Adaptive Labeling Algorithms for the Dynamic Assignment Problem

Warren B. Powell, Wayne Snow, Raymond K. Cheung

Institutions: Princeton University, Hong Kong University of Science and Technology

Published on: 01 Feb 2000 - Transportation Science (INFORMS)

Topics: Heuristics, Assignment problem and Scheduling (computing)

Adaptive Labeling Algorithms for the Dynamic Assignment Problem

WARREN B. POWELL AND WAYNE SNOW

Department of Operations Research and Financial Engineering, Princeton University, Princeton, New Jersey 08544

RAYMOND K. CHEUNG

Department of Industrial Engineering and Engineering Management, Hong Kong University of Science and Technology, Clearwater Bay, Kowloon, Hong Kong

We consider the problem of dynamically routing a driver to cover a sequence of tasks (with no consolidation), using a complex set of driver attributes and operational rules. Our motivating application is dynamic routing and scheduling problems, which require fast response times, the ability to handle a wide range of operational concerns, and the ability to output multiple recommendations for a particular driver. A mathematical formulation is introduced that easily handles real-world operational complexities. Two new optimization-based heuristics are described, one giving faster performance and the second providing somewhat higher solution quality. Comparisons to optimal solutions are provided, which measure the quality of the solutions that our algorithms provide. Experimental tests show that our algorithms provide high quality solutions, and are fast enough to be run in real-time applications.

We consider the problem of routing and scheduling a heterogeneous set of drivers to cover a known set of tasks. There is a reward for covering each task, and not all the tasks have to be covered. The reward received can depend on when a task is covered, and the cost of covering a task can be dependent, in an arbitrary way, on the characteristics of the driver. After a driver has finished one task, he may be free to cover another task, although subsequent assignments have to obey hours of service regulations and service commitments on tasks. A driver may or may not have to return home at the end of a shift, and we may be planning tours for the driver for several days into the future. A driver can handle only one task at a time; there is no in-vehicle consolidation. A task might represent a job that has to be performed at a specific location, but our work is motivated by applications where the task involves moving a load of freight from one location to another, thereby adding a spatial element to the problem.

The construction of each tour is subject to various constraints, such as government regulations on driver hours and constraints associated with each task. For example, each task has an associated time window and the driver must arrive at the origin and destination of the task between their respective time windows. Other constraints might reflect driver capabilities and load characteristics; for example, the load might require a driver with certain training or a specific type of tractor. Because there are very few hard constraints in real-time operations, assignments that violate stated goals are assessed penalties according to a function that varies according to the constraint violation. Similar penalty functions can also be constructed for other assignment rules and calibrated to simulate the human decision process.

This paper presents two algorithms, which are optimization-based heuristics, that solve this type of problem, which we refer to as the dynamic assignment problem. Our work is motivated by the need to solve these problems in a real-time setting. This need imposes three constraints on our solution approach. First, the run times must be exceptionally fast. In real problems, data updates can come in every few seconds, so the algorithm must be able to produce a revised solution very quickly for problems with perhaps 500 drivers. As a result, the algorithm must be amenable to finding new solutions given

Transportation Science, © 2000 INFORMS
Vol. 34, No. 1, February 2000

0041-1655/00/3401-0050 $05.00
pp. 50–66, 1526-5447 electronic ISSN

small perturbations to the problem, with sufficiently high quality that a dispatcher cannot readily find a better solution. Second, operational problems can be extremely complex. There is a lot more data available and issues to be considered in a real-time setting than is generally considered in more classical (static) planning problems. As a result, the modeling approach must be able to handle high levels of detail and complex operational issues very easily.

Third, we need to output not only a recommended tour, but alternative recommendations. In an operational problem, it is simply impossible to get all of the data right all of the time. As a result, a model that outputs a specific tour is typically of limited usefulness. Instead, an optimization model needs to give alternative task assignments for a particular driver, and alternative driver assignments for a particular task. An experienced dispatcher can use this information to make an assignment that considers other information that may not be in the computer. Thus, we need not only primal information (what the model recommends) but also dual information, so that the dispatcher can, in real-time (and without requiring a reoptimization by the model) identify sensible alternatives when the recommended solution is deemed to be inappropriate.

In this paper, we consider only static snapshots of data. We present our two algorithms and compare the solutions generated by these algorithms to an optimal solution, a comparison that can only be performed in a static setting. Our goal is to establish the flexibility of the modeling framework, and the quality of the solution in a static setting. The testing of these algorithms on a dynamic data set will be presented in a subsequent paper. However, we demonstrate that our methods provide fast solutions that are of very high quality for problems that are likely to arise in practice.

The problem we address in this paper has been handled in the literature under titles such as the vehicle scheduling problem, the full-truckload problem, or the crew scheduling problem. The vehicle scheduling problem, or the full-truckload problem, are typically cast in relatively simple terms, where a set of vehicles needs to cover a set of loads. Tours may need to cover tasks within a time window, and the length of the tour is typically constrained to the maximum number of hours a driver can spend on the road (see BALL et al., 1981; BODIN et al., 1983; ATKINSON, 1994, for example). This earlier work primarily used tour construction and tour improvement procedures. There now exists an extensive body of literature on local search and improvement procedures that have been primarily developed in the context of vehicle routing problems (which involve in-vehicle consolidation), but which could just as easily be adapted for vehicle scheduling problems. Examples can be found in GOLDEN and ASSAD (1986), SOLOMON (1987), SOLOMON, BAKER, and SCHATTER (1988), SAVELSBERGH (1985) and the more recent literature on tabu search, including GLOVER (1989, 1990) and GENDREAU, HERTZ, and LAPORTE (1994).

The use of local search heuristics in a real-time setting has received relatively little attention. A nice discussion of some of the issues is given in PSARAFTIS (1988). A significant drawback is the response time. The problem is not the speed of the algorithms, but rather the speed of communication between an on-line dispatch system and an optimization model. A dispatcher may receive a phone call with new information, and may have to assign a driver to the request while on the phone. There may not be enough time, in many practical settings, to send the information to an optimization algorithm and then wait for a response. An effective strategy that has proved successful in the truckload motor industry is to use the dual solution to offer a series of recommendations (based on the reduced cost of the assignment).

An alternative technology is the column generation/set partitioning strategy developed in the context of airline crew scheduling (see, for example, MARSTEN and SHEPARDSON, 1981; DESROSIERS, SOUMIS, and DESROCHERS, 1984; and DESROSIERS, SOLOMON, and SOUMIS, 1995). These techniques offer the ability to incorporate a wide variety of complex work rules, and do offer dual solutions from which alternative solutions can be generated. A significant weakness is their slow execution time (even with dual variables, we need to reoptimize as quickly as possible) especially in the presence of wide time windows (as we have). Also, column-generation strategies ensure that we have generated the optimal columns, but it is not clear if we have generated a high quality set of suboptimal columns. For example, assume a single driver can cover five loads during a day, and, each time he is assigned to a load, there are 20 loads to choose from. There are, then, potentially $20^5 = 320{,}000$ columns to choose from. A column-generation code will sample only a small percentage of these, and thus may not provide enough options for a production system where so-called optimal tours are, in fact, not at all optimal because of data problems.

This paper makes the following contributions:

- We present a formulation that readily handles a high degree of operational detail within a very simple mathematical framework. The formulation is math programming-based, and provides the dual information needed to produce alternative recommendations, as needed in real-time applications. We believe that our formulation of this problem is new.

- We present two new heuristic algorithms for solving this problem class. The algorithms are similar, and are denoted by the name RAPID (Resource Allocation Procedure for the Integrated Dynamic Assignment Problem). The first, RAPID-SL (single label), is exceptionally fast and provides high quality solutions to problems where the number of drivers or vehicles is sufficient to handle most of the tasks presented to the fleet. The second, RAPID-ML (multi-label), is somewhat slower but provides higher quality solutions than RAPID-SL, especially when the problems are resource constrained (more tasks than can be covered by the available drivers or vehicles).
- We compare solutions produced by both algorithms to optimal solutions in a variety of problem settings (using static data sets) to provide a rigorous estimate of the quality of the solutions provided by both algorithms. The work shows that RAPID-SL provides high quality solutions very quickly, although solution quality deteriorates in more tightly capacitated problems. RAPID-ML provides comparable results in problems that are not resource constrained, but much better results in problems that are resource constrained.

The RAPID algorithms belong in the same family of procedures as column generation methods. The primary difference is that, although column generation uses a master problem/subproblem format, we integrate the dual adjustment and column generation steps into a single procedure.

Our two algorithms differ in terms of how we represent the attributes of a driver (or vehicle) following the completion of a task. We begin in Section 1 with a formulation that uses a single label to describe a driver after a task completion. Next, Section 2 extends this concept to a multi-label concept, which uses a set of labels to represent potential drivers at the end of a task. Then, Section 3 develops an algorithm using the multi-label concept. Section 4 describes a comprehensive set of numerical experiments, where the algorithms are compared to each other and to an optimal solution based on a column-generation technique. Finally, Section 5 summarizes the results of the research.

