# Exact Train Pathing

Viswanath Nagarajan* Abhiram G. Ranade†

**Abstract**

Suppose we are given a schedule of train movements over a rail network into which a new train is to be included. The origin and the destination are specified for the new train; it is required that a schedule (including the path) be determined for it so as to minimize the time taken without affecting the schedules for the old trains. In the standard formulations of this single train pathing problem, the time taken by the train to traverse any block (track segment guarded by a signal) in the network is deemed to be a fixed number independent of how the train arrived onto that block. In other words, the standard formulations of train pathing do not accurately model the acceleration/deceleration restrictions on trains.

In this paper we give an algorithm to solve the single train pathing problem while taking into account the maximum allowed acceleration and deceleration as well as explicitly modeling signals. For trains having 'large' maximum acceleration and deceleration, our algorithm runs in polynomial time. On the other hand, if the train to be pathed is capable of only very small acceleration so that it must take a long time to reach full speed, our algorithm takes exponential time. However, we prove that the pathing problem is NP-complete for small acceleration values, thus justifying the time required by our algorithm.

Our algorithm can be used as a subroutine in a heuristic for multiple train pathing. If all trains have large (but possibly different) accelerations this algorithm will run in polynomial time.

**Keywords:** train pathing, dynamics, signalling, train simulation, train scheduling.

## 1 Introduction

The Train pathing/scheduling problem [3] is an interesting and important optimization problem. Given a train network and a set of trains (specified by their origins and destinations and perhaps a partial or full description of their paths), the problem is to compute a schedule for the train movement so as to optimize different parameters while satisfying certain constraints. One possible parameter to optimize could be the average time taken by the trains to finish their journey, or the power consumption, and so on. At the core of the pathing problem, is a *sequencing problem*, the central question in it being: if several trains need to cross a block of track, in what order should they do so? The sequencing problem has to be solved to optimize the chosen parameter, and at the same time satisfying some natural constraints relating to train dynamics and signalling/safety. By train dynamics we mean the problem of controlling the speed of the trains given the capabilities of the locomotives. How the achievable acceleration of a train varies as a function of the load (and perhaps other features such as gradients on the track) is a very complex problem. The signalling/safety constraints are more obvious: later train movement must obey the signals guarding the different blocks of track as set by the preceding trains etc.

*Tepper School of Business, Carnegie Mellon University, Pittsburgh, USA. Email: viswa@cmu.edu. Supported in part by NSF grant CCF-0728841.

†Department of Computer Science and Engineering, Indian Institute of Technology, Mumbai, India. Email: ranade@cse.iitb.ac.in

There appear to be two streams of research on train pathing. The first focuses on sequencing aspect, while the second on the signalling and dynamics aspects. In the first stream, train dynamics is not explicitly considered. Instead, the input to the problem gives for each train the time it takes to cross each relevant block of the track. Typically, signalling aspects are also ignored: trains passing consecutively on a single block are required to be separated by a certain *ad hoc* headway, whereas in reality how closely one train can follow another would be a function of the signalling system. With these assumptions, the pathing problem is typically represented as a Mixed Integer Program (MIP). Since this is typically intractable to solve optimally, various heuristics are used. More directly, the pathing problem can also be thought of as a shop scheduling problem (also NP-complete) [13, 11] and heuristics related to such problems can also be used. Constraint Logic Programming has also been used to solve these problems [12] (this might internally use backtracking/branch-and-bound techniques as well as integer programming). Since the problem is very hard, there has been a lot of work on *single track scheduling*, where the network is a single long track with stations on it, where trains can pass each other [4, 12, 2]. The focus in all this work is on the sequencing considerations, and train dynamics and the signalling aspects are modeled only summarily. This is often adequate, especially if trains can accelerate/decelerate very quickly. If the time taken by a train to reach its maximum velocity (or alternatively come to a stop from its maximum velocity) is comparable to the time required by the train to traverse a block, it will be necessary to model the acceleration/deceleration times as well. This may become important in urban/suburban sections where signals might be closely spaced, or while modeling the movement of freight trains which typically take substantial time to reach their full speed.

The second stream of work relating to train pathing focuses more on the dynamics and signalling questions, and mostly avoids the sequencing problems. In fact, the goal here is to build Decision Support Systems that *assist* humans to design train schedules [10, 14]. The sequencing information (including path selection) is typically provided by the human expert, and the computer program manages the dynamics and signalling. Thus given the path and the locomotive characteristics etc., the computer program computes and displays the space-time trajectory that the train would follow. There are many commercial packages which provide such functionality. Computing the trajectories of multiple trains simultaneously has also been considered, e.g. in the work by Blendiger et al [1]. Given a timetable they show how a control theoretic approach can be used to generate precise schedules (i.e. precise timings of train acceleration and deceleration with fidelity to train dynamics and signalling) so as to minimize different objective functions. Moreira and Oliveira [10] tackle a different aspect of this problem: given a timetable, they use integer programming to detect and remove track occupancy conflicts. They do have a "run-time calculator" which appears to model acceleration and deceleration, however, this work does not appear to consider the problem of selecting paths. The paths are fixed and only the train timing is changed to remove conflicts. In general, the focus in this entire stream is not on sequencing or path selection.

The work by Kraay et al. [5] seems to be one of the early papers that explicitly models train dynamics. In this, the authors consider the problem of pacing a set of trains (i.e., computing velocity profiles) in order to minimize some measure of cost. Kraay et al. [5] consider sequencing multiple trains, but they do not model signals, and their algorithm does not appear to handle path selection. In more recent work, Lu et al. [6] consider the multiple train pathing problem incorporating aspects of sequencing, path selection, and train dynamics. Again, instead of modeling signals, they require a certain headway between trains. In this setting, they present a *heuristic* for minimizing travel times for the trains in the system.

Our work considers path selection under train dynamics constraints while explicitly modeling signals. The main problem we consider is as follows. Suppose we are given a schedule of train movements over a rail network into which a new train is to be included. The origin and the destination are specified for the new train, as well as restrictions on it such as the maximum allowed acceleration

& deceleration, and the maximum velocity. Further, the rail network is divided into *blocks* where each block is guarded by a signal (say capable of showing the standard 3 colours: red, yellow, green). It is required that a trajectory (path + schedule of movement along the path) be determined for the new train so as to minimize its journey time without affecting the schedules for the old trains. A heuristic solution to this problem could be to discretise time (and velocity) as in [9]. In this paper, we solve the single train pathing problem *exactly*, over continuous time and velocity.[1]

### 1.1 Main Results

We present an algorithm to solve the exact train pathing problem. The time taken by the algorithm depends upon the *values* of the maximum allowed acceleration and deceleration for the new train. This may seem surprising, but we show later that this must be the case unless P=NP.

We say that a train has *large* acceleration and deceleration if it can start from rest, accelerate to maximum speed, and decelerate to a halt all within a single block. In such a case, we show that our algorithm runs in time polynomial in the size of the network and the number of block occupancies (due to trains already in the schedule). Polynomial runtime is also established for some other conditions, as described later.

We also show that for very small values of acceleration and deceleration, the problem is NP-complete. In fact, the proof also shows that in general, exact train pathing is not approximable to any factor in polynomial time. We note that this result is not of direct practical significance; however it indicates that the either the algorithm or the time taken by it must depend upon values of the acceleration and deceleration.

### 1.2 Outline

We begin in Section 2 by discussing our model and outlining the algorithm. Section 3 defines the main data structure. Section 4 describes in detail the basic operation (procedure EXPAND) used by the algorithm, and Section 5 gives a schematic description of our algorithm TRAIN-PATH. In Section 6, we analyse the runtime of this algorithm on some special cases that are likely to arise in practice. Our basic model considers trains as point objects, and we show how to extend this model to incorporate train lengths in Section 7. In Section 8, we discuss some other extensions of the model to which our algorithm applies. Finally, in Section 9, we show that for general values of the problem parameters, it is NP-hard to approximate exact train pathing to any factor.

## 2 Model and Problem Definition

Our model of a rail network was developed in consultation with K. Madhusudan and B. Gopi Singh of the Indian Railway Institute of Signal Engineering and Telecommunications [7]. This model was also used in Rangaraj et al. [15].

In our model, a rail network is a directed graph $G = (V, A)$ with $V$ being the set of vertices and $A$ the set of edges. Edges represent track segments guarded by a signal, which are also referred to as *blocks*. The vertices represent *branching points* in the network, and there are no branching points within blocks. We assume that every vertex has either only one incoming block or only one outgoing block. For block $e = (u, v)$, vertex $u$ is the entry point, and vertex $v$ is the exit point. Every block has a signal at its entry point, that reflects reservations on blocks ahead of it. There is a resource constraint on each block, which requires that at any time, at most one train may occupy the block. We note that it is not necessary to explicitly place such resource constraints on vertices- this is enforced

[1] This is assuming that the algorithm can perform exact computation over reals.

by the resource constraints on blocks (due to the fact that each vertex has either a single incoming block or a single outgoing block). Our model of a rail network may appear simplistic, but we use this model only to make the description of our algorithm easier. As shown in Section 8, our model and algorithms can be extended to handle *junctions* and *bi-directed* blocks as well.

An example of a rail network in our basic model is shown in Figure 1. This example is used throughout the paper in order to illustrate concepts.

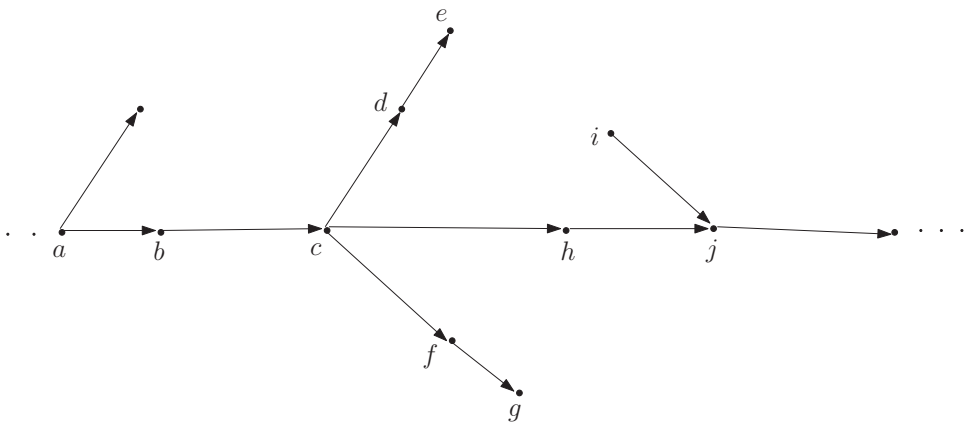


Figure 1: A portion of a rail network in our basic model

The two ends of a train are referred to as its *head* and *tail*; the end of the train facing the direction of movement is its head. A block is said to be occupied by a train if any part of the train is in the block. We say that a train enters block $(a, b)$ when its head enters the block (i.e., at $a$), and a train leaves block $(a, b)$ when its tail leaves the block (i.e., at $b$). Three parameters are specified for the train to be pathed, viz. its maximum velocity $V$, maximum acceleration $A$, and maximum deceleration $D$. These are referred to as the *train dynamics constraints*. Each train also has specific origin and destination positions[2], and a departure time.

The signal in a block indicates reservations made on subsequent blocks. Every signal is capable of displaying any of $C$ colours $0, 1, \cdots, C-1$. The most common signalling system has $C = 3$ with colours 0,1,2 representing red, yellow and green respectively. If the signal in block $e$ shows colour $c$ then all blocks located within $c$ blocks ahead of $e$ are unreserved by any train. If a train enters block $e$ when the signal colour is $c$, it can choose any value $c' \leq c$ to be its *signal aspect*. This means that the train is given exclusive access to the $c'$ blocks following block $e$, until any time that it leaves them. Formally, the constraints imposed by the signals are expressed in terms of signal rules and a driver rule, stated below.

**Definition 1** *The* **signal rules** *are as follows:*

1. *If the signal of block $e$ shows colour $c$, then $c$ blocks following the signal (including block $e$) on every possible path ahead of the signal are currently unreserved. Additionally, the $c+1$th block on at least one of the paths ahead of the signal is currently reserved, for $c < C-1$.*

2. *A train can enter block $e$ under signal aspect $c'$ only if the signal colour in $e$ is $c \geq c'$.*

3. *At any point of time, the blocks reserved by a train are as follows. All blocks occupied by the train are deemed reserved for the train. Additionally, if $e$ denotes the block in which the head is*

[2]See Definition 7 for a formal definition. In the first part of the paper, we will assume that train lengths are 0, and in this case a position is just a vertex in $G$.

*present and $c'$ the signal aspect under which the train entered $e$, then $c'-1$ blocks ahead of block $e$ on all branches are reserved for the train.*

**Definition 2** *The* **driver rule** *is as follows: If the head of a train enters block $e=(u,v)$ under signal aspect $c'$ (at vertex $u$), its velocity when the head reaches vertex $v$ must be such that its head can come to a stop within the shortest $c'-1$ block path out of $v$.* [3]

In other words, the driver rule ensures that the train can always come to a halt within its currently reserved blocks. The colours shown at the different signals are represented by a sequence of quintuples $(p,q,c,t_0,t_1)$, which indicates that the signal at the head of block $(p,q)$ has colour $c$ during the time interval $[t_0,t_1]$. We refer to these quintuples as **signal intervals**.

