A MODEL AND STRATEGY FOR TRAIN PATHING WITH
CHOICE OF LINES, PLATFORMS, AND ROUTES

MALACHY CAREY
Faculty of Business and Management, University of Ulster, Northern Ireland,
BT37 0QB, United Kingdom

(Received 1 February 1993; in revised form 23 October 1993)

Abstract—Train pathing is concerned with assigning trains and train times for a set of rail links,
stations stops, etc., so as to meet a system of constraints on headways, trip times, dwell times, etc.
while minimizing delays or costs and meeting travel demands. In a previous paper we presented a
model, algorithms, and strategy for pathing trains of different speeds and stopping patterns for a
double track rail line dedicated to trains in one direction. Here we extend this to more general
more complex rail networks, with choice of lines, station platforms, etc, as is more typical of the
high density scheduled passenger railways in Britain and Europe. We apply the model to a small
network and find acceptable solution times. Applying addition search strategies from the previous
paper should reduce solution times by further orders of magnitude.

1. INTRODUCTION

A previous paper (Carey & Lockwood, 1994) set out a model and algorithms for train
pathing and planning. The basic model considered a line with multiple stations and
multiple trains of different speeds, and with all trains on the line travelling in the same
direction, as is usual on say British and European railways. Here, we extend the model to
a more general network, introducing explicit choices among multiple lines in each direc-
tion between stations (or other points), choice of platforms to use at stations, shared
platforms, various types of intersections, trains in one or both directions, etc. We find
that the model and algorithms from the previous paper extend very naturally to the
present more complex rail context.

Mathematical programming models and methods for train pathing have been devel-
oped by Szpigel (1972), Sauder and Westerman (1983), Petersen, Taylor, and Martland
(1988), and others. The last three papers appear to present the most general and most
powerful models for train pathing to date. The approach in Kraay, Harker, and Chen
is similar to that in Jovanovic and Harker, except that the former also considers the train
acceleration/deceleration patterns and the effects these have on fuel consumption and
hence on costs. We do not consider this here, partly because in Britain acceleration
patterns are already largely determined by other considerations.

The train pathing models just described focus on single track rail lines, with trains in
both directions and meets and passes taking place at sidings located at intervals along the
single line. We are concerned with more complex rail contexts, including general rail
networks. The focus of the above research on single track corridors is appropriate for rail
networks such as those in North America, which are mainly single track, long distance,
low density freight railways. In Britain and Europe rail systems are mainly multi-track,
shorter distance, higher density intercity passenger, or commuter networks. Headways
are often tight, with a train every few minutes, and there is usually a choice of two or
more unidirectional lines in each direction between stations. At stations there is often a
choice of two or more "platforms" at which a train can stop for passengers to board or
alight. Though such choices give managers some flexibility, they also introduce additional
types of potential conflicts between trains. The conflicts are widespread and interdepen-
dent, due to congestion, restrictions on line speeds and on dwell times, intersections,
desired departure or arrival times, etc. However, some of the restrictions that make it
harder for humans to generate good train plans make it easier for computer algorithms, as they reduce the search space.

In formulating the various features of train pathing, and algorithms for handling these, we first closely observed train pathers and planners at work, using traditional manual graphical methods for a wide variety of pathing and planning problems. This was done to ensure that the problem solved is consistent with and includes the problems perceived by train planners. It was also done because the underlying combinatorial problem is intractable unless we introduce some problem specific heuristics to solve it. The heuristics evolved over time by expert train planners are an ideal starting point for formalised algorithms. By formalizing and extending these heuristics in computer implemented models, as proposed here, we can improve on existing expertise, by considering many more options and looking further ahead, and doing this much faster. The basic solution algorithm used (see Section 4 and Carey & Lockwood, 1994) consists of steps similar to those in the sequential one-at-a-time train pathing methods associated with the traditional string-line diagrams (Fig. 1). The assumptions below about headways, dwell times, trip times, platforming, etc. are based on observation of train planning practice.

In Section 2, we set out briefly the basic single line train pathing model (Program P) from Carey and Lockwood (1994), for reference and to introduce notation. In Section 3, we extend the model of Section 2 to multiple lines, multiple platforms, choice or routes, etc. We refer to this as program $P^M$. This model requires introducing a substantial amount of new notation and new features of train pathing. The strategies that we adopt for solving program $P^M$ are set out in Section 4, and are similar to those introduced in Carey and Lockwood (1994) for the simpler model set out there and in Section 2 below. The first step in the strategy is to decompose program $P^M$ into a set of subproblems $P^M_y$, each of which consists of pathing one train while temporarily holding fixed the sequence order, but not the times, of all already pathed trains on all links. These subproblems are derived and discussed in Section 5. In Section 6, we present a numerical application, of program $P^M$ and the decomposition solution strategy, to a small rail network. The network is large enough to allow trains a choice of routes but is small enough to present the solution here.

2. A BASIC SINGLE LINE MODEL

We set out here, as briefly as possible the basic train planning and pathing model from Carey and Lockwood (1994). The notation for the model is as follows.

![Fig. 1. Train time-distance (pathing) diagram with conflicts.](image)
Subscripts

$s$ denotes the link joining station, stop, or timing point $s - 1$ to $s$, $s \in S$.
$v$ and $u$ denote trains, $\mu, v \in V$.
$v_1$ and $v_2$ denote the first and last train respectively on link $s$.

The assumption that the identities (indices) of the first and last trains on a link are known in advance involves no loss of generality, since we can always introduce “artificial” first and last trains (“ghost” trains) on each link $s$.

Variables

\begin{align*}
d_{sv} &= \text{departure time of train } v \text{ from station } s - 1, \text{ on link } s \\
a_{sv} &= \text{arrival time of train } v \text{ at station } s, \text{ at the end (exit) of link } s \\
x_{uv} &= \begin{cases} 1 & \text{if train } u \text{ immediately precedes train } v \text{ on link } s \\ 0 & \text{otherwise} \end{cases}
\end{align*}

Parameters. For each train $v$,

\begin{align*}
K^t_s &= \text{minimum time needed to traverse link } s \\
K^w_s &= \text{minimum dwell time required at station } s \text{ (for boarding, alighting, etc..)} \\
K^{d_{sv}} &= \text{minimum time interval (headway) required between the departure time of train } v \\
&\quad \text{and departure time of the preceding train } u \text{ on link } s \\
K^{a_{sv}} &= \text{minimum time interval (headway) required between the arrival time of train } v \\
&\quad \text{and arrival time of the preceding train } u \text{ on link } s.
\end{align*}

Cost functions. For each train $v$,

\begin{align*}
c^{c}_{sv}(a_{sv} - d_{sv}) &= \text{cost associated with trip time } (a_{sv} - d_{sv}) \text{ on link } s \\
c^{w}_{sv}(d_{sv} - a_{sv}) &= \text{cost associated with dwell time } (d_{sv} - a_{sv}) \text{ at station } s + 1 \\
c^{d}_{sv}(d_{sv}) &= \text{cost associated with departing at time } d_{sv} \text{ from station } s \\
c^{a}_{sv}(a_{sv}) &= \text{cost associated with arriving at time } a_{sv} \text{ at station } s
\end{align*}

Each of the above cost functions will often be simply proportional to time (a constant cost per minute) but some may be piecewise linear or could be more general convex functions.