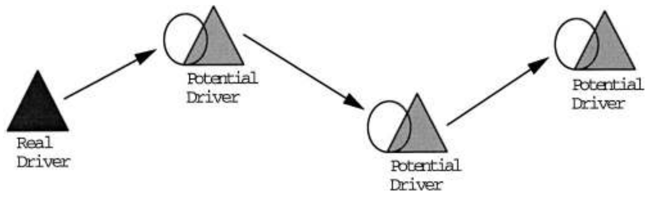

Fig. 1. Potential drivers.

1. THE SINGLE-LABEL LABELING ALGORITHM

IN OUR FIRST ALGORITHM, we represent the attributes of a driver by a label. The labels for the drivers are iteratively updated in the algorithm. In the following, we first present the notation, then the mathematical formulation, followed by a description of the algorithm.

1.1 Notation

We consider the notations used for tasks, drivers, the decisions for assigning drivers to tasks, and the costs involved.

Task Attribute and Driver Labels

Associated with a task are two sets of labels—one describes the attributes of that task and the other describes the attributes of the driver who has just completed that task. There are two types of drivers: real or potential. The real drivers are those whose attributes are known, typically the set of initial drivers. After a real driver covers a task, a new driver is generated, which we refer to as a potential driver. This potential driver can then cover other tasks, resulting in other potential drivers as illustrated in Figure 1.

To simplify our notation, we assume that the set of initial drivers are generated by some dummy tasks. Let

$\mathcal{L}^0$ = set of dummy tasks that generate the initial drivers
$\mathcal{L}^a$ = set of actual tasks to be covered
$\mathcal{L}$ = set of all tasks = $\mathcal{L}^0 \cup \mathcal{L}^a$
ℓ = index of a particular task, $\ell \in \mathcal{L}$.

The number of attributes for a task is arbitrary and depends on the applications. Let

$\mathcal{B}$ = space containing all possible task attribute labels
b_ℓ = the attribute label associated with task ℓ.

In our presentation, we assume that there are only five attributes for each task ℓ, defined as

$$b_\ell = \begin{bmatrix} b_{\ell,1} \\ b_{\ell,2} \\ b_{\ell,3} \\ b_{\ell,4} \\ b_{\ell,5} \end{bmatrix} = \begin{bmatrix} \text{origin} \\ \text{destination} \\ \text{start of origin time window} \\ \text{end of origin time window} \\ \text{length of task} \end{bmatrix}.$$

For the driver label, let

$\mathcal{A}$ = space containing all possible driver attribute labels
a_ℓ = the driver label associated with the driver generated by task ℓ.

We assume that there are only four attributes for each driver r:

$$a_\ell = \begin{bmatrix} a_{\ell,1} \\ a_{\ell,2} \\ a_{\ell,3} \\ a_{\ell,4} \end{bmatrix} = \begin{bmatrix} \text{location} \\ \text{time of availability} \\ \text{hours of service elapsed at time } a_{\ell,2} \\ \text{daily duty time allowance} \end{bmatrix}.$$

Costs

The costs and travel times are defined over the attribute space because a task can be covered by a real driver or a potential driver whose attributes are not known initially. Let $\Re$ be the set of real numbers. For each $a \in \mathcal{A}$ and $b \in \mathcal{B}$ define

$v: \mathcal{A} \to \Re$ = salvage function for a driver label a
$\tau: \mathcal{A} \times \mathcal{B} \to \Re$ = total travel time incurred in driver label a covering a task with attribute label b
$c: \mathcal{A} \times \mathcal{B} \to \Re$ = net contribution associated with driver label a covering a task with attribute label b.

The number $v(a)$ represents the value of stranding a driver with attribute a. This may be a cost in the case where a driver has to return home, or it may be an expected revenue derived from historical data. Notice that $v(a)$ is a useful construct in problems that may not have a well-defined end of horizon. The revenue and cost functions are defined over driver and task labels, which allows us to incorporate a great deal of complexity into these functions as the application requires. Notice that $c(a_{\ell'}, \ell)$ represents the contribution of covering task ℓ by the driver generated by task ℓ'. For fixed ℓ' and ℓ, the value of $c(a_{\ell'}, \ell)$ can vary because it depends on the attributes of ℓ' and ℓ.

Driver–Task Assignment

When the driver generated by a task ℓ' with attribute label $a_{\ell'} \in \mathcal{A}$ covers a task with attribute label $b_\ell \in \mathcal{B}$, it generates a new driver with label a_ℓ, defined by

$$\mathcal{M}(a_{\ell'}, b_\ell) = a_\ell = \begin{bmatrix} a_{\ell,1} \\ a_{\ell,2} \\ a_{\ell,3} \\ a_{\ell,4} \end{bmatrix} = \begin{bmatrix} b_{\ell,2} \\ \tau(a_{\ell'}, b_\ell) + a_{\ell',2} \\ \tau(a_{\ell'}, b_\ell) + a_{\ell',3} \\ a_{\ell',4} \end{bmatrix}.$$

The mapping, $\mathcal{M}: \mathcal{A} \times \mathcal{B} \to \mathcal{A}$, defines the attributes of a new potential driver r that will be available at the destination of task ℓ.

Decision Variables

A driver (real or potential) has two options: covers a task or is unassigned. Thus, the decision variables are given by

$$x_{\ell',\ell} = \begin{cases} 1 & \text{if the driver generated by task } \ell' \text{ covers task } \ell \\ 0 & \text{otherwise,} \end{cases}$$

$$z_\ell = \begin{cases} 1 & \text{if the driver generated by task } \ell \text{ is unassigned to any task.} \\ 0 & \text{otherwise.} \end{cases}$$

Subtours

A subtour contains a sequence of chained driver–task assignments. Let

$\mathcal{S}$ = set of subtours
s = index of a subtour, $s \in \mathcal{S}$
$|s|$ = number of arcs in the subtour.

A subtour s can be represented by a sequence of task pairs,

$$\{(\ell_{s1}, \ell_{s2}), (\ell_{s2}, \ell_{s3}), \ldots, (\ell_{|s|}, \ell_{s1})\}.$$

In the subtour, the potential driver generated by task ℓ_{s_1} will eventually cover task ℓ_{s_1}, creating a loop that is not allowed in the solution.

1.2 Formulation

Using the notation presented so far, the problem formulation is given by

$$\underset{x,z}{\text{Maximize}} \sum_{\ell' \in \mathcal{L}} \sum_{\ell \in \mathcal{L}} c(a_{\ell'}, b_\ell) x_{\ell',\ell} + \sum_{\ell' \in \mathcal{L}} z_{\ell'} v(a_{\ell'}) \quad (1)$$

subject to

$$\sum_{\ell' \in \mathcal{L}} \mathcal{M}(a_{\ell'}, b_\ell) x_{\ell',\ell} - a_\ell = 0 \quad \forall \ell \in \mathcal{L}^a \quad (2)$$

$$\sum_{\ell' \in \mathcal{L}} x_{\ell',\ell} - \left(\sum_{\ell'' \in \mathcal{L}} x_{\ell,\ell''} + z_\ell \right) = 0 \quad \forall \ell \in \mathcal{L}^a \quad (3)$$

$$\sum_{\ell'' \in \mathcal{L}} x_{\ell,\ell''} + z_\ell = 1 \quad \forall \ell \in \mathcal{L}^0 \quad (4)$$

$$\sum_{\ell' \in \mathcal{L}} x_{\ell',\ell} \leq 1 \quad \forall \ell \in \mathcal{L} \tag{5}$$

$$\sum_{\ell \in \mathcal{L}} x_{\ell',\ell} \leq 1 \quad \forall \ell' \in \mathcal{L} \tag{6}$$

$$z_{\ell'} \in \{0, 1\} \quad \forall \ell' \in \mathcal{L} \tag{7}$$

$$x_{\ell',\ell} \in \{0, 1\} \quad \forall \ell', \ell \in \mathcal{L} \tag{8}$$

$$\sum_{(\ell', \ell) \in s} x_{\ell',\ell} + 1 - |s| \leq 0 \quad \forall s \in \mathcal{S}, \quad |s| \geq 2 \tag{9}$$

Constraint 2 represents the definition of the attribute vector of a future driver label (introducing, at the same time, a complex nonlinear constraint). Constraint 3 ensures that flow conservation is maintained for each task. Constraint 4 means that an initial driver must either cover a task or be unassigned. Constraints 5 and 6 are the bundle constraints for each task and driver. These constraints make sure that each task will be covered by at most one driver, and each driver can cover at most one task at a time. Constraints 7 and 8 are the integrality constraints on the decision variables. Finally, constraint 9 prevents subtours.

