ to $\sigma_{k,i'}$;
4.3.2.           $done \leftarrow$ BT($j, \sigma, \sigma^*$);
4.3.3.           if not $done$ then (backtrack) undo Step 4.3.1;
4.4.     if $done$ then copy $\sigma$ to $\sigma^*$;
5.   endif
6.   if $done$ then return 1 else return 0.
```

Each state of the backtracking process either begins a new shift (Step 3) or assigns one current shift (Step 4). When $*$-shifts are assigned, we say that slack units are being *consumed*. Thus, in each state, 0 to $I-1$ slack units are consumed, by definition of potential assignability. When there are no more slack units, each cluster C_i must contain exactly one workstretch whose leftmost slot is at position i. In that case, BT collapses to the greedy algorithm presented in [9].

5.2 Worst-Case Computational Complexity

We show that BT is pseudo-polynomial with respect to S (number of slack units). Let T be the worst-case backtrack tree generated by BT for a given input instance of M-CSAP, where the nodes represent the states of recursion. We estimate the worst-case complexity of BT by counting the number of nodes in T. Define an *(i,j)-assignment* to be a series of assignments to the schedule to meet the demand for some shift j on some day i. Let $T(i,j)$ denote the subtree of T associated with an (i,j)-assignment.

Consider an arbitrary $T(i,j)$. Let v_0 be the root of $T(i,j)$. For each node v in $T(i,j)$, let $p(v)$ denote the number of slack units consumed in v. Clearly, $p(v_0) = 0$. For every non-leaf node v of T, let $deg(v)$ denote the branching factor (i.e. the number of child nodes) of v, and let $w_1, w_2, ..., w_{deg(v)}$ denote the child nodes of v. Then, for all non-leaf nodes v:

1. $deg(v) = I - p(v)$; and
2. $0 \le p(v) \le p(w_1) < p(w_2) < ... < p(w_{deg(v)}) \le I-1$.

The structure of $T(i,j)$ for $I = 4$ is as shown in Fig. 6.

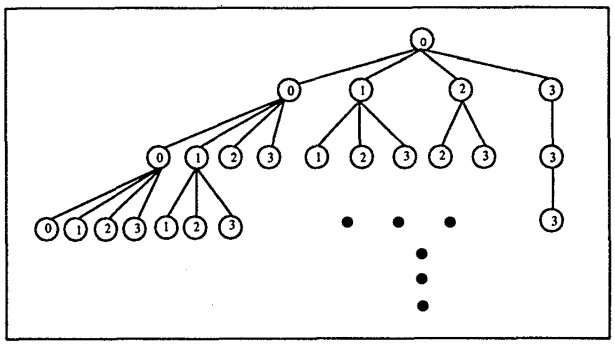


Fig. 6: Example of $T(i,j)$. Numbers in nodes show the slack units consumed.

Suppose the depth of $T(i,j)$ is d. For any leaf node v_d, the path from the root v_0 to v_d can be uniquely identified by a distinct non-decreasing sequence $(p(v_0), p(v_1), ..., p(v_d))$, The total number of slack units consumed by the path is given by $\sum_{i=0}^{d} p(v_i)$. Thus, the number of different paths in $T(i,j)$ which consume some n slack units is equivalent to the number of integer partitions of n into d or less parts such that each part does not exceed $I-1$, which is the coefficient of x^n of the following generating function.