Then the train planning and pathing model can be set out as follows.

\begin{align*}
P: \quad &\min \sum_{v \in V} \sum_{s \in S} [c^{c}_{sv}(a_{sv} - d_{sv}) + c^{w}_{sv}(d_{sv+1} - a_{sv}) + c^{d}_{sv}(d_{sv}) + c^{a}_{sv}(a_{sv})] \tag{2.1}
\end{align*}

subject to, for all $v \in V_s, s \in L$,

\begin{align*}
d_{sv} + K^t_s &\leq a_{sv} \tag{2.2} \\
a_{sv} + K^w_s &\leq d_{sv+1} \tag{2.3} \\
d_{sv} + K^{d_{sv}} &\leq a_{sv} + (1 - x_{uv})M, \text{ for all } u \in V_s \tag{2.4} \\
a_{sv} + K^{a_{sv}} &\leq a_{sv} + (1 - x_{uv})M, \text{ for all } u \in V_s \tag{2.5} \\
\sum_{u \in V_s} x_{uv} &= 1, v \neq v_s \tag{2.6} \\
\sum_{u \in V_s} x_{uv} &= 1, v \neq v_2 \tag{2.7} \\
x_{uv} &= 0 \text{ or } 1, \text{ for all } u, v, \text{ and } s \tag{2.8}
\end{align*}

where $M$ is an arbitrarily large number. Constraints (2.2) and (2.3) ensure that minimum trip times on links and minimum dwell times at stations are respected. Constraints (2.4)
and (2.5) ensure that minimum headways for departing and arriving trains, respectively are respected. Constraints (2.6)-(2.7) are needed to define the \(x_{uv}\) variables and ensure that each train can have one and only one immediate predecessor or successor.

3. A MODEL WITH MULTIPLE LINES AND PLATFORMS

In this section we set out a train pathing model to handle much more general rail networks and train pathing policies than can the single line model set out above. However, the easiest way to present and understand the model is perhaps to see it as starting from the above model.

In the above model, trip times on links between stations and dwell times at stations are treated separately, in (2.2) and (2.3). However, to simplify the formal presentation below we treat stations and the lines between stations as simply two different types of 'links' \(s\). Then the variables and constraints for stations and for lines between stations become formally the same, so that both can be discussed together as 'links'. For example, when link \(s\) represents a station then the dwell time on \(s\) is the time needed for boarding, alighting, etc., and when \(s\) represents a line between stations then the dwell time on \(s\) represents a link trip time.

3.1. Notation

As well as the notation below we also introduce some additional notation in the subsections 3.2.7 and 3.3, on "Infeasible paths" and "Costs," respectively.

Subscripts

\(u\) and \(v\) denote trains.

\(s\) denotes a link joining two nodes.

\(i\) denotes a node at which two or more links meet, converge, or diverge.

Index sets

\(L = \) the set of all links. \(L_v = \) set of links which train \(v\) is allowed to use. \(L'_v = \) set of links which train \(v\) is \(required\) to use \((L'_v \subseteq L_v)\). \(L''_v = \) set of links pointing into (before) node \(i\) and which can be used by train \(v\). \(L'_v = \) set of links pointing out of (after) node \(i\) and which can be used by train \(v\). \(L'\) = set of links on each of which only one train at a time is allowed.

\(N = \) the set of all nodes. \(N_v = \) the set of nodes which train \(v\) is allowed to use. \(N'_v = \) set of nodes which train \(v\) is \(required\) to use. \(N''_v = \) set of nodes which train \(v\) can optionally use \((N''_v = N'_v \cup N''_v)\). \(N^* = \) set of nodes each of which has only one link pointing into it, or only one link pointing out of it (or both).

\(V = \) set of all trains. \(V_v = \) set of trains which are allowed to use link \(s\).

\(I, = \) set of link pairs \((s,s') \in I,\) such that if train \(v\) uses link \(s\) then it can not use link \(s'\), and \(vice versa\) (see Section 3.2.7).

Most inadmissible link pairs do \textit{not} have to be explicitly listed in \(I,\), since if certain "key" pairs of links are infeasible others are automatically infeasible. Links \((s,s') \in I,\) usually share a common node: e.g., if \(s\) is the first link on one route, \(s'\) may be the first link on an alternative route.

Variables

\(x_{uv} = \begin{cases} 1 \text{ if train } u \text{ immediately precedes train } v \text{ on link } s \\ 0 \text{ otherwise} \end{cases}\)

\(x_{vs} = \begin{cases} 1 \text{ if train } v \text{ uses link } s \\ 0 \text{ otherwise} \end{cases}\)

The values of many of the \(x_{uv}\)'s, and some of the \(x_{vs}\)'s, can usually be prespecified as 1 or 0, so that these become constants rather than variables. (See also \(\delta_{uv}\) following).
\[ a_{vs} = \text{time at which train } v \text{ arrives on (enters) link } s \]
\[ d_{vs} = \text{time at which train } v \text{ departs from link } s \]

The time taken to “cross-over” a node, from the exit of one link to the beginning of the next link, can usually be treated as zero, if necessary by slightly adjusting the times for traversing the previous or subsequent links (see also Section 3.2.6 below). Because of this, we can set the departure times \( d_{vs} \) from links pointing into a node equal to the arrival times \( a_{vs} \) on links pointing out of the node. Thus, at each node \( i \), we can replace the \( d_{vs} \)’s and \( a_{vs} \)’s with a single time variable say \( t_{vi} \). This eliminates a large number of variables from the mathematical programs set out below (see Section 3.2.6). However, it is convenient to retain the present notation (\( d_{vs} \)’s and \( a_{vs} \)’s) for expositional purposes, and to substitute \( t_{vi} \)’s for these only at the computational stage. (To do this, simply replace the link subscript \( s \) everywhere with \( ij \), where \( i \) and \( j \) are the nodes at the beginning and end respectively of link \( s \). Then replace all link arrival times \( (a_{vs}, a_{us}, etc.) \) with \( (t_{vi}, t_{ui}, etc.) \) and replace all link departure times \( (d_{vs}, d_{us}, etc.) \) with \( (t_{vj}, t_{uj}, etc.) \).)

**Parameters/constants**

The constants \( K_{va}, K_{tv}, \) and \( K_{uvs} \) are as already defined in Section 2. However, link \( s \) can now denote a station platform, in which case \( K_{va} \) denotes the minimum dwell time required (for boarding, alighting, etc.) at the platform.

\[ K_{uvs}^{da} = \text{minimum headway required between the departure time } d_{us} \text{ of train } u \text{ and the arrival time } a_{vs} \text{ of the next train } v \text{ on link } s \text{ if link } s \text{ can hold only one train at a time.} \]

\[ M = \text{an arbitrarily large constant. (See end of Section 4.)} \]

\[ \delta_{uvs} = \begin{cases} 1 & \text{if train } u \text{ is permitted to (immediately) precede train } v \text{ on link } s \\ 0 & \text{otherwise.} \end{cases} \]

In many cases, equations with terms containing a zero \( \delta_{uvs} \) multiplier will reduce to simpler forms, or will be eliminated altogether. There are several reasons why \( \delta_{uvs} \) may be 0.

(i) \( \delta_{uvs} = 0 \) if either \( u \) or \( v \) are not allowed to use link \( s \), i.e., if \( u \notin V_s \) or \( v \notin V_s \). Also, because a train cannot precede itself \( \delta_{uvs} = 0 \) if \( u = v \).

(ii) The first and last train on a link \( s \) are \( V_s \) and \( V_s \), respectively (see Section 2). By definition, a train cannot precede train \( V_s \), nor follow train \( V_s \), hence \( \delta_{uvs} = 0 \) if \( v = V_s \) or \( u = V_s \).

(iii) As a matter of operating policy, certain trains \((u \text{ and } v)\) may not be allowed to follow each other on a link in which case \( \delta_{uvs} = 0 \).

(iv) If there are given upper and lower bounds on the arrival (or departure) times of trains \( u \) and \( v \) on link \( s \) and the lower bound on the arrival (departure) time of train \( u \) exceeds the upper bound on the arrival (departure) time of train \( v \), then \( v \) must precede \( u \), hence \( \delta_{uvs} = 0 \). This can be tightened by including headways in the calculations.

(v) We may know in advance that an optimal (or even a feasible) solution will not allow say a very slow train \( u \) to immediately precede a fast train \( v \) on a link. To perhaps save some computing time, we could exclude this possibility in advance by setting \( \delta_{uvs} = 0 \).

Items (i) and (ii) above can be summarized as: \( \delta_{uvs} = 0 \) if \((u \notin V_s \text{ or } v \notin V_s \text{ or } u = v \text{ or } u = V_s \text{ or } v = V_s \).