1.3 RAPID-SL Algorithm

The difficulties in solving the optimization defined by 1–9 come from two aspects. First, the problem is an integer program that can contain a large number of subtour constraints. Second is the presence of the highly nonlinear constraint in Eq. 2. In the RAPID-SL algorithm, we iteratively fix the labels a_ℓ, $l \in \mathcal{L}^a$ to a value a^k_ℓ (in iteration k). This eliminates Eq. 2 and also allows us to define

$$\bar{c}^k_{\ell',\ell} = c(a^k_\ell, b_\ell).$$

We may now rewrite the objective function as

$$\mathrm{F}(a^k) = \underset{x,z}{\text{Maximize}} \sum_{\ell' \in \mathcal{L}} \sum_{\ell \in \mathcal{L}} \bar{c}^k_{\ell',\ell}, x_{\ell',\ell} + \sum_{\ell' \in \mathcal{L}} z_{\ell'} v(a_{\ell'}), \tag{10}$$

subject to 3–9. The problem $\mathrm{F}(a^k)$ is now just a driver scheduling problem. If we relax the subtour constraints 9, the resulting problem is a pure network that can be solved easily.

We propose an algorithm with an outer iteration that fixes the labels a^k, and an iteration that penalizes violations of the subtour constraints. Our updating rules are quite simple. The important research result is that the algorithm works quite well under many situations that arise in practice, and easily handles very large problems (e.g., hundreds of vehicles) in real-time.

Let

$$\beta_s = \text{a multiplier (penalty cost)}$$
$$\text{for the } s\text{th subtour constraint.}$$

The steps and the interpretation of the algorithm are as follows.

Step 1. Initialization

$k = 0$.

For each $\ell \in \mathcal{L}$, generate a driver label a_ℓ such that

$$a_{\ell,1} = b_{\ell,2}$$
$$a_{\ell,2} = b_{\ell,3} + b_{\ell,5}$$
$$a_{\ell,3} = b_{\ell,5}$$

Compute $\bar{c}^k_{\ell',\ell} = c(a^k_{\ell'}, b_\ell)$, $\forall r \in \mathcal{R}$, $\ell \in \mathcal{L}$.

Explanation. RAPID-SL builds an initial estimate of a driver label by giving it the most optimistic set of attributes possible given that we know it has covered this task. For example, if the time window on the task origin starts at 3:00 P.M. and the task takes one hour to complete, then the most optimistic time of availability is 4:00 P.M. The other attributes are set using similar reasoning and are continually revised at each iteration as tasks are covered.

Step 2. Flow adjustment

Set $k = k + 1$, solve the relaxation problem (P1) below as a minimum cost flow problem. For subtour s, set $\beta_s = 0$

Explanation. After relaxing the subtour constraint, the resulting optimization is

$$\text{(P1)} \quad \underset{x,z}{\text{Maximize}} \sum_{\ell' \in \mathcal{L}} \sum_{\ell \in \mathcal{L}} \bar{c}^k_{\ell',\ell}\, x_{\ell',\ell} + \sum_{\ell' \in \mathcal{L}} z_{\ell'} v(a^k_{\ell'}) - \sum_{\{s \in \mathcal{S}, |s| \geq 2\}} \beta_s \left\{ \sum_{\ell, \ell' \in s} x_{\ell,\ell'} + 1 - |s| \right\}$$

$$\text{subject to} \quad \text{constraints 3–8}$$

Initially, β_s are 0 but will be updated in Step 3. Problem P1 is a pure network flow problem that can be solved by standard optimization technique such as network simplex algorithm.

Step 3. Identify and eliminate subtours

Find a subtour s with positive flow, that is, to find s such that

$$s = \{(\ell_{s1}, \ell_{s2}), (\ell_{s2}, \ell_{s3}), \ldots, (\ell_{|s|}, \ell_{s1})\}$$

where $x_{\ell_{s1},\ell_{s2}} = 1$, $x_{\ell_{s2},\ell_{s3}} = 1, \ldots,$ and $x_{\ell_{|s|},\ell_{s1}} = 1$.

Calculate

$$\gamma_s = \min_{(\ell', \ell)\in s} \{\bar{c}^k_{\ell',\ell} + p_{\ell'} - p_\ell\} \tag{11}$$

Update $\beta_s = \beta_s + \gamma_s$
Solve Problem P1 again.
Repeat this step until no subtour exists in the optimal solution; or until a prespecified number of times.

Explanation. We first identify the subtours created by the optimal solution of P1. Then, we use the dual variables to increase the cost of the arcs in the subtours for penalizing the formation of such subtours when P1 is re-optimized.

Step 4. Label and cost adjustment
Update the labels for potential drivers:

```
For each $\ell' \in \mathscr{L}^0$:
  while $x_{\ell',\ell} = 1$ for some $\ell \in \mathscr{L}$ do the following:
    Obtain $a^k_\ell = \mathcal{M}(a^k_{\ell'}, b_\ell)$.
    Compute $\bar{c}^k_{\ell',\ell} = c(a^k_\ell, b_{\hat{\ell}})\ \forall \hat{\ell} \in \mathscr{L}$.
    $\ell' \leftarrow \ell$.
```

Explanation. Here we trace out the assignments made for each driver and update the potential driver attributes to those of the driver that now covers that task. Having changed the potential driver, we then re-cost all the arcs out of that potential driver.

Step 5. Termination check
If no assignments have changed from the last iteration or k reaches a pre-specified maximum allowable number of iterations, then we terminate the algorithm. Otherwise, go to Step 2.

The strengths of this procedure are that it lends itself very well to real-time updating, requiring only a few iterations to perform real-time updates. One clear disadvantage is the development of subtours that can have a significant impact on solution quality. We use a Lagrangian approach to mitigate the formulation of subtours, but this does not completely eliminate the problem. In addition, the labels can alternate between discrete values, creating another source of instability.

2. MULTI-LABEL FOR DRIVER-TASK ASSIGNMENT

WE NOW EXTEND the notion of a single potential driver at the end of each task to a set of path-dependent drivers. With this new notion, we develop an algorithm RAPID-ML whose details are described shortly. There are two major differences between the labels used in RAPID-SL and those in RAPID-ML. First, in RAPID-SL, after a task is covered, a potential driver is generated and we approximate its attributes. In RAPID-ML, a set of possible drivers are associated with a task. For each possible driver, we keep track of the path to reach the current task. Second, in RAPID-ML, using these driver labels, we can generate tour labels, which correspond to possible sequences of tasks that any driver with a specific set of attributes may perform in the future. These tour labels are similar to those found in column-generation approaches to this problem. Thus, at any task ℓ, in addition to the task labels that are the same as those in RAPID-SL, there exist two sets of labels: driver labels and tour labels. To completely represent a task, we must include all the labels at that task. Each driver label describes the past of a specific driver that may cover that task. In contrast, each tour label describes one possible future for a driver that covers this task. In RAPID-SL, we do not use tour labels. In this section, we describe the labels in detail.

2.1 Driver Labels

The attribute label a_ℓ for a potential driver generated by task ℓ in RAPID-SL is now replaced by a set of driver labels describing the possible drivers generated by this task. Let

$\mathcal{R}_\ell$ = set of possible path-dependent drivers at the completion of task ℓ, $\forall \ell \in \mathscr{L}$
$\mathcal{A}_\ell$ = set of all possible driver labels associated with the path-dependent drivers in $\mathcal{R}_\ell$ at the completion of task ℓ, $\forall \ell \in \mathscr{L}$
$\mathcal{R} = \cup_{\ell\in\mathscr{L}} \mathcal{R}_\ell$.

We now use a cone to denote the set of drivers at a task and a triangle to represent a single driver label in that set. Figure 2 shows the various ways for each driver to cover task 2. Each different path produces a separate driver label at task 2 that is specific to the path taken by that driver. The two drivers have a total of five different paths to reach task 2, and thus create five different driver labels at task 2. For each driver label, we also include additional attributes to specify the path to reach the current task. To simplify our discussion, let us introduce two functions defined for a driver $r \in \mathcal{R}_\ell$ with the label $a_r \in \mathcal{A}_\ell$:

prev_task(a_r) = the last task before task ℓ on the path that generate driver r
prev_label(a_r) = the driver label for the last driver before task ℓ on the path that generates driver r.

For example, in Figure 2, prev_task(a_6) returns task 1 and prev_label(a_6) returns a_1.

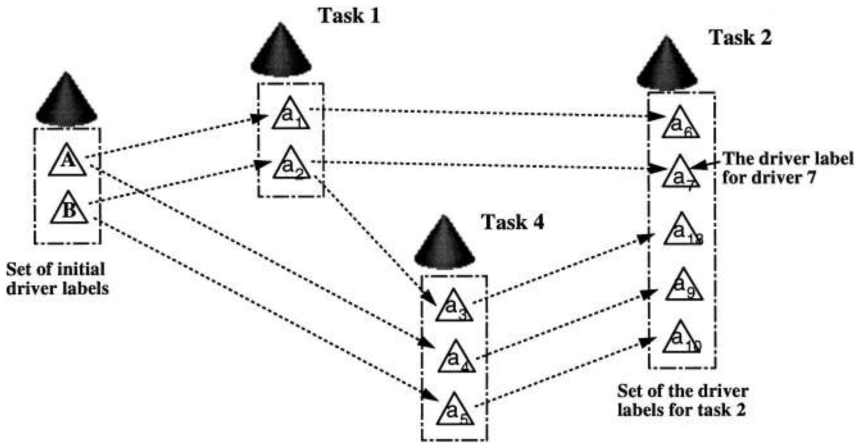
Task 1
Task 2
Task 4
The driver label for driver 7
Set of initial driver labels
Set of the driver labels for task 2
A
B
a1
a2
a3
a4
a5
a6
a7
a8
a9
a10

Fig. 2. Illustration of driver labels.

