

The task assignment problem for unrestricted movement between workstation groups

Jonathan F. Bard · Lin Wan

© Springer Science + Business Media, LLC 2006

Abstract The purpose of this paper is to investigate the problem of assigning tasks to workers during their daily shifts. For a homogeneous workforce, a given set of workstation groups, and a corresponding demand for labor, the objective is to develop a disaggregated schedule for each worker that minimizes the weighted sum of transitions between workstation groups. In the formulation of the problem, each day is divided into 48 1/2-hour time periods and a multi-commodity network is constructed in which each worker corresponds to a unique commodity and each node represents a workstation group-time period combination. Lunch breaks and idle time are also included in the model.

Initial attempts to solve large instances with a commercial code indicated a need for a more practical approach. This led to the development of a reduced network representation in which idle periods are treated implicitly, and a sequential methodology in which the week is decomposed into 7 daily problems and each solved in turn. To gain more computational efficiency, a tabu search procedure was also developed.

All procedures were tested using data obtained from a U.S. Postal Service mail processing and distribution center. Depending on the labor category, anywhere from 3 to 28 workstation groups and up to 311 full-time and part-time workers had to be scheduled together. The results were mixed. While small problems could be solved to near-optimality with the integer programming approaches, tabu search was the best alternative for the very large instances. However, the excessive number of swaps needed to gain marginal improvements, undermined its effectiveness. Combining the two provided a good balance in most cases.

Keywords Multi-commodity network · Integer programming · Tabu search · Task assignment problem

This work was supported in part by the National Science Foundation under grant # DMI-0218701.

J. F. Bard (✉)

Graduate Program in Operations Research and Industrial Engineering, 1 University Station C2200, The University of Texas, Austin, Texas 78712
e-mail: jbard@mail.utexas.edu

L. Wan

Amazon.com 605 5th Ave. S, Seattle, WA 98104
e-mail: linwan@amazon.com

1. Introduction

In most organizations, personnel planning and scheduling problems are approached hierarchically. In the long run, the goal is to determine the size and composition of the permanent workforce. This is typically done by solving a shift scheduling problem to satisfy some percentage of the average daily demand (the specification of this percentage is a management decision that represents a tradeoff between cost and quality of service; see Bard, 2004). What results is sometimes called a *tour* or *bid job*, which specifies the weekly or monthly schedules for each employee. Included are the workdays, their length, the daily start times, and perhaps the lunch breaks.

At the weekly level, it is common to specify individual task assignments for each shift in fixed time increments. In many industries, this is an easy problem to solve because workers spend the majority of their day at a single location. When demand varies from one time period to the next or when different products are assembled in a flexible environment, it is necessary to reposition the workforce periodically to maximize the use of both equipment and manpower. However, this presents two difficulties. First, it is easier to supervise a stationary workforce than a mobile one, and second, it is impossible to avoid the loss of some productivity due to travel time and the reorientation that takes place after each move.

These issues give rise to what is called the *task assignment problem* (e.g., see Ernst et al., 2004)—the focus of this paper. The objective is to construct daily schedules for each member of the workforce that minimize the weighted sum of transitions between workstation groups (WSGs) while ensuring that all demand is satisfied.

In the next section, we highlight the relevant literature on personnel scheduling. In Section 3, we give a formal statement of the task assignment problem (TAP) along a complexity proof. The corresponding mathematical model is presented in Section 4 followed in Section 5 by a description of several solution methods. The first is based on an implicit treatment of idle time that leads to a simplified integer programming (IP) representation of the problem. The second involves decomposing the full problem into seven daily IP subproblems to obtain greater computational tractability. The third is a tabu search procedure that was developed with the goal of obtaining good feasible solutions in a short amount of time and makes use of a greedy heuristic or one of the previous two methods to provide a starting point. We conclude with a series of computational tests using data obtained from a U.S. Postal Service (USPS) mail processing and distribution center for several categories of labor. The largest group consisted of over 300 full-time and part-time workers, and up to 28 WSGs. The results showed that tabu search in combination with the greedy heuristic was not able to match the solution quality of the IP approaches but, on balance, was the best alternative for the very large data sets.

2. Literature review

Typical issues in personnel scheduling include tour construction, break assignments, leave management, and the use of part-time and casual labor. In tour construction, the first component is to find the optimal workforce size and the daily shift assignments (Aykin, 1996). A shift may vary in length depending on whether it is associated with a full-time or part-time employee, and may extend into the following day. The second part requires the specification of the days off (Burns and Carter, 1985). The number and characteristics of those days (weekend days, weekdays or combinations thereof) vary according to the organization and the type of industry in which it operates. A general consideration is that sufficient slack must be provided throughout the week so that the days-off requirement is satisfied for every worker. The third component is the lunch

break. All shifts, except for those that are shorter than a specific number of periods, require a break. Labor contracts determine the start time and the duration of this break. General practice is to create a break window for every shift – a set of consecutive periods during which a break may be given – and to assign a break within the window (Bechtold and Jacobs, 1990). Depending on the nature of the work being performed, the breaks could be staggered or coincident.

Jarrah et al. (1994) were the first to address the days-off scheduling and shift scheduling in a unified manner. They presented a new methodology for solving the large-scale combined shift and days-off scheduling problem when the labor requirements span less than 24 hours per day. They began with an IP formulation and then introduced a set of aggregate variables and related cuts to facilitate the computations. When the aggregate variables are fixed the original problem decomposes into seven subproblems (one for each day of the week) that are much easier to solve. Solutions were obtained for problems with up to 1,400 integer variables and 1,500 constraints.

Brusco and Jacobs (1998) presented a two-stage solution procedure for the restricted start time tour scheduling problem, which may be described as the determination of appropriate subsets of shift starting times for full-time and part-time employees, as well as the assignment of employees to tours associated with these starting times. To find solutions, they developed a construction/improvement heuristic and a three-stage procedure for reducing the density of the A -matrix associated with the technological constraints of the underlying IP model. Computational experience was presented for a large set of real-world problems that contained on the order of 1344 pure integer variables, 192 binary variables, and 867 constraints.

Given a permanent workforce, the midterm scheduling problem must be solved on a periodic basis, usually weekly or monthly, to provide individual schedules or rosters. For a review of the nurse rostering problem, see Burke et al. (2004) and Cheang et al. (2003); issues related to airline crew scheduling can be found in Dawid et al. (2001).

Variations of the basic personnel scheduling problem include non-homogeneous workforce in terms of skill, various labor requirements such as maximum work stretches, break definitions, off-days and off weekend policies, as well as maximum and minimum workforce constraints, start time regulations, and objective function definitions (e.g., see Nanda and Browne, 1992). Overtime allocation and personal preferences are also management concerns.

While much has been written on the classical assignment problem and its variants, little research exists on the type of assignment problem addressed in this paper. At the midterm planning level, Mukherjee and Gilbert (1997) considered the problem of scheduling instructors in executive development programs run by universities and other institutions. They developed a 0-1 integer programming formulation but the many restrictions and the dynamic nature of the environment led to an unsolvable model. As an alternative, they developed four heuristics based on Lagrangian relaxation. Under various conditions, each was shown to be fast, accurate and reasonably suitable for both random and real problems that required the scheduling of up to 547 classes.

Lewis et al. (1998) modeled an administrative office as a closed queuing network and developed an allocation scheme for grouping workers and tasks. The allocation component of the problem was formulated as a nonlinear integer program with the assumption that all tasks in the office were interdependent rather than independent or serial. The proposed solution approach first identified the bottleneck workstation and then allocated workers optimally.

Applications of the assignment problem somewhat related to our work can be found in the field of transportation. Hall and Lospitch (1996), for example, presented a multi-commodity network flow model for lane assignment on an automated highway, where the commodities represented trip destinations (i.e., exit onto ramps). A static formulation was given that did not consider the time distribution of demand. Apart from the normal network flow constraints, the model incorporated bundle constraints to account for traffic that enters, exits and passes through

each lane within a segment of the highway. The objective was to maximize total flow, subject to a fixed origin/destination pattern expressed on a proportional basis. Tests were conducted for highways with up to 80 segments, 20 destinations and 5 lanes.

Campbell and Diaby (2002) developed an assignment heuristic for allocating cross-trained workers to multiple departments at the beginning of a shift. Each worker had different qualifications with respect to each department. The problem was formulated as a variant of the generalized assignment problem with a concave objective function that measured department preferences. The authors presented a comparison of their linear approximation heuristic with a greedy approach and a Lagrangian heuristic.

Another generic problem that is closely related to ours is the multi-period assignment problem. Franz and Miller (1993) studied the problem of assigning medical residents to training rotations and clinic stints. The objective was to maximize the residents' schedule preferences while meeting the hospital's training goals. A decision support system was designed to review naturally occurring infeasibilities due to the complexity of the scheduling problem and to make decisions about altering conflicting constraints.

Aronson (1986) developed a branch and bound algorithm for the multi-period assignment problem by transforming it into a multi-commodity network flow problem. His model included the cost of assigning a person to an activity in each time period as well as the cost of transferring a person from one activity to another in successive time periods. Arc capacities prohibited the assignment of an activity to more than one person in each period while the pure minimum cost network flow structure limited the assignment of a person to no more than one activity in each time period. An exact algorithm was proposed to find solutions based on solving a relaxation of the multi-commodity network flow problem obtained by eliminating the mutual capacity constraints. The relaxation led to a series of shortest path subproblems whose solutions were used to establish branching rules and to provide a lower bound on the original objective function.