**3.2. Constraints**

3.2.1. **Defining \( x_{vs} \) variables, as sums of \( x_{uvs} \) variables.** If train \( v \) uses link \( s \) (i.e., \( x_{vs} = 1 \)) and \((v \neq V_s)\) then it must have exactly one immediate predecessor on link \( s \) (i.e., \( \sum_{u \in V_s} \delta_{uvs} x_{uvs} = 1 \)). Also, if train \( v \) does not use link \( s \) (i.e., \( x_{vs} = 0 \)) then it has no immediate predecessor on link \( s \) (i.e., \( \sum_{u \in V_s} \delta_{uvs} x_{uvs} = 0 \)). Thus in either case,
for all \( v \in V, v \neq v_s, s \in L \). These equations and the \( x_{vs} \) variables need not appear explicitly in the program, since the equations can be used to substitute for \( x_{vs} \) wherever it appears.

### 3.2.2. Train continuity or conservation equations.

If train \( v \) arrives at node \( i \) (i.e., on some links \( s \in L^v_i \)) then it must also depart from node \( i \) (i.e., on some link \( s \in L^v_i \)). Recall that if train \( v \) uses link \( s \) then \( x_{vs} = 1 \), otherwise it equals 0. Hence,

\[
\sum_{s \in L^v_i} x_{vs} = \sum_{s \in L^v_i} x_{vs} \quad \text{for all } v \in V, i \in N_i.
\]

These constraints do not apply at the origin or destination node for train \( v \). If the links to be used by train \( v \) entering and/or exiting node \( i \) are prespecified (by \( x_{vs} = 1 \)), then the left and/or right of (3.2) will be 1, and the equation is redundant if it reduces to 1 = 1.

### 3.2.3. Trip times on links and dwell times at platforms.

For train \( v \) the trip time on link \( s \) (or dwell time at platform \( s \)) is \( d_{vs} - a_{vs} \). To ensure that this can not be less than the prespecified minimum trip time \( K_{vs} \) needed by train \( v \) to traverse the link write

\[
a_{vs} + K'_{vs} \leq d_{vs} + (1 - x_{vs})M \quad \text{for all } v \in V, s \in L
\]

The \((1 - x_{vs})M\) term ensures that these constraints are enforced if and only if train \( v \) actually uses link \( s \) (i.e., if \( x_{vs} = 1 \)): if \( x_{vs} = 0 \) then the right hand side of (3.3) is \( \geq M \) so that (3.3) is not binding. (Note that an \( a_{vs} \) and \( d_{vs} \) will be generated by (3.3) even for links \( s \) which it turns out that train \( v \) does not use. However, this is not a problem, because these variables will also be generated for the link which \( v \) does use.)

If we wish to hold the trip time fixed at \( K_{vs} \) on link \( s \) then write (3.3) as a strict equality. Also, if train \( v \) is not scheduled/allowed to stop at a station then the dwell times for trains \( v \) on all links (platforms) at this station should be set to zero, i.e., \( a_{vs} = d_{vs} \) for all links \( s \) at the station.

### 3.2.4. Headways on links.

For reasons of signalling, safety, etc., a minimum time interval (called headway) is required between each train and its predecessor on each link. Also, because trains can travel at very different speeds (see link trip time constraints), it is necessary here to impose these headway constraints on trains on entering and exiting from each link.

#### Headway on entering (arriving on) a link.

To ensure at least the minimum headway \( K_{uv}^{a} \) between the arrival times (\( a_{us} \) and \( a_{vs} \)) of trains \( u \) and \( v \) on link \( s \), write

\[
a_{us} + K_{uv}^{a} \leq a_{vs} + (1 - x_{uv})M \quad \text{for all } u, v, s | \delta_{uv} = 1
\]

The \((1 - x_{uv})M\) term ensures that the constraint is enforced if and only if train \( u \) immediately precedes train \( v \) on link \( s \).

#### Headway on exiting (departing) from a link.

These are the same as the above headway constraints on entering the link, except that here all arrival times (\( a's \)) are replaced by departure times (\( d's \)). Thus

\[
d_{us} + K_{uv}^{d} \leq d_{vs} + (1 - x_{uv})M \quad \text{for all } u, v, s | \delta_{uv} = 1
\]

Combining the headways for entering and exiting from a link. If the link trip time constraints (3.3) for a pair of trains on link \( s \) are strict equalities then one of the above pair of headway constraints is redundant. To see which one, use (3.3) to substitute for \( a_{us} \) and \( a_{vs} \) in (3.4a). This gives (3.4b), but with \( K'_{uv} \) replaced by \((K_{uv}^{a} + K'_{uv} - K_{uv}^{d})\). If the latter constant is greater than the former then (3.4b) is redundant, if the reverse is true then (3.4a) is redundant, and if they are equal then either of (3.4a) or (3.4b) can be dropped. Note that above substitution of (3.3) in (3.4a) also yields \( M[(1 - x_{uv}) + (1 - x_{us}) - (1 - x_{vs})] \) on the r.h.s., but this takes the same value as \( M(1 - x_{uv}) \).
3.2.5. One (or more) train at a time on some links, e.g., at platforms, some short links and "block signalled" sections. Most platforms at stations allow only one train at a time. The next train v is not allowed to enter the platform at time \( a_{uv} \) until the previous train u has exited, at time \( d_{uv} \), and a minimum headway \( K_{uv}^{d} \) has elapsed. A similar restriction is required for so called "block signalled" sections of track: on such stretches of track a train is not allowed to enter until the preceding train has exited. These restrictions can be written as,

\[
d_{uv} + K_{uv}^{d} \leq a_{uv} + (1 - x_{uv})M \quad \text{for all } (s \in L^{p}, u, v | \delta_{uv} = 1) \tag{3.5}
\]

where \( L^{p} \) is the set of links on which only one train at a time is allowed. The \( (1 - x_{uv})M \) term ensures that these constraints are enforced if and only if train u immediately precedes v on link s.

In some cases, where there is a relatively long platform, or short trains, two or even more trains may be allowed at a platform at the same time. To represent this, we can simply treat the platform as a set of end to end shorter platforms and represent each of these as a separate link. A train which enters the first of these links has to pass through all of them but has a nonzero minimum dwell time \( K_{uv}^{m} \) for only one of the links.

3.2.6. Headways at nodes: junctions, intersections, station entrances/exits etc. The simplest form of junction of links consists of two links one of which is a continuation of the other, as in Fig. 2a. However, we would usually treat these as a single link. The next simplest form of junction consists of three links meeting at a point. This can occur as in Fig. 2b or c, and both of these occur frequently in practice. For example, b may represent two lines leading into a single platform, or a single line leading out from two platforms. Similarly for c. Crossovers from one line to another, as in Fig. 2d are also very common in practice but can be decomposed into b and c.

Each node in a–f has either a single link pointing into it or a single link pointing out of it: we will refer to such junctions as n-furcations, or reverse n-furcations. The headway constraint already imposed on the single link pointing into (or out of) the node is designed to ensure that headways will be adequate when crossing through the node. We assume that crossing through the node takes a negligible amount of time, or takes a fixed time which can be included in the trip time on the single link pointing into (or out of) the node. Hence, to model train movements through n-furcations (as in a–f), simply set the departure or exit times \( d_{vs} \) from link(s) pointing into the node equal to the arrival or entry times \( a_{vs} \) on link(s) pointing out of the node. Thus,

\[
a_{vs} = d_{vs} \quad \text{for all } s \in L^{p}_{v}, s' \in L^{q}_{v}, v \in V \tag{3.6}
\]

These constraints can be eliminated by replacing all \( a_{uv} \)'s and \( d_{uv} \)'s with \( t_{uv} \)'s, as noted just after introducing the \( a_{uv} \)'s and \( d_{uv} \)'s in Section 3.1.