2.2 Tour Labels

Each tour label describes one possible sequence of tasks that a driver may perform after it has completed a task. To be sure that the tour is feasible for a given driver, each label contains information about the tour. Let

$\mathcal{T}$ = space containing all possible tours labels
$\mathcal{T}_\ell$ = set of tour labels at the completion of task ℓ, $\forall \ell \in \mathcal{L}$.

Each tour label t, $t \in \mathcal{T}_\ell$, has the following attributes:

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = \begin{bmatrix} \text{future revenue attainable from covering the tour} \\ \text{time required to complete the tour and attain } t_1 \\ \text{latest arrival time at task } \ell \text{ for attaining } t_1 \text{ in time } t_2 \\ \text{terminating location for the tour} \end{bmatrix}.$$

The set of tour attributes can be expanded as the application requires.

We use a sphere to represent the set of tour labels and a circle represents a single tour label. Some examples of tour labels are shown in Figure 3. Label t^1 describes the tour that covers task 2 after task 1 and then terminates. Label t^3 describes the tour that covers task 4 and then 2, after task 3 has been completed. Notice that there is only one label, t^2, at task 2. This is because all the tours that cover this task terminate there and, hence, will have the same tour label.

2.3 Label Generation

Suppose that the driver $r' \in \mathcal{R}_{\ell'}$ will cover task ℓ as shown in Figure 4, a new driver r is generated at the completion of task ℓ. The attributes of the newly created driver label a_r are similar to those of the driver label in RAPID-SL, but with the additional

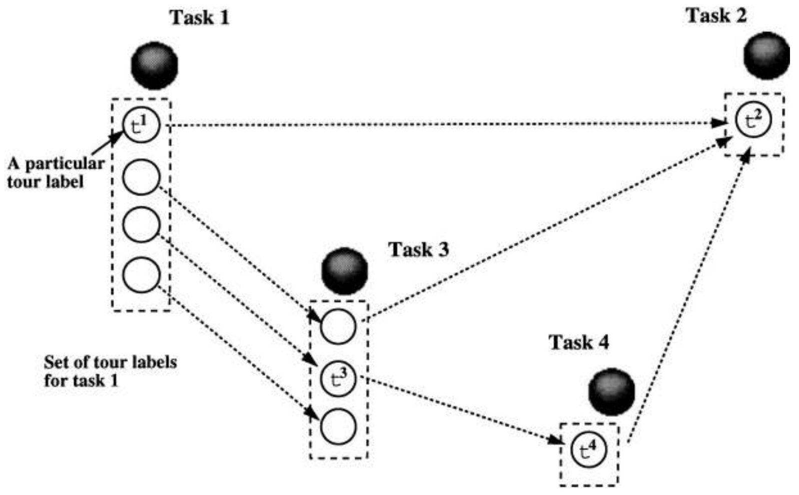

Fig. 3. Forward tours represented by each tour label: a tour label captures the characteristics of a tour that might start with a particular task.

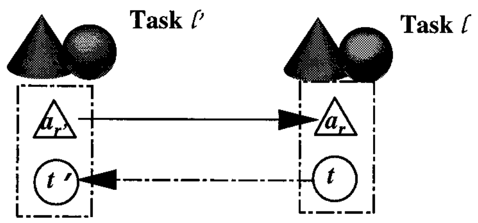

Fig. 4. Generation of labels.

attributes for tracking the last task visited (i.e., task ℓ') and the corresponding driver label in the last task (i.e., $a_{r'}$).

After the driver labels have been generated, which correspond to a sequence of tasks in a tour, we can use these labels to generate tour labels for that tour. For example, assume that there is nothing that the driver can do after completing task ℓ in Figure 4. The corresponding tour label t at task ℓ is

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ b_{\ell,4} \\ b_{\ell,2} \end{bmatrix}.$$

Then, a new label t', $t' \in \mathcal{T}_{\ell'}$ will be generated by using a mapping $\mathcal{F}$: $\mathcal{A} \times \mathcal{B} \times \mathcal{T} \to \mathcal{T}$, where

$$\mathcal{F}(a_{r'}, b_\ell, t) = t' = \begin{bmatrix} t'_1 \\ t'_2 \\ t'_3 \\ t'_4 \end{bmatrix} = \begin{bmatrix} t_1 + c(a_{r'}, b_\ell) \\ t_2 + \tau(a_{r'}, b_\ell) \\ \min\{b_{\ell',4}, t_3 - \tau(a_{r'}, b_\ell)\} \\ t_4 \end{bmatrix},$$

where $c(a_{r'}, b_\ell)$ is the contribution that the driver with label $a_{r'}$ makes by covering task ℓ.

2.4 Decision Variables and Revenue Functions

Generating the driver and tour labels enables us to calculate how much revenue a driver r is capable of earning after it covers a task ℓ by solving a simple optimization problem,

$$\mu_{r,\ell} = \operatorname*{Max}_{t \in \mathcal{T}_\ell} t_1 \qquad (12)$$

$$\text{subject to} \quad a_{r,3} + \tau(a_r, b_\ell) + t_2 \leq a_{r,4}. \qquad (13)$$

Constraint 13 ensures that the driver does not exceed its maximum daily driving limit for that day; other constraints can be included as the application requires.

Given we now have multiple drivers at each task, we must now expand the set of decision variables to recognize flow between a specific driver pair. For each $r' \in \mathcal{R}$ and $r \in \mathcal{R}_\ell$ for each $\ell \in \mathcal{L}$, let $x_{r',r}$ be the flow from driver r' to driver r. That is,

$$x_{r',r} = \begin{cases} 1 & \text{if driver } r' \text{ covers task } \ell \\ & \text{and becomes driver } r \\ 0 & \text{otherwise.} \end{cases}$$

Because a task can only be covered by a single driver, we impose the constraint

$$\sum_{r \in \mathcal{R}_\ell} \sum_{r' \in \mathcal{R}_{\ell'},\, \ell' \in \mathcal{L}} x_{r',r} \leq 1 \quad \forall \ell \in \mathcal{L}$$

in our formulation.

2.5 Initial Tour Generation

Before we apply the RAPID-ML algorithms, we need to generate initial tours and the initial sets of labels. The tour-generation logic is based on a breadth-first approach by generating driver labels at the m most profitable tasks and putting them into a queue. We then repeat this procedure for each driver in the queue, which generates new driver labels as the tour progresses until the queue is empty. Then, we use the backward recursive strategy discussed earlier in Section 2.3, generating the tour labels.

Define

Q = queue for (real or potential) drivers
n_r = number of possible tasks that can be covered by driver r.

For each distinct initial driver, say r^0, we generate a set of tours (and labels) as follows.

Procedure ILG (Initial label generation)

LG1: Initialize $Q = \{r^0\}$.
LG2: If Q is empty, go to LG5. Otherwise, remove the top element from Q and denote it as r. If $n_r = 0$, then repeat this step until $n_r > 0$.
LG3: For all the n_r tasks that can be covered by r, compute and rank the contribution of covering these tasks such that: $c(a_r, b_{\ell_1}) \geq \cdots \geq c(a_r, b_{\ell_{n_r}})$.
LG4: For each task ℓ_i, $i \leq \min\{m, n_r\}$, create a new driver $\hat{r}$ and update the labels as

$$a_{\hat{r}} = \mathcal{M}(a_r, b_{\ell_i})$$

$$\mathcal{R}_{\ell_i} = \mathcal{R}_{\ell_i} \cup \hat{r}$$

Add $\hat{r}$ to the bottom of Q. Go to LG2.
LG5: Let ℓ be the last task of the tour just found,

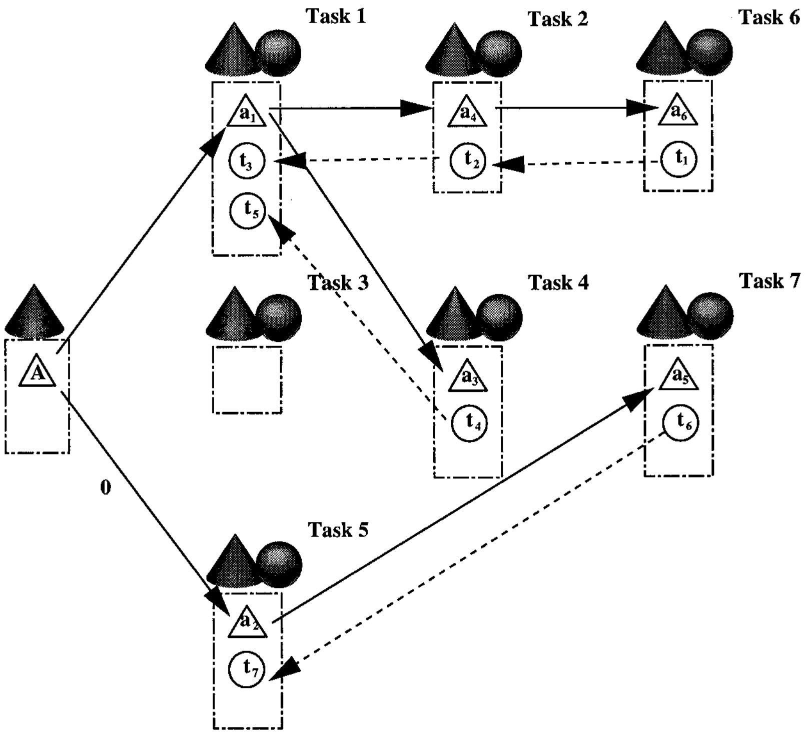

Fig. 5. Illustration of creating new driver labels and tour labels initially.

and a_r be the driver label generated by task ℓ where $n_r = 0$ (no task can be covered by r).