As an example of the signal and driver rules, consider the signal in block $(b,c)$ of Figure 1, and say it is capable of showing $C=4$ colours. If this signal shows colour 2, then the blocks $(b,c),(c,d),(c,f)$ & $(c,h)$ are currently unreserved, and at least one of the blocks $(d,e),(f,g)$ & $(h,j)$ is currently reserved. Suppose that the lengths of blocks $(c,d),(c,f)$ & $(c,h)$ are $0.25\ km, 1\ km$ & $0.75\ km$ respectively, and a train (with maximum deceleration $D=5000\ km/hr^2$) enters block $(b,c)$ under signal aspect 2. Then the maximum allowed velocity of the train when it reaches vertex $c$ is $50\ km/hr$, so that it can come to a halt within $0.25\ km$ from $c$.

For simplicity of exposition, in the first part of the paper (until Section 7), we assume that trains are *point objects*. Later in Section 7, we describe how our algorithm can be modified to handle train lengths. In Section 8, we mention some other extensions of this basic model to which our algorithm can be applied.

A **trajectory** of a train is a path $P$ starting and ending at specified vertices, the signal aspects under which it enters each block, and a plot of velocity along $P$, satisfying the train dynamics constraints and the driver rule. A set $\mathcal{T}$ of trajectories is said to be **realizable** if no signal rules are violated when the trajectories are simultaneously followed by the corresponding trains. In other words, when the trajectories in $\mathcal{T}$ are followed, no two trains reserve a common block of $G$ at the same time.

The *exact train pathing* problem can now be defined as follows:
**Input:** A directed graph $G$, a set $\mathcal{T}$ of realizable trajectories, and a new train specified by its origin & destination vertices ($s$ and $d$), departure time $t_0$, and velocity and acceleration limits $(V,A,D)$.
**Output:** A trajectory $\tau$ for the new train, departing $s$ from rest at time $t_0$ and ending at $d$ with zero velocity, such that $\mathcal{T}\bigcup\tau$ is realizable, and $\tau$ has the minimum travel time.

## 2.1 The non-interference condition

It may be tempting to think that it is enough to schedule the new train so that the signal aspect under which it enters a block is equal to the signal colour it sees while entering. However, this could upset the trajectories of already pathed trains in $\mathcal{T}$. It turns out that it suffices for the new train to *not interfere* with the trains already pathed in a sense we define shortly. This idea is at the base of our algorithm.

For a set of realizable trajectories $\mathcal{T}$, define the **signal record** $S(\mathcal{T})$ to be the set of all signal intervals that arise (in each block, over time) when the trajectories in $\mathcal{T}$ are realized. Note that $S(\mathcal{T})$ can be easily computed by simulating $\mathcal{T}$, and using the signal rules.

**Definition 3 Non-interference Condition:** *Suppose that $\tau$ is a trajectory of the new train from the origin $s$ to vertex $v$, on a path consisting of blocks $b_0,b_1,\cdots,b_k$, where block $b_0$ begins at $s$ and*

[3] Due to branchings in the network, there may be many $c'-1$ block paths following $e$. So the distance within which the train has to come to a stop depends on the path that it plans to take. For now, we assume that the train has to come to a stop within the shortest $c'-1$ block path out of $e$. We show how to remove this assumption in Section 8.

*block $b_k$ ends at $v$. For each block $b_i$, let $t_i$ denote the time the train enters $b_i$, and $t_{i+1}$ the time it leaves $b_i$. We say that $\tau$ does not interfere with $\mathcal{T}$ if, for every $0 \leq i \leq k$, the signal aspect under which the train enters $b_i$ is at most the minimum signal colour in block $b_i$ (as per signal record $S(\mathcal{T})$) during the time interval $[t_i, t_{i+1}]$.*

As an example, consider block $(b, c)$ in the network of Figure 1, where its signal shows colour 2 in the time interval $[1, 3]$ and colour 1 during $[3, 5]$. If a train enters block $(b, c)$ at time 2 under signal aspect 2 and leaves this block at time 4, then it is said to interfere with the existing trajectories.

**Lemma 1** *Given a set of realizable trajectories $\mathcal{T}$, and a trajectory $\tau$ of the new train, $\mathcal{T} \bigcup \{\tau\}$ is realizable if and only if $\tau$ does not interfere with $\mathcal{T}$.*

**Proof:** Suppose $\tau$ is a trajectory as in Definition 3, that does not interfere with $\mathcal{T}$. For each $i = 0, \cdots, k$, let $a_i$ denote the signal aspect under which the train enters block $b_i$. The reservations made by $\tau$ are as follows (see the signal rules, Definition 1): for each $i = 0, \cdots, k$, it reserves all the blocks that are within $a_i$ blocks from $b_i$ for the time interval $[t_i, t_{i+1}]$. Since $a_i$ is at most the minimum signal colour seen in $b_i$ during $[t_i, t_{i+1}]$, it is clear that there are no reservation conflicts in $\mathcal{T} \bigcup \{\tau\}$. In other words, $\mathcal{T} \bigcup \{\tau\}$ is realizable.

For the other direction, suppose $\tau$ is a trajectory for the new train that interferes with $\mathcal{T}$. Let $b_i$ be the block where this happens, and $c_i$ the minimum signal colour seen in $b_i$ during $[t_i, t_{i+1}]$. *i.e.* $\tau$ enters block $b_i$ under signal aspect $a_i > c_i$. This means that $\tau$ reserves all the blocks located up to $a_i$ blocks ahead of $b_i$ during the entire period $[t_i, t_{i+1}]$ (while it occupies $b_i$). But at some time in $[t_i, t_{i+1}]$, the signal in block $b_i$ shows colour $c_i < a_i \leq C$; *i.e.* some train in $\mathcal{T}$ reserves a block located $c_i + 1$ blocks from $b_i$. This conflicts with $\tau$ reserving all blocks up to $a_i \geq c_i + 1$. Thus $\mathcal{T} \bigcup \tau$ is not realizable. ■

Our algorithm uses this idea and finds the fastest trajectory for the new train from its source to destination that does not interfere with already pathed trains.

### 2.2 Deviation from trajectories

In the rest of this paper we will assume that trains do not deviate from the trajectories prescribed by the algorithm. Our guarantee of computing the fastest schedule $\tau$ is given under this assumption. In real life however, it is conceivable that trains may have to run slower than their prescribed speeds because of unforseen circumstances.[4] If this happens, we should be assured that neither collisions nor deadlocks will result. Appendix A provides this assurance.

### 2.3 Actual representation of signal intervals

The non-interference condition suggests a different representation of signal intervals, which is also helpful in describing the algorithm for exact train pathing. The signal intervals that we refer to are those in the signal record $S(\mathcal{T})$, resulting from the realization of trajectories in $\mathcal{T}$. Similar to the original representation, a signal interval in the new representation is a quintuple $(p, q, c, t, t')$ – however, the interpretation is different. Such a quintuple will now indicate that $c$ is the *minimum* colour seen in the signal at block $(p, q)$ in the time interval $[t, t']$. Further, we require that $[t, t']$ be a maximal time interval satisfying this condition.

Here is an example. Suppose on some block $(p, q)$, the signal is red at time 0, turns yellow at time 1, turns green at time 5, turns red at time 7, turns yellow at time 9, and turns green at time 10 and stays green subsequently. In the old representation we would have the intervals $(p, q, red, 0, 1)$,

[4] We assume there are no circumstances that force a train to run *faster* than its prescribed speed!

$(p, q, yellow, 1, 5)$, $(p, q, green, 5, 7)$, $(p, q, red, 7, 9)$, $(p, q, yellow, 9, 10)$ and $(p, q, green, 10, \infty)$. In the new representation we will have $(p, q, red, 0, \infty)$, $(p, q, yellow, 1, 7)$, $(p, q, yellow, 9, \infty)$, $(p, q, green, 5, 7)$, $(p, q, green, 10, \infty)$.

These new signal intervals can be generated from the old representation easily, which in turn can be obtained from the trajectories in $\mathcal{T}$. Furthermore, the number of intervals in the new representation will be no larger than in the old one. Also note that some optimization could be possible for the representation, e.g. red signal intervals can be dropped since they do not participate in the algorithm. In what follows, we will assume that the signals are given as per the new representation. We use $\mathcal{I}$ to denote the total number of signal intervals. Clearly $\mathcal{I}$ is polynomial in the input size.

### 2.4 Algorithm Overview

The idea of our algorithm is similar to the well known algorithm by Dijkstra for shortest paths. We systematically construct all possible trajectories leaving the origin $s$ of the new train at its given departure time (which can be assumed to be 0 without loss of generality). In Dijkstra's algorithm, paths are grown away from the source in shortest path first order; likewise our algorithm progressively computes for each vertex $u$ a set of pairs $(t, v)$ such that it is possible to reach vertex $u$ (without interference) at time $t$ with velocity $v$. Of course, such pairs form an unbounded set, and so we need a special data structure which we call an *arrival profile* using which the sets of pairs can be compactly represented.

The algorithm is initialized with a profile $\{(0, 0)\}$ (signifying that the train has velocity 0 at time 0) for the origin vertex of the new train. As the algorithm progresses, a process we call *profile expansion* is used to generate more and more arrival profiles, signifying that the algorithm has detected more valid $(t, v)$ pairs for various vertices. The algorithm terminates when it is clear that the arrival profile for the destination is found to contain a pair $(t, 0)$ for the smallest possible $t$, signifying that the algorithm has found the earliest possible time at which the train can arrive at the destination and be in a state of rest. How to manage the expansions so that their number remains small, while adhering to the signal and driver rules, are the main issues in designing our algorithm.

## 3 Arrival Profiles

We assume, without loss of generality, that the departure time of the new train $t_0 = 0$. So, the new train is at rest at the origin $s$, at time 0. Also, when we say a trajectory is non-interfering, it is assumed to be w.r.t. the given set $\mathcal{T}$ of trajectories. If the train passes a vertex $p$ with velocity $v$ at time $t$, we refer to the tuple $(t, v)$ as a time-velocity tuple at vertex $p$. The basic idea of our algorithm is to construct for each vertex $p$, a region in the time-velocity plane, which represents all "feasible" time-velocity tuples of the train at vertex $p$.

**Definition 4** *A set $R(p)$ of points in the $\mathbb{R}^2$ plane is said to be an* **arrival profile** *at vertex $p$ (or simply a profile at $p$) if it satisfies:*

$$R(p) \subseteq \left\{ (t, v) \in \mathbb{R}^2 \quad | \quad \begin{array}{l} \exists \text{ non-interfering trajectory } \tau \text{ departing } s \text{ from rest} \\ \text{at time 0, and reaching } p \text{ at time } t \text{ with velocity } v \end{array} \right\}$$

The *complete arrival profile* at vertex $p$ (denoted $R^*(p)$) is the profile that satisfies the above inclusion with equality. Note that $R^*(p)$ represents *all* possible time-velocity tuples at which the train can be at vertex $p$, while obeying the non-interference condition. Constructing these arrival profiles is the central question in the paper. However, it should be clear that if $R^*(p)$ is known for all vertices

$p$, we are done. The shortest time required to reach the destination $d$ is simply the smallest time at which the train can be at rest at $d$, i.e.

$$\min_{(t,0)\in R^*(d)} t$$

Our algorithm maintains an arrival profile $R(p) \subseteq R^*(p)$ at every vertex $p$. These profiles $R(p)$ represent estimates for the complete profile $R^*(p)$. However, these estimates $R(p)$ can have complicated shapes. So we represent them as a union of possibly overlapping atomic profiles.

**Definition 5** *A profile $R$ is* **atomic** *if it is of the form:*

$$R = \{(t, v) \mid V_R \geq v \geq v_R,\ E_R(v) \leq t \leq L_R(v)\}$$

*where the boundary curves $E_R(v)$ and $L_R(v)$ are non-increasing in $v$, and $v_R$ and $V_R$ are respectively the minimum and maximum velocities in $R$.*
*The atomic profile $R$ is represented by the tuple $(v_R, V_R, E_R(v), L_R(v))$.*

In the above definition, $E_R$ is called the *early curve* of $R$, and $L_R$ is the *late curve* of $R$. The boundary of an atomic arrival profile refers to its early and late curves. An example of an atomic arrival profile is shown in Figure 2. We will see in Section 4.2 that the early and late curves of all profiles generated in our algorithm are piecewise algebraic. Note that the "top left corner" of an atomic profile $R$ i.e. the point $(E_R(V_R), V_R)$ simultaneously minimizes time and maximizes velocity. Likewise the "bottom right corner" (which could be at infinity) simultaneously minimizes velocity and maximizes time. In the rest of the paper, all profiles that we deal with will be atomic.