Other more complex junctions and intersections can be decomposed into equivalent simple bifurcation or n-furcation components as in Fig. 2b, c, e, or f. (The exceptions are those which include links which can be traversed in either direction, but these can be handled as in Section 3.2.5 above.) For example, an intersection or crossover g can be replaced with h, by introducing an artificial link AB. For this link AB:

(a) Set the trip time on link AB equal to zero, or to the time, if any, needed to pass through node A in g.
(b) Impose minimum headway constraints on trains on link AB. Let these be the minimum headways needed between trains passing through node A in g.

Similarly, train movements through a node such as in Fig. 2i may be modelled by replacing i with Fig. 2j, and then proceeding as in e–f above. More generally, much more complex junctions, intersections, station entrances, exits, etc., can be decomposed into sets of components each of which is one of the simpler forms already discussed. For example, the station entrance shown in k contains three nodes of type e, two of type c,
Fig. 2. Reducing junctions, crossovers, station entrances, etc., to equivalent simpler forms (n-furcations) as in 3.2.6.

and three intersections of type g. The latter three can be decomposed as in g \rightarrow h. The
station entrance in Fig. 2l contains only simple nodes of the form b, c, e, and f.

3.2.7. Infeasible paths. At an intersection having more than one entry link and more
than one exit link, if a train enters on say link s then it may not be possible for it to exit
on say link s' . The track layout (points), signals, etc. may not allow it, or it may be
against operating policy. For example, in Fig. 2g and h it may not be possible for a train
to enter on say link 1 and exit on link 3. To ensure this, we simply write \( x_{13} + x_{1'} \leq 1 \).
More generally, we can prevent any illegal combination of link pairs \((s, s') \in I, v \in V\)
being used by train \( v \) by writing,

\[
(x_{sv} + x_{s'v}) \leq 1 \quad \text{for all } (s, s') \in I, v \in V
\]

(3.7a)
Alternatively, instead of the link combination \((s,s')\) being completely forbidden, there may simply be a high penalty or cost (say \(c_{ss'}\)) associated with using this link pair. This can be enforced by writing,

\[
(x_{ss} + x_{ss'}) \leq 1 + x_{ss'},
\]

for all \((s,s') \in I_s\),

and including \((c_{ss'}x_{ss'})\) in the cost function to be minimized. Assuming \(c_{ss'} > 0\), the \(x_{ss'}\)'s will automatically take 0 or 1 integer values (since \(x_{ss}\) and \(x_{ss'}\) are 0–1), and \(x_{ss'} = 1\) if and only if \(x_{ss} = x_{ss'}\) = 1. Thus the cost \(c_{ss'}\) will be incurred if and only if train \(v\) uses the link pair \((s,s')\).

3.2.8. Consistency of the 0–1 variables \(x_{uv}\). By definition, \(x_{uv} = 1\) if and only if train \(u\) is the immediate predecessor of \(v\) on link \(s\). Each train \(v\) \((v \neq v_s)\) must have one and only one immediate predecessor on link \(s\), if train \(v\) is required to use link \(s\). To ensure this, write

\[
\sum_{u \in \mathcal{V}_s} \delta_{uv} x_{uv} = 1 \quad \text{for all } v \in \mathcal{V}, v \neq v_s, s \in \mathcal{L}_v
\]

where \(L'_v\) is the set of links which train \(v\) is required to use. However, this does not prevent train \(u\) being the immediate predecessor of more than one other train \(v\). Hence to prevent this, write

\[
\sum_{v \in \mathcal{V}_s} \delta_{uv} x_{uv} = 1 \quad \text{for all } v \in \mathcal{V}, u \neq v_s, s \in \mathcal{L}_v
\]

If train \(v\) has to choose one out of a set of links \(L'_v\) pointing out of node \(i\), then in the above equations we sum the left hand side over this set, so that (3.8a) and (3.8b) become,

\[
\sum_{i \in \mathcal{N}_v} \left( \sum_{u \in \mathcal{V}_s} \delta_{uv} x_{uv} \right) = 1 \quad \text{for all } v \in \mathcal{V}, i \in \mathcal{N}_v
\]

and

\[
\sum_{i \in \mathcal{N}_v} \left( \sum_{v \in \mathcal{V}_s} \delta_{uv} x_{uv} \right) = 1 \quad \text{for all } u \in \mathcal{V}, i \in \mathcal{N}_u
\]

where \(\mathcal{N}_v\) is the set of nodes which train \(v\) is required to use. These constraints ensure that each train uses exactly one of the links pointing out of node \(i\) in \(\mathcal{N}_v\).

Furthermore, if there is a set of nodes such that train \(v\) has to use one and only one of these nodes then we should sum the left hand side of the above equations (3.8c) and (3.8d) over this set of nodes. More generally, let \(L'_v\) denote any set of links (the \(I'th\) set) such that train \(v\) is required to use one and only one of these links. In this case, replace \(\Sigma_{i \in \mathcal{L}_v} \delta_{uv} x_{uv}\) in the above two sets of equations with \(\Sigma_{i \in \mathcal{L}_v} \delta_{uv} x_{uv}\) and replace \(i \in \mathcal{N}_v\) on the right hand side with \(i \in J\) where \(J\) is the set of such sets of links.

Finally, since introducing additional constraints helps cut computing time in integer programs, it can be useful to introduce additional constraints of the form indicated in the previous paragraph, even if these are not strictly necessary. For example, if \(L'_v\) is the set of lines between two stations or the set of platforms at a station, and we know that train \(v\) has to use one of these lines (or platforms), then it may be worth introducing constraints of the above form for this set, even if these constraints are already logically implied by other sets of consistency constraints.

3.3. Costs

The timing points of interest to passengers are those which occur at stations, or other designated points for boarding, alighting or waiting. Let \(R\) denote the set of such points, which we will refer to as stations. Let \(L_r\) denote the set of links on which train \(v\) can dwell at station \(r \in R\). Define station arrival and departure times, \(a_v\) and \(d_v\), as
The dwell time of train $v$ at station $r$ is $(d_r - a_v)$ and the cost of this can be written as $c_v^r(d_r - a_v)$, so that the total cost over all stations is $\sum_{r \in \mathcal{R}} \sum_{v \in \mathcal{V}_r} c_v^r(d_r - a_v)$, where $V$ is the set of trains which are scheduled to stop at station $r$.

For train $v$ let $r_v^+$ denote the next station at which train $v$ is scheduled/required to stop after departing from station $r$. The trip time from $r$ to $r_v^+$ is $(a_{r_v^+} - d_r)$ and the cost of this can be written as $c_v^r(a_{r_v^+} - d_r)$ so that the total cost over all inter-station trips is $\sum_{r \in \mathcal{R}} \sum_{v \in \mathcal{V}_r} c_v^r(a_{r_v^+} - d_r)$.

There may also be penalties or costs associated with trains departing from or arriving at stations at times other than those which are desired for marketing or revenue reasons. These costs can be stated as $c_v^x$ and $c_v^x$, respectively.

There is often a preferred platform for each train $v$ at each station, and a preferred line for train $v$ to use between stations. There may also be a real or notional cost or penalty associated with using links (lines or platforms) other than the preferred links. These costs can be written as $c_v^x$, which reduces to $c_v^x$ if train $v$ uses link $s$ (i.e., if $x_v = 1$) and reduces to 0 if train $v$ does not use link $s$.