For task ℓ, initialize a tour label t as

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ b_{\ell,4} \\ b_{\ell,2} \end{bmatrix}$$

Define the tour labels for the tasks on the path just found recursively:

If r is not an initial real driver, repeat the following

$\ell' = \text{prev\_task}(a_r)$

$r' = \text{prev\_driver}(a_r)$

$\mathcal{T}_{\ell'} = \mathcal{T}_{\ell'} \cup t'$ where $t' = \mathcal{F}(a_{r'}, b_\ell, t)$

$r \leftarrow r'$

An example of this procedure is given in Figure 5 when we discuss the steps of the RAPID-ML algorithm.

3. THE RAPID-ML ALGORITHM

DEFINE:

$\mathcal{C}$ = candidate list of drivers to be assigned or reassigned

p_ℓ = estimate of the dual price for constraint 3 for task ℓ at the current iteration

ν_r = estimated revenue of the best assignment for driver r

w_r = estimated revenue of the second-best assignment for driver r

k = iteration counter

β = discount factor for dual price reduction (described later)

N = number of iteration before each dual price reduction (described later)

ϵ = a positive number used to avoid cycling in the algorithm.

Notice that, in the algorithm, the sets such as $\mathcal{R}_\ell$ and $\mathcal{T}_\ell$ and the prices and revenues (p_ℓ, ν_r, w_r) are iteratively updated. In the description below, these sets and numbers are implicitly assumed as the most current ones.

3.1 Basic Ideas

RAPID-ML partially relaxes the flow conservation constraints given by Eqs. 3 and 4 to allow for the case where the flow into any given task or terminal may exceed the flow out. In other words, the "=" in 3 is replaced by a "≥".

The algorithm RAPID-ML is based around a candidate list that contains a list of all the drivers available for assignment. At each iteration, we retrieve an element from this list and, using the tour labels, assign it to its best alternative using a flow-adjustment process. After updating the sets of both driver and tour labels, we re-assign some of the drivers to tasks based on estimated dual prices. The newly assigned drivers will create new flow. At the same time, some assigned drivers can become unassigned. As a result, the downstream flow from these drivers needs to be eliminated. We repeat this process until the candidate list is empty. In RAPID-SL, we solve a sequence of minimum cost flow problems to get the dual prices. In RAPID-ML, we use an approach adopted from the auction algorithm of BERTSEKAS (1988) for deriving the dual prices. When evaluating the best decision for a driver, we take three elements into consideration. The first is the revenue earned from covering the task, $c(r, \ell)$. The second, $\mu_{r,\ell}$, is how much revenue the driver can earn after it has completed task ℓ, and the third term, $\bar{p}_\ell$, is the estimated price of the task. This is how much the driver must be prepared to pay if it wants to cover this task. It measures the level of competition for the task, the more intense the competition, the higher the price.

3.2 The Steps

We now formally present each step of the algorithm together with intuitive explanations and illustrations.

Step 1. Initialization

Set $p_\ell = 0$, $\forall \ell \in \mathscr{L}$ and $\epsilon = \epsilon^0$ (a user specified parameter)

Set $\mathscr{C}$ = {set of initial drivers}

For each task ℓ, set $\mathscr{R}_\ell = \emptyset$ and initialize $\mathscr{T}_\ell = \{t\}$ where

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ t_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ b_{\ell,4} \\ b_{\ell,2} \end{bmatrix}$$

Explanation. There is no driver label initially. However, we generate a tour label for each task, reflecting the option of doing nothing after completing that task.

Step 2. Initial label generation

We use the procedure ILG described in Section 2.5 to create the initial sets of drivers and tour label.

Example. Figure 5 illustrates an example of this process. We start with the initial driver A and loop over the seven tasks. In this example, we take $m = 2$. The most profitable tasks for driver A are 1 and 5; task 1 being the most profitable. We then generate the new driver labels a_1 and a_2 at tasks 1 and 5, respectively. For driver 1 (with label a_1), the best and the second-best tasks are tasks 4 and 2, and thus we create a_3 and a_4. The driver cannot feasibly cover any more tasks after task 4, so the tour stops here. We go on to create other driver labels: in our example, a_5 and a_6 (from drivers 2 and 4, respectively). We now trace back along the tasks that were covered and generate a new tour label, which describes the completion of the tour from that task on. For task 6, we have an initial tour label t_1. Tracing back the path, we create t_2 and t_3 for tasks 2 and 1, respectively. By repeating the procedure, we obtain the tour labels t_4, t_5, t_6, and t_7.

Step 3. Flow adjustment

FA1: Adjust iteration counter and retrieve a driver from the candidate list

$k = k + 1$

Let r = first element of $\mathscr{C}$.

FA2: If r is unassigned (i.e., $\Sigma_{r' \in \mathscr{R}} x_{r,r'} = 0$), determine the task to cover next

Estimate the revenue of the best assignment,

$$\nu_r = \max_{\ell \in \mathscr{L}} \{c(a_r, b_\ell) + \mu_{r,\ell} - p_\ell\} \qquad (14)$$

Let ℓ^* be the argmax of equation (14). Go to FA4.

FA3: If r is assigned, determine whether re-assignment should be made

Find r'' such that $x_{r,r''} = 1$ (r'' is the current best assignment for r) and identify task ℓ'' such that $r'' \in \mathscr{R}_{\ell''}$.

Re-estimate the revenue of the new second-best assignment;

$$w_r = \max_{\ell \neq \ell''} \{c(a_r, b_\ell) + \mu_{r,\ell} - p_\ell\}, \qquad (15)$$

where $\mu_{r,\ell}$ is defined by Eq. 12

If $w_r < \nu_r - \epsilon$

No new assignment is made. Remove driver r from $\mathscr{C}$ and insert the previous driver r' (with $x_{r',r} = 1$) to $\mathscr{C}$.

Go to FA1.

else (i.e., need to change of the assignment of r)

Let ℓ^* be the argmax of Eq. 15.

Go to FA4.

FA4: Identify the driver who has covered task ℓ^* and remove the downstream flow out of it

If $\exists r' \in \mathscr{R}_{\ell^*}$ and $\hat{r} \in \mathscr{R}$ such that $x_{\hat{r},r'} = 1$

While $x_{\hat{r},r'} = 1$ for some $r' \in \mathscr{R}$

Set $\mathscr{C} = \mathscr{C} \cup \hat{r}$, $x_{\hat{r},r'} = 0$, and $\hat{r} \leftarrow r'$.

FA5: Create new labels and assign flow to task ℓ^*

Obtain $a_{r^*} = \mathscr{M}(a_r, b_{\ell^*})$, set $\mathscr{R}_{\ell^*} = \mathscr{R}_{\ell^*} \cup a_{r^*}$ and set $x_{r,r^*} = 1$.

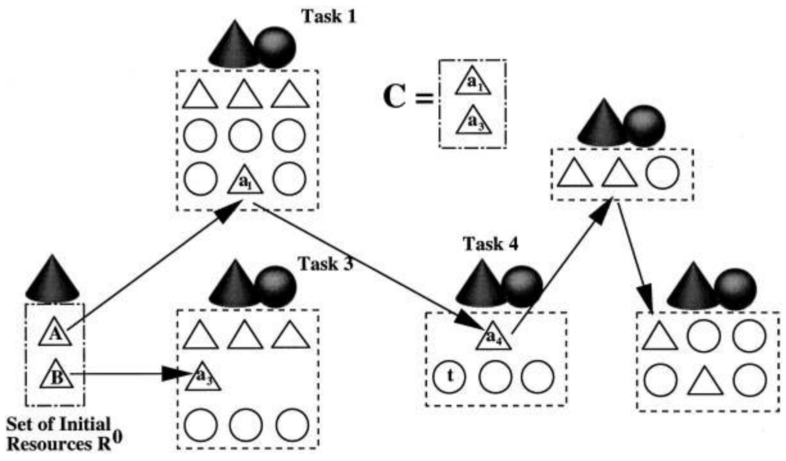

Fig. 6. An illustration of driver and tour labels, and a current set of tours.