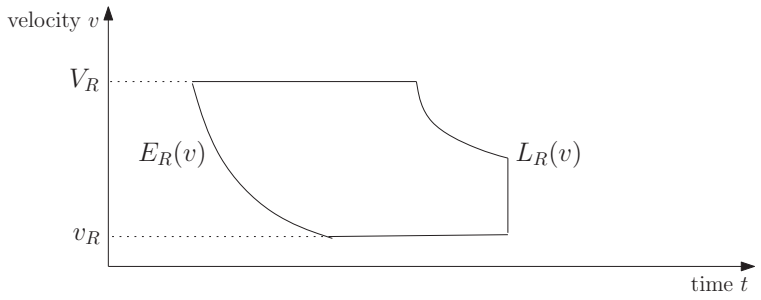


Figure 2: An atomic arrival profile $R$

# 4 Expansion

Suppose we are given an atomic arrival profile $R$ at vertex $p$. If $(p, q)$ is an edge, we simulate the movement of the train along the edge to construct the resulting arrival profile at $q$. This process is called *expansion*. It is accomplished in two steps. In the first step (Section 4.1), we only consider train dynamics and determine the set $S$ of time-velocity pairs at which the train can reach $q$ starting with some time-velocity pair in $R$. This computation does not consider the signal intervals on $(p, q)$ and hence the corresponding trajectories may interfere with the trajectories in $\mathcal{T}$ as the train passes through block $(p, q)$. We then show (Section 4.2) how the interfering trajectories can be removed from $S$ to complete the expansion process.

## 4.1 Expansion under train dynamics

The following definition is convenient.

**Definition 6** *Let $R$ be an atomic arrival profile at vertex $p$ and $(p,q)$ an edge. Then a dynamics limited expansion $S$ of $R$ at $q$ is the set*

$$S = \left\{ (t,v) \in \mathbb{R}^2 \quad | \quad \begin{array}{l} \exists (t_0, v_0) \in R\text{: the train can reach } q \text{ at time } t \text{ with} \\ \text{velocity } v\text{, starting from } p \text{ at time } t_0 \text{ and velocity } v_0. \end{array} \right\}$$

The following theorem says that $S$ can be characterized as an atomic arrival profile, and shows how it can be constructed. The statement is a bit daunting, however, it really only uses rules relating velocities, accelerations, time taken and distance covered from elementary kinematics. The basic idea in this Theorem is that the early (resp., late) curve of $S$ is derived from the early (resp., late) curve of $R$.

**Theorem 1** *Let $(p,q)$ be a block of length $d$. Let $R$ be an atomic profile at $p$, with parameters ($v_R$, $V_R$, $E_R(v)$, $L_R(v)$). Then the dynamics limited expansion $S$ of $R$ at $q$ is the set*

$$S = \{(t,v) \mid V_S \geq v \geq v_S,\ E_S(v) \leq t \leq L_S(v)\}$$

*where $V_S, v_S, E_S, L_S$ are defined as*

$$v_S = \sqrt{\max\{0, v_R^2 - 2Dd\}}$$

$$V_S = \min\{V, \sqrt{V_R^2 + 2Ad}\}$$

$$E_S(v) = \begin{cases} E_R(V_R) + t_{min}(V_R, v) & \text{for } V_S \geq v \geq v_{thresh} \\ E_R(\sqrt{v^2 + 2Dd}) + t_{min}(\sqrt{v^2 + 2Dd}, v) & \text{for } v_{thresh} > v \geq v_S \end{cases}$$

$$L_S(v) = \begin{cases} L_R(v_R) + t_{max}(v_R, v) & \text{for } v_S \leq v \leq v'_{thresh} \\ L_R(\sqrt{v^2 - 2Ad}) + t_{max}(\sqrt{v^2 - 2Ad}, v) & \text{for } v'_{thresh} < v \leq V_S \end{cases}$$

*where*

$$v_{thresh} = \sqrt{\max\{V_R^2 - 2Dd, 0\}}$$

$$t_{min}(u,v) = \begin{cases} \frac{1}{A}(\frac{V}{2} - u + \frac{u^2}{2V}) + \frac{1}{D}(\frac{V}{2} - v + \frac{v^2}{2V}) + \frac{d}{V} & \text{If } L(u,v) \leq d \\ \sqrt{\frac{1}{A} + \frac{1}{D}}\sqrt{\frac{u^2}{A} + \frac{v^2}{D} + 2d} - \frac{u}{A} - \frac{v}{D} & \text{If } L(u,v) > d \end{cases}$$

$$v'_{thresh} = \min\{V, \sqrt{v_R^2 + 2Ad}\}$$

$$t_{max}(u,v) = \begin{cases} \frac{u}{D} + \frac{v}{A} - \sqrt{\frac{1}{A} + \frac{1}{D}}\sqrt{\frac{u^2}{D} + \frac{v^2}{A} - 2d} & \text{If } L'(u,v) > d \\ \infty & \text{If } L'(u,v) \leq d \end{cases}$$

*with $L(u,v) = \frac{V^2 - u^2}{2A} + \frac{V^2 - v^2}{2D}$ and $L'(u,v) = \frac{u^2}{2D} + \frac{v^2}{2A}$.*

**Proof:** We first observe that the function $t_{min}(u,v)$ represents the minimum time taken by the train to cover block $(p,q)$, starting with a velocity $u$ from $p$, and ending with velocity $v$ at $q$ (if it is possible). To achieve minimum time the train must maintain maximum possible velocity between $p$ and $q$. This happens only if the train first accelerates to as high a velocity as possible and then decelerates to velocity $v$ just in time when it reaches $q$. If $L(u,v) \leq d = \text{length}(p,q)$, then between the accelerating portion and the decelerating portion we will have some period during which the train will run at its maximum allowed velocity $V$. If $L(u,v) > d$, then the train accelerates at $A$ to its highest velocity (smaller than $V$) and immediately decelerates at $D$ to reach vertex $q$ at velocity $v$. A similar argument shows that $t_{max}(u,v)$ represents the maximum time taken by the train to cover this block, with the desired initial and final velocities.

We now consider the two cases for the early curve $E_S$.

1. $E_S(v) = E_R(V_R) + t_{min}(V_R, v)$ for $V_S \geq v \geq v_{thresh}$. Starting with velocity $V_R$ at $p$, it *is* possible to reach $q$ with any velocity $v \in [v_{thresh}, V_S]$. It follows then that any point $(E_S(v), v)$ for $v \in [v_{thresh}, V_S]$ evolves from the top left corner of $R$, i.e. the point $(E_R(V_R), V_R)$ of $R$. This is because all other points in $R$ have either larger time, or lower velocity, or both. Thus starting from any of these points, even if we could arrive at $q$ with velocity $v$, we would arrive later, and hence not be on the early curve $E_S$. In order to reach the point $(E_S(v), v)$ on the early curve we further require that the travel from $p$ to $q$ must happen in minimum possible time. This is precisely $t_{min}(V_R, v)$.

2. $E_S(v) = E_R(\sqrt{v^2 + 2Dd}) + t_{min}(\sqrt{v^2 + 2Dd}, v)$, for $v_{thresh} > v \geq v_S$. For velocities $v < v_{thresh}$, it is not possible to reach $q$ at $v$, starting from $p$ at velocity $V_R$. Again, to minimize time, the train must maintain as high a velocity as possible. This implies starting from $p$ at a velocity of $\sqrt{v^2 + 2Dd}$ and using maximum deceleration $D$, to reach $q$ at velocity $v$. The earliest time that the train can leave $p$ with this initial velocity is given by $E_R(\sqrt{v^2 + 2Dd})$, and the minimum travel time is $t_{min}(\sqrt{v^2 + 2Dd}, v)$.

The two cases for the late curve are similar.

1. $L_S(v) = L_R(v_R) + t_{max}(v_R, v)$, for $v_S \leq v \leq v'_{thresh}$. Starting with velocity $v_R$ at $p$, it *is* possible to reach $q$ with any velocity $v \in [v_S, v'_{thresh}]$. It follows then that any point $(L_S(v), v)$ for $v \in [v_S, v'_{thresh}]$ evolves from the bottom right corner of $R$, i.e. the point $(L_R(v_R), v_R)$ of $R$.

2. $L_S(v) = L_R(\sqrt{v^2 - 2Ad}) + t_{max}(\sqrt{v^2 - 2Ad}, v)$, for $v'_{thresh} < v \leq V_S$. In order to maximize the time spent in the block, the train must maintain as small a velocity as possible. This implies starting from $p$ with velocity $\sqrt{v^2 - 2Ad}$ and accelerating at $A$ to reach $q$ at velocity $v$. The expression now follows.

For any velocity $v \in [v_S, V_S]$, we have shown that $E_S(v)$ and $L_S(v)$ are the earliest and latest times that the train can reach $q$ with velocity $v$, given that it leaves $p$ with a time-velocity tuple in $R$. Since $R$ is atomic, it is easy to see that the train can reach $q$ with velocity $v$ at *any* time $E_S(v) \leq t \leq L_S(v)$, while leaving $p$ with some time-velocity tuple in $R$.■

In Theorem 1, we note that if $R$ is an atomic profile with piecewise algebraic boundaries, then $S$ is also an atomic profile with piecewise algebraic boundaries. Furthermore, the number of pieces in the boundary curves increase by at most 2.

### 4.2 Expansion over a signal interval

The next step is to account for the signal intervals over the blocks. Each signal interval is treated separately. Let $(p, q)$ be a block, and $I = (p, q, c, t_1, t_2)$ a signal interval on it. Suppose $\Pi$ is an atomic profile at vertex $p$. Then $I(\Pi)$ is an arrival profile at $q$ obtained when the train begins at $p$ with velocity $v$ and time $t$ where $(t, v) \in \Pi$ and traverses edge $(p, q)$ during the interval $[t_1, t_2]$ without interfering with the trajectories in $\mathcal{T}$. Formally,

$$I(\Pi) = \left\{ (t, v) \in \mathbb{R}^2 \quad | \quad \begin{array}{l} \exists \quad (t_0, v_0) \in \Pi,\ t_1 \leq t_0 \leq t_2, \text{ and a } \textit{non-interfering} \text{ trajectory } \tau \text{ departing } s \\ \text{from rest at time 0, reaching } p \text{ at time } t_0 \text{ with velocity } v_0, \text{ and entering} \\ \text{block } (p, q) \text{ under signal aspect } c, \text{ to reach } q \text{ at time } t \text{ with velocity } v. \end{array} \right\}$$

The following procedure EXPAND shows how to compute $\Sigma = I(\Pi)$.

1. If the signal colour $c = 0$ (red), then we set $\Sigma \leftarrow \phi$ and terminate the construction.

2. Restrict $\Pi$ to the interval $[t_1, t_2]$: set $\Pi' \leftarrow \Pi \cap \{(t, v) \in \mathbb{R}^2 : t_1 \leq t \leq t_2\}$.

3. Compute the dynamics limited expansion $\Sigma'$ of $\Pi'$ using Theorem 1.

4. Set $V_I$ to be the maximum velocity such that, starting at $q$ with velocity $V_I$, the train can come to a halt at the end of the shortest $c-1$ block path following $q$.

5. Set $\Sigma \leftarrow \Sigma' \cap \{(t,v) : t_1 \leq t \leq t_2, v \leq V_I\}$.

See Figure 3 for the sequence of profiles generated a typical execution of EXPAND. The resulting profile $I(\Pi)$ is atomic: this follows from Theorem 1, and the observation that steps 2 & 5 do not make an atomic profile non atomic. We note that if $\Pi$ has piecewise algebraic boundary curves, so does $I(\Pi)$. Additionally, the increase in the number of pieces in the boundary curves is at most 4; steps 2 & 5 increase the number of pieces by at most one each, and step 3 increases this by at most 2. As mentioned in Section 3, we start our algorithm with a single profile $R_0(s)$, whose boundaries are trivially algebraic. Since every profile generated by our algorithm is obtained from $R_0(s)$ by a sequence of expansions, all atomic profiles in our algorithm have piecewise algebraic boundaries.

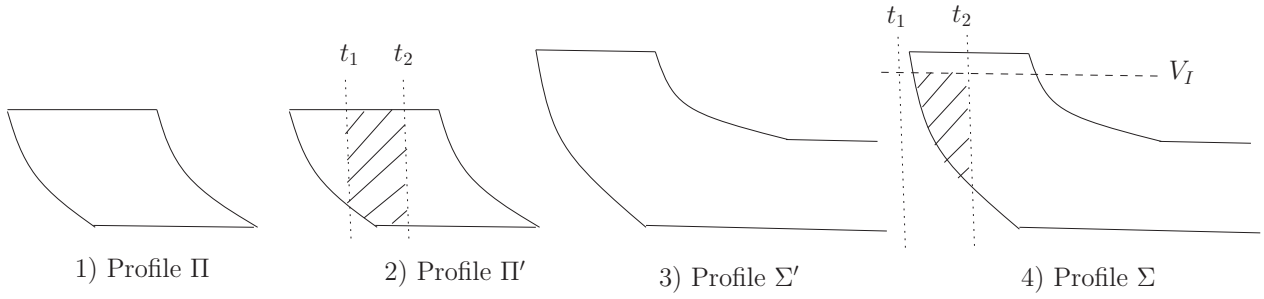


Figure 3: The steps in procedure EXPAND.