With regard to postal operations, most of the existing research has centered on tour construction. In an early study, Showalter et al. (1977) developed a simple building heuristic aimed at specifying shift start times, work center assignments, and mail class responsibility for each employee. Assignments were made subject to a number of constraints, including days off requirements, work center capacities, mail arrival volumes, and mail flow patterns through the system. For more recent work, see Bard et al. (2003), Berman et al. (1997), and Malhotra et al. (1992).

3. Problem description

The application that has motivated this work arises in USPS mail processing and distribution centers (P&DCs). These facilities are like high volume factories that run 24 hours a day, 7 days a week, and are staffed by a skilled workforce comprising full-time, part-time, and casual employees. Our analysis begins with a fixed workforce whose composition has been determined to meet long-term performance and budgetary goals. An implementation of the model developed for this purpose, called the Scheduling Optimization System (SOS), is now being used by the USPS at each of their P&DCs (see Bard et al., 2003).

Having solved the weekly staffing problem, each full-time regular (FTR) and part-time flexible (PTF) worker must be given a daily assignment of tasks for each day he or she is scheduled to work. Generally, these tasks are associated with a specific machine or workstation. It is desirable, but not mandatory, that each worker be assigned to a single machine for a full shift. In most cases, however, this is not possible because equipment schedules, which are derived from mail arrival

profiles, do not match shift lengths. Some operations might only be two or three hours long while others may run up to 12 hours.

A “good” assignment of tasks is characterized by as few switches among machines as possible; that is, one that minimizes some function of the total number of transitions over the day for each worker category. It is not possible to take transitions into account in the weekly staff-scheduling model that determines the daily shift for each worker because demand is specified by worker category only.

In defining the problem, we make the following assumptions.

1. Each shift longer than 6 hours includes a 1/2-hour lunch break.
2. Demand requirements for a particular category of worker (e.g., clerks, mail handlers) can be partitioned into a finite number of groups so the problem decomposes by worker category.
3. All demand must be satisfied.
4. Shifts can spill over from one day to the next so the problem does not decompose by day.
5. The assignment of workers to a specific WSG in numbers greater than the demand in any particular period represents idle time.

In the solution to the weekly scheduling problem, a sufficient number of shifts are generated to guarantee that assumption 3 can be met. Also, if there is only one WSG, there is no need to make assignments because all transitions between machines in a WSG have zero cost. In the more general case, when there are, say, m WSGs, we have the following result.

Theorem 1. *The task assignment problem is NP-hard in the strong sense.*

Proof: We will show that a variant of the set-covering problem (SCP), which is NP-hard in the strong sense (Garey and Johnson, 1979), can be polynomially transformed into the task assignment problem (TAP). Recall that in the standard set-covering problem, there are u items that form the set S and v subsets $S_i (i = 1, \dots, v)$ of S comprising one or more of the items such that $\cup_{i=1,v} S_i = S$, where each subset S_i has cost c_i . The objective is to choose a finite number of subsets so that the entire set S is included in their union and the overall cost is minimized.

The variant that we are interested in, denoted by SCP-SOS, is one in which the subsets are divided into n groups and only one subset can be selected from each group, where $v \geq n$ and the groups do not have to be mutually exclusive. Here, SOS is a reference to special ordered set type 1 constraints. To show that SCP-SOS is NP-hard we make use of the *restriction principle*, which states that if a special case of a problem is NP-hard, then the problem itself is NP-hard. In our situation, when $v = n$ and the groups are mutually exclusive, SCP-SOS reduces to SCP, which implies that the former is NP-hard.

To define an instance of SCP-SOS, let S_{kl} be the l th subset in group k and c_{kl} the corresponding cost, $k = 1, \dots, n, l = 1, \dots, n(k)$. The problem is to select one subset S_{kl} from each group k such that the total cost is minimized and all items are included in their union.

For an instance of TAP, let there be n workers, t time periods over which assignments are to be made, and m WSGs. Let $d_{jp} = 1$ be the demand for workers in WSG j during period p . If $J(k)$ is the set of WSGs and $P(k)$ is the set of periods that are feasible for worker k over the planning horizon, then $(j, p) \in J(k) \times P(k)$ is a feasible WSG-period. Let S_{kl} be the l th collection of feasible WSG-period pairs that define a complete schedule for worker k and let c_{kl} be the corresponding cost. The problem is to select one subset S_{kl} for each worker such that the total cost is minimized and all demand is covered. For compatibility, we assume without loss of generality that $t = u/m$ (number of items / number of WSGs) is integral. If this is not the case, then it is always possible to add $\theta = m \lceil u/m \rceil - u$ items to SCP-SOS and include all of them in each subset S_{kl} , leaving the cost coefficient c_{kl} unchanged. We would also add θ WSG-period

pairs to TAP of the form $(m - \theta + 1, t), \dots, (m, t)$ and include each of them in duplicates of subsets S_{kl} that didn't contain the pairs $(1, t), \dots, (m - \theta, t)$. The corresponding cost coefficient c_{kl} would similarly be left unchanged.

Now, given any instance of SCP-SOS, it is straightforward to map it into an instance of TAP in linear time by associating an item in the l th subset in group k in SCP-SOS with a feasible pair (j, p) in the l th schedule for worker k in TAP. A demand of 1 for all (j, p) pairs means that all items must be selected. If more than one feasible pair is assigned to, say WSG 1 in period 1, then all but one of the workers assigned to WSG 1 would be idle in period 1.

Similarly, a solution to TAP can be translated directly into a solution of SCP-SOS. Moreover, the simple observation that any assignment for TAP can be checked for feasibility in $O(nt)$ time implies that it is in NP. This leads to the conclusion that the TAP is NP-hard. \square

Mail processing environment. Each day, mail arriving at a P&DC undergoes the following three high-level operations: (i) if a stamp exists, it is cancelled, (ii) the address is read and a bar code is sprayed on the envelop, and (iii) each piece of mail is sorted to its final destination. Four types of automation equipment are used for these activities: (1) an advanced face-canceller system (AFCS); (2) a multi-line optical character reader (MLOCR); (3) a remote barcode sorter (RBCS); and (4) a delivery barcode sorter (DBCS). Workstation groups are typically formed by combining similar pieces of equipment located in the same area; e.g., all AFCSs may constitute a single WSG. When a large number of machines of the same type are present, they may be partitioned into several WSGs.

In practice, a workday is divided into 48 periods, each 30 minutes long. FTRs normally work an 8 1/2-hour shift that includes a 1/2-hour lunch break. This translates into 17 consecutive periods per day. In contrast, PTFs may be scheduled to work as few as 1 day or as many as 6 days per week, depending on the expected demand. Part-time shifts vary from 8 to 17 periods. Whenever supply exceeds demand, one or more workers will be idle in the corresponding period.

4. Model formulation

The task assignment problem has several different generic interpretations. We will view it as a variant of the minimum cost, multi-commodity network flow problem in which the individual workers are the commodities. In this section, we describe how the problem can be modeled as a network and give the corresponding IP formulation. Several special cases are discussed. A more detailed model that includes an explicit representation of idle time and lunch breaks is then given.

4.1. Basic model

Let n be the number of workers, m the number of WSGs, and t the number of time periods. For each worker $k \in K$ we define a directed subgraph $G_k = \{N_k, A_k\}$ with node set N_k and arc set A_k . Each node $i \in N_k$ represents a WSG - time period pair, which collectively form a rectangular grid, or t -layered network. For completeness, it is necessary to add source and sink nodes (s_k and t_k) for each worker. In all, $|N_k| = 2 + tm$. Each arc $(i, j) \in A_k$ represents a permissible transition from one WSG to another in adjacent time periods and has "cost" c_{ij}^k , which may be viewed more appropriately as a penalty. At a minimum, this value includes the cost of switching from WSG i to WSG j but may also include the cost of being assigned to j . It is assumed that cost is not a function of time.