Explanation of Step 3. When a driver r is retrieved from the candidate list, it may be either unassigned or it may be assigned to some task, ℓ^*. If the driver is unassigned, then we must calculate v_r to determine which task to cover next using Eq. 14. Having determined ℓ^*, we then check to see if that task is already covered by another driver (say $\hat{r}$) and remove all flow out of that driver. If, alternatively, the driver is already assigned, then we evaluate the next-best assignment using Eq. 15 to check if the current assignment is still the best assignment (within ϵ) for this driver. If it is, then we move on to the next element in the candidate list. Otherwise, we redirect the flow out of driver r to its new best assignment.

Example. Consider Figure 6 where driver 1 (with label a_1) is the first in the candidate list that is currently assigned to task 4. After verifying that the best strategy for this driver is still task 1, we remove driver 1 from the candidate list and move on to the next element driver 3 (with label a_3). This driver is currently unassigned, and, on evaluating Eq. 14, we find that the best task for this driver is task 4 using tour label t. This task is currently covered by the driver with label a_1 so we remove all the flow out of a_1 and add 1 to the candidate list as an unassigned driver. We now generate the new driver label a_5 at task 4, assign flow from a_3 to a_5, and place driver 5 into the candidate list. The new situation is shown in Figure 7.

Step 4. Dual price adjustment

Given that driver r is assigned to task ℓ^*. Compute

$$w_r = \max_{\ell \neq \ell^* \in \mathscr{L}} \{c(a_r, b_\ell) + \mu_{r,\ell} - p_\ell\} \tag{16}$$

$$p_{\ell^*} = p_{\ell^*} + v_r - w_r + \epsilon \tag{17}$$

Explanation of Step 4. Having assigned flow from r to ℓ^*, we now evaluate the second-best alternative for driver r using Eq. 15. Equation 17 then gives the reduction in revenue incurred if driver r was diverted to its second-best alternative. This is our estimate for p_ℓ. This is the adjustment given in Bertsekas' (1988) auction algorithm for the assignment problem. However, in the assignment problem, after a driver is assigned to some task, it remains covered at each subsequent iteration. In our problem, when a driver is knocked off a task, it also frees up all the other tasks that it may have covered. The problem we face is what to do with the prices on these newly uncovered tasks. This issue is addressed in Step 6. To prevent the algorithm from entering into price wars over a task by making bids of zero ($\nu_r = w_r$), we perturb the price by a positive amount ϵ each time the price is adjusted.

```
Step 5. Update tour labels
  Let task $\ell^*$ be the last task of the tour and $a_{r^*}$ be the driver label generated by task $\ell^*$ (i.e., $c(a_{r^*}, b_\ell) = -\infty, \forall \ell \in \mathscr{L}$). Update the tour labels along the tour recursively
    Initialize $t \in \mathscr{T}_{\ell^*}$ as the tour label at task $\ell^*$ representing doing nothing after task $\ell^*$ (see LG5 in Section 2.5).
    Set $r = r^*$, $\ell = \ell^*$, and $\mathscr{C} = \mathscr{C} \cup r'$.
    If $r$ is not an initial real driver, repeat the following:
      Find previous task $\ell'$ and driver $r'$ of $r$ by identifying $\ell'$ such that $\exists r' \in \mathscr{R}_{\ell'}$ such that $x_{r',r} = 1$.
      Create a label by $t' = \mathscr{F}(a_{r'}, b_\ell, t)$.
      if $t' \notin \mathscr{T}_{\ell'}$ then $\mathscr{T}_{\ell'} = \mathscr{T}_{\ell'} \cup t'$.
      $\ell \leftarrow \ell'$, $t \leftarrow t'$, $r \leftarrow r'$.
```

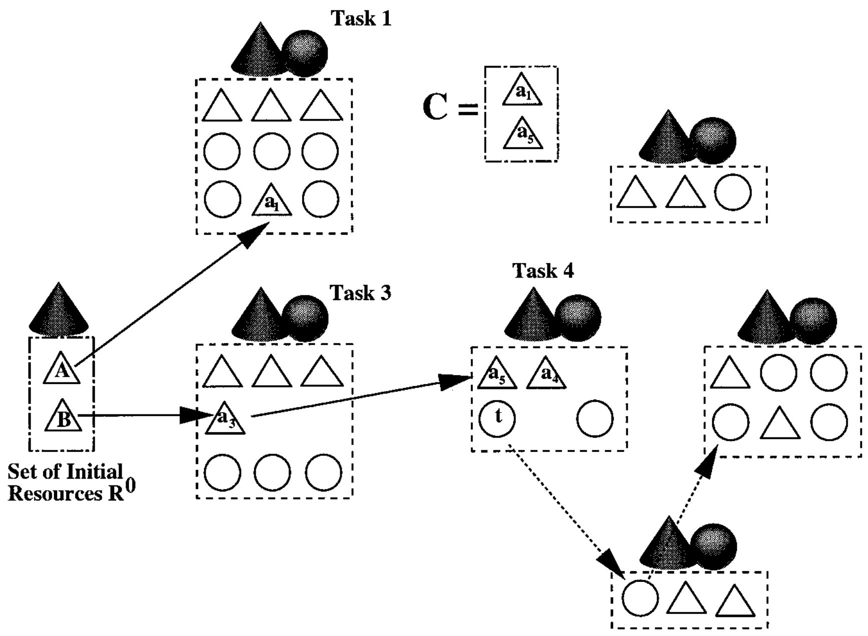

Fig. 7. Illustration of how adjusting the flow changes the flow into specific driver labels.

Explanation of Step 5. After a driver has completed the last task in its tour, we must check to see if the tour is represented by the existing set of tour labels $\mathcal{T}$. If not, then we augment the set of tour labels at each task to represent the new tour. The generation of these labels follows the same recursive strategy as the initial label-building process. In practice, it is this continual updating of $\mathcal{T}$ that gives us more accurate estimates of $\mu_{r,\ell}$ than the labels developed from the initial greedy approach. After the tour is completed, we now place the driver that covers the last task in the tour back into the candidate list so that we can check the validity of this decision at some subsequent iteration.

Step 6. Secondary dual price adjustment
For every N iterations (i.e., k is divisible by N), reduce the dual prices for uncovered task: $p_\ell = p_\ell \cdot \beta$ for some $0 \leqslant \beta < 1$.

Explanation of Step 6. When a driver is knocked off a tour consisting of several tasks, some of the tasks that were in the tour may remain uncovered during subsequent iterations. If the prices on these tasks are not reduced, then a driver may ignore these tasks because of their high price. The strategy that we have adopted is to reduce the price by a factor β every n iterations as long as these tasks remain uncovered. At some point, the price may decrease sufficiently to become attractive to some driver.

Step 7. Termination check

```
If $\mathcal{C} \neq \emptyset$ then go to Step 3
else adjust the value of $\epsilon$
    Set $\epsilon = \epsilon/\psi$.
    If $\epsilon > \epsilon_{\min}$ then
        for every unassigned driver $r$, set $\mathcal{C} = \mathcal{C} \cup r$
        go to Step 3.
```

Explanation of Step 7. The epsilon reduction strategy is similar to the approach adopted by Bertsekas (1988). We discuss this in greater detail in the next section.

Step 8. Uncovered task dual price adjustment
For all uncovered task ℓ set $p_\ell = 0$
For all unassigned initial driver r, set $\mathcal{C} = \mathcal{C} \cup r$
Repeat Steps 3 to 7 and then terminate.

Explanation of Step 8. Because it is possible that, after several iterations uncovered tasks with positive prices will remain, we reduce all the uncovered task prices to zero and perform one more iteration.

4. NUMERICAL EXPERIMENTS

THIS SECTION PRESENTS a comparison of the two algorithms RAPID-SL and RAPID-ML. The algorithms were tested on data sets with various properties. We use several criteria as measures for performance.