Given an atomic profile $\Pi$ at vertex $p$ and block $(p,q)$, it is easily seen that all possible ways in which the train can arrive without interference at $q$ starting at some point in $\Pi$ is given by

$$\bigcup_{I \text{ on } (p,q)} I(\Pi) = \left\{ (t,v) \in \mathbb{R}^2 \quad | \quad \begin{array}{l} \exists \quad (t_0, v_0) \in \Pi, \text{ and a non-interfering trajectory } \tau \text{ departing } s \\ \text{from rest at time 0, reaching } p \text{ at time } t_0 \text{ with velocity } v_0, \text{ and} \\ \text{reaching } q \text{ at time } t \text{ with velocity } v. \end{array} \right\}$$

# 5 The Algorithm

We start with the profile representing the train at its origin at time 0. We then use expansion steps to progressively build the complete arrival profile $R^*(p)$ at each vertex $p$. At each point in the execution, an estimate of $R^*(p)$ (for every vertex $p$) is maintained as a collection of atomic profiles. Some amount of pruning is also done, e.g. if one atomic profile is found to be a subset of another. In addition, the algorithm maintains a floating time horizon $T$ which is the earliest time found so far, at which the new train can reach its destination and come to a stop. Before expanding a profile, it is checked whether the earliest time in the profile is less than $T$. If not, this profile can not participate in the optimal path, and can be safely ignored. So it is useful to keep the unexpanded profiles in a priority queue $H$, which is ordered by the earliest times of the profiles. Expansion occurs in the order of increasing earliest time. Below, we describe our algorithm TRAIN-PATH.

1. Initialize:
   $H \leftarrow \{(s, (0,0,\mathbf{0},\mathbf{0}))\}$ *// Heap containing tuples to be processed.*
   $F \leftarrow \phi$ *// Set of processed tuples.*
   $T \leftarrow \infty$ *// Estimate of arrival time at the destination.*

2. While ($H \neq \phi$) do

   (a) Extract the tuple $(p, \Pi)$ with minimum earliest time (i.e., $t_{min} = E_\Pi(V_\Pi)$) from $H$.

   (b) If $t_{min} > T$, end. // *Algorithm has found the optimal path.*
   Else, if $p$ is the destination and $t_\Pi = \min_{(t,0)\in\Pi} t < T$, set $T \leftarrow t_\Pi$.

   (c) For every block $(p, q)$ out of vertex $p$ :
   For every signal interval $I$ on $(p, q)$ :

      i. Use procedure EXPAND to obtain profile $\Sigma = I(\Pi)$ at vertex $q$.

      ii. Check for possible pruning: If there is a tuple $(q, S') \in H \cup F$ such that $S' \supseteq \Sigma$, do nothing.

      iii. More pruning: If there is a tuple $(q, S'') \in H$ such that $S'' \subseteq \Sigma$, replace tuple $(q, S'')$ by $(q, \Sigma)$.

      iv. If neither condition (ii) nor (iii) holds, insert $(q, \Sigma)$ into $H$.

   (d) $F \leftarrow F \cup \{(q, \Pi)\}$.

The steps 2c(ii)-(iv) just ensure that the algorithm does not store redundant profiles (these are not necessary for the correctness). It is fairly straightforward to show that algorithm TRAIN-PATH is correct, and terminates in finite time. We outline a proof in Appendix B. However, this algorithm does not run in polynomial time in general. The number of profiles that need to be processed might grow exponentially. But given some fairly natural assumptions on the input instance, we show that only polynomially many profiles need to be processed, and algorithm TRAIN-PATH runs in polynomial time. The next section introduces an important notion that is used in proving the runtime of our algorithm.

## 5.1 History of a profile

Any profile $\Pi$ generated during the algorithm can naturally be associated with a 'parent' profile $\Pi_1$, which gave rise to $\Pi$ in step 2c(i) of some iteration. In particular, we would have $\Pi = I_1(\Pi_1)$ for some signal interval $I_1$. If we now look at the parent of $\Pi_1$, and then its parent and so on, we would eventually reach the point profile $R_0(s)$ at the source $s$. This is because we start the algorithm with $R_0(s)$ as the only profile. Thus we could write $\Pi = I_1(I_2(\cdots(I_k(R_0(s)))))$. We define this sequence of signal intervals, $I_1, \cdots, I_k$ as the *history* of profile $\Pi$. Note that $I_1$ is the most recent and $I_k$ the most distant signal interval in the history of $\Pi$. We also define the *$l$-history* of $\Pi$ to be the sequence of the $l$ most recent signal intervals in its history, namely $I_1, \cdots, I_l$.

In many special cases, we will prove that if two profiles $\Pi$ and $\Pi'$ have identical $l$-histories (for appropriately small values of $l$), then one is contained in another. Due to the conditions in steps 2c(ii)-(iv), this allows us to bound the number of profiles processed by the algorithm, and hence its running time. Note that the number of distinct $l$ histories is at most $\mathcal{I}^l$, where $\mathcal{I}$ is the total number of signal intervals. In Sections 6.1 & 6.2, we look at some special cases of exact train pathing that can be solved in polynomial time by algorithm TRAIN-PATH.

### 5.1.1 A numerical example

We now illustrate the notion of profile merging (steps 2c(ii)-(iv) in algorithm TRAIN-PATH) with an example. Consider a train specified by dynamics $V = 100\ km/hr$ and $A = D = 1250\ km/hr^2$, a block $(p, q)$ of length $d = 6\ km$, and a signal interval $I = (p, q, 2, 2.0\ hr, 2.2\ hr)$. $\Pi_1$ and $\Pi_2$ shown in Figure 4 are two incomparable profiles at vertex $p$; the velocity range for $\Pi_1$ is 0 to 50 $km/hr$, and for $\Pi_2$ is 0 to 100 $km/hr$. We will show that expanding these two profiles over signal interval $I$ results in profiles $\Sigma_1 = I(\Pi_1)$ and $\Sigma_2 = I(\Pi_2)$ at vertex $q$ such that $\Sigma_1 \subseteq \Sigma_2$. Hence at vertex $q$, only

$\Sigma_2$ would be processed by the algorithm: $\Sigma_1$ is a redundant profile and would be eliminated. In this simple example, profiles $\Sigma_1$ and $\Sigma_2$ have a common 1-history, and one is contained in the other.

We now describe the process of expanding $\Pi_1$ to obtain $\Sigma_1 = I(\Pi_1)$ in detail (the expansion of $\Pi_2$ is similar). Restricting $\Pi_1$ to time interval $[2.0, 2.2]$ results in a profile $\Pi_1'$ (step 2 in EXPAND) with top-left point $(u_1, t_1) = (50\ km/hr, 2.0\ hr)$ and bottom-right point $(0\ km/hr, 2.2\ hr)$. Next, applying Theorem 1 (using the values of $u_1, d, V, A, D$) to expand $\Pi_1'$ to obtain a profile $\Sigma_1'$ (step 3 in EXPAND), one can easily verify that the early curve of $\Sigma_1'$ evolves entirely from the top-left point and the late curve of $\Sigma_1'$ is $\infty$. In particular, the early curve of $\Sigma_1'$ is given by $E_{\Sigma_1'}(v) = \frac{v^2}{2DV} - \frac{v}{D} + \alpha_1$, where $\alpha_1 = t_1 + \frac{V}{2A} + \frac{V}{2D} + \frac{d}{V} + \frac{u_1^2}{2AV} - \frac{u_1}{A} = 2.11\ hr$ and the late curve $L_{\Sigma_1'}(v) = \infty$. In an identical fashion, one can verify that the profile $\Sigma_2'$ resulting in step 3 of EXPAND applied to $I(\Pi_2)$ satisfies $E_{\Sigma_2'}(v) = \frac{v^2}{2DV} - \frac{v}{D} + \alpha_2$ (where $\alpha_2 = 2.10\ hr$) and $L_{\Sigma_2'}(v) = \infty$. From this it is clear that $\Sigma_1' \subseteq \Sigma_2'$, and applying the common velocity and time restrictions (step 5 in EXPAND) to each of these, we obtain that $I(\Pi_1) \subseteq I(\Pi_2)$.

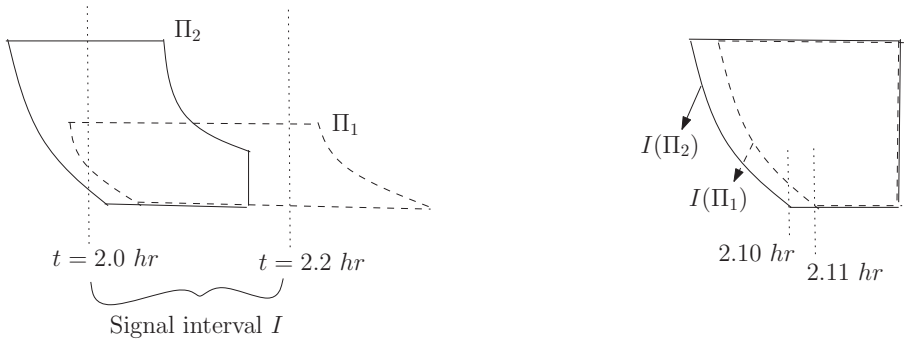


Figure 4: Example of profile merging

### 5.2 Avoiding issues of numerical round-off

At first sight it seems that algorithm TRAIN-PATH suffers from numerical round-off issues due to the complicated calculations arising during expansion. However, note that any profile is precisely defined by its history. So we can represent each profile in the algorithm by its history, which is just a sequence of signal intervals. This representation of profiles has the advantage that we store all profiles exactly, without any round-off. This also avoids having to explicitly store the profiles in terms of their boundary curves.

Numerical calculations are required in steps 2a, 2b, and 2c(ii)-(iv). The calculations in steps 2a & 2b are straightforward; given the history of profile $\Pi$ at vertex $p$, it is not difficult to write procedures that compute expressions for the earliest arrival time in $\Pi$ (for step 2a), and the earliest time in $\Pi$ at which the train comes to a halt at $p$ (for step 2b). A procedure similar to this is described in Lu et. al. [6]. Steps 2c(ii)-(iv) test whether a profile is redundant. In general, this may be susceptible to a round-off. However, in the cases that we consider (Section 6), this test can be replaced by a comparison of the histories of the profiles involved, and a comparison as in step 2a. Thus for our purposes, steps 2c(ii)-(iv) are free of any numerical round-off.

If the numerical calculations mentioned above can be performed exactly over reals, algorithm TRAIN-PATH will compute the exact optimal value. If reals are only stored to a certain precision, the algorithm computes the optimal value at a corresponding level of precision. Further, assuming a RAM model (where each basic numerical operation takes unit time), the running time of the algorithm does not depend on the precision.

# 6 Polynomial runtime

In this Section, we establish the polynomial runtime of algorithm TRAIN-PATH under some assumptions on the train dynamics (i.e., $V, A, D$).

## 6.1 Graphs with *long* blocks

A block is defined to be *long* if it has length at least $\frac{V^2}{2A} + \frac{V^2}{2D}$. This means that the train can start at any initial velocity, come to a halt, and then accelerate to any final velocity, all within a single long block. For example, if a train has maximum allowed velocity $V = 100\ km/hr$ maximum aceleration $A = 1250\ km/hr^2$ and maximum deceleration $D = 5000\ km/hr^2$, then it is capable of accelerating (resp., decelerating) from 0 to 100 $km/hr$ (resp., 100 to 0 $km/hr$) within a distance of 4 $km$ (resp., 1 $km$). In this case, any block of length *at least* 5 $km$ is said to be long. Note that requiring blocks to be long is equivalent to requiring the acceleration/deceleration of the train to be sufficiently large.

In this section, we show that if all blocks in the graph are long, then exact train pathing can be solved in polynomial time. The next lemma allows us to bound the number of profiles processed by algorithm TRAIN-PATH, when all blocks are long. The key idea is that if a block $(p, q)$ is long, then from any initial velocity (at $p$), the train will be able to come to a halt, and then reach $q$ with any final velocity. Similarly, from any initial velocity (at $p$), the train will be able to attain the maximum velocity $V$, and then reach $q$ with any final velocity. This implies that, after a dynamics limited expansion, the shapes of the early & late curves remain simple (in fact, the late curve will always lie at infinity).