The total of all the above costs is then,

$$
\sum_{r \in \mathcal{R}} \sum_{v \in \mathcal{V}_r} [c_v^r(a_{r_v^+} - d_r) + c_v^r(d_r - a_v) + c_v^r(d_r) + c_v^r] + \sum_{s \in \mathcal{L}} \sum_{v \in \mathcal{V}_s} c_v^s x_v (3.9b)
$$

3.4. The optimization model $P^M$

The train planning and pathing problem can now be set out as: minimize (3.9b), subject to, (3.1)-(3.9a). Thus:

$$
P^M: \min \quad (3.9b),
\quad \{x_v, d_r, x_v, x_v\}
$$

subject to:

- Defining the $x_v$ variables, as sums of $x_{vst}$, for all $v \in \mathcal{V}$, $v \neq v_s$, $s \in \mathcal{L}$,

$$
x_v = \sum_{w \in \mathcal{V}_s} \delta_{vst} x_{vst} (3.1)
$$

- Train continuity or conservation equations, for all $v \in \mathcal{V}$, $i \in \mathcal{N}_v$ except for origin and destination nodes for train $v$,

$$
\sum_{s \in \mathcal{L}^+} x_{vst} = \sum_{s \in \mathcal{L}^-} x_{vst} (3.2)
$$

- Trip times on links, for all $s \in \mathcal{L}$, $v \in \mathcal{V}_s$,

$$
a_v + K_v^t \leq d_v + (1 - x_v)M (3.3)
$$

- Headways on links, for all $u$, $v$, and $s$ such that $\delta_{uv} = 1$,

$$
a_{uv} + K_{uv}^s \leq a_v + (1 - x_{uv})M (3.4a)$$

and

$$
a_{uv} + K_{uv}^s \leq d_v + (1 - x_{uv})M (3.4b)
$$

One train at a time on certain links (platforms, etc.), for all $s \in \mathcal{L}^e$, $u$ and $v$ such that $\delta_{uv} = 1$. 

a_v = a_v^t and d_v = d_v^t for all $s \in \mathcal{L}_v$ (3.9a)
Train pathing model

\[ d_{uv} + K^u_{uv} \leq a_{uv} + (1 - x_{uv})M \]  

(3.5)

Connections at nodes, for all \( s \in L^b_u \), \( s' \in L^a_v \), \( v \in V \),

\[ a_{uv} = d_{uv}. \]  

(3.6)

Infeasible paths, for all \((s, s') \in I, v \in V\),

\[ x_{uv} + x_{v's'} \leq 1 \]  

(3.7)

Consistency of the 0-1 variables \( x_{uv} \),

\[ \sum_{s \in L} \left( \sum_{u \in V} \delta_{uv} x_{uv} \right) = 1 \quad \text{for all } v \in V, i \in N_v \]  

(3.8c)

\[ \sum_{s \in L} \left( \sum_{u \in V} \delta_{uv} x_{uv} \right) = 1 \quad \text{for all } u \in V, i \in N_u \]  

(3.8d)

Integer restriction on \( x_{uv} \) and \( x_{uv} \) variables, for all \( v \in V, s \in L \),

\[ x_{uv} = 0 \text{ or } 1 \]

\[ x_{uv} = 0 \text{ or } 1 \text{ for all } u \in V, \delta_{uv} = 1. \]

Recall that the number of variables and constraints in the above formulation can be greatly reduced as follows.

(i) The 0–1 variables \( x_{uv} \) and the equality constraints (3.1) can be eliminated from the above program by using the latter equations to substitute for \( x_{uv} \) in (3.2) and (3.7), and in the cost function (3.9b).

(ii) The constraint (3.6), and the corresponding variables \( d_{uv} \) and \( a_{uv} \), can be eliminated from the program (see also Section 3.2.6 above) by substituting \( t_{uv} = (d_{uv}, \text{ for all } s \in L^b_u) = (a_{uv}, \text{ for all } s \in L^a_v) \).

4. SOLVING THE PATHING PROBLEM \( P^M \)

The 0–1 integer program \( P^M \) set out above can not be solved by simply applying any available general integer programming package to the program as it stands. For example, Carey and Lockwood (1994) found that a good IP package (GAMS/ZOOM, see Singhal et al (1989a), Brooke et al (1988)) took days of computing time on a SUN 4 to solve a simpler program than \( P^M \), having only 4 trains and 6 links.

To solve problem \( P^M \) we propose adopting strategies analogous to those adopted by “expert” train pathers using traditional manual graphical methods. Such strategies are set out in Carey and Lockwood (1994) and are summarized next. The basic strategy is to “path” the trains one at a time until all trains are pathed once, and if necessary iteratively repath trains until an acceptable solution is found. To “path” train \( v \) we solve the following program.

\( P^M_v = \) the program constructed from program \( P^M \) by holding fixed the sequence order (but not the times) of all already pathed trains, other than train \( v \), on all links.

Here “all links” includes all platforms and any other points (intersections, etc.) where headways have to be enforced between trains. We will refer to train \( v \) in program \( P^M_v \) as the “current” train. Thus program \( P^M_v \) finds the lowest cost time-space path through the network for the current train \( v \), while holding fixed the sequence order, but not times, of all other trains on all links. Note that holding the sequence order of all trains (except \( v \))
fixed on each link implies that the set of trains (other than \( v \)) on each link is also held fixed.

A major advantage of solving \( P^M \) by pathing trains one at a time (solving programs \( P^M_v \)) is as follows. The number of 0–1 variables in program \( P^M \) increases if we increase the problem size by introducing more trains. However, the number of 0–1 variables in program \( P^M_v \) need not increase if we introduce more and more later or earlier trains, if they do not intersect with train \( v \).

As well as decomposing program \( P^M \) into a sequence of subproblems \( P^M_v \), Carey and Lockwood (1994) also introduced a number of other complementary strategies to further reduce the computing time needed to solve \( P^M \). These strategies are based on analogies with how experienced or expert train planners path trains, using traditional manual graphical methods. They include the following. Though these strategies may be simple to state, embedding them in a mathematical programming package was not simple. For example, as noted below, treating links in the order in which they are traversed by the train necessitated months of effort, finding and switching to a different source code and substantially revising this.

1. **Searching paths in link traversal order.** In traditional graphical train pathing methods, trains are pathed by considering links in the same sequence order as that in which trains traverse the links. By analogy, when branching in the branch-and-bound search, we wish to introduce the 0–1 variables associated with choice of rail link in the same order as the links are traversed by the trains. The GAMS/ZOOM branch and bound algorithm did not allow introducing the 0–1 variables in this order. We therefore revised the Fortran source code for XMP (Singhal et al. 1989a, 1989b) to introduce this feature. We found that this very greatly reduced computing times. In the majority of 15 test problems (each having 10 trains and 10 links) the computing times reduced by over 30 times and sometimes by as much as 1,000 times.

2. **Time windows.** Experienced train pathers using manual methods are able to find acceptable solutions to large scale train pathing problems, in part by restricting their search to a time "window" for each train. We emulate this by placing reasonable bounds on train paths, and relaxing these if necessary. These bounds are then used to eliminate 0–1 variables and constraints.

3. **Fixing train paths.** In the traditional pathing process, planners usually hold fixed not only the sequence order but also, as far as possible, the times of all trains other than the one currently being pathed. In contrast, in program \( P^M_v \) we allow the times of all trains to vary, within bounds. However, to partially emulate this practice of train pathers, we can fix the timetable of any train whose time window has no overlap with that of the current train \( v \), and which is "sufficiently far" from the time window and expected path of the current train \( v \). This heuristic reduces the number of ordinary LP constraints and variables but does not in itself reduce the number of 0–1 variables. However, after applying this heuristic we can (re)apply the following bounds recalculation to eliminate further 0–1 variables.

4. **Dynamic recalculation of bounds.** After pathing train \( v \) (solving \( P^M_v \)) we use the solution to fix the sequence order of train \( v \) with respect to all other already pathed trains, and proceed to path the next train \( v + 1 \). Fixing the sequence order of train \( v \) still allows its times (and those of other trains) to vary but is likely to restrict the range within which these times can vary. The resulting new upper and lower bounds on all the arrival and departure times of all trains can be calculated by simple arithmetic (see Carey & Lockwood 1994). These new bounds can then be used to eliminate 0–1 integer variables as follows.

5. **Preprocessing bounds to eliminate 0–1 variables, and constraints.** Many of the 0–1 variables \( \{x_{uv}\} \) and the corresponding headway constraints, in program \( P^M_v \) as set out above are likely to be redundant due to the upper and lower bounds and other constraints on departure and arrival times. The 0–1 variable \( x_{uv} \) in program \( P^M_v \) is zero and can be eliminated if there is not sufficient "space" (time) between trains \( u \) and \( u + 1 \) to slot in train \( v \). Even if train \( u \) departs as early as possible and the next train
Train pathing model 345
departs as late as possible, there may not be sufficient room between them to allow the headways needed to slot in another train v. This is trivially easy to test by using a preprocessing program, but it usually greatly reduces the difficulty of program \( P_v^M \).