We also describe some of the strengths and weaknesses of both types of algorithms and discuss their adaptability to solving real-time problems. The algorithms were tested on both randomly generated

TABLE I

Medium Tour Length

	Excess, $n = 50$			Constrained, $n = 10$		
			$m = 30$			
Mixed	SL	ML	DWCG	SL	ML	DWCG
Profit	31477	31246	31703	28190	31480	32177
% Cov.	100	100	100	88	100	100
Idle.	30	30	30	0	0	0
% Empty	75	76	73	73	75	68
ATL (MTL)	1.3 (3)	1.5 (3)	1.5 (4)	2.6 (4)	3 (6)	3 (7)
CPU Time	1	6	47	1	5	24
			$m = 50$			
Mixed	SL	ML	DWCG	SL	ML	DWCG
Profit	42055	42727	43496	35230	39760	43824
% Cov.	97	100	100	81	91	100
Idle.	20	21	21	0	0	0
% Empty	74	75	71	73	72	68
ATL (MTL)	1.5 (2)	1.7 (3)	2.2 (4)	4.1 (6)	4.6 (6)	5 (7)
CPU Time	3	10	23	2	3	13
			$m = 30$			
Wide	SL	ML	DWCG	SL	ML	DWCG
Profit	25994	25526	26002	24053	26484	26981
% Cov.	100	100	100	90	100	100
Idle.	22	23	24	0	0	0
% Empty	65	69	64	52	58	48
ATL (MTL)	1 (3)	1.1 (3)	1.2 (3)	2.7 (4)	3 (4)	3 (4)
CPU Time	1	5	49	1	5	86

and real-world problems obtained from a truck-load motor carrier. All tests were done on an Indy Silicon Graphics Workstation (250 MHz IP22 Processor).

4.1 Problem Sets

The characteristics of the randomly generated and real-world problems are described below.

Real World Problems

The real-world problems used in this comparison are an example of the drayage problem and have a very specific structure. In drayage problems, the tasks typically consist of loads that have to be transported either to or from a rail terminal. In this case, a typical driver tour will start from a terminal where he will collect a load and take it to its destination. He will then move empty to pick up a second load and bring that load back to the terminal. A driver will typically perform several of these moves in a day.

Randomly Generated

Although the real-world data sets are interesting from a practical standpoint, the randomly generated data sets allow us more freedom to construct examples that highlight the properties of each algorithm.

To assess both algorithms in multiple situations, we generated a number of problems with three different time-window constraints, wide, tight and mixed; and also with different tour lengths. The wide time windows stretched the length of the entire planning horizon, which, in this case, was one day. The mixed category contained approximately 25% perfectly tight time windows, 25% wide and 50% were uniformly distributed between the two. The task origin and destinations were uniformly distributed over a given set of locations that were based around a central hub. The algorithms were tested on two different tour lengths. The first set exhibits an average tour length between 1 to 5 tasks and the results for these problems are given in Table I. The second set of results, given in Table II, are for problems where the average tour length ranges from 2 to 9 tasks.

4.2 Implementation Issues

Several issues arose in implementation of both RAPID-SL and RAPID-ML. The main ones were as follows.

4.2.1 Arc Generation

One issue common to both algorithms is the generation of arcs from drivers to tasks. The size of the problems we are dealing with prohibit the calculation of every arc in the network so we have used a space-filling procedure (BARTHOLDI and PLATZMAN, 1988) to only generate arcs between drivers and near tasks. (In practice, the origin of a task should be sufficiently close enough to a driver to justify a deadhead movement.)

TABLE II

Long Tour Length

	Excess, $n = 50$			Constrained, $n = 10$		
			$m = 60$			
Wide	SL	ML	DWCG	SL	ML	DWCG
Profit	51047	52358	53324	40777	43047	54173
% Cov.	96	100	100	75	80	100
Idle.	19	21	21	0	0	0
% Empty	59	62	55	46	52	45
ATL (MTL)	1.8 (4)	2 (5)	2 (6)	4.5 (5)	4.8 (7)	6 (7)
CPU Time	5	19	1918	3	3	954
			$m = 100$			
Mixed	SL	ML	DWCG	SL	ML	DWCG
Profit	75116	86258	DNR	49127	52690	80119
% Cov.	86	100		56	59	90
Idle.	42	43		0	0	0
% Empty	72	74		69	61	65
ATL (MTL)	1.8 (3)	2.3 (4)		5.6 (7)	5.9 (8)	9 (11)
CPU Time	19	59		8	7	1009
			$m = 120$			
Wide	SL	ML	DWCG	SL	ML	DWCG
Profit	84367	89657	DNR	44463	45032	DNR
% Cov.	79	88		40	41	
Idle.	13	11		0	0	
% Empty	55	72		48	57	
ATL (MTL)	2.1 (3)	2.7 (5)		4.8 (5)	4.9 (7)	
CPU Time	24	51		15	7	

DNR: CPU times exceeded 7 hours.

4.2.2 Stability of RAPID-SL

The subtour logic in RAPID-SL does not eliminate subtours at each iteration. This results in some degree of instability in the solution, and, to prevent termination problems, we only allow RAPID-SL to run for a maximum number of iterations and then terminate the algorithm. The solution we take is the best one up to and including that iteration. This number was set to 20. The assignment problem was solved using a network simplex code.

Although subtours can occur in any problem class, they are more likely to occur as the tour length increases; hence one would expect the solution quality for RAPID-SL in these types of problems to be significantly lower. This hypothesis is tested in the next section, where we compare RAPID-SL and RAPID-ML on problems with varying tour lengths. In real-time implementation of RAPID-SL, the problem is solved on a rolling horizon fashion and only the assignments of the initial drivers will be implemented. Thus, a task that has been covered will not be covered again, and subtours do not exist in practice (see subsection 4.4).

4.2.3 Control Parameters for RAPID-ML

The main parameters in RAPID-ML to be set are β, N, and ϵ. The first two parameters control the periodic price reduction of each uncovered task, defined in Step 8 of the algorithm. After performing numerous experiments, we found that the values that, on average, gave the highest objective function values were $\beta = 0.6$ and $N = 25$. The parameter ϵ is discussed next.

4.2.4 Epsilon Strategies

In setting a value for ϵ, we followed the approach suggested by Bertsekas (1988), that is, we solved the problem with a relatively high ϵ (around 100) and then iteratively reduced it until it fell below a specified minimum, $\epsilon_{\min}$. In practice, we set ψ (the amount we divide ϵ by at each iteration) to 4 and $\epsilon_{\min}$ (the stopping point) to 5. Different ϵ strategies can affect solution quality as well as CPU times. We tested various ϵ strategies on several assignment problems and the results are summarized in Table III. The solution is given as a percentage of the optimum solution.

In strategy 1, ϵ^0 was set to 100, and we terminated the algorithm at Step 8. Strategy 2 was the same as

TABLE III

ϵ Strategies

Strategy	Solution (%)	CPU (Average)
1	78	3
2	99	150
3	94	5

TABLE IV

Short-Haul Truckload Problems

	Excess			Constrained		
		$n = 60, \quad m = 91$			$n = 10, \quad m = 91$	
Real	SL	ML	DWCG	SL	ML	DWCG
Profit	37895	37501	37900	18896	21380	22414
% Cov.	59	70	59	19	20	20
Idle.	14	5	14	0	0	0
% Empty	50	50	49	52	44	44
ATL (MTL)	1.2 (2)	1.2 (3)	1.2 (3)	1.7 (2)	1.8 (2)	1.8 (2)
CPU Time	4	28	121	2	4	31
		$n = 46, \quad m = 65$			$n = 23, \quad m = 65$	
Real	SL	ML	DWCG	SL	ML	DWCG
Profit	3993	4935	5290	3032	4139	5013
% Cov.	76	82	54	45	75	79
Idle.	15	9	19	0	0	0
% Empty	54	54	53	48	50	48
ATL (MTL)	1.6 (2)	1.4 (3)	1.4 (2)	1.3 (2)	2.1 (3)	2.2 (3)
CPU Time	4	38	56	4	26	130
		$n = 36, \quad m = 27$			$n = 10, \quad m = 27$	
Real	SL	ML	DWCG	SL	ML	DWCG
Profit	1132	1082	1132	516	1050	1067
% Cov.	85	85	85	55	59	59
Idle.	12	12	12	0	0	0
% Empty	69	69	69	56	56	56
MTL	1 (1)	1 (2)	1.1 (2)	1.5 (2)	1.6 (3)	1.6 (3)
CPU Time	<1	2	4	<1	1	2
		$n = 52, \quad m = 48$			$n = 10, \quad m = 48$	
Real	SL	ML	DWCG	SL	ML	DWCG
Profit	3885	3826	3887	3099	3114	3182
% Cov.	64	66	64	39	41	39
Idle.	24	22	24	0	0	0
% Empty	78	77	78	54	55	54
MTL	1.2 (2)	1.1 (2)	1.2 (2)	1.9 (2)	2 (2)	1.9 (2)
CPU Time	<1	9	9	<1	4	6

strategy 1 but ϵ^0 was set to 1. Strategy 3 is the method described by Bertsekas (1988) and reviewed above.

As expected, strategy 1 terminated quicker but gave the worst solution quality, in accordance with the epsilon optimality properties of the auction algorithm. The second gave a better solution quality but the CPU requirements were too large for the class of problems that we are aiming to solve. The third option gave a good balance between the two.

4.3 Discussion of Results

The abbreviations used in Tables I, II, and IV are as follows.