**Lemma 2** *Suppose atomic profiles $S = (v_S, V_S, E_S, L_S)$ and $S' = (v_{S'}, V_{S'}, E_{S'}, L_{S'})$ at $q$ have identical 1-histories, with $E_S(V_S) \leq E_{S'}(V_{S'})$. Then $S \supseteq S'$.*

**Proof:** Let $I = (p, q, c, t, t')$ denote the common 1-history of profiles $S$ and $S'$. Note that the signal aspect $c$ must be at least 1. Suppose profiles $S$ and $S'$ are generated from profiles $Q$ and $Q'$ at vertex $p$. The rest of this proof looks at the expansion of profiles $Q$ and $Q'$ over signal interval $I$, according to the procedure EXPAND in Section 4.2. The expansion of $Q$ (respectively, $Q'$) first restricts it to the time interval $[t, t']$, to get a profile $R$ (respectively, $R'$). We next expand $R$ (resp., $R'$) using Theorem 1, to obtain a profile $P$ (resp., $P'$). Using Theorem 1 for the dynamics limited expansion of $R$ to $P$ (resp., $R'$ to $P'$) over a long block, we have (in the notation of Theorem 1) $v_{thresh} = 0$ and $L(u, v) \leq d$ for all velocities $u, v$, which implies:

$$v_P = v_{P'} = 0$$

$$V_P = V_{P'} = V$$

$$E_P(v) = E_R(V_R) + \frac{v^2}{2DV} - \frac{v}{D} + \frac{V}{2}(\frac{1}{A} + \frac{1}{D}) - \frac{1}{A}(\frac{V_R^2}{2V} - V_R) + \frac{d}{V} \text{ , for all } 0 \leq v \leq V$$

$$E_{P'}(v) = E_{R'}(V_{R'}) + \frac{v^2}{2DV} - \frac{v}{D} + \frac{V}{2}(\frac{1}{A} + \frac{1}{D}) - \frac{1}{A}(\frac{V_{R'}^2}{2V} - V_{R'}) + \frac{d}{V} \text{ , for all } 0 \leq v \leq V$$

$$L_P(v) = L_{P'}(v) = \infty \text{ , for all } 0 \leq v \leq V$$

Note that the coefficients of $v$ and $v^2$ in the early curves $E_P(v)$ and $E_{P'}(v)$ are the same. Thus we have $E_P(v) = E_{P'}(v) + a$, $\forall v \in [0, V]$, where $a$ is some constant. So $P$ and $P'$ are identical except for their early curves, which are translations of one another. Clearly, the one with the earlier early curve must contain the other.

The last step of the expansion restricts both profiles $P$ and $P'$ to the time interval $[t, t']$ and velocity range $[0, V_I]$, to get profiles $S$ and $S'$. But this does not change the containment of one profile in the other.■

Note that there can be only $\mathcal{I}$ distinct 1-histories. Using the preceding lemma, we see that over the entire execution of the algorithm, at most $\mathcal{I}$ distinct tuples will need to be processed. So we obtain the following.

**Theorem 2** *Algorithm TRAIN-PATH solves exact train pathing in polynomial time if all blocks in the rail network are long.*

We note that if the acceleration and deceleration are both infinite, then irrespective of their length, all blocks become trivially long. This case was analyzed by Malde [8], who gave a polynomial time algorithm for it. However, Malde's algorithm is not applicable for finite acceleration and deceleration.

### 6.2 Graphs with $\frac{1}{k}$-*long* blocks

For $k \geq 1$, a block is defined to be $\frac{1}{k}$-*long* if its length is at least $\frac{1}{k}\left(\frac{V^2}{2A} + \frac{V^2}{2D}\right)$. In other words, such a block has length at least $\frac{1}{k}$ times the minimum length of a long block. We believe that for every fixed $k \geq 1$, algorithm TRAIN-PATH runs in polynomial time if all blocks in the graph are $\frac{1}{k}$-long. In Section 6.1 we proved this for $k = 1$, but we have not been able to prove this for larger values of $k$.

In this section, we prove that for any $k \geq 1$, algorithm TRAIN-PATH runs in polynomial time if all blocks in the graph are $\frac{1}{k}$-long, and in addition the train has infinite deceleration ($D = \infty$). Under this assumption, the train can instantly come to a halt from any velocity; however it still takes non-zero time to accelerate to full speed. We note that even in this special case, exact train pathing remains NP-complete (see Section 9). We discuss this case at some length because it uses an argument which we feel could be useful for the general result. The main idea in this result is as follows. Let $p_0, p_1, \cdots, p_k$ be a path in the graph, and suppose an arrival profile $\Pi_0$ at $p_0$ is successively expanded over the blocks of this path to obtain profiles $\{\Pi_i\}_{i=1}^k$ at each vertex $\{p_i\}_{i=1}^k$. Then, the arrival profile $\Pi_0$ has a progressively decreasing effect (as $i$ increases) on the shape of the profile $\Pi_i$ resulting from it at $p_i$. Further, if all blocks are $\frac{1}{k}$-long, then $\Pi_0$ has *no* effect on the shape of the profile $\Pi_k$ at $p_k$. In other words, the profiles at any vertex can only have a small number of shapes, and the number of non-redundant profiles can be bounded.

Since all early curves are decreasing and the deceleration is infinite, the early curve is always a vertical line. Now, if two profiles have the *same* late curve, then one must contain the other since the early curves are vertical lines. Thus it is important to estimate the number of distinct late curves at each vertex. The next lemma formalizes the intuition described in the previous paragraph, that the shape of a profile depends only on the $k$ most recent expansions through which it is generated.

**Lemma 3** *If two profiles $S$ and $S'$ at a vertex $p$ have the same $k$-histories, then they must have the same late curves.*

**Proof:** We prove the following statement by induction on $i$ ($1 \leq i \leq k$): if two profiles $S$ and $S'$ have identical $i$-histories, then their late curves must be identical in the velocity range $[0, V\sqrt{i/k}]$. Setting $i = k$ in this statement gives us the lemma.

For the base case ($i = 1$), let $I$ be the 1-history of $S$ and $S'$. Let profiles $R$ and $R'$ be such that $S = I(R)$ and $S' = I(R')$. Since the deceleration is infinite, the lower velocity limit is 0 for all profiles, in particular for $R$ and $R'$. Now consider the expansion of $R$ and $R'$ over $I$. Even after restricting these profiles in step 2 of procedure EXPAND, the velocity lower limit is 0. Now from Theorem 1, performing a dynamics limited expansion of any such profile over a $\frac{1}{k}$-long block results in a late curve

that is $\infty$ in the velocity range $[0, \frac{V}{\sqrt{k}}]$. Now it is easy to see that even after the restriction in step 5 of EXPAND, the resulting profiles (namely $S$ and $S'$) have identical late curves in the range $[0, \frac{V}{\sqrt{k}}]$.

Now consider two profiles $S$ and $S'$ that have identical $i$-histories ($i > 1$). Let $I$ be the common 1-history of both profiles, and suppose $S = I(R)$ and $S' = I(R')$. Then $R$ and $R'$ have identical $i-1$-histories. By the induction hypothesis, the late curves of $R$ and $R'$ are identical in the range $[0, V\sqrt{i-1/k}]$. In computing $I(R)$ (resp., $I(R')$), let $P$ (resp., $P'$) denote the profile just before step 3 of EXPAND, and $Q$ (resp., $Q'$) the profile just after step 3 of EXPAND. So $Q$ results from the dynamics limited expansion (Theorem 1) of $P$; similarly $Q'$ results from $P'$. Since $R$ and $R'$ have identical late curves in $[0, V\sqrt{i-1/k}]$, so do $P$ and $P'$.

We now argue that $Q$ and $Q'$ have the same late curve in $[0, V\sqrt{i/k}]$. Consider a point $(t, v)$ on the late curve of $Q$ with $v \in [0, V\sqrt{i/k}]$. Using Theorem 1 on the dynamics limited expansion of $P$ to $Q$, we see that the point $(t, v)$ must result from a point $(t_0, u)$ in $P$ where,

1. $(t_0, u)$ is on the late curve of $P$.

2. $u$ is the smallest initial velocity such that it is possible to reach velocity $v$ within distance $d$. Here $d$ is the length of the block corresponding to $I$. Since all blocks are $\frac{1}{k}$-long, $d \geq \frac{V^2}{2Ak}$. So,

$$u = \sqrt{\max\{0, v^2 - 2Ad\}} \leq \sqrt{V^2\frac{i}{k} - 2A\frac{V^2}{2Ak}} \leq V\sqrt{\frac{i-1}{k}}$$

Thus $(t_0, u)$ lies on the portion of $P$'s late curve that is common with $P'$. So $(t, v)$ is also on the late curve of $Q'$. This shows that $Q$ have $Q'$ have the same late curve in $[0, V\sqrt{i/k}]$. Finally, $S$ and $S'$ are obtained by restricting $Q$ and $Q'$ to the same region, and the late curves of $S$ and $S'$ are also identical in the range $[0, V\sqrt{i/k}]$. This completes the induction, and also the proof.■

There are at most $\mathcal{I}^k$ distinct $k$-histories, and thus only so many different late curves and profiles are processed by the algorithm. This implies the following Theorem.

**Theorem 3** *For any constant $k$, algorithm TRAIN-PATH solves exact train pathing in polynomial time if all blocks in the rail network are $\frac{1}{k}$-long, and the train has infinite maximum deceleration.*

We will see in section 9 that exact train pathing remains NP complete even under the assumption of infinite deceleration. So we do not expect to see a polynomial time algorithm for general values of $k$.

# 7 Modeling train length

So far we have worked with the simplifying assumption that the train is a point object. Here, we discuss the changes needed in our algorithm to model positive train lengths. As mentioned earlier, the two ends of the train are its *head* and *tail*, and a block is said to be occupied by a train if any part of the train is in the block. We assume that the length of the train is such that the train never occupies more than $g$ (a small constant) blocks at a time. Under the assumption that train length is zero, the only positions of interest were when the train moved from one block to the next (i.e. at vertices of the network). But when the train has a positive length, it is important to keep track of positions where *either* the head or tail of the train moves from one block to the next.

**Definition 7** *The* position *of a train in the network is a tuple $(\beta, b)$, where $\beta = \beta_1, \cdots, \beta_l$ is a sequence of $l$ contiguous blocks occupied by the train such that the head is in block $\beta_1$ and the tail is in block $\beta_l$. $b$ is a boolean variable indicating the position of the head/tail of the train. If $b$ = head, the head of the train is at the start of $\beta_1$; if $b$ = tail, the tail of the train is at the start of $\beta_l$.*

For example, consider a train in the network of Figure 1 that occupies blocks $(a,b),(b,c)$ & $(c,h)$ such that its head is at vertex $c$ and its tail is between vertices $a$ & $b$ in block $(a,b)$. Then its position is given by the tuple $(\langle(c,h),(b,c),(a,b)\rangle, head)$.

We assume that for each train, there are separate initial & final blocks that lie *outside* the rail network $G$. These are referred to as *terminal blocks*, and each terminal block is incident to a particular vertex in $G$. The initial/final position of each train corresponds to occupying its initial/final block. This ensures that when a train is at its initial or final position, it does not reserve any block in the rail network $G$. Further, these terminal blocks do not participate in the signal rules since each terminal block is reserved for a single train and there can be no conflicts on such blocks. We denote the initial and final positions of the new train by $(\beta_s, b_s)$ and $(\beta_d, b_d)$ respectively. This assumption is made only to keep the description of the algorithm simple.

The definition of a *trajectory* is essentially the same: it consists of a path $P$ from one position of the train to another, the signal aspects under which the head enters each block in $P$, and a plot of velocity along $P$ satisfying train dynamics and the driver rule. As before, a set $\mathcal{T}$ of trajectories is said to be *realizable*, if no signal rules are violated when the trajectories in $\mathcal{T}$ are simultaneously followed by the corresponding trains. In this definition, it suffices to restrict attention to signals in blocks of $G$ since blocks not in $G$ (the terminal blocks) do not participate in the signal rules. The non-interference condition becomes slightly more complicated when modeling train lengths.

**Definition 8 Non-interference Condition:** *Suppose that $\tau$ is a trajectory of the new train from its initial position $(\beta_s, b_s)$ to some position $(\beta, b)$, on a path consisting of blocks $b_0, b_1, \cdots, b_k$. Here $b_0$ is the first block in $G$ that the train enters, and the head of the train in position $(\beta, b)$ is in block $b_k$. For each block $b_i$, let $h_i$ denote the time the head of the train enters $b_i$, $h_{i+1}$ the time the head leaves $b_i$, and $l_i$ the time the tail of the train leaves $b_i$. If any of these events does not occur in a realization of $\tau$, set the corresponding time to $\infty$. We say that $\tau$ does not interfere with $\mathcal{T}$ if the following hold:*

1. *For every $i = 0, \cdots, k$ where $h_{i+1} < \infty$, the signal aspect under which the train enters block $b_i$ is at most the minimum signal colour (as per signal record $S(\mathcal{T})$) in block $b_i$ during the time interval $[h_i, h_{i+1}]$.*

2. *For every $i = 0, \cdots, k$ where $l_i < \infty$, the minimum signal colour (as per signal record $S(\mathcal{T})$) in block $b_i$ during time interval $[h_i, l_i]$ is strictly greater than 0.*

For example, consider block $(b,c)$ in the network of Figure 1, where its signal shows colour 2 in the time interval $[1,3]$, colour 1 during $[3,5]$ and colour 0 during $[5,7]$. If the new train (i.e., its head) enters block $(b,c)$ at time 2 under signal aspect 2 and its head leaves $(b,c)$ at time 4, then it interferes with the existing trajectories. Another example is the following: if the head of the new train enters block $(b,c)$ at time 1.5, the head leaves $(b,c)$ at time 2.5, and the tail of the train leaves $(b,c)$ at time 6, then also it interferes with the existing trajectories.