6. Preventing repassing. In practice, pairs of trains usually do not pass each other more than once on a trip, because the relative speeds of trains are usually similar on successive links. In program \( P_v^M \), forbidding repassing eliminates a large proportion of the 0-1 integer variables. Alternatively, placing a high penalty on repassing means many branches of the branch and bound tree need be searched only as a last resort.

7. Depth-first search. Following initial experiments we found it best (much faster) to use a diving or depth first search strategy. The particular form of this is analogous to the approach adopted, and found very successful, by train pathers in practice, because their manual methods are closely analogous to a diving strategy. Also, it reflects their concern with quickly finding “acceptable” feasible or viable solutions, rather than with global optimality.

8. Flighting trains. This refers to the common practice of dispatching trains in groups having similar speeds. This usually makes it possible to fit in more trains, with larger headways and fewer potential conflicts, than would dispatching a mixture of trains having very different speeds. Introducing this flighting policy in program \( P_v^M \) restricts the number of trains which train v is eligible to go just before or just after. This greatly reduces the number of 0-1 variables, hence makes the problem easier to solve.

Setting the M values. Another tactic we introduced, which is only indirectly related to strategies of train pathers, concerns setting the value of \( M \). In many constraints in programs \( P \) and \( P_v^M \), an \( M \) is introduced as a coefficient of 0-1 integer variables. Though logically any arbitrarily large positive constant will suffice for \( M \), we found [as is often the case (see Williams, 1990)] that computing times are substantially reduced by choosing a just sufficiently large value of \( M \). Consider a typical constraint \( a \leq d + (1 - x)M \), where \( a \) and \( d \) are continuous variables, with lower and upper bounds \((a, d)\) and \((\bar{a}, \bar{d})\) and \( x \) is a 0-1 variable. \( M \) should be just large enough to ensure that if \( x = 0 \) the constraint will be nonbinding (redundant) for all feasible \( a \) and \( d \). Setting \( x = 0 \) gives \( (a - d) \leq M \) and the maximum of \( (a - d) \) is \( (\bar{a} - \bar{d} + k) \), hence the \( M = (\bar{a} - \bar{d}) \) is the just sufficiently large value of \( M \).

To obtain an initial relaxed LP solution for each program \( P_v^M \), \( v = 1,2,3, \ldots \), any standard method can be used to find a starting basis. However, the number of simplex iterations can be reduced by constructing a starting basis for program \( P_v^M \) by taking the final LP basis from solving program \( P_v^M \), and making appropriate modifications.

5. REDUCING PROBLEM \( P^M \) TO PROBLEMS \( P_v^M \)

Though program \( P_v^M \) is informally defined above we set it out more fully and explicitly below. We do this because, in the algorithm proposed here for solving program \( P^M \), it is not necessary to ever explicitly set up program \( P^M \) as a computer program: instead we repeatedly set up and solve programs of the form \( P_v^M \). Also, program \( P_v^M \) is useful in its own right, because train planners frequently path or repath single trains without wishing to disturb the order of other trains. Furthermore, setting out program \( P_v^M \) highlights the reduction in the numbers of 0-1 variables, headway constraints, etc. as compared to program \( P^M \).

5.1. Notation

Indices and index sets. These are the same as in \( P^M \), with the following additions.

Let,

- \( U_s \) = the ordered set of trains \( u (u \neq v) \) which are already assigned to link \( s \) (by previous programs \( P_v^M \)).
- \( u^+ \) = the train immediately after train \( u \) in the already given ordering \( U_s \) of trains on link \( s \). (Strictly this should be \( u^+_s \) but using \( u^+ \) will not cause any confusion).
Variables. The $a_{uv}$ and $d_{uv}$ variables are the same as in program $P^M$. The train order on each link is held fixed in $P^M$ for all trains except $v$, hence the values of the train order variables $(x_{u,w}, u \neq v, w \neq v)$ are already fixed (at 0 or 1) and are no longer variables in $P^M$. This leaves only the variables $(x_{u,v}, u \neq v)$, and because the $v$ subscript in all of these variables denotes the same current train $v$ it can be omitted. This reduces the train order variables $x_{u,v}$ to,

$$w_{uv} = \begin{cases} 1 & \text{if train } u \text{ precedes train } v \text{ on link } s \text{ (i.e., train } v \text{ is slotted in between trains } u \text{ and } u^* \text{ on link } s) \\ 0 & \text{otherwise.} \end{cases}$$

As in program $P^M$, the $x_{u,v}$ variables are (merely) definitional and need not appear explicitly in the program. If train $v$ uses link $s$ (i.e., $x_{u,v} = 1$) then it must be preceded by some other train $u$ on link $s$ (i.e., $w_{uv} = 1$ for some $u \in U_i$), and conversely. (Recall that the first and last train on link $s$ are known and already included in $U_i$). Hence,

$$x_{u,v} = \left( \sum_{u' \in U_i} w_{u'u} \right) \quad \text{for all } s \in L_v \quad (4.0)$$

This can be used to substitute for $x_{u,v}$ wherever $x_{u,v}$ appears.

Parameters/constants. These are the same as in program $P^M$.

5.2. Constraints

5.2.1. Consistency of the $w_{uv}$ variables. The constraint sets (3.8) in program $P^M$ reduce as follows for program $P^M_v$. To ensure that the current train $v$ is pathed between one and only one pair of existing adjacent trains departing from node $i$, write

$$\sum_{s \in L_i} \sum_{u \in U_i} w_{uv} = 1 \quad \text{for all } i \in N \quad (4.1)$$

If using node $i$ is optional for train $v$ then the $=$ becomes $\leq$.

5.2.2. Continuity or conservation equations. These are the same as equations (3.2) in $P^M$, but are needed only for the current train $v$.

5.2.3. Trip times on links, and dwell times at platforms. For train $v$ the constraints are the same as (3.3) for program $P^M$, except that $x_{u,v}$ is replaced by the equivalent $(\sum_{u \in U_i} w_{u,v})$, hence

$$a_{uv} + K'_{uv} \leq d_{uv} + \left( 1 - \sum_{u \in U_v} w_{uv} \right) M \quad \text{for all } s \in L_v \quad (4.3a)$$

For trains $u (u \neq v)$, the link trip time constraints are the same as above except that the $+(\ldots)M$ term is omitted, thus

$$a_{uv} + K'_{uv} \leq d_{uv} \quad \text{for all } u \in U_v, s \in L \quad (4.3b)$$

5.2.4. Headways on links. Recall from program $P^M$ that there were two sets of headway constraints for each link, namely arrival time headways (3.4a) and departure time headways (3.4b). Consider these in turn.

Headway on entering a link. For the present program $P^M_v$, (3.4a) reduces to two sets of constraints, one set for the current train $v$ and one set for the remaining trains $u \neq v$. For train $v$ we have to ensure headway with respect to the train $u$ immediately preceding it, and the train $u^+$ immediately succeeding it, in the currently given train order $U_v$. To ensure the former,
Train pathing model 347

\[ a_{us} + K_{us}^w \leq a_{us} + (1 - w_{us})M \quad \text{for all } u \in U_v, s \in L_{us}, \delta_{us} = 1 \] (4.4a)

and to ensure the latter,

\[ a_{us} + K_{us}^w \leq a_{us} + (1 - w_{us})M \quad \text{for all } u \in U_v, s \in L_{us}, \delta_{us} = 1 \] (4.4b)

For trains other than the current train \( v \), the headway constraints are,

\[ a_{us} + K_{us}^w \leq a_{us} + (1 - w_{us})M \quad \text{for all } u \in U_v, u \neq v, s \in L \] (4.4c)

Headway on exiting from a link. Same as (4.4a)-(4.4c) above but with all \( a \)'s in the constraints replaced by \( d \)'s. We refer to these constraints as (4.4d)-(4.4f).