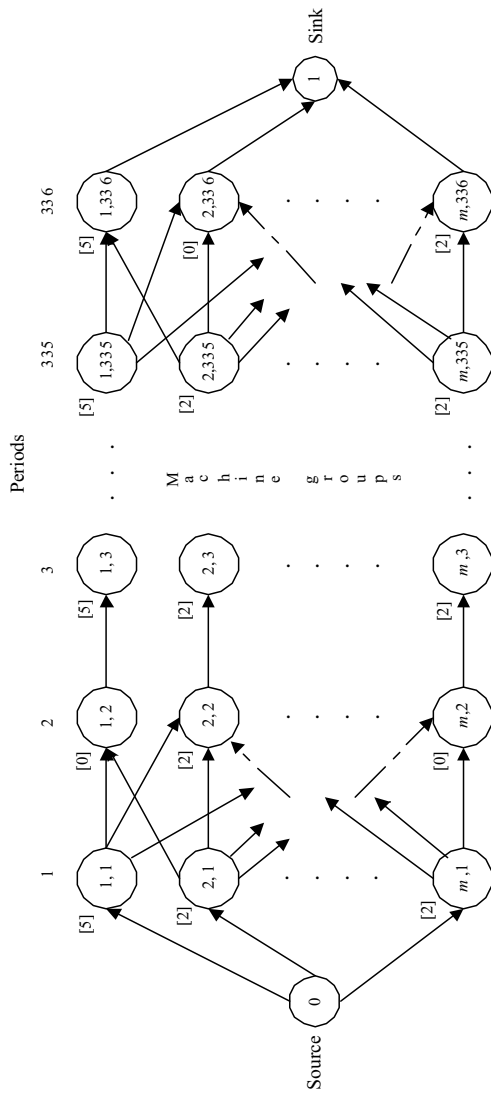


Fig. 1 Multi-period, single-commodity network representation for a week

Figure 1 depicts a “complete” graph for one worker for 336 periods or 1 week. The numbers in square brackets adjacent to the nodes represent demand d_{jp} , while the numbers inside the nodes denote the WSG j and the period p , respectively. In reality, it is only necessary to include nodes that correspond to the shifts that worker k is assigned over the week. Accordingly the nodes that correspond to the predetermined lunch breaks and the periods between shifts can be omitted. This greatly reduces the size of N_k .

The full graph G_k for each worker k has the same node set except for s_k and t_k . The arc sets, however, are all disjoint. The full graph for the problem is $G = (N, A) = \bigcup_{k \in K} G_k = (\{N_1 \cup \dots \cup N_n\}, \{A_1 \cup \dots \cup A_n\})$ and is likely to be very sparse because arcs only exist between nodes in successive periods, except for the two situations just mentioned. That is, when there is a lunch break in period p , the nodes and arcs associated with period p can be skipped for this worker and new arcs added that join the nodes in period $p - 1$ to the nodes in period $p + 1$. Similarly, when a shift ends in period p and the next shift starts in period $p + q$, the intermediary arcs and nodes can be skipped.

If we specify a supply of 1 at the source node s_k and a demand of 1 at the sink node t_k for all $k \in K$, the task assignment problem is to find a minimum cost flow from each s_k to each t_k such that the demand, d_{jp} , at each intermediate node jp is satisfied. When $d_{jp} = 0$ for all nodes but s_k and t_k , the overall problem decomposes into n shortest path problems, one for each worker.

For the TAP to be feasible, we must have $\sum_{j=1}^m d_{jp} \leq n$ for all p . When $\omega(p) \equiv n - \sum_{j=1}^m d_{jp} > 0$, $\omega(p)$ workers in period p will be idle. The only remaining modeling issue concerns the initial conditions. There are two ways to view the overall problem, which leads to two different formulations. If a new solution is required every week because the demand changes, we need to take into account the last assignment of each worker k in the last period ($t = 336$) of the previous week and define the arc costs, $c_{s_k, j}^k$, accordingly. Assuming that there is no transition cost when a shift starts, if worker k is off in period 336, $c_{s_k, j}^k = 0$ for $j = 1, \dots, m$; otherwise, $c_{s_k, j}^k$ will reflect the cost of going from the WSG he is assigned in period 336 in the previous week to perhaps a new WSG in period 1 in the current week.

Alternatively, if the demand remains constant from week to week, then the problem only has to be solved once. In this case, we would remove all source and sink nodes, add arcs between the nodes associated with the last working period and the nodes associated with the first working period, and interpret the solution as a circulation thereby obviating the need to account for initial conditions. The model presented below reflects the first view because it is more general. In the developments, we make use of the following notation, which is somewhat different than the notation previously defined.

Indices

i, j = indices for nodes
 k = index for workers

Sets

I = set of nodes
 K = set of workers
 $A(k)$ = set of nodes corresponding to the periods during the week that worker k is on duty
 $F(k, i)$ = set of nodes that are immediate successors of node i for worker k
 $P(k, i)$ = set of nodes that are immediate predecessors of node i for worker k

Parameters

c_{ij}^k = cost of a transition from node i to node j for worker k
 D_i = demand at node i

Decision variables

x_{ij}^k = (binary) equal to 1 if node i is immediate predecessor of node j for worker k ; 0 otherwise

Model

$$\text{Minimize } z = \sum_{k \in K} \sum_{i \in I} \sum_{j \in F(k,i)} c_{ij}^k x_{ij}^k \tag{1a}$$

$$\text{subject to } \sum_{j \in F(k,l)} x_{ij}^k - \sum_{j \in P(k,j)} x_{ji}^k = 0, \quad \text{for all } k \in K, i \in A(k) \tag{1b}$$

$$\sum_{j \in F(k,s_k)} x_{s_k j}^k = 1, \quad \text{for all } k \in K \tag{1c}$$

$$\sum_{j \in P(k,t_k)} x_{j t_k}^k = 1, \quad \text{for all } k \in K \tag{1d}$$

$$\sum_{k \in K} \sum_{j \in F(k,i)} x_{ij}^k \geq D_i, \quad \text{for all } i \in I \tag{1e}$$

$$x_{ij}^k = 0 \text{ or } 1, \quad \text{for all } k \in K, i \in A(k), j \in F(k, i) \tag{1f}$$

The objective function (1a) minimizes the total transition costs for the workforce during the week. The coefficient c_{ij}^k is defined for each employee k and appropriate nodes i and j . It is positive only if i and j are in different WSGs. For the calculations, if i and j correspond to successive working periods, we set $c_{ij}^k = 1$; if there is a lunch break period between nodes i and j , $c_{ij}^k = 0.5$; and if there is a shift break between i and j , $c_{ij}^k = 0.1$. Alternative definitions are possible. For example, if it were more desirable for worker k to be assigned to WSG g_1 than g_2 , c_{ij}^k could be adjusted so that solutions were biased towards g_1 .

The inherent network structure of the problem is embodied in the definition of the set $A(k)$ which is of size $O(mt)$. Constraint (1b) requires conservation of flow at each node, while Eqs. (1c) and (1d) ensure that exactly one unit of flow is allowed in the network for each commodity from source to sink. Constraint (1e) ensures that all the demand is met and constraint (1f) places binary restrictions on the variables.

Model (1) is not a pure minimum cost multi-commodity network flow problem for at least two reasons [see Garey and Johnson (1979) for a definition]. First, the network on which it is defined has a special structure; second, such problems require that the flow on each arc be restricted by some upper bound, $u(i, j)$, rather than the flow through each node be restricted by some lower bound which is the case here. Although it would be possible to split each node i in our model into two nodes, say, i_1 and i_2 , join them by a single arc and then associate the demand D_i with the arc capacity, say, $u(i_1, i_2)$, the sense of the inequality in constraint (1e) would be in the wrong direction. Nevertheless, several special cases arise when some workers have the same schedule.

Proposition 1. *Let $K_1 \cup K_2 \cup \dots \cup K_b = K$ be a partition of K into b sets such that the each worker in $K_k, k = 1, \dots, b$ has the same schedule. Then $k \in K$ can be replaced with $k = 1, \dots, b$ in model (1) and Eqs. (1c) and (1d) can be replaced with*

$$\sum_{j \in F(k, s_k)} x_{s_k j}^k = |K_k|, \quad \text{for all } k = 1, \dots, b \tag{2a}$$

$$\sum_{j \in F(k, t_k)} x_{j t_k}^k = |K_k|, \quad \text{for all } k = 1, \dots, b \tag{2b}$$

where $x_{ij}^k = 1$ now denotes a transition from node i to node j for a worker in set K_k .

Proof: Equations (2a) and (2b) represent the aggregation of Eqs. (1c) and (1d) for all workers in a particular set K_k so any point that is feasible to the new formulation is feasible to model (1). The right-hand sides of (2a) and (2b) indicate that there must be $|K_k|$ units of flow in the corresponding aggregated network. Because the $|K_k|$ workers are indistinguishable from each other, any feasible solution to (1b), (2a), (2b), (1e), (1f) obtained after replacing $k \in K$ with $k = 1, \dots, b$ will be feasible to (1b) – (1f); that is, the flow in the aggregated network maps directly into the flows in the individual networks. \square

Note that if two workers have the same shift schedule but different lunch breaks, Proposition 1 is not valid. If the two networks were combined, it might happen that one worker took two lunch breaks and the other none, or that they both took lunch breaks in the same period rather than at different periods. Either situation could lead to an incorrect solution.