Note that when the train length is zero, $l_i = h_{i+1}$ in the above definition, and this definition coincides with Definition 3. Similar to Lemma 1, we have the following equivalence which characterizes feasible trajectories of the new train.

**Lemma 4** *Given a set of realizable trajectories $\mathcal{T}$, and a trajectory $\tau$ of the new train from its initial to final positions, $\mathcal{T} \bigcup \{\tau\}$ is realizable if and only if $\tau$ does not interfere with $\mathcal{T}$.*

**Proof:** Suppose $\tau$ is a trajectory from $(\beta_s, b_s)$ to $(\beta_d, b_d)$ along path $b_0, \cdots, b_k$ that does not interfere with $\mathcal{T}$ (as in Definition 8). Note that the final position of the train corresponds to occupying only one terminal block, which is not in $G$; in particular $b_k \notin G$. For each $i = 0, \cdots, k-1$, let $a_i$ denote the signal aspect under which the head of the train enters block $b_i$. The reservations made by $\tau$ *in*

$G$ are the following (see the signal rules, Definition 1): (R1) for each $i = 0, \cdots, k-1$, it reserves all blocks in $G$ within $a_i$ blocks from $b_i$ for the time interval $[h_i, h_{i+1}]$, and (R2) for each $i = 0, \cdots, k-1$, it reserves block $b_i$ for the time interval $[h_i, l_i]$. Since the final position of the train corresponds to occupying only block $b_k$, in a realization of $\tau$, the tail of the train leaves each block $b_i$ (for $i < k$); i.e. for $i = 0, \cdots, k-1$, $h_{i+1}, l_i < \infty$. The first condition in Definition 8 implies that $a_i$ is at most the minimum signal colour seen in $b_i$ during $[h_i, h_{i+1}]$ (for $i < k$); so none of the (R1) reservations made by $\tau$ conflict with $\mathcal{T}$. The second condition in Definition 8 implies that block $b_i$ is unreserved (by $\mathcal{T}$) during $[h_i, l_i]$ (for $i < k$); so none of the (R2) reservations made by $\tau$ conflict with $\mathcal{T}$. In other words, $\mathcal{T} \bigcup \{\tau\}$ is realizable.

For the other direction, suppose $\tau$ is a trajectory for the new train that interferes with $\mathcal{T}$. There are two ways that this can happen.

1. The first condition in Definition 8 is violated at some block $b_i$. Clearly, $b_i$ is not a terminal block, so $b_i \in G$. Let $c_i$ denote the minimum signal colour in block $b_i$ during $[h_i, h_{i+1}]$. Since the first condition is violated, $\tau$ enters block $b_i$ under signal aspect $a_i > c_i$. This means that $\tau$ reserves all the blocks in $G$ located up to $a_i$ blocks ahead of $b_i$ during the entire period $[h_i, h_{i+1}]$ (while the head of the train is in $b_i$). But at some time in $[h_i, h_{i+1}]$, the signal in block $b_i$ shows colour $c_i < a_i \leq C$; *i.e.* some train in $\mathcal{T}$ reserves a block located $c_i + 1$ blocks from $b_i$. This conflicts with $\tau$ reserving all blocks within $a_i \geq c_i + 1$ blocks from $b_i$.

2. The second condition in Definition 8 is violated at some block $b_i$ (again $b_i$ must be in $G$). This means that the minimum signal colour in $b_i$ during $[h_i, l_i]$ is 0; i.e., block $b_i$ is reserved by a train in $\mathcal{T}$ at some time in $[h_i, l_i]$. This conflicts with the new train occupying block $b_i$ in the duration $[h_i, l_i]$.

Thus in both cases, $\mathcal{T} \bigcup \tau$ is not realizable. ■

We now describe the algorithm for exact train pathing in the presence of positive train length. It is useful to state it in comparison to the algorithm TRAIN-PATH for zero length trains. The algorithm TRAIN-PATH, may be viewed as an efficient simulation of all possible ways in which the train can move starting from its origin at time 0. The interesting events in this simulation correspond to the train (a point object) entering a block, and indeed, the priority queue $H$ described in TRAIN-PATH maintains these events. The algorithm for positive length trains will also be a simulation, however, the interesting events in this case are: (a) the train head entering a block, and (b) the train tail entering a block. Our modified algorithm precisely keeps track of such events.

The algorithm computes arrival profiles at different *positions* of the train. An arrival profile at a position of the train is a straightforward extension of Definition 4. The algorithm now maintains tuples of the form $h = (\beta, b, \Pi)$, where $(\beta, b)$ is a position of the train, and $\Pi$ is an atomic arrival profile at this position. We can now describe the procedure EXPAND, for positive train length. EXPAND takes a tuple $h = (\beta, b, \Pi)$ as input, and computes the set of all new tuples resulting from $h$. This corresponds to extending non-interfering trajectories ending at position $(\beta, b)$ to non-interfering trajectories ending at every position that can follow $(\beta, b)$. Let $B_h$ (resp., $B_t$) denote the block containing the head (resp., tail) of the train in $h$. First we determine the next position of the train. Consider what happens first when the head of the train moves forward in its current block. Set $b' = head$ if the head of the train leaves block $B_h$, and $b' = tail$ if the tail of the train leaves block $B_t$. Let $d_{exp}$ denote the distance moved by the head of the train from the position $(\beta, b)$ until one of the preceding events occurs. There are four cases depending on the values of $b$ and $b'$.

1. $b = b' = head$.
   For every signal interval $I_h = (p_h, q_h, c_h, t_1^h, t_2^h)$ on $B_h$:
   For every block $B$ following $B_h$:

(a) Set $\beta' \leftarrow \beta \cup B$. // $(\beta', b')$ *is the next position of the train.*

(b) Set $\Pi' \leftarrow \Pi \cap \{(t, v) : t_1^h \leq t \leq t_2^h\}$.

(c) Expand $\Pi'$ to $\Sigma'$ over length $d_{exp}$ using Theorem 1.

(d) Set $\Sigma \leftarrow \Sigma' \cap \{(t, v) : t_1^h \leq t \leq t_2^h,\ 0 \leq v \leq V(I_h)\}$; $(t_1^h, t_2^h)$ is the duration of signal interval $I_h$, and $V(I_h)$ is the velocity limit imposed by the driver rule for $I_h$.

(e) Return tuple $(\beta', b', \Sigma)$. // $history((\beta', b', \Sigma)) = history(h) \cdot I_h$

2. $b = head$, $b' = tail$.
For every signal interval $I_h = (p_h, q_h, c_h, t_1^h, t_2^h)$ on $B_h$:

(a) Set $\beta' \leftarrow \beta \setminus B_t$.

(b) Set $\Pi' \leftarrow \Pi \cap \{(t, v) : t_1^h \leq t \leq t_2^h\}$.

(c) Expand $\Pi'$ to $\Sigma'$ over length $d_{exp}$ using Theorem 1.

(d) Set $\Sigma \leftarrow \Sigma' \cap \{(t, v) : t \leq t_{red}\}$, where $t_{red}$ is the first time the signal in block $B_t$ turns red *after* the signal interval in which the train entered $B_t$.

(e) Return tuple $(\beta', b', \Sigma)$. // $history((\beta', b', \Sigma)) = history(h) \cdot I_h$

3. $b = tail$, $b' = head$.
For every block $B$ following $B_h$:

(a) Set $\beta' \leftarrow \beta \cup B$.

(b) Expand $\Pi$ to $\Sigma'$ over length $d_{exp}$ using Theorem 1.

(c) Set $\Sigma \leftarrow \Sigma' \cap \{(t, v) : t_1^h \leq t \leq t_2^h,\ 0 \leq v \leq V(I_h)\}$, where $I_h$ is the signal interval in which the train entered block $B_h$, and $t_1^h, t_2^h, V(I_h)$ are as in step 1d above.

(d) Return tuple $(\beta', b', \Sigma)$. // $history((\beta', b', \Sigma)) = history(h)$

4. $b = b' = tail$.

(a) Set $\beta' \leftarrow \beta \setminus B_t$.

(b) Expand $\Pi$ to $\Sigma'$ over length $d_{exp}$ using Theorem 1.

(c) Set $\Sigma \leftarrow \Sigma' \cap \{(t, v) : t \leq t_{red}\}$, where $t_{red}$ is the first time the signal in block $B_t$ turns red *after* the signal interval in which the train entered $B_t$.

(d) Return tuple $(\beta', b', \Sigma)$. // $history((\beta', b', \Sigma)) = history(h)$

Here is an example for the modified EXPAND procedure. Consider a tuple $h$ at the train position $(\beta_0, b_0) = (\langle (c, h), (b, c), (a, b) \rangle, head)$ in the network of Figure 1. Suppose that the distance from the tail of the train (in position $(\beta_0, b_0)$) to vertex $b$ is 0.8 $km$ and the length of block $(c, h)$ is 1 $km$. Then the next position of the train is given by $(\beta_0', b_0') = (\langle (c, h), (b, c) \rangle, tail)$, and case 2 of this procedure is applied in expanding tuple $h$. Additionally, the distance used in the dynamics limited expansion (step 2c) is $d_{exp} = 0.8$ $km$.

It can be easily verified that the above description of EXPAND correctly enforces the new non-interference condition. Given this description of expansion, algorithm TRAIN-PATH remains essentially the same. It starts with the single tuple, $T_0 = (\beta_s, b_s, \{0, 0\})$, corresponding to the train being at rest in its initial position; and step 2c is replaced by the modified procedure EXPAND applied to the tuple corresponding to the earliest arrival time.

Since the algorithm starts with a single tuple $T_0$, all other tuples are derived from this initial tuple. For any tuple $h$ generated in the algorithm, let $I_k, \cdots, I_1$ be the sequence of signal intervals

(over contiguous blocks) under which $T_0$ was expanded to generate $h$. The *history* of tuple $h$ is the sequence $I_k, \cdots, I_1$, with $I_1$ being the most recent signal interval. This computation of history of a tuple is also shown in procedure EXPAND in comments. As before, for any number $l$, the $l$-history of $h$ is the sequence $I_1, \cdots, I_{l-1}$ of the $l$ most recent signal intervals.

## 7.1 Polynomial runtime

We saw that algorithm TRAIN-PATH runs in polynomial time, given some assumptions on the input instance (Section 6). Here we argue that even with positive train length, our algorithm runs in polynomial time, under similar assumptions. The important fact is that under both these assumptions, we can bound the number of arrival profiles at each position of the train. This is formalized in Lemmas 5 & 6, and their proofs are along the same lines as Lemmas 2 & 3 respectively.

### 7.1.1 Graphs with *long* blocks

To deal with train lengths, we need a stricter notion of long blocks. Define a block to be *long* if its length is at least $\frac{V^2}{A} + \frac{V^2}{D}$. We claim that if all blocks of the graph are long, then the modified TRAIN-PATH algorithm (as described above) runs in polynomial time. In this case, we have the following (similar to Lemma 2).

**Lemma 5** *Suppose $h_0 = (\beta, b, \Pi)$ and $h'_0 = (\beta, b, \Pi')$ are two tuples at the same position $(\beta, b)$, that have the same $g+2$-history. Then $\Pi \subseteq \Pi'$ or $\Pi' \subseteq \Pi$.*

**Proof:** Since the $g+2$ history is same for tuples $h_0$ and $h'_0$, the sequence of operations in the last two calls to EXPAND before generating $h_0$ and $h'_0$ are identical. Let $h_2$ & $h_1$ denote the last two tuples through which tuple $h_0$ was generated, and similarly $h'_2$ & $h'_1$ the last two tuples for $h'_0$. For $i = 1, 2$, let $d_i$ denote the common distance over which expansion takes place (i.e., the distance $d_{exp}$ in EXPAND) in generating $h_{i-1}$ from $h_i$ and $h'_{i-1}$ from $h'_i$. From the description of EXPAND, it is easy to see that over any two consecutive expansions, the train completely covers at least one block at its head or tail. So $d_1 + d_2 \geq \frac{V^2}{A} + \frac{V^2}{D}$, i.e., $\max\{d_1, d_2\}$ is at least $\frac{V^2}{2A} + \frac{V^2}{2D}$. Using Theorem 1 for dynamics limited expansion over this longer distance, we see that after this operation, one of the resulting profiles is contained in the other (same argument as Lemma 2). The lemma now follows from the fact that all remaining operations (which are same for $h_0$ and $h'_0$) do not change the containment of one profile in the other.■

From this lemma, we see that the number of distinct tuples that need to be processed is at most $\mathcal{I}^{g+2}$. Since $g$ (maximum number of blocks the train can occupy) is a constant, we get a polynomial time algorithm.

### 7.1.2 Graphs with $\frac{1}{k}$-*long* blocks

Recall that a block is $\frac{1}{k}$-long if its length is at least $\frac{1}{k}(\frac{V^2}{2A} + \frac{V^2}{2D})$, for a constant $k$. We showed in section 6.2 that algorithm TRAIN-PATH runs in polynomial time if all blocks are $\frac{1}{k}$-long, *and* the train is capable of infinite deceleration. Here we extend this result to the case when the train has a positive length. In this case, we have the following (similar to Lemma 3).