If the link trip constraints (4.3a) and (4.3b) are strict equalities, then they can be used to eliminate about half of the headway constraints (4.4) above, as explained at the end of Section 3.2.4.

5.2.5. One (or more) train at a time at platforms, etc. For the current train \( v \) these constraints are as (3.5) in \( P^M \), but with \( x_{us} \) replaced by \( w_{us} \), thus

\[ d_{us} + K_{us}^w \leq a_{us} + (1 - w_{us})M \quad \text{for all } u \in U_v, s \in L_{us}, \delta_{us} = 1 \] (4.5a)

\[ d_{us} + K_{us}^w \leq a_{us} + (1 - w_{us})M \quad \text{for all } u \in U_v, s \in L_{us}, \delta_{us} = 1 \] (4.5b)

The first constraint ensures headway between train \( v \) and the preceding train \( u \), and the second constraint ensures headway between \( v \) and the succeeding train \( u^+ \). For the remaining trains \( u \neq v \), these constraints are

\[ d_{us} + K_{us}^w \leq a_{us} + (1 - w_{us})M \quad \text{for all } u \in U_v, u \neq v, s \in L^p \] (4.5c)

5.2.6. Headways at nodes: junctions, etc. These headways are ensured in the same way as in program \( P^M \) for all trains including the current train \( v \). That is, headways at nodes are ensured by,

(a) suitably decomposing junctions, etc., if necessary, into component nodes and links so that there is only a single link pointing into (or alternatively, out of) each node, and

(b) equating arrival and departure time variables at these nodes, by applying constraints (3.6).

5.2.7. Infeasible paths. The formulation here is the same as for program \( P^M \), except that the constraints (3.7) are needed only for the current train \( v \). The remaining trains \( u \neq v \) have already been assigned to paths which are feasible.

5.3. Costs

The costs for program \( P^M_v \) are the same as for program \( P^M \), except that the values of the 0-1 variables \( \{x_{us} \text{ for all } u \neq v\} \) are held fixed. This reduces the expression \( \sum_{s \in L_v} \sum_{u \in U_v} c^v_{us} x_{us} \) to \( \sum_{s \in L_v} c^v_{vs} x_{vs} \) + a constant, namely \( \sum_{s \in L_v} \sum_{u \neq v} c^v_{us} x_{us} \).

5.4. The optimization model \( P^M_v \)

The train pathing model for a single train \( v \) can now be set out as, minimize (3.9b), subject to (4.1)-(4.7). Thus

\[ P^M_v: \{ \min_{\{u_{us}, \delta_{us}\}} (3.9b) \}, \text{subject to:} \]

\[ \sum_{u \in U_v} c^v_{us} x_{us} \text{ reduced to } c^v_{vs} x_{vs} = c^v_{vs} \sum_{u \in U_v} w_{us} \]
Consistency of the 0-1 variables, \( w_{ui} \)

\[
\sum_{s \in L^P_i} \sum_{u \in U_s} w_{ui} = 1 \quad \text{for all } i \in N
\] (4.1)

Train continuity or conservation equations. For all \( i \in N \), except the origin and destination nodes for \( v \),

\[
\sum_{s \in L^P_i} \left( \sum_{u \in U_s} w_{ui} \right) = \sum_{s \in L^P_i} \left( \sum_{u \in U_s} w_{ul} \right)
\] (4.2)

Trip times on links,

\[
d_{us} + K_{us} \leq a_{us} + (1 - \sum_{w \in U_s} w_{us})M \quad \text{for all } s \in L^P
\] (4.3a)

\[
d_{us} + K_{us} \leq a_{us} \quad \text{for all } u \in U_s, s \in L
\] (4.3b)

Headways on links, for all \( u \in U_s, u \neq \overline{v}_s \). Headways on entering links:

\[
a_{us} + K_{us} \leq a_{us} + (1 - w_{us})M \quad \text{for all } s \in L^P, \delta_{us} = 1
\] (4.4a)

\[
a_{us} + K_{us} \leq a_{us} + (1 - w_{us})M \quad \text{for all } s \in L^P, \delta_{us} = 1
\] (4.4b)

\[
a_{us} + K_{us} \leq a_{us} \quad \text{for all } u \neq \overline{v}_s, s \in L
\] (4.4c)

Also, repeat (4.4a)-(4.4c), with \( a \)'s replaced by \( d \)'s, to ensure headways on exiting from links.

One train at a time on certain links, platforms, etc. for all \( u \in U_s, u \neq \overline{v}_s \),

\[
d_{us} + K_{us} \leq a_{us} + (1 - w_{us})M \quad \text{for all } s \in L^P, \delta_{us} = 1
\] (4.5a)

\[
d_{us} + K_{us} \leq a_{us} + (1 - w_{us})M \quad \text{for all } s \in L^P, \delta_{us} = 1
\] (4.5b)

\[
d_{us} + K_{us} \leq a_{us} \quad \text{for all } u \neq \overline{v}_s, s \in L^P
\] (4.5c)

Headways at nodes. Same as in program \( P^M \), thus,

\[
a_{us} = d_{us} \quad \text{for all } s \in L^P_{ui}, s' \in L^P_{ui}, u \in V
\] (4.6)

Infeasible paths. Same as in program \( P^M \) but needed only for the current train \( v \).

Integer restriction on \( w_{us} \),

\[
w_{us} = 0 \text{ or } 1 \quad \text{for all } u \in U_s, s \in L
\]

We can slightly simplify the appearance of the above program by replacing \( \sum_{w \in U_s} w_{us} \) in constraints (4.1) and (4.2) with \( x_{us} \) and introducing definitional constraints \( x_{us} = \sum_{w \in U_s} w_{us} \).

6. NUMERICAL EXAMPLE

In this section, we set out a numerical example to illustrate the models and solution method set out above. The example is kept sufficiently simple to be conveniently set out here, while at the same time illustrating the main features of the model. It includes choice of lines and routes, junctions with multiple crossing lines, one and two-way trains, and nodes which are origins or destinations for some trains but intermediate stops for others.

The rail network used in the example is illustrated in Fig. 3a. It has 10 nodes and 28 links, 14 “northbound” and 14 “southbound”, the direction of train movement being indicated by an arrow, and an “N” or “S” by the link number. The slightly darker, shorter
Train pathing model

Fig. 3a. The network used in the numerical example.

Fig. 3b. Expanded node 2.

Fig. 3c. Expanded node 7.

Fig. 3d. Expanded node 5.
Fig. 3c. Expanded node 6.

Fig. 3f. Expanded node 9.

links (1 and 15, 5 and 20, etc) represent platforms (denoted “p”) for boarding and alighting at stations A to E. The remaining links represent rail lines joining the stations. These lines are marked “f” or “s”, indicating a fast or slow line, though the distinction is not adhered to in the present application.

To simplify the figure, the network is set out schematically rather than giving the exact topology of the tracks. However, the train movements permitted across some of the nodes is not obvious from this figure. We therefore expand the nodes as illustrated in Figs. 3b–f. In expanding each node we are careful to decompose it into an equivalent set of subnodes such that each junction is a simple bifurcation of n-furcation, as in Section 3.2.6. and Fig. 2. We do this to ensure that the resulting network can be handled by the model set out in Sections 3 and 4 above.

For example, node 2 has path crossover conflicts, such as path 16–15 crossing paths 1–2 and 1–3. These can be handled as in Fig. 2g–h of Section 3.2.6. Thus, node 2 is expanded to three subnodes (a, b, c) in Fig. 3b, by introducing artificial links l₁ and l₂. Nodes a and c have the same form as in Fig. 2e, that is, they have multiple entering links but only one exit link. Node b is of the same form as in Fig. 2f, that is, it has multiple exit links but only one entering link.