Proposition 2. *When each worker k has the same schedule, (1a)–(1f) can be transformed into a pure min-cost flow problem.*

Proof: When all the workers have the same schedule, they are indistinguishable from one another so we can drop the index k from the formulation and set $A(k) = I, s_k = s, t_k = t, F(k, i) = F(i),$ and $P(k, i) = P(i)$. To guarantee that there are n units of flow in the network, we set the right-hand sides of (1c) and (1d) to n . Finally, to ensure that all demand is met we use the node splitting idea and set a lower bound on the arc that joins the two new nodes, say, nodes i_1 and i_2 , to D_{i_1} . Letting I' be the set of nodes in the new network and E be the set of arcs that join the split nodes, model (1) becomes

$$\text{Minimize } z = \sum_{i \in I'} \sum_{j \in F(i)} c_{ij} x_{ij} \tag{3a}$$

$$\text{subject to } \sum_{j \in F(i)} x_{ij} - \sum_{j \in P(i)} x_{ji} = 0, \quad \text{for all } i \in I' \setminus \{s, t\} \tag{3b}$$

$$\sum_{j \in F(s)} x_{sj} = n \tag{3c}$$

$$\sum_{j \in P(t)} x_{jt} = n \tag{3d}$$

$$x_{ij} \geq D_i, \quad \text{for all } (i, j) \in E \tag{3e}$$

$$x_{ij} \geq 0 \text{ and integer, for all } i \in I', j \in F(i) \tag{3f}$$

which is a pure min-cost network flow problem. \square

When solving integer programs with exact methods, it is often helpful to add valid inequalities that remove symmetry in the problem definition (Sherali and Smith, 2001). The goal is to prevent fractional solutions that look different but are actually reflections of each other appearing at different nodes in the branch and bound tree. For the TAP, symmetry is present when two or more workers have the same schedule for all or part of the week. The following result addresses this issue.

Proposition 3 (Symmetry). *For the TAP defined for a single week only, assume that workers k and $k + 1$ have the same schedules and lunch breaks from period q through period 336. Let $I_p = \{i_{p1}, \dots, i_{pm}\}$ ($p = q, \dots, 336$) represent the set of nodes in the network during period p and let $J_p = I_{p+s}$, where $p + s$ is the first working period after p . Then the following constraints are valid for model (1) for all values of $p = q, \dots, 336$ that correspond to working periods.*

$$x_{i_{pg}j_{ph}}^k \leq \sum_{l=h}^m x_{i_{pg}j_{pl}}^{k+1}, \quad \forall i_{pg} \in I_p; j_{ph} \in J_p; g, h = 1, \dots, m \tag{4}$$

Proof: For period p , when workers k and $k + 1$ are at node i_{pg} and worker k transits from WSG g to WSG h , constraint (4) restricts worker $k + 1$ to transitions from WSG g to WSG h or higher. This prevents workers k and $k + 1$ from switching assignments in period $p + s$ and hence breaks the symmetry. Because k and $k + 1$ have exactly the same schedule from period p through the end of the week they appear in the model to be identical from p forward, so no advantage can be gain by interchanging their assignments after node i_{pg} . This validates the imposition of constraint (4). □

Corollary 1. *For the TAP defined as a circulation, a sufficient for constraint (4) to be valid for model (1) is that workers k and $k + 1$ have exactly the same schedule for the entire week.*

4.2. Model with idle time and lunch breaks

The objective of the TAP is to minimize the number of transitions between different WSGs for all employees. The following are the most common types of transitions:

1. immediate, from one WSG to another
2. after lunch break (different WSG before and after lunch)
3. after idle period (different WSG before and after idle period)
4. between shifts (changing WSGs)

The first is the most undesirable because it involves a change of location, supervisor, and job content for the employee. The others are less costly as measured by time and inconvenience. The easiest way to represent this preference structure is to assign unique costs to the coefficients in (1a). This approach is possible for transitions after the lunch break, between WSGs, and between shifts because the schedule of each worker is known in advance. We must wait until a solution is found, though, before we can identify the idle periods.

In the network description of model (1), each node is associated with a WSG-time period pair (j, p) . To accommodate idle time explicitly, when the number of workers who are available in time period p is greater than the total demand in period p , we split the corresponding node in two creating a *work node* and an *idle node*. In a solution, if a worker is assigned to the former, it

means that he is active in the period under consideration; if he is assigned to the latter, then he is idle.

In the multi-commodity network, these two nodes are distinguished only by the costs on their leaving arcs. If nodes i and j are in different WSGs and i is an idle node, then arc (i, j) corresponds to a transition after an idle period and should be assigned a cost that reflects this type of transition. Note that it is not necessary for a worker to move to the idle node in another WSG because the worker can always choose moving to the idle node in the current WSG without increasing the total cost. So the corresponding arcs can be removed except when there is a lunch break or a shift break right after the associated period. Because the size of model (1) is proportional to the square of the number of WSGs, the augmented model is roughly twice as large as the original model when the unnecessary arcs are omitted.

A second extension of model (1) involves the option of assigning lunch breaks along with the tasks. In the USPS application, the lunch break is part of the bid job and is scheduled in advance by SOS. In practice, management has some discretion in rescheduling the lunch break to better match the current day's supply and demand of labor. For an 8-hour shift, the break can be assigned any time within a 3-hour window starting two hours into the shift.

Because a transition after the lunch break is more acceptable than an immediate transition, the rescheduling option may yield better results. To build this into the model, it would be necessary to add a lunch break node in each time period covered by the break window of a shift. As was the case for idle periods, one additional node would be needed for each WSG. This construction is based on the assumption that the transition cost is a function of the WSG assigned just prior to the lunch break. If there is no transition cost after the lunch break, it could be modeled as just another WSG. To guarantee that one and only one break is assigned to each shift, the following constraint should be added to model (1):

$$\sum_{i \in E(k,d)} \sum_{j \in F(k,i)} x_{ij}^k = 1, \quad \text{for all } k \in K, d \in C(k) \quad (1g)$$

where $E(k, d)$ is the set of lunch break nodes in the break window of worker k on day d and $C(k)$ is the set of days in which worker k requires a lunch break.

In the remaining sections, we focus on the explicit representation of idle time only. Because the lunch breaks can be treated in a similar manner, they don't affect the nature of the problem or the solution methodology.

5. Solution methodologies

Because most realistic instances of model (1) were too big to solve with a high performance commercial code, we developed several different solution approaches. The first is based on a greedy heuristic; the second on the decomposition of the problem into seven daily subproblems; and the third on a hybrid tabu search procedure. Each is discussed in the following subsections.

5.1. SOS heuristic

The first algorithm developed for the TAP at P&DCs was a greedy heuristic (referred to as the SOS heuristic) that viewed the week as a circular array of 336 ($= 48 \times 7$) $1/2$ -hour periods. In this design, no starting point exists so every element in the array always has a successor (the elements in period 336 are followed by the elements in period 1). The two data structures used by the heuristic are as follows.

1. A list of workstations with corresponding demand. The demand is represented by an array of 336 integers that give the number of workers required for each period of the week. The workstations are grouped in sections or WSGs.
2. A list of shifts satisfying all demand for the given week. Each shift is associated with a single worker and has a starting period (a value between 0 and 335), a length, and possibly a lunch period. Shifts are grouped by starting period giving rise to 336 < shift/period > lists; the shift/period list [p] contains all the shifts starting at period < p >. During execution, a shift starting at < p > may be assigned to a workstation for a duration < d > which is less than the shift length < l >. In this case, the partially assigned shift is added to the shift/period list [$p + d$] for further assignment during the remaining < $l - d$ > periods. Thus, each shift/period list contains two types of shifts: (1) unassigned shifts that can be assigned to a workstation in any section, and (1) partially assigned shifts that should be assigned, if possible, to a workstation in the same section.

The first step is to compute the period of the week with least demand. Let

$$p_{\min} = \operatorname{argmin} \left\{ \sum_{j=1}^m d_{jp} : p = 0, \dots, 335 \right\}$$

The greedy SOS heuristic cycles through the circular array of 336 shift/period lists, starting at p_{\min} . For each period, an assignment is given to all the shifts on the shift/period list thereby emptying it. However, each shift not completely full is added to a later shift/period list. Processing the last few lists re-introduces shifts to the first few lists so we have to cycle through the circular array until all lists are empty. Additional data structures are needed to track partial assignments and unsatisfied demand; additional logic is needed to determine which workstation to select for the next shift assignment in light of the objective of minimizing the number of transitions. The details are available from the authors.

In an effort to improve on this heuristic, we have developed two separate approaches that can be combined into a single methodology. Each is discussed below.

5.2. Delayed idle period assignment and daily decomposition algorithm

When idle time is included in model (1) explicitly, the number of nodes in the multi-commodity network doubles. This leads to a linear increase in the number of constraints, which is $O(nmt)$, and a quadratic increase in the number of variables, which is $O(nm^2t)$. Our experience showed that only very small instances could be solved optimally at this level of generality. As a compromise, we solved model (1) in its original form and scheduled the idle time in a post-processing phase. Although this approach should be viewed as a heuristic, the often negligible impact of idle time suggests that the resultant solutions should be near-optimal.

The problem of assigning idle time in the post-processing phase can be modeled as an integer program but for the reason just mentioned, we take a greedy approach. In particular, when there is a surplus of workers in a period we must decide which of them should be considered active and which should be considered idle. To resolve the issue, we note that the only way that the objective function can be reduced at this point is by inserting an idle period immediately before or after a transition from one WSG to another in a worker's schedule. This means that the priority for assigning idle time in a particular period should be given to those workers who have immediate transitions right before or after the period in which a surplus of idle time exists. If no workers have such schedules, then the idle periods can be assigned arbitrarily.