Fact 1. *The generating function for integer partition into d or less parts such that each part does not exceed k is*

$$G(x) = (1 + x^1 + ... + x^d)(1 + x^2 + ... + x^{2d})...(1 + x^k + ... + x^{kd}) = \prod_{l=1}^{k} \sum_{m=1}^{d} x^{lm}.$$

Hence, the number of paths in T which consumes S slack units is given by the coefficient of x^S in the generating function

$$F(x) = \prod_{i=1}^{I} \prod_{j=1}^{J} \prod_{l=1}^{I-1} \sum_{m=1}^{D_{j,i}} x^{lm}.$$

As there is no known formula that computes integer partitions, we upper-bound the above generating function by the following:

$$F(x) = \left(\frac{1}{1-x}\right)^{I^2 J}.$$

Fact 2. *Given differentiable functions f and g over x such that $f(x) = g(x)^r$ for some integer constant r, then the nth derivative of f with respect to x may be expressed as*

$$f^n(x) = \sum_{i=0}^{n-1} \left[r\binom{n-1}{i} - \binom{n-1}{i-1} \right] f^i(x) \cdot \frac{g^{n-i}(x)}{g(x)}.$$

Fact 3. *If $g(x) = \frac{1}{1-x}$, then $g^n(0) = n!$.*

From facts 2 and 3, we get

$$F^n(x) = \sum_{i=0}^{n-1} \left[r\binom{n-1}{i} - \binom{n-1}{i-1} \right] F^i(x) \cdot (n-i)!$$

where $r = I^2 J$. Thus, the coefficient of x^n in $F(x)$ is given by

$$\frac{F^n(0)}{n!} = \sum_{i=0}^{n-1} \left[r\frac{n-i}{n} - \frac{i}{n} \right] \frac{F^i(0)}{i!}.$$

By induction, one can prove that the above recurrence has the closed form

$$\frac{F^n(0)}{n!} = \binom{r+n-1}{n}.$$

Relating back to BT, the total number of paths in T which consume S or less units is thus given by

$$P_{\leq S} = \sum_{n=0}^{S} \frac{F^n(0)}{n!} = \binom{I^2 J + S}{S} \leq \left(\frac{e(I^2 J + S)}{I^2 J}\right)^{I^2 J} \leq e^{I^2 J} \cdot \left(1 + \frac{S}{I^2 J}\right)^{I^2 J} \leq e^{I^2 J} \cdot O(S^{I^2 J}).$$

Hence, $P_{\leq S}$ is independent of K. Furthermore, if we assume that I and J are constants, then $P_{\leq S}$ is polynomial in S. Now the number of nodes in T is at most the depth of T times $P_{\leq S}$, which equals $O(KI) \cdot P_{\leq S}$. Every node represents one recursion step which incurs $O(K)$ time. Thus, the worst-case complexity of BT given S slack units, denoted $time_{BT}(S)$ is given by

$$time_{BT}(S) = O(e^{I^2 J} S^{I^2 J} \cdot K^2 I)$$

which is pseudo-polynomial assuming that I and J are constants. Comparing the time complexities of algorithms EG and BT, we may conclude that BT is superior to EG unless

$$\frac{S}{logS + 1} < I^2 \cdot \frac{J}{\log J}.$$

5.3 Experimental Results

We test the performance of BT with random instances of M-CSAP. Recall that I is the scheduling period, J is the number of shifts, K is the number of workstretches and S is the number of slack units. An instance of M-CSAP is generated as follows. Randomly generate a $K \times I$ cyclic show-up schedule and for each day, distribute the S slack units in proportion to the number of slots. (In the real world setting, this is often the case.) The demands for all J shifts are then randomly generated based on the cyclic schedule and distribution of slack units.

First, we fix $I = 7$, $J = 3$ and vary K and S. For each pair of K and S, we generate 1000 instances which include both feasible and infeasible instances. Table 1 shows the results of the experiment. The results indicate that our branch and bound algorithm works hardest when the ratio of slack units to 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.

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 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.

Table 1. $I = 7$ 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

Table 2. K=20, $I = 7$ and ϵ=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

6 Conclusion

In this paper, a sequel to [9], we have presented combinatorial algorithms to solve the more difficult cases of the Changing Shift Assignment Problem (CSAP). For the most general version, we proposed an algorithm based on network flows. One could have proposed other approaches such as branch and bound methods, but we believe that network-based approaches give much more insights into the operation and structure of the problem. Furthermore, network flows are well-studied and there exist efficient computational procedures which we can use rather than invent new ones. It remains an open problem whether there exists a polynomial-time algorithm for CSAP with slack demands and monotonic shift change constraints.

Acknowledgements

I would like to thank Osamu Watanabe for discussions and comments.

References

[1] N. Balakrishnan and R. T. Wong. A network model for rotating workforce scheduling problem. *Networks*, 20:25-32, 1990.

[2] L. Bianco, M. Bielli, A. Mingozzi, S. Ricciardelli, and M. Spadoni. A heursitic procedure for the crew rostering problem. *Euro. J. Ops. Res.*, 58:272–283, 1992.

[3] F. T. Boesch and J. F. Gimpel. Covering the points of a digraph with point-disjoint paths and its application to code generation. *J. Assoc. Comput. Mach.*, 24(2):192–198, 1977.