Similarly, node 7 has path conflicts such as path 7–14 crossing paths 28–26 and 28–27. This can be handled as in Fig. 3c, where nodes 7a and 7c have only one exit link each, and node 7b has only one entering link. To ensure that trains cannot enter link 14 (north) from link 28 (south) in Fig. 3a, do not allow these trains on the artificial link l₂.

Nodes 3 and 4 can be expanded in a very similar way to node 7. Nodes 5, 6, and 9 are expanded fairly simply as in Figs. 3d–f.

We consider 10 trains (1–10) on the network: trains (1, 3, 5, 7, 9) northbound and trains (2, 4, 6, 8, 10) southbound, with origins and destinations as in Table 1. The set of links L_v, which each train v is allowed to use consists of all links which lie between its origin and destination node (Table 1) and which point in the appropriate (north or south) direction as shown in Fig. 3.

The minimum trip times on links, minimum dwell times at platforms, and minimum permitted headways between trains are computed by rounding numbers randomly generated from normal distributions with means 10, 5, and 2, and SD’s 2, 2, and 0.5, respectively. The trip time are then multiplied by 1.25 for links (4, 10, 16, 23), multiplied by 3 for link 7, and by 1.5 for link 25. The time required to pass through a node is taken as 1 minute.
Train pathing model

Table 1. Origins and destination nodes for trains 1 to 10

<table>
<thead>
<tr>
<th>Train</th>
<th>Northbound</th>
<th>Southbound</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 3 5 7 9 2 4 6 8 10</td>
<td></td>
</tr>
<tr>
<td>Origin node</td>
<td>1 1 1 1 3 8 8 8 6 8</td>
<td></td>
</tr>
<tr>
<td>Destination node</td>
<td>8 8 8 10 8 1 1 1 3 9</td>
<td></td>
</tr>
</tbody>
</table>

The costs are defined as (a) the sum of the link trip times and station dwell times plus (b) a cost of one unit for each minute each train is later or earlier than a preferred or target arrival time at the destination. These preferred arrival times are taken as half way between the earliest and latest arrival times consistent with given preferred start times. The preferred start times for trains (1,3,5,7,9) are (2,7,12,7,12), and for trains (2,4,6,8,10) are again (2,7,12,7,12).

Using the above data we set up a program of the form $P$, to path the 10 trains on the network (Fig. 3) so as to minimize costs. We solved $P$, as indicated in Section 4, by pathing the 10 trains one at a time until no further improvement is found. Pathing train $v$ involves forming and solving a mixed integer 0-1 subproblem of the form $P_v$. To solve these we used GAMS-ZOOM (Singhal, Marsten, & Morin, 1989; Brooke et al. 1988) on a SUN 4 workstation. This package does not (currently) permit exploiting the specialized branching and pruning strategies for train pathing developed and found effective in Carey and Lockwood (1994) (see Section 4), and does not provide a source code to adapt. For example, it does not allow the user to specify the order or groupings of variables for fixed order branching. Hence, we could not introduce the 0-1 variables in the same order in which the corresponding rail links are traversed by the trains. Despite this, by using the remaining strategies (pathing trains one at a time, re-tightening bounds, etc,) we were able to solve an otherwise intractable problem in reasonable time.

The problem solving phase took 53.9 secs cpu time, for the simplex iterations: for example 0.14 secs to path the first train and 13.2 secs to path the last train. Of the 53.9 secs, 38.6 were for iterations in the branch and bound phase (2596 iters) and the rest for iterations in the initial (923 iters) and final (4248 iters) LP phase. We used the GAMS code to generate the LP subproblems $P_v$ to be solved, and this took relatively a very long time. It took about 6 minutes for GAMS to generate the matrix generators for the subproblems, and even longer to execute these to generate the 0–1 integer LP's to be solved. This was due to the then current version of GAMS not having certain features we needed for conditionals in the generation phase. However, this is not a problem, as we were using GAMS only for its convenience in initial problem exploration and development. We have since written a “C” code to generate similar size train pathing problems in a few seconds (see Carey & Lockwood, 1994). Such times are easily achieved as the subproblems have only up to a few hundred variables and constraints. We introduced the trains in numerical order, from 1 to 10, and found no further improvement in the solution after pathing all the trains once.

The optimal solution obtained is given in Table 2. The item in brackets after the path list for each train is the time the train takes to traverse the path. The solution is intuitively reasonable. For example, different trains chose different paths between the same pair of origin-destination nodes, so as to reduce delays which would otherwise occur and which would affect following trains or crossing trains.

Table 2. Solution paths for trains 1 to 10

<table>
<thead>
<tr>
<th>Train</th>
<th>Path (links used)</th>
<th>Train</th>
<th>Paths (links used)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-4-6-10-12-14</td>
<td>2</td>
<td>28-26-23-20-17-15</td>
</tr>
<tr>
<td></td>
<td>(65)</td>
<td></td>
<td>(70)</td>
</tr>
<tr>
<td></td>
<td>(66)</td>
<td></td>
<td>(63)</td>
</tr>
<tr>
<td>5</td>
<td>1-2-7-14</td>
<td>6</td>
<td>28-26-24-21-20-18-15</td>
</tr>
<tr>
<td></td>
<td>(63)</td>
<td></td>
<td>(63)</td>
</tr>
<tr>
<td>7</td>
<td>1-4-6 (22)</td>
<td>8</td>
<td>24-22-20</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(16)</td>
</tr>
<tr>
<td>9</td>
<td>5-8-11-12-14</td>
<td>10</td>
<td>28-26-24-23-19</td>
</tr>
<tr>
<td></td>
<td>(46)</td>
<td></td>
<td>(43)</td>
</tr>
</tbody>
</table>
7. CONCLUDING REMARKS

We set out a detailed mathematical programming model for train pathing and planning. This greatly extends the basic single line pathing model set out in Carey and Lockwood (1994). We allow trains a choice of lines, station platforms, and routes. To make it tractable to solve the mathematical program, we decompose it into a sequence of simpler mathematical programming subproblems. Each of these corresponds to pathing a single train while temporarily holding fixed the sequence order but not the timings of all other already pathed trains. To reduce the computing time needed to solve each of these subproblems we can use strategies and heuristics analogous to those used by train pathers using traditional manual methods. Such strategies are introduced in Carey and Lockwood (1994) and are summarized here in Section 4. We implemented the models on a SUN 4 computer. We obtained favourable computational results for a small network, even without using any of our search reduction strategies, other than pathing trains one at a time. Based on the experience in Carey and Lockwood (1994) we can further greatly reduce these computing times.

The models and algorithms of the present paper are extended to handle two-way track in Carey (1994). Also, in later research we plan to apply these models and algorithms to larger networks. In Britain, and in the rest of Europe, there are many train stations having a choice of multiple platforms for boarding and alighting. We may have to substantially restrict these choices if we are to obtain acceptable computing times with the present algorithms. We therefore plan to develop computationally more efficient ways of representing and handling such complex junctions and stations. One possibility is a two stage recursive process, iterating as needed between a main network model and a more detailed station model. This is analogous to the traditional decentralized approach traditionally adopted in practice, hence it is in keeping with our general approach of emulating existing expertise.

Acknowledgements—This research was supported by Science and Engineering Research Council Grants GR/H/50449 and GR/H/48033. It also had earlier support from a British Rail/Fellowship of Engineering Senior Research Fellowship which the author held at Oxford University. I thank British Rail, The Fellowship of Engineering, and the Science and Engineering Research Council for their support and cooperation. An earlier draft of this article was presented but not published at TRISTAN I: Triennial Symposium on Transportation Analysis held in Montreal, Canada, June 5-11, 1991. I would also like to thank the anonymous referees for their helpful comments.

I would especially like to thank David Lockwood for his valuable participation in this work and for undertaking computations in Section 6. Dave unfortunately has recently had to take retirement at a young age due to serious illness.

REFERENCES


Szpigel, B. (1972). Optimal train scheduling on a single track railway (pp. 343-352). In M. Ross (ed.), Operational Res. '72, North-Holland, Amsterdam.