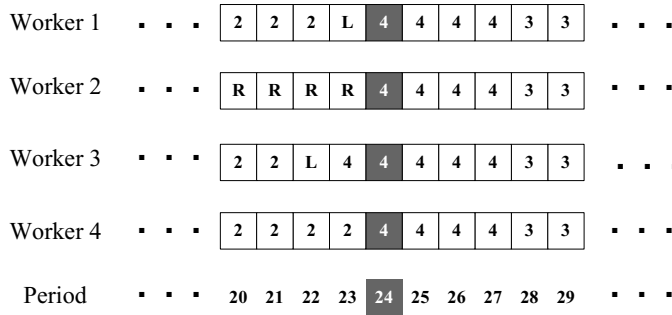


Fig. 2 Various types of transitions for given schedules

Figure 2 illustrates the types of transitions that might occur for a worker during the week. The numbers in the boxes represent the WSG, “L” denotes a lunch break, and “R” indicates an unscheduled period. Given the following cost structure: immediate transition = 1.0, transition after lunch = 0.5, transition after idle time = 0.5, transition between shifts = 0.1, suppose that WSG 4 has a demand of three during period 24, and that the solution to model (1) assigns all four workers to that WSG. As shown in the figure, worker 1 has a lunch break in the preceding period with a cost of 0.5, while worker 2 is scheduled to begin a new shift at this time (in the example, worker 2’s assignment in the last period of the previous shift does not affect the results). Therefore, no benefit can be gained by allowing either of them to be idle in period 24. The same can be said for worker 3 who is assigned to WSG 4 during periods 23, 24 and 25. Making period 24 an idle period will not affect the cost of his schedule because he is assigned to WSG 4 in periods 23 and 25. As such, no cost will be incurred by the transition from the idle period.

The best choice is to assign the idle time to worker 4 who has an immediate transition in period 23. Because the transition will now be after an idle period rather than a WSG, the objective function will decrease from its current value by the difference between the two costs; that is, cost reduction = 1 – 0.5 = 0.5. Summarizing, the cost for each worker before idle time is assigned is given by the vector (1.5, 1.0, 1.5, 2.0); the costs after are (1.5, 1.0, 1.5, 1.5). The algorithm used in the post-processing phase to assign idle time is given in Fig. 3.

A second method to reduce the problem size is to decompose model (1) into seven subproblems, one for each day. When this is done, the solution from the first day provides the initial conditions for the second day, and so on. These conditions are a function of the shifts that spill over from one day to the next, where day 7 can be thought of as day 0. Because the number of arcs in the network (and hence the number of variables) grows linearly with the number of periods t in the planning horizon, the complexity of the integer program (1a) – (1e) grows exponentially with t .

Although solving the daily subproblems separately cannot be expected to yield the optimal solution for the week, the decomposition approach, combined with delayed idle period assignment, does yield high quality solutions quickly for instances with up to 300 workers and 7 WSGs. Larger instances require a smaller grid than a day if solutions are to be obtained in a reasonable amount of time. Combining these two ideas leads to even greater reductions in problem size and run times without much degradation in solution quality, as shown in the section on computational results.

```

Let  $W_i$  = set of workers assigned to node  $i$ , and let  $p_i$  = time period associated with  $i \in I$ .
Set  $no\_priority = false$ 
For (all nodes  $i \in I$ ) {
  While ( $|W_i| > D_i$ ) {
    For (all workers  $k \in W_i$ ) {
      If ( $k$  has a transition right before or after period  $p_i$  or  $no\_priority = true$ ) {
        Assign  $k$  idle time during period  $p_i$ 
         $W_i \leftarrow W_i \setminus \{k\}$ 
        Continue
      }
    }
    If (all workers have been checked at node  $i$  and  $W_i \neq \emptyset$ ) {
       $no\_priority = true$ 
    }
  }
}

```

Fig. 3 Heuristic for assigning idle time

5.3. Tabu search

Tabu search (Glover and Laguna, 1997) is a powerful metaheuristic that has been used successfully in solving many types of large-scale optimization problems that have resisted exact methods. In the computations, the neighborhood of the incumbent is explored and the best feasible solution found becomes the new incumbent, regardless of quality. To prevent returning to the same local solution within a fixed number of iterations, recently visited points are placed on a tabu list.

The essential components of the procedure are the (1) initial solution, (2) neighborhood definition, (3) tabu list, and (4) search strategy. Although there has been continued debate as to whether a good initial solution leads to greater overall efficiency, our experience has shown that the better the initial solution, the better the results, at least for the TAP under reasonable run time limits. We investigated three options: (1) the solution obtained with the SOS heuristic, (2) the solution obtained with the delayed idle period assignment and daily decomposition (DIPA&DD) algorithm, and (3) the solution obtained by solving a series of shortest route problems (SSRA). The first two have already been discussed; we now focus on the third.

5.3.1. Initial solution with shortest route algorithm

An implication of Proposition 2 is that when the demand constraints (1e) are relaxed, model (1) can be decomposed into n independent pure network flow problems with integral solutions. Because each network is cyclic and contains only one unit of flow, the problem is equivalent to finding the shortest loop in the network. We used CPLEX to solve this problem.

Of course, ignoring the demand constraints and solving all n flow problems separately is not likely to produce a feasible solution to (1). To achieve feasibility, we used a simple greedy algorithm in which the workers, arbitrarily ordered, are scheduled in series (see Fig. 4). For the first worker, the shortest route problem is solved and the demand associated with each node in the solution is reduced by 1. When the demand at a node reaches 0, it is removed from the network. The process is repeated until all workers have been scheduled.

```

For (all workers  $k \in K$ ) {
  Set up the single-commodity network shown in Fig. 1 for current worker  $k$ .
  For (all nodes  $i \in I$ ) {
    If (remaining demand at node  $i = 0$ ) {
      Remove node  $i$  and all arcs connected to it in the network.
    }
  }
  Solve the shortest loop problem to get a schedule for current worker  $k$ 
  For (all nodes  $i \in I$ ) {
    If (node  $i$  is on the schedule of worker  $k$ ) {
      Deduct 1 from the remaining demand at node  $i$ .
    }
  }
}

```

Fig. 4 Sequential shortest route method

5.3.2. Neighborhood definition

A neighborhood is defined as a set of points that can be reached from the current solution by performing one or more exchanges, which characterize a move. The algorithm proceeds from an initial feasible solution or incumbent to a new solution by searching the current neighborhood for the “best” solution. Depending on the strategy used, the new solution may or may not be feasible.

For the task assignment problem, we define a neighborhood as all feasible points that can be reached by a swap move that switches the positions of two workers in the same period. To ensure that only neighborhood points that are feasible and different than the current solution are considered, some restrictions must be placed on a move. First, at least two workers must be scheduled in the period under consideration or no swap can take place. Second, the selected workers must be assigned to different WSGs in the chosen period otherwise the exchange will have no effect on the solution.

Figure 5 gives an example of a swap move for workers 1 and 2 in period 24. This is an admissible move because they are assigned to different WSGs in this period. Before the swap, both workers have two immediate transitions in the portion of schedule shown. The total penalty induced by these transitions is 2. After the swap, the first immediate transition in both cases is replaced by a transition after lunch, which brings the penalty down to 1.5 for each worker. Therefore, the value of the swap (*move_value*) is -1 . In the algorithm, all feasible moves are considered and the one with the smallest value is chosen, giving the new incumbent.

The size of the neighborhood is $O(n^2t)$, where n is the number of workers and t is the number of periods, so it is reasonable to conduct an exhaustive search. A second advantage is that every feasible point in the solution space can be reached if enough iterations are performed. Nevertheless, a significant amount of effort may still be required to conduct one neighborhood search for t fixed even though the size of the neighborhood is only quadratic in n . Finally, we note that more complex neighborhoods were considered, including three way swaps and multiple period swaps, but they proved to be more time consuming and no more effective than the one selected.

5.3.3. Tabu list and aspiration criterion

The fundamental process that tabu search uses to transcend local optimality is to make certain moves forbidden (tabu) on a temporary basis. The process is operationalized with a tabu list

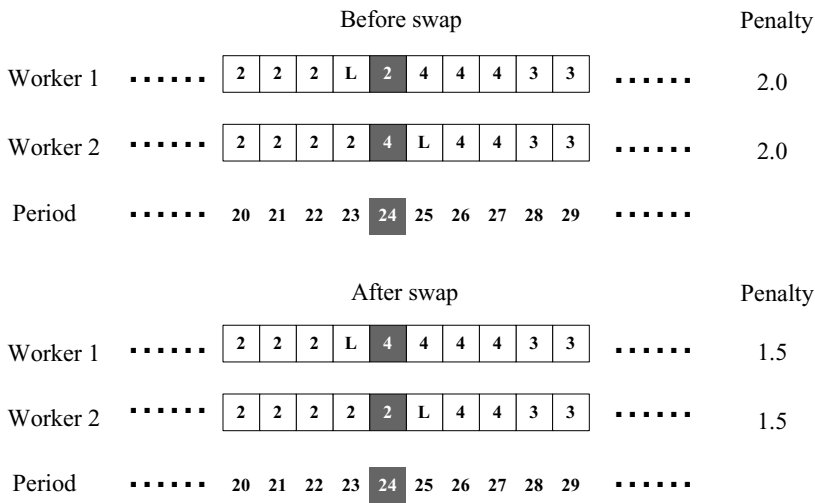


Fig. 5 Example of swap move

whose length determines how many iterations a certain move is forbidden. Each entry on the list is a combination of a worker and a period; i.e., (worker i , period p). During execution, any move that leads to a solution that includes a combination on the list is disallowed. After the current neighborhood is searched and a new incumbent is identified, the two workers associated with the move and time period in which it occurs are added to the tabu list.

Nevertheless, the tabu status of a move can be overridden when a certain aspiration criterion is satisfied. In our implementation, a move on the tabu list is accepted if it leads to a solution better than the best solution found so far.

5.3.4. Search strategy

There are two important characteristics for a successful tabu search procedure: exploration and exploitation. Exploration is the ability of the algorithm to diversify the search throughout the solution space, while exploitation is the ability of the algorithm to intensify the search in the areas of the solution space that show promise. In practice, long-term memory in conjunction with the short-term memory provided by the tabu list, are used to implement the exploration strategy. The long-term memory in our algorithm is in the form of an $n \times 336$ matrix M , where an element M_{ip} represents the number of times the pair (worker i , period p) has been involved in a move. Any move associated with this pair is additionally penalized in accordance with the value of M_{ip} . For example, if a candidate swap calls for workers i and j to change positions in period p , then the actual value used in the neighborhood comparisons is $move_value + 0.1(M_{ip} + M_{jp})$ rather than $move_value$ alone.

As mentioned, the size of the neighborhood is polynomial in n but still large for most real-world instances. To reduce the computational effort, it is common to limit the search to a certain number of randomly selected candidates within a neighborhood. The certain number, $candidate_num$, is a parameter that is set adaptively according to the size of the problem and the progress of the algorithm, and represents the exploitation strategy. In our implementation, it oscillates between two values: *smallscan* and *bigscan*. Whenever an improved solution is encountered,

candidate_num is increased to *bigscan* to allow the neighborhood of the solution to be searched more thoroughly; it is reset to *smallscan* after 200 consecutive iterations without improvement.

5.3.5. Algorithm description

Figure 5 highlights the major components of the tabu search procedure. The first step is to generate an initial feasible solution with one of the methods described above; i.e., the greedy SOS heuristic, the delayed idle period assignment and daily decomposition algorithm, or the sequential shortest route algorithm. The results are saved as both the *current solution* and the *best solution* found so far; the corresponding objective function value is used to initialize the data element *best_cost*. The *move_value* and the *tabu_list* are also initialized. Because our implementation depends on the independent generation of a starting point, it should really be viewed as a hybrid approach that combines tabu search with one of several feasibility heuristics.

Given an initial solution, the algorithm iterates until a prespecified limit (*max_iterations*) is reached or there is no improvement in *max_no_improve* iterations. The move to be made at a given iteration is found by computing the sum of *move_value* and *move_penalty* of all candidate swaps in the neighborhood of the current solution. Because we are minimizing the transition cost, the best candidate move is the one associated with the smallest sum. A move is admissible if (1) it is not tabu or (2) its tabu status can be overridden by the aspiration criteria. The *best_move* is then performed and the tabu data structures are updated. The best overall solution (*best_soln*) is updated if the current value of the total cost is less than the objective value of the incumbent.

The data structures used to describe the heuristic are as follows.

- *num_iterations*: current iteration number
- *max_iterations*: limit on total number of iterations that can be performed
- *curr_cost*: sum of the cost of transitions for all workers in a week (objective function value)
- *best_cost*: best total transition cost for all workers obtained so far (incumbent objective function value)
- *curr_soln*: current solution obtained after executing the best move
- *best_soln*: best solution obtained so far (incumbent)
- *move_value*: difference in the cost of a transition before and after the swap of the two workers being considered in a certain period
- *best_move_value*: lowest *move_value* among all candidate moves
- *move_penalty*: additional penalty imposed by the long-term memory *freq_matrix* M ; when workers i and j are selected two switch positions in period p , $move_penalty = 0.1(M_{ip} + M_{jp})$
- *best_move_penalty*: *move_penalty* for the best move of all candidate moves
- *no_improve*: number of consecutive iterations during which no better solutions are found
- *max_no_improve*: maximum number of consecutive iterations permitted during which no better solutions are found: 5,000 for all data sets
- *tabu_size*: total number of iterations for which a move is held tabu: 20 in our implementation
- *tabu_list*: short-term memory function that stores recent moves that are forbidden in subsequent iterations
- *freq_matrix*: long-term memory function that records the number of times that each possible move is revisited
- *candidate_num*: number of candidate moves that are scanned during each iteration (can be one of two values, $smallscan = n^2/100$ and $bigscan = n^2/50$, depending on the situation)

```

1. Generate initial feasible solution and save it as curr_sol and best_sol.
2. Evaluate curr_cost and set best_cost = curr_cost.
3. Initialize tabu_list and freq_matrix.
4. Set candidate_num = smallcan.
5. Set no_improve = 0.
6. Do{
    best_move_value = ∞
    for (all candidate moves)
    {
        if (move_status ≠ tabu or move_value is improving)
        {
            if (move_value + move_penalty < best_value + best_move_penalty)
            {
                best_move_value = move_value
                best_move_penalty = move_penalty
                best_move = current_move
            }
        }
    }
    execute best_move [ update current solution by swapping ]
    curr_cost = curr_cost + best_move_value
    update tabu_list
    update freq_matrix
    if (curr_cost < best_cost)
    {
        best_cost = curr_cost
        best_soln = curr_soln
        candidate_numb = bigscan
        no_improve = 0
    }else{
        no_improve = no_improve + 1
    }
} While (num_iterations < max_iterations or no_improve < max_no_improve)

```

Fig. 6 Tabu search procedure

- *candidate move*: solution that results when two workers assigned to different WSGs in the current period swap position
- *admissible move*: candidate move that either satisfies the aspiration level criterion or is not tabu

6. Computational experience

The various algorithms and initialization procedures developed for solving the task assignment problem were tested using data obtained from the Boston P&DC. Table 1 lists each approach. All algorithms were coded in Java and run on a Linux workstation with dual Xeon 1.8G CPUs and 1 gigabyte of memory. (Our licensing agreement, however, did not allow for parallel processing.) CPLEX 9.0 was used to solve the embedded IPs and LPs. For the IPs, setting the CPLEX emphasis parameter to “feasibility” rather than “optimality” worked best, as did setting the heuristic frequency parameter to every 15 nodes.

The data sets used in the computations are shown in Table 2 and reflect the range of problem sizes that might be encountered in a large facility (contact authors to obtain data). Five different worker categories are included with anywhere from 17 to 311 workers and 3 to 28 WSGs. This diversity should allow us to draw some general conclusions from the results.

Table 1 Approaches investigated

Methodology	Abbreviation
Solve model (1) with idle period nodes	EXACT
Delayed idle period assignment	DIPA
Delayed idle period assignment and daily decomposition	DIPA&DD
Tabu search	TS
Greedy SOS heuristic	SOS
Sequential shortest route algorithm	SSRA

Table 2 Data sets for computational experiments

Problem no.	Worker category	Number of workers (n)	Number of WSGs(m)
<i>Small data sets</i>			
1	P5-MPC	17	5
2	P5-FSMO	34	7
3	P5-PPDMO	45	3
4	P5-MPC	85	3
5	P5-MPC	93	3
6	P5-MPC	105	3
7	P5-MPC	116	4
8	MH5-EO	68	10
<i>Large data sets</i>			
9	P5-MPC	197	4
10	MH4	33	10
11	P5-MPC	222	6
12	P5-MPC	311	6
13	P5-MPC	288	7
14	MH4	171	28

Of the two groups of data sets in Table 2, the instances associated with the first group are relatively small and can be solved to optimality with EXACT within 1 hour. For EXACT, DIPA and DIPA&DD, a 1% optimality gap was used as the stopping criterion for small data sets to ensure that high quality solutions were found within an acceptable amount of time. For discussion purposes, the solution provided by EXACT will be termed the “optimal solution” and used as the reference point for evaluating the performance of the other algorithms.

The instances associated with the second group of data sets are larger than those in the first and could not be solved to within 10% of optimality within 2 hours. In many cases, not even a lower bound could be found after several hours of computations because their LP relaxations could not be solved. Nevertheless, it is still useful to compare the results obtained with the other methods to get a relative understanding of their performance.

Table 3 lists the number of variables, number of constraints, and number of nonzero elements in the A matrix of the IP associated with model (1) for all data sets. The first set of statistics includes idle time nodes in the formulation and the second does not. The exclusion of idle

Table 3 Size of weekly problem for model (1)

Data set	Model (1) with idle periods			Model (1) without idle periods		
	No. of variables	No. of constraints	No. of nonzeros	No. of variables	No. of constraints	No. of nonzeros
<i>Small data sets</i>						
1	27,736	9,186	68,049	13,077	5,520	29,598
2	20,417	7,493	51,330	7,246	3,211	17,317
3	23,345	11,417	55,870	13,385	7,660	31,134
4	97,722	42,088	221,875	49,825	27,395	113,404
5	123,823	51,548	281,878	57,741	31,443	131,372
6	154,363	62,356	351,605	66,825	35,981	151,850
7	186,408	70,563	427,638	90,504	44,284	204,464
8	110,852	37,390	249,636	50,041	26,252	112,399
<i>Large data sets</i>						
9	380,172	137,358	866,920	168,261	78,686	377,552
10	132,335	27,321	331,790	80,899	21,481	195,628
11	1,059,070	259,894	2,328,239	624,624	186,411	1,351,720
12	1,387,969	248,252	3,073,161	647,369	209,292	1,405,198
13	1,223,668	308,335	2,733,587	718,946	219,333	1,558,397
14	7,482,642	547,204	36,518,862	3,217,536	383,043	16,433,488

time nodes reduced the dimensions of these problems by approximately 50%. The symmetry constraints (4) were not included in the testing because the size of the LPs and not the number of nodes in the branch and bound trees was the limiting factor. Adding these constraints would have only aggravated the situation.