**Lemma 6** *Suppose $h = (\beta, b, \Pi)$ and $h' = (\beta, b, \Pi')$ are two tuples at the same position $(\beta, b)$, that have the same $2k + g$-history. Then $\Pi$ and $\Pi'$ have the same late curve, and hence one is contained in the other.*

**Proof:** We only give a sketch of this proof, as the details are similar to Lemma 3.

Let $(\beta_{2k}, b_{2k}, \Pi_{2k}), \cdots, (\beta_1, b_1, \Pi_1), (\beta_0, b_0, \Pi_0) = h$ denote the sequence of the last $2k$ tuples through which tuple $h$ was generated; similarly $(\beta_{2k}, b_{2k}, \Pi'_{2k}), \cdots, (\beta_1, b_1, \Pi'_1), (\beta_0, b_0, \Pi'_0) = h'$ denote the last $2k$ tuples for $h'$. Due to the common $2k + g$ history, profiles $\Pi_i$ and $\Pi'_i$ refer to the same position of the train, and the operations performed in expanding $\Pi_{i+1}$ to $\Pi_i$ are *identical* to those in expanding $\Pi'_{i+1}$ to $\Pi'_i$ (for $i = 2k-1, \cdots, 0$).

Let $d_i$ denote the distance ($d_{exp}$ in EXPAND) over which the expansion of $\Pi_{i+1}$ to $\Pi_i$ (also $\Pi'_{i+1}$ to $\Pi'_i$) took place. Since all operations are identical, we can prove inductively (as in Lemma 3) that the late curves of $\Pi_j$ and $\Pi'_j$ agree for velocities in the interval $[0, \sqrt{2A(d_{2k-1} + \cdots + d_j)}]$ (for $j = 2k-1, \cdots, 0$). As observed earlier, the sum of the distances $d_i + d_{i+1}$ of any two consecutive expansions is at least the minimum block length, $\frac{1}{k}\frac{V^2}{2A}$. Now using $j = 0$ in the inductive statement above, the late curves of $\Pi$ and $\Pi'$ agree for velocities in $[0, V]$, i.e. they are the same. Since $D = \infty$, the early curve is always a vertical line; so one of $\Pi$ & $\Pi'$ is contained in the other.■

Using this lemma, we see that the number of tuples processed by the algorithm is at most $\mathcal{I}^{2k+g}$. Thus our algorithm runs in polynomial time in this case.

# 8 Some Extensions

In this section, we consider some extensions of the basic model and outline how our algorithm can be modified to handle these.

**Velocity limit in the driver rule:** In the driver rule (Definition 2), we require that a train entering a block $e = (u, v)$ under signal aspect $c'$ be able to stop within the shortest $c' - 1$ block path after $e$. Clearly, this is too restrictive if the train actually has to take a different (longer) path out of $v$ - in which case we should allow it to pass $v$ with a higher velocity. We would like to relax the driver rule as: If a train enters block $e = (u, v)$ under signal aspect $c'$ (at vertex $u$), its arrival velocity at vertex $v$ must be such that it can come to a stop within $c' - 1$ blocks after $v$, along the path that it has to take.

Our algorithm can be modified to handle this detail. The basic idea is to use a separate profile to represent each possible path out of vertex $q$. Each profile would have a *label* consisting of at most $C - 2$ blocks, which indicates what future path the train must follow in order for that profile to be considered. Procedure EXPAND is modified as follows. A profile can only be expanded along the first block in its label. In expanding a profile $\Pi$ over signal interval $I = (p, q, c, t_1, t_2)$, we will generate a separate profile for each $c - 1$ block path $P$ out of $q$ that agrees with the label of $\Pi$. The profiles for these paths differ only in their velocity threshold - the threshold for the profile associated with path $P$ is $V_{I,P} = \min\{V, \sqrt{2DL_P}\}$ where $L_P$ is the length of path $P$.

For example, consider the network of Figure 1 with $C = 4$ signal colours. In this case, each arrival profile is labeled with a path of at most 2 blocks that represents the future path along which the profile can be expanded. Suppose $\Pi$ is a profile at vertex $a$ with label $\langle (a, b), (b, c) \rangle$. Then $\Pi$ can only be expanded over a signal interval in block $(a, b)$. The expansion of $\Pi$ over a colour 3 signal interval in block $(a, b)$ results in three different profiles $\Pi_d, \Pi_f, \Pi_h$ at vertex $b$, having labels $\langle (b, c), (c, d) \rangle$, $\langle (b, c), (c, f) \rangle$ & $\langle (b, c), (c, h) \rangle$ respectively. In particular, if the lengths of the blocks are $(b, c)$: 0.8 $km$, $(c, d)$: 0.45 $km$, $(c, f)$: 1 $km$ & $(c, h)$: 1.65 $km$, the maximum deceleration of the train is $D = 4000$ $km/hr^2$ and the maximum velocity is 150 $km/hr$, then the velocity thresholds in the three profiles $\Pi_d, \Pi_f, \Pi_h$ are 100, 120 & 140 $km/hr$ respectively.

The important change in the running time analysis is that the labels must be considered a part of the history. This is natural, since the label represents a commitment to the path to be taken that the train has *already* made. Thus we define the history of a profile $\Pi$ at a vertex $p$ as a pair: the sequence of signal intervals that gave rise to $\Pi$ (as before) and the path $P$ out of $p$ that $\Pi$ is labelled with.

With this change, Lemma 2 and Lemma 3 (hence the polynomial-time results) still hold. The only difference is in counting how many profiles can have the same $l$-history. Since there can be at most $\Delta^{C-2}$ different labels ($\Delta$ is the maximum number of outgoing blocks at any vertex), this number is at most $\mathcal{I}^l \cdot \Delta^{C-2}$. For the common case of $\Delta = 2$ and $C = 3$ this is only doubling the work.

**Modeling junctions and bi-directed blocks:** A junction is a crossover point between blocks. Our basic model does not include junctions as all vertices are assumed to be branching points: each vertex has either only one incoming block or only one outgoing block. First note that a zero-length junction can be modeled as a vertex having several incoming and outgoing blocks. Our algorithm extends to this case as follows. In addition to blocks, each vertex representing a junction is also treated as a resource that can be occupied by at most one train. Based on the trajectories of the earlier trains, each block has a set of signal intervals as before (the *signal record*), and additionally each junction vertex has a set of time intervals when it is unoccupied. Now during expansion, when an arrival profile passes through a junction vertex, it is restricted to only those time intervals when that vertex is unoccupied. It is now straightforward to also model junction lengths: in this case, each time a profile passes through a junction, the algorithm needs to perform an expansion over the length of the junction (as in the case of blocks).

A bi-directed block is one which trains can traverse in either direction. There is one signal at each end of the block, signalling the block occupancies in the respective directions. Our model required the rail network to be a directed graph only for simplicity; it is easy to see that the algorithm works even when there are bi-directed blocks.

Thus we can extend our algorithm to model junctions and bi-directed blocks as well; this gives us the general network model of Lu et al. [6]. Even in this model, if the block lengths are 'long' enough, our algorithm will run in polynomial time.

**Variations in maximum velocity:** In our model, we assumed that the maximum velocity of the train is constant over the entire network. However, there may be different velocity limits in different blocks. Note that our algorithm performs operations for each block separately; so it can easily handle this case as well. Long blocks are now defined relative to the maximum velocity, acceleration & deceleration in that block. With this definition of a long block, it is easy to see that our algorithm runs in polynomial time if all blocks are long.

It is also possible that the maximum velocity varies over portions of the same block. Again, our algorithm can be modified to handle this case. In the expansion based on train dynamics (step 3 in procedure EXPAND), we perform a sequence of expansions, one for each velocity limit within the block. For example, suppose a block $(p, q)$ consists of the two regions $(p, r)$ & $(r, q)$, each with a different velocity limit. Then the dynamics limited expansion of a profile $\Pi$ over block $(p, q)$ is obtained as follows: first compute the profile $\Pi'$ resulting from expanding $\Pi$ over the distance of $(p, r)$ subject to its velocity limit; then expand $\Pi'$ over a distance of $(r, q)$ (under $(r, q)$'s velocity limit). However, the equivalent of a 'long' block in this setting is not clear, and so the results on polynomial runtime do not extend directly to this case.

# 9 Hardness of Exact Train Pathing

In this Section, we show that the exact train pathing problem in its full generality is NP-complete. An algorithm for a minimization problem $\mathcal{P}$ is said to be a $c$-approximation algorithm, if for every instance $I$ of $\mathcal{P}$, the algorithm obtains a solution having value at most $c$ times the optimal value of $I$. The approximation factor $c$ may also be a function $c(|I|)$ of the size of the input instance $I$. Our proof also shows that unless P=NP, there is no polynomial time $c$-approximation algorithm for exact train

pathing, for any factor $c$. We note that exact train pathing with $C = 2$ signal colours is polynomially solvable by our algorithm. On the other hand, as we show next, the problem becomes NP-complete with even $C = 3$ signal colours.

**Theorem 4** *The exact train pathing problem is NP-complete. Additionally, it is NP-hard to approximate this problem to any factor that is at most exponential in the input size.*

**Proof:** We reduce from the subset sum problem, which is known to be NP complete. The *subset sum* problem is defined as follows: Given a set $N = \{a_1, \cdots, a_n\}$ of positive integers, and a positive integer $B$, does there exist a subset $S \subseteq N$, such that $\sum_{a \in S} a = B$? Given any instance of the subset sum problem we construct an instance of exact train pathing as follows.

The network $G$ is as in Figure 5, consisting of $2n + 3$ nodes lying on a path, and the signals are capable of showing $C = 3$ colours. Note that each vertex in $G$ has either at most one incoming block or at most one outgoing block: so it satisfies the assumption of our basic model (Section 2). The blocks in network $G$ are as follows: (a) For each $i = 1, \cdots, n$, there are blocks $(p_i, p'_i)$ and $(p'_i, p'_{i+1})$, each of length $\frac{B}{2}$; (b) For each $i = 1, \cdots, n$, there is another block $(p_i, p'_i)$ of length $B + a_i$; (c) Blocks $(p_{n+1}, p_{n+2})$ and $(p_{n+2}, p_{n+3})$ of length $B$ each. The train to be pathed has origin $p_1$, destination $p_{n+3}$, and departure time 0. This train has maximum velocity $V = \sqrt{2(n+3)BA}$, maximum acceleration $A$, and maximum deceleration $D = \infty$.[5] The set $\mathcal{T}$ of realizable trajectories of old trains consists of two trajectories, both for trains with origin $p_{n+2}$ and destination $p_{n+3}$. Train 1 reserves block $(p_{n+2}, p_{n+3})$ during the time interval $[0, \sqrt{2B(n+1)/A}]$, and train 2 reserves block $(p_{n+2}, p_{n+3})$ during the time interval $[\sqrt{2B(n+3)/A}, M]$ (here $M$ is some large value). In the signal record $S(\mathcal{T})$, the signal at $p_{n+2}$ shows green only in the intervals $[\sqrt{2B(n+1)/A}, \sqrt{2B(n+3)/A}]$ and $[M, \infty)$, and is red at other times. Similarly the signal at $p_{n+1}$ is green only in these intervals, and is *yellow* otherwise. All other signals are green throughout.

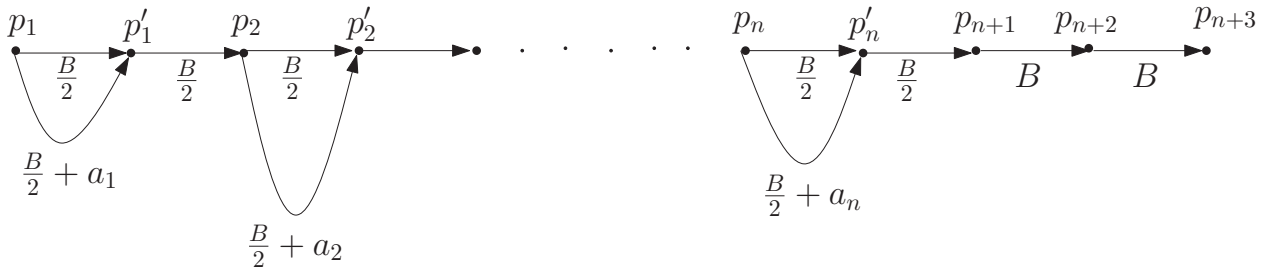


Figure 5: The network $G$ used in the reduction.

Suppose that the given subset sum instance has a solution, i.e. there is a set $S \subseteq N$ such that $\sum_{a \in S} a = B$. From this a trajectory can be constructed for the train as follows. The path is: for each $i = 1, \cdots, n$ if $a_i \in S$, use the long block $(p_i, p'_i)$ (length $\frac{B}{2} + a_i$), otherwise use the short block $(p_i, p'_i)$ (length $\frac{B}{2}$). Throughout the journey, the train accelerates at the maximum acceleration $A$. The length of the path traversed by the train until it reaches $p_{n+1}$ is $\sum_{a \notin S} \frac{B}{2} + \sum_{a \in S}(\frac{B}{2} + a) + n \cdot \frac{B}{2} = Bn + \sum_{a \in s} a = B(n+1)$. Thus it reaches $p_{n+1}$ at time $\sqrt{2B(n+1)/A}$, and it sees a green signal at $p_{n+1}$. Likewise, the distance to $p_{n+2}$ from $p_1$ is $B(n+2)$ and hence the arrival time at $p_{n+2}$ is $\sqrt{2B(n+2)/A}$ and the signal is green. The last block $(p_{n+2}, p_{n+3})$ has length $B$, so the train passes through this block just in time, reaching $p_{n+3}$ at time $\sqrt{2B(n+3)/A}$. So in this case, the optimum travel time is $\sqrt{2B(n+3)/A}$.

[5]This assumption on $D$ is not critical in the proof, and is only made for simplicity. A slight modification of this proof shows the NP hardness of instances with any given maximum deceleration.

Suppose now that the exact train pathing instance has a solution with travel time less than $M$. We first argue that in such a solution the train must see a green signal at $p_{n+1}$. Suppose the train sees a yellow, then it will have to come to rest at $p_{n+2}$. Then the fastest that it can cover the last block is in time $\sqrt{2B/A} > \sqrt{2B(n+3)/A} - \sqrt{2B(n+1)/A}$, the available window. Thus the signal must be green. Since the train must see a green signal at $p_{n+1}$, it has only time $\sqrt{2B(n+3)/A} - \sqrt{2B(n+1)/A}$ in which to cover the last two blocks. As seen above, starting with a velocity of $\sqrt{2AB(n+1)}$ at $p_{n+1}$, and continuously accelerating at $A$, the train barely manages to cover these two blocks in the available time window. Hence, in any solution, the velocity at $p_{n+1}$ is at least $\sqrt{2AB(n+1)}$.

Suppose now that the train covers distance $L$ in getting to $p_{n+1}$. Clearly, $\sqrt{2AL} \geq$ velocity at $p_{n+1} \geq \sqrt{2AB(n+1)}$. So $L \geq B(n+1)$. Now when the train reaches $p_{n+3}$, say at time $T$, we know it covers distance $L + 2B$. Starting from 0 velocity, the time required to cover distance $L + 2B$ is at least $\sqrt{2(L+2B)/A}$. But the arrival at $p_{n+3}$ must be before the signal on the last block $(p_{n+2}, p_{n+3})$ turns red. i.e., $\sqrt{2B(n+3)/A} \geq T \geq \sqrt{2(L+2B)/A}$. So $L \leq B(n+1)$. Thus we have $L = B(n+1)$. Let

$$I = \{i \in \{1, \cdots, n\} \mid \text{long block } (p_i, p_i') \text{ is traversed by train } \}$$

Then we know that $\sum_{i \in I} a_i = B$. Thus a solution for the subset sum problem can be constructed from this solution of exact train pathing.

Note that in this proof, when subset sum has a solution, the optimal value of exact train pathing is $\sqrt{2B(n+3)/A}$; and when subset sum has no solution, the optimal value is at least $M$. The ratio of these values is at least $M/\sqrt{2B(n+3)/A}$. This means that if exact train pathing has a polynomial time $c$-approximation algorithm for any $c < M/\sqrt{2B(n+3)/A}$, then the subset sum problem can be solved in polynomial time. However, for any value of $M$ that has length polynomial in $\log B$ and $n$, this reduction remains polynomial time. The size of the resulting exact train pathing instance is also polynomial in $\log B$ and $n$. So, taking an adequately large value of $M$, this proof also shows that it is NP-hard to approximate exact train pathing to any factor that is exponential in the input size. ■

The train pathing instances that we construct above may appear somewhat contrived. Nevertheless, they are legal instances which any algorithm that attempts to solve the exact train pathing problem as defined will have to contend with. We have already indicated how the problem could be modified (e.g. insisting that all blocks be long) so as to make it more tractable as well as perhaps more in conformance with instances arising in practice. This hardness result highlights the need for studying such restricted versions.

# 10 Concluding Remarks

We believe that our algorithms are of practical interest. If all blocks are long as defined earlier, then clearly the algorithms will run in polynomial time. If all blocks are not long, then there is a danger that they may be too slow. However, as shown in the previous section, this will likely happen only for somewhat special track topologies. For most track topologies arising in practice, we feel that our strategies for pruning redundant profiles will work well and will keep the algorithm runtime modest.

We have not considered many complications that would arise in developing complete pathing programs: e.g. the existence of so called *Warner signals* which are positioned upstream from the real signal and which simply show the same colour as that shown by the main signal. When the real signal becomes less restrictive (e.g. changes from red to yellow) an approaching train learns of this at the warner itself, and thus can start accelerating right away. We believe that most such additional complications could be accommodated into our algorithm, albeit with some increase in execution time.

Finally, we note that as mentioned by Carey and Lockwood [3], an algorithm for single train pathing can be used in a heuristic algorithm for multiple train pathing. This idea, which has been used in other papers as well, is very natural: assign a priority to the trains (which could simply be

their earliest possible departure time, but need not be so) and schedule them in the priority order, ensuring that the $i$th train does not upset the schedules of the $i-1$ previously scheduled trains. If the train network is such that our algorithm runs in polynomial time for single train pathing (i.e. if one of the cases we considered apply), then it will run in polynomial time for pathing many trains as well.

We note that the above approach does not address the sequencing aspect of the multiple train pathing problem, and hence may not result in a good overall schedule. An alternate heuristic for scheduling multiple trains can be obtained by modifying the single train pathing problem as follows. Instead of asking for the minimum time schedule for the new train such that the schedules of the previous trains are not disturbed, one may ask for a feasible schedule for the new train with its departure and arrival times in specified time windows such that the number of disruptions to the previous trains is minimized. How to extend the ideas in this paper to solve this modified single train pathing problem is an important question to be explored. It would also be interesting to determine if these ideas help in obtaining better heuristics for the multiple train pathing problem.

**Acknowledgements:** We thank Narayan Rangaraj for many conversations and suggestions. We also thank the two anonymous reviewers, whose comments have substantially improved the paper.

## References

[1] C. Blendinger, V. Mehrmann, A. Steinbrecher, and R. Unger. Numerical simulation of train traffic in large networks via time-optimal control. *Technical Report, Institute of Mathematics, Technical university of Berlin*, February 2002.

[2] X. Cai and C. J. Goh. A fast heuristic for the train scheduling problem. *Computers in Operations Research*, 21(5):499–510, 1994.

[3] M. Carey and D. Lockwood. A model, algorithms and strategy for train pathing. *Journal of the Operational Research Society*, 46:988–1005, 1995.

[4] A. Higgins, E. Kozan, and L. Ferreira. Optimal scheduling of trains on a single line track. *Transportation Research-B*, 30(2):147–161, 1996.

[5] David Kraay, Patrick T. Harker, and Bintong Chen. Optimal pacing of trains in freight railroads: model formulation and solution. *Operations Research*, 39(1), 1991.

[6] Quan Lu, Maged Dessouky, and Robert Leachman. Modelling Train Movements Through Complex Rail Networks. *ACM Transactions on Modelling and Computer Simulation*, 14(1):48–75, 2004.

[7] K. Madhusudan and B. Gopi Singh, 2002. Personal Communication.

[8] Sanket Malde. Train scheduling, 2001. BTech Project Report, Department of Computer Science and Engineering, I.I.T. Bombay.

[9] A. I. Mees. Railway scheduling by network optimization. *Mathematical and Computer Modelling*, 15(1):33–42, 1991.

[10] Alexandre B.S. Moreira and Rui C. Oliveira. A decision support system for operational planning in railway networks. *8th conference on Computer-Aided Scheduling of Public Transport.*, June 2000.

[11] M.R.Garey, D.S.Johnson, and R.Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics Operation Research*, 1:117–129, 1976.

[12] Elias Olivera and Barbara M Smith. A job-shop scheduling model for the single track railway scheduling model. *Research Report, School of Computer Studies, University of Leeds*, August 2000.

[13] Christos H. Papadimitriou and Paris C. Kanellakis. Flowshop scheduling with limited temporary storage. *J. Assoc. Comput. Mach.*, 27:533–549, 1980.

[14] G. Raghuram and V. Rao. A decision support system for improving railway line capacity. *Public Enterprise*, 11:64–72, 1991.

[15] N. Rangaraj, A. Ranade, K. Moudgalya, R. Naik, C. Konda, and M. Johri. A Simulator for estimating Railway Line Capacity. In *Proceedings of the Sixth International Conference of the Association of Asia-Pacific Operational Research Societies within IFORS*, pages 347–355, 2003.

# A Safety and deadlock freedom

We assume that a train does not occupy any block when it is at rest in its origin or destination. At all other points of time, it occupies some block. A collision is said to occur if two trains occupy the same block at the same time. The following claim follows from definitions 1 & 2.

**Claim 1** *If the signal and driver rules are followed, there will be no collisions, even in the presence of delays.*

A *deadlock* is said to occur if there are two or more trains, each waiting for a block occupied by some other. We would like the trajectories generated by our algorithm to be free of deadlocks, even in case of delays. We demonstrate one such strategy, which ensures a deadlock free schedule for all the trains.

**Claim 2** *Given a set $\mathcal{T}$ of realizable trajectories, there is a strategy for allocating blocks to trains such that, even in the presence of delays, every train reaches its destination without deadlocks. Additionally, in this strategy, every train continues to follow its original path in $\mathcal{T}$.*

**Proof:** For a set of realizable trajectories $\mathcal{T}$, we define a *resource-rule*, which requires that each block $b$ be allocated to different trains in the same order as in a (delay-free) realization of $\mathcal{T}$. Clearly, in a delay-free scenario, there are no deadlocks, and the resource rule is followed. We will show that if the resource rule is followed and the trains follow their prescribed paths in $\mathcal{T}$, no deadlocks will occur even in presence of delays.

Suppose (for a contradiction) that after some delays, there is a deadlock involving trains $t_1, t_2, \cdots, t_n$. Let $b_i$ denote the block currently occupied by train $t_i$. Suppose that the deadlock is caused by train $t_1$ waiting for block $b_2$, train $t_2$ waiting for block $b_3$, and so on, train $t_n$ waiting for block $b_1$. Let $T(i, j)$ denote the time at which train $t_i$ enters block $b_j$ in a delay-free realization of $\mathcal{T}$. Since the resource rule is followed, the fact that train $t_i$ is waiting for block $b_{i+1}$ (occupied by train $t_{i+1}$) implies that $b_{i+1}$ is allocated to $t_i$ after $t_{i+1}$. This means that for $i = 1, \cdots n-2$,

$$T(i, i+1) = \text{time } t_i \text{ enters } b_{i+1} > \text{time } t_{i+1} \text{ leaves } b_{i+1} > \text{time } t_{i+1} \text{ enters } b_{i+2} = T(i+1, i+2)$$

So $T(1,2) > T(n-1, n)$. The same argument for trains $t_{n-1}$, $t_n$ and $t_1$, implies that $T(n-1, n) > T(n, 1) > T(1, 2)$, which is a contradiction. ■

We emphasize that the guarantees in this paper are most interesting in delay-free conditions. The two claims in this section essentially say that even in case of delays, all the trains *can* be routed to completion without any collision or deadlocks.

# B Correctness of algorithm TRAIN-PATH

In this section, we argue the correctness and finite termination of algorithm TRAIN-PATH. We first show that the algorithm terminates in finite time. Given an instance of exact train pathing, let $\delta$ denote the minimum time required to cover any block, and $U$ denote any finite upper bound on the minimum travel time of the new train from origin $s$ to destination $d$. Note that $\delta = \frac{d_{min}}{V} > 0$, where $d_{min}$ is the length of the shortest block in the graph. In every expansion, the earliest time of the profile increases by at least $\delta$. So any atomic profile $\Pi$ that has its earliest time, $E_\Pi(V_\Pi) < U$ can have a history of length at most $U/\delta$. Thus the number of such atomic profiles is finite. Since every profile processed by algorithm TRAIN-PATH is of this form, the algorithm has only a finite number of iterations.

We prove the correctness of algorithm TRAIN-PATH by induction. In iteration $k$ of the algorithm, let $t_k$ denote the minimum earliest time in step 2a. For each vertex $p$, let $R_k(p)$ denote the union of all profiles at vertex $p$ in $H \cup F$ at the start of iteration $k$. Then we have the following.

**Lemma 7** *For every $k \geq 1$, for each vertex $p$, $R_k(p) \supseteq R^*(p) \cap \{(t, v) : 0 \leq t \leq t_k\}$.*

**Proof:** We prove this statement by induction on $k$. When $k = 1$, it is trivial since $t_1 = 0$ (expanding the point profile $R_0(s)$). For $k > 1$, if $t_k = t_{k-1}$, the statement follows from induction on $k - 1$. Otherwise $t_k > t_{k-1}$. Suppose (for a contradiction) that the statement were not true; then there is a time-velocity tuple $(t, v) \in R^*(p) \setminus R_k(p)$ at some vertex $p$ such that $t_{k-1} < t \leq t_k$. Among all such tuples $(t, v)$ over all vertices $p$, consider the one with minimum $t$. Since $(t