[4] R. N. Burns and G. J. Koop. A modular approach to optimal multiple-shift manpower scheduling. *Ops. Res.*, 35(1):100–110, 1987.

[5] P. Carraresi and G. Gallo. A multi-level bottleneck assignment approach to the bus drivers' rostering problem. *Euro. J. Ops. Res.*, 16:163–173, 1984.

[6] G. M. Guisewite and P. M. Pardalos. Minimum concave-cost network flow problems: Applications, complexity and algorithms. *Annals Ops. Res.*, 25:75–100, 1990.

[7] C. M. Khoong and H. C. Lau. ROMAN: An integrated approach to manpower planning and scheduling. In O. Balci, R. Sharda, and S. Zenios, editors, *CS & OR: New Development in Their Interfaces*, pages 383-396. Pergammon Press, 1992.

[8] M. M. Kostreva and K. S. Jennings. Nurse scheduling on a microcomputer. *Computers Ops. Res.*,

18(8):106–117, 1991.

[9] H. C. Lau. Manpower scheduling with shift change constraints. In *Proc. 5th Int'l. Symp. Algorithms and Computation (ISAAC)*, pages 616–624. Springer Verlag Lect. Notes Comp. Sci. (834), 1994. Full version in *Trans. Inf. Proc. Soc. Japan*, 36(5) 1995.

[10] S. Martello and P. Toth. A heuristic approach to the bus driver scheduling problem. *Euro. J. Ops. Res.*, 24(1):106–117, 1986.

[11] J. Tien and A. Kamiyama. On manpower scheduling algorithms. *SIAM Review*, 24(3):275–287, 1982.

Hoong Chuin Lau
Dept. Computer Science
Tokyo Institute of Technology
Meguro-ku, Ookayama
Tokyo 152, Japan
(hclau@cs.titech.ac.jp)

Appendix A: Construction of Fixed-Cost Network

In the following procedure, the ranges of variables are as follows: $1 \leq i \leq I, 1 \leq j \leq J, 1 \leq k \leq K, 1 \leq j_1 \leq J, 1 \leq j_2 \leq J, 1 \leq p \leq D_{j,i}, 1 \leq q \leq DSL_i, 1 \leq p_1 \leq D_{j_1,i_1}, 1 \leq q_1 \leq DSL_{i_1}, 1 \leq p_2 \leq D_{j_2,i_2}$, and $1 \leq q_2 \leq DSL_{i_2}$.

procedure `Construct-Network` (i.e. Step 2 of Algorithm F):

1. For all k, create W-node $W[k]$ and set $b(W[k]) = |\sigma_k|$.
2. For all j, i and p, create D-node $D[j,i,p]$ and set $b(D[j,i,p]) = -1$.
3. For all j, i and q, create $*$-node $*[j,i,q]$ and set $b(*[j,i,q])$=0.
4. For all i, create slack node $SL[i]$ and set $b(SL[i]) = DSL_i$.
5. For all j_1, j_2, p_1, p_2, q_1, q_2, and adjacent layers i_1 and i_2 (i.e. $1 \leq i_1 \leq I-1$ and $i_2 = i_1 + 1$), do the following: add arcs from $D[j_1,i_1,p_1]$ to $D[j_2,i_2,p_2]$ if $\delta_{j_1,j_2} = 1$; add arcs from $D[j_1,i_1,p_1]$ to $*[j_2,i_2,q_2]$ if $j_1 = j_2$; add arcs from $*[j_1,i_1,q_1]$ to $D[j_2,i_2,p_2]$ if $\delta_{j_1,j_2} = 1$; and add arcs from $*[j_1,i_1,q_1]$ to $*[j_2,i_2,q_2]$ if $j_1 = j_2$. For each such arc (u,v), set $lb_{uv} = 0$, $ub_{uv} = \infty$ and $h_{uv} = 1$.
6. For all k, j, i, p and q, add arcs from $W[k]$ to $D[j,i,p]$ and $*[j,i,q]$ if workstretch k starts on day i. For each such arc (u,v) where $u = W[k]$, set $lb_{uv} = 0$, $ub_{uv} = \infty$ and $h_{uv} = 1$.
7. For all j, i and q, add arcs from all $*[j,i,q]$ to $SL[i]$. For each such arc (u,v), set $lb_{uv} = 0$, $ub_{uv} = 1$ and $h_{uv} = 0$.