6.1. Results for small data sets

The objective function values and solution times obtained with all methods for the small instances are reported in Table 4. When EXACT and DIPA were used, a 1-hour time limit was set for solving model (1) for the week. For DIPA&DD, the time limit was set to 5 minutes for each single-day problem. When tabu search was used, the SOS solution served as the starting point, and “no improvement in 5,000 consecutive iterations” served as the stopping criterion.

The EXACT solutions provided the baseline against which the other solutions were compared. In Table 4, the percentage under each “objective value” and “solution time” entry represents the ratio between the EXACT result and results obtained from the respective heuristics. For example, the best solution provided by DIPA&DD for data set 4 is 53.9, which is 115.91% of the optimal solution, 46.5. Similarly, the time for DIPA&DD to find this solution was 4.0 seconds, which is 1.51% of the time used by EXACT.

Because of the need to perform “what if” analysis in the real operating environment, the EXACT solution times are acceptable for only the very small data sets. Empirically speaking, we found that any data set having an $n \times m$ value larger than 1,000 could not be solved by EXACT, and most of the time, even the LP relaxation could not be solved within several hours. In fact, the optimal solution was found by CPLEX’s build-in heuristic at the first node of branch

Table 4 Computational result of small data sets

Data set	EXACT		DIPA		DIPA&DD		TS	
	Objective value	Solution time (sec)	Objective value	Solution time (sec)	Objective value	Solution time (sec)	Objective value	Solution time (sec)
1	72.5	56.3	78.8	11.4	78.7	3.2	83.8	118.7
	100.00%	100.00%	108.69%	20.25%	108.55%	5.68%	115.59%	210.83%
2	94.6	9.0	98.8	3.6	99	5.2	98.5	245.2
	100.00%	100.00%	104.44%	40.18%	104.65%	58.04%	104.12%	2736.61%
3	68.6	12.5	78.3	8.5	74.2	3.8	94.1	71.6
	100.00%	100.00%	114.14%	68.00%	108.16%	30.40%	137.17%	572.80%
4	46.5	265.0	48.5	30.6	53.9	4.0	60.7	110.2
	100.00%	100.00%	104.30%	11.55%	115.91%	1.51%	130.54%	41.58%
5	74.3	547.9	81.6	59.7	88.6	6.8	95.4	115.0
	100.00%	100.00%	109.83%	10.90%	119.25%	1.24%	128.40%	20.99%
6	81.3	1762.5	85.2	55.5	86.8	5.2	108.2	245.1
	100.00%	100.00%	104.80%	3.15%	106.77%	0.30%	133.09%	13.91%
7	89.9	2722.7	97.1	126.7	103.3	12.7	141.4	175.2
	100.00%	100.00%	108.01%	4.65%	114.91%	0.47%	157.29%	6.43%
8	27.2	3268.2	28.0	484.5	36.1	15.9	43.7	173.8
	100.00%	100.00%	102.94%	14.82%	132.72%	0.49%	160.66%	5.32%
Avg	100.00%	100.00%	107.14%	21.69%	113.86%	12.26%	133.36%	451.06%

and bound tree for most instances, except for 3 and 8. The majority of the computational effort went into solving the LP relaxations, generating cuts, and running the built-in heuristic.

The size of model (1) without idle period nodes is about 50% of the size of the exact model, and thus can be solved much faster. On average, DIPA found solutions that were only 7.14% above the optimum, on average, in approximately 1/5 of the time. For the small data sets, this method is effective but cannot be relied on as the problem size increases, as discussed in the next section. DIPA&DD is more efficient in terms of the solution time, which is under 13% of the time required by EXACT, on average, while objective function values remained within 14% of the optimum. Finally, tabu search showed no advantage with respect to solution quality for small instances. In part, this was due to the relatively poor quality of the initial solution provided by the SOS heuristic. This issue is further examined in Section 6.3.

The graphs in Figs. 7 and 8 plot the performance of the different methods for the small data sets. From Fig. 7, it is easy to see that relative solution quality with respect to the objective function value remains the same for most instances. EXACT gives the best results in general, while DIPA consistently provides solutions that are very close to the optimum. The DIPA&DD solutions are close to or even better than those obtained with DIPA for data sets 1, 2, 4 and 6, but are noticeably worse for data sets 3, 5, 7 and 8. The tabu search results were consistently inferior to those obtained with the other methods.

Figure 8 depicts relative run times. For the first three problems, tabu search took much longer than the other methods due to the nonproductive time required to satisfy the termination criterion “no improvement in 5,000 consecutive iterations.” The corresponding points are out of range.

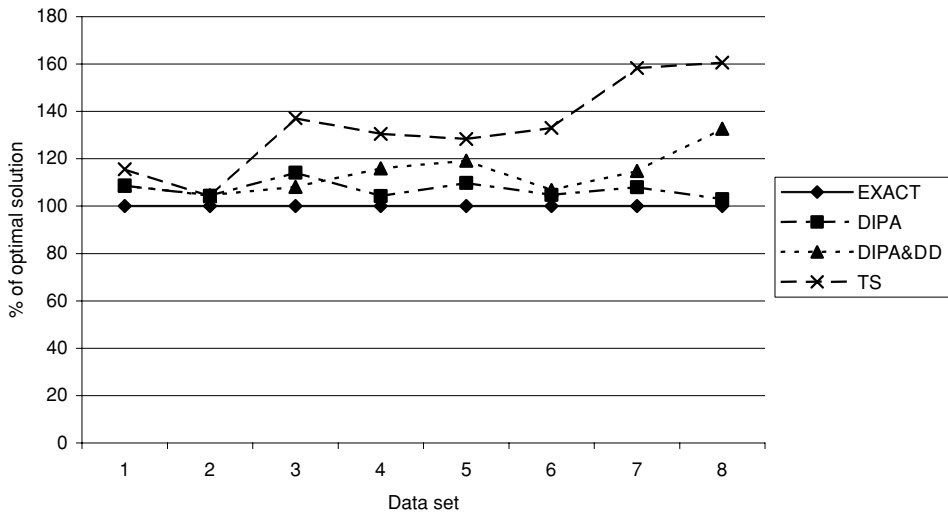


Fig. 7 Objective function comparisons for small data sets

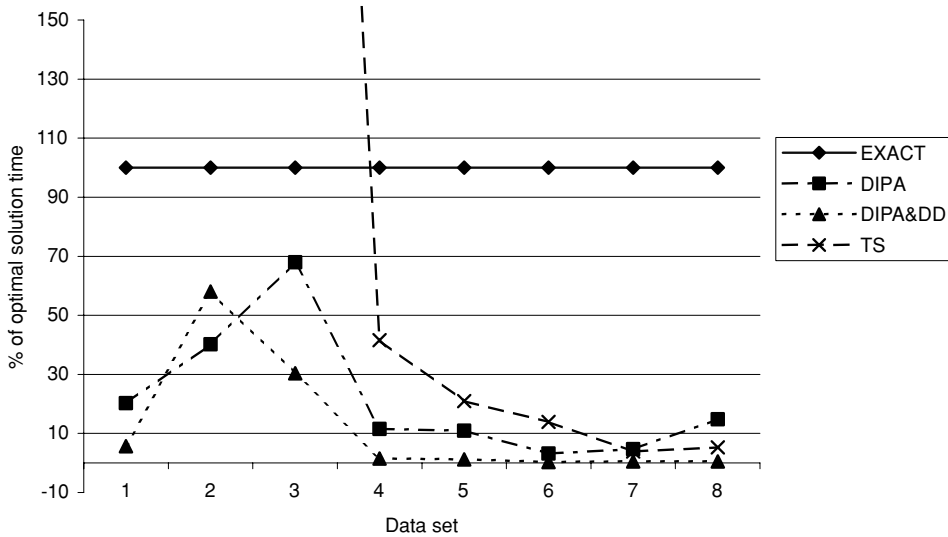


Fig. 8 Run time comparisons for small data sets

For the remaining problems, tabu search was more competitive but still not as effective. Of the IP approaches, DIPA&DD provided the fastest times, EXACT the slowest times, and DIPA was somewhere in between. The relative run time for tabu search decreased steadily as the problem size increased, and eventually dropped below that of DIPA for data set 8.

6.2. Results for large data sets

Table 5 presents the computational results obtained with DIPA, DIPA&DD, TS and SOS for the large data sets. The SOS results are included to provide a baseline for the more difficult

Table 5 Computational results for large data sets

Data set	DIPA		DIPA&DD		TS		SOS objective value
	Objective value	Solution time (sec)	Objective value	Solution time (sec)	Objective value	Solution time (sec)	
9	52.7	2468.2	61.2	20.2	102.8	753.5	119.8
10	181.9	5785.8	185.4	141.6	200.3	105.8	223.3
11	–	–	300.4	1105.2	421.3	1002.3	451.9
12	–	–	323.0	153.4	440.4	4139.5	551.2
13	–	–	415.6	2948.6	556.7	2008.3	657.2
14	–	–	–	–	815.0	801.7	929.0

instances, and reflect the performance of the system now in use at P&DCs. Because none of the LP relaxations was solvable by EXACT, no lower bounds are available.

The relative performance of the four methods tested was the same as was observed for the small data sets. DIPA provided the best solutions but could solve only the first two instances; i.e., data sets 9 and 10. Using a 20-minute time limit for each single-day problem, DIPA&DD was able to find relatively good solutions in acceptable time, except for the largest instance — data set 14. None of the daily LP relaxations could be solved within the allotted time.

The solutions found by tabu search, again, were not as good as those provided by DIPA or DIPA&DD, but it did manage to improve the initial SOS solution by an average of 13.15%. As a consequence of the randomness inherent in the search strategy, no correlation between problem size and computational effort was evident.

Whether the additional time required by any of the proposed methods to improve the initial solution can be justified, depends on the time available for the analysis. In the current system, the SOS heuristic runs in a matter of seconds. This represents a negligible amount time when compared with the options being considered here.

6.3. Initializing tabu search

As opposed to implicit enumeration methods, metaheuristics such as tabu search, require a feasible solution to get started. Although this might be viewed as a disadvantage, when coupled with other methods, they can be used in a complementary way to improve upon results obtained from those methods. In addition, because the computations can be stopped at any point, metaheuristics are often the best alternative for solving real-world problems that are intractable and not subject to decomposition.

Table 6 reports the improvements obtained with tabu search when started from the “final” solutions given in Table 5 for DIPA, DIPA&DD and SOS. Also included are the results obtained when started with the SSRA solutions. In each case, the final objective value, the run time, and the percentage improvement are listed.

The first observation is that the better the quality of the initial solution, the less improvement provided by tabu search. This is seen in the bottom row of Table 6 and is consistent with intuition; i.e., poorer solutions leave more room for improvement, although more improvement usually means that more time must be spent on the search. A second observation is that while tabu search can improve a solution quite a bit, the quality of the final solution is a function of the quality of the

Table 6 Tabu search started from different solutions

Data set	DIPA			DIPA&DD			SOS			SSRA		
	Obj. value	Time (sec)	Improvement	Obj. value	Time (sec)	Improvement	Obj. value	Time (sec)	Improvement	Obj. value	Time (sec)	Improvement
9	52.2	258.5	0.95%	60.7	273.4	0.82%	102.8	753.5	14.19%	140.2	1317.9	61.18%
10	177.0	62.8	2.69%	179.3	92.1	3.29%	200.3	105.8	10.30%	232.8	175.4	29.11%
11	-	-	-	292.8	929.6	2.53%	421.3	1002.3	6.77%	684.2	2420.2	42.85%
12	-	-	-	307.2	1897.1	4.89%	440.4	4139.5	20.10%	572.4	1253.8	51.97%
13	-	-	-	407.2	1586.5	2.02%	556.7	2008.3	15.29%	897.9	1723.5	38.19%
14	-	-	-	-	-	-	815.0	801.7	12.27%	1036.3	1200.3	39.75%
Avg	-	-	1.82%	-	-	2.71%	-	-	13.15%	-	-	43.84%

starting solution. For example, tabu search improved the SSRA solutions by 43.84% on average, but only improved the DIPA solutions by 1.82%. Despite this huge difference, the objective function values at termination were respectively 2.7 and 1.3 times greater when the tabu search was started with the SSRA solutions for data sets 9 and 10 than when it was started with the DIPA solutions.

7. Summary and conclusions

In this paper, both exact and heuristic methods were developed and tested for solving the task assignment problem. Included were a delayed idle period assignment algorithm, a daily decomposition algorithm, and tabu search. The computational results indicated that the exact method is only practical for very small instances, while daily decomposition can reliably provide near-optimal solutions for problems of moderate size. The advantages of tabu search only came into play for the large instances that were not solvable with the other methods.

As part of the analysis, we also investigated the effectiveness of tabu search when started from different feasible solutions. The results underscored the importance of good initial solutions, at least for the TAP. Although tabu search has the ability to transcend local optimality, we found that an excessive amount of time was needed to make up the difference in objective function values when started from a poor solution compared with starting from a good solution.

Other approaches to solving model (1) exactly that might be worth investigating include branch and price and Lagrangian relaxation. Both of these have the potential to exploit the fact that when the demand constraint (1e) is removed from the model, the TAP decomposes into n shortest route problems, one for each worker. Although neither branch and price nor Lagrangian relaxation will provide better lower bounds than the LP relaxation, they are likely to reduce the overall computational effort because the accompanying subproblems would be much smaller than the full LP relaxation. With some additional computations, they may also provide good feasible solutions for initializing tabu search or some other metaheuristic.

References

- Aronson, J. E., "The multi-period assignment problem: a multi commodity network flow model and specialized branch and bound algorithm," *European Journal Of Operational Research*, **23**, 367–381, (1986).
- Aykin, T., "Optimal shift scheduling with multiple break windows," *Management Science*, **42**(4), 591–602, (1996).
- Bard, J.F., "Selecting the appropriate input data set when configuring a permanent workforce," *Computers & Industrial Engineering*, **47**(4), 371–389, (2004).
- Bard, J.F., C. Binici, and A.H. deSilva, "Staff scheduling at the united states postal service," *Computers & Operations Research*, **30**(5), 745–771, (2003).
- Bechtold, S. E and L.W. Jacobs, "Implicit modeling of flexible break assignments in optimal shift scheduling," *Management Science*, **36**(11), 1339–1351, (1990).
- Berman, O., R.C. Larson, and E. Pinker, "Scheduling workforce and workflow in a high volume factory," *Management Science*, **43**(2), 158–172, (1997).
- Brusco, M. J and L. W. Jacobs, "Personal tour scheduling when starting time restrictions are present," *Management Science*, **44**(4), 534–547, (1998).
- Burke, E. K., P. D. Causmaecker, G. Vanden Berghe and H. Van Landeghem, "The state of the art of nurse rostering," *Journal of Scheduling*, **7**(6), 441–499, (2004).
- Burns, R. N, and M. W. Carter, "Work force size and single shift schedules with variable demands," *Management Science*, **31**(5), 599–607, (1985).
- Campbell, G. M. and M. Diaby, "Development and evaluation of an assignment heuristic for allocating cross-trained workers," *European Journal of Operational Research*, **138**(1), 9–20, (2002).
- Cheang, B., H. Li, A. Lim, and B. Rodrigues, "Nurse rostering problems—A bibliographic survey," *European Journal of Operational Research*, **151**, 447–460, (2003).

- Ernst, A.T., H. Jiang, M. Krishnamoorthy, and D. Sier, "Staff scheduling and rostering: A review of applications, methods and models," *European Journal of Operational Research*, **153**(1), 3–17, (2004).
- Dawid, H., J. Konig, and C. Strauss, "An enhanced rostering model for airline crews," *Computers & Operations Research*, **28**, 671–688, (2001).
- Franz, L.S and J.L. Miller, "Scheduling medical residents to rotations: Solving the large-scale multi-period staff assignment problem," *Operations Research*, **41**(2), 269–279, (1993).
- Garey, M.R and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman: New York, NY, 1979.
- Glover, F. and M. Laguna, *Tabu Search*, Kluwer Academic Publishers: Boston, MA, 1997.
- Hall, R.W. and D. Lotspeich, "Optimized lane assignment on an automated highway," *Transportation Research, Part C: Emerging Technology*, **4**(4), 211–229, (1996).
- Jarrah, A.I.Z., J.F. Bard, and A.H. deSilva, "Solving large-scale tour scheduling problems," *Management Science*, **40**(9), 1124–1145, (1994).
- Lewis, L.H., A. Srinivasan and E. Subramanian, "Staffing and allocation of workers in an administrative office," *Management Science*, **44**(4), 548–570, (1998).
- Malhotra, M.K., L.P. Ritzman, W.C. Benton, and G.K. Leong, "A model for scheduling postal distribution employees," *European Journal of Operational Research*, **58**, 374–385, (1992).
- Mukherjee, A.K. and K.C. Gilbert, "Lagrangian heuristics for instructor scheduling in executive development programmes," *Journal of Operations Research Society*, **48**(4), 373–382, (1997).
- Nanda, R. and J. Browne, *Introduction to Employee Scheduling*, Van Nostrand Reinhold: New York, NY, 1992.
- Sherali, H.D. and J. C. Smith, "Improving discrete model representations via symmetry considerations," *Management Science*, **47**(10), 1396–1407, (2001).
- Showalter, M.J., L.J. Krajewski, and L.P. Ritzman, "Manpower allocation in US postal facilities: A heuristic approach," *Computers & Operations Research*, **4**(4), 257–269, (1977).