- n = number of drivers
- m = number of tasks
- Profit = total contribution earned
- % Cov = percentage of task covered
- % Empty = percentage of miles traveled empty by the drivers
- Idle = number of idle drivers
- ATL = average tour length (number of tasks covered/number of drivers used)
- MTL = maximum tour length
- CPU = CPU time in seconds

Tables I, II, and IV compare the performance of RAPID-SL (denoted as SL) and RAPID-ML (denoted as ML) to a Dantzig–Wolfe column generation procedure (denoted as DWCG) that can produce an optimal solution on randomly generated and real-world problem sets. Each of the task sets were tested on two fleet sizes, excess and constrained. In the excess case (left column) there were idle drivers who did not cover any tasks, whereas in the constrained case (right column) all drivers covered at least one task.

In general, RAPID-SL performs much better under excess fleet capacity, attaining solutions that, on average, are within 5% of the optimal solution. However, in the constrained case, the shortage of drivers forces both algorithms to cover the tasks with fewer drivers, resulting in an increased tour length. As the tour length increases, so does the occurrence of subtours, resulting in a significantly lower solution quality for the constrained case, and one can see from Table II that RAPID-SL covers significantly fewer tasks than both RAPID-ML and DWCG.

From Tables I and II, we can see that RAPID-ML

TABLE V
Objective Function Gap

	Excess		Constrained	
Problem	SL	ML	SL	ML
30-Medium	99.3	98.6	87.6	97.8
30-Medium	99.9	98.2	88.9	97.9
50-Medium	96.7	98.2	80.4	90.7
50-Long	95.7	98.2	75.3	79.5
100-Long	DNR	DNR	61.3	65.6
100-Long	DNR	DNR	DNR	DNR
Real world-91	99.9	98.9	84.3	95.4
Real world-65	75.4	93.3	60.5	82.6
Real world-27	100.0	95.6	48.4	98.4
Real world-48	99.9	98.4	97.4	97.9
Average	95.8	97.4	76.0	89.5

and RAPID-SL are approximately equivalent on unconstrained data sets. However, in the constrained case, RAPID-ML consistently outperforms RAPID-SL. RAPID-ML is clearly better at using the available duty time, covering more tasks than RAPID-SL in every data set.

We now turn our attention to the real-world problems given in Table IV. In the unconstrained examples, the algorithms have approximately the same objective functions, but, in the constrained cases, RAPID-SL is not as efficient at using available duty time. RAPID-ML consistently outperforms RAPID-SL in both objective value and coverage, but at the cost of a significant increase in CPU times in some cases. This is mainly due to the initial label-building strategy that generates increasingly more labels as the possible tour length increases.

The performance of RAPID-SL and RAPID-ML versus the optimal solution is summarized in Table V. Here, we clearly see the highly competitive performance of RAPID-SL in the cases of more loosely constrained data sets. By contrast, RAPID-ML noticeably outperforms RAPID-SL in the case of tightly constrained data sets.

4.4 Real Time Implementation

In cases where the driver's tour length is short, typically only one or two tasks long, RAPID-SL will provide optimal or near-optimal solutions to these problems. We have successfully implemented a real-time version of RAPID-SL in a truckload application where the problem structure exhibits the above properties.

In our implementation, the problem is solved from scratch after the system is initialized at 7:00 A.M. Updates, which occur approximately every 20 seconds, are incorporated into the solution as the day progresses.

The initial network usually consists of approximately 1000 drivers and 1500 tasks at any one solution pass. The initial solution takes approximately one or two minutes to reach a solution, but updates require considerably less computational effort and take approximately 5 seconds.

The updating procedures for RAPID-SL are as follows. For each change to an existing driver, we recost all the arcs out of that driver and resolve using the modified network. If a new driver is added, then we simply add in the arcs from the new driver to the tasks and re-solve.

The addition of a new task is similar. For each new task, we add in the arcs from each potential and real driver to that task and from the potential driver at that task to all the other tasks in the network and re-solve. Changes to a task's attributes are incorporated using a similar strategy.

The addition of a driver simply means reinitializing the candidate list and repeating the algorithm from Step 3. The addition of a task requires the generation of a set of tour labels for that task, which can be accomplished using the existing sets of tour labels at other tasks. Once the tour labels are generated, we can then reinitialize the candidate list and repeat from Step 3 to check if this task, which we initially price at zero, is attractive to any driver.

5. CONCLUSIONS

THE OBJECTIVE OF THIS PAPER was to develop a class of heuristics to solve the Dynamic Assignment Problem in a real-time environment caused by the dynamic nature of the problem that we wish to solve. This means that the algorithms must be able to incorporate updates to the problem and produce good quality solutions in a matter of seconds. We also require that the procedures are flexible enough to simulate the complex environment of a real-world situation.

We have described two algorithms, RAPID-SL and RAPID-ML, which satisfy all of the above criteria. We have tested both algorithms on a variety of data sets, both real-world and randomly generated. We have also compared both solutions to an optimal column-generation procedure and can draw the following conclusions:

- RAPID-SL produces results that, on average, are within 4.1% of optimality for problems that are not tightly driver constrained.
- RAPID-SL's performance decreases as the number of tasks in a tour increases and the problem becomes more driver constrained.
- RAPID-ML outperforms RAPID-SL on problems with longer tours, which are prone to instability brought about by subtours.

- RAPID-ML is better at using available driver time, resulting in increased coverage over RAPID-SL on all data sets.
- Both algorithms provide high quality solutions when there are enough drivers to cover most of the tasks, with RAPID-ML outperforming RAPID-SL by an average of 1.6%. When drivers are tightly constrained, RAPID-ML outperforms RAPID-SL by an average of 13%.
- CPU times for RAPID-ML are greater than those for RAPID-SL but are still compatible with its use in a real-world environment.
- Both algorithms are easily adaptable to real-time updates, and provide dual information needed to produce alternative solutions.

ACKNOWLEDGMENTS

THIS RESEARCH WAS supported in part by grant AFOSR-F49620-93-1-0098 from the United States Air Force Office of Scientific Research. The work of the third author was partially supported by the Research Grant Council in Hong Kong (CERG: HKUST6047/97E).

REFERENCES

J. B. ATKINSON, "A Greedy Look-Ahead Heuristic for Combinatorial Optimization: An Application to Vehicle Scheduling with Time Windows," *J. Opns. Res.* **45,** 673–684 (1994).

M. BALL, B. GOLDEN, A. ASSAD, AND L. BODIN, "Planning for Truck Fleet Size in the Presence of a Common Carrier Option, *Decision Sci.* **14,** 103–120 (1981).

J. J. BARTHOLDI, III AND L. K. PLATZMAN, "Heuristics Based on Spacefilling Curves for Combinatorial Problems in Euclidean Space," *Management Sci.* **34,** 291–305 (1988).

D. P. BERTSEKAS, "The Auction Algorithm: A Distributed Relaxation Method for the Assignment Problem," *Ann. Opns. Res.* **14,** 105–123 (1988).

L. BODIN, B. GOLDEN, A. ASSAD, AND M. BALL, "Routing and Scheduling of Vehicles and Crews," *Comp. Opns. Res.* 63–211 (1983).

J. DESROSIERS, M. SOLOMON, AND F. SOUMIS, "Time Constrained Routing and Scheduling, in *Handbook in Operations Research and Management Science*, Volume on *Networks*, C. Monma, T. Magnanti, and M. Ball (eds), North Holland, Amsterdam, 1995.

J. DESROSIERS, F. SOUMIS, AND M. DESROCHERS, "Routing with Time Windows by Column Generation," *Networks* **14,** 545–565 (1984).

M. GENDREAU, A. HERTZ, AND G. LAPORTE, "A Tabu Search Heuristic for the Vehicle Routing Problem," *Management Sci.* **40,** 1276–1290 (1994).

F. GLOVER, "Tabu Search. Part I," *ORSA J. Comp.* **1,** 190–206 (1989).

F. GLOVER, "Tabu Search. Part II," *ORSA J. Comp.* **2,** 4–32 (1990).

B. GOLDEN AND A. ASSAD, "Vehicle Routing with Time Window Constraints," *Amer. J. Math. Management Sci.* **6,** 251–260 (1986).

R. E. MARSTEN AND F. SHEPARDSON, "Exact Solution of Crew Scheduling Problems Using the Set Partitioning Model: Recent Successful Applications," *Networks* **11,** 167–177 (1981).

H. PSARAFTIS, "Dynamic Vehicle Routing Problems," in *Vehicle Routing: Methods and Studies*, B. L. Golden and A. A. Assad (eds), 223–248, North Holland, Amsterdam, 1988.

M. W. P. SAVELSBERGH, "Local Search in Routing Problem with Time Windows," *Ann. Opns. Res.* **4,** 285–305 (1985).

M. SOLOMON, "Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints," *Opns. Res.* **35,** 254–265 (1987).

M. SOLOMON, E. BAKER, AND J. SCHAFFER, "Vehicle Routing and Scheduling Problems with Time Window Constraints: Efficient Implementation of Solution Improvement Procedures," in *Vehicle Routing: Methods and Studies*, B. L. Golden and A. A. Assad (eds), Vol. 16, 85–106, North Holland, Amsterdam, 1988.

(Received: September 1997; revisions received: September 1998; accepted: October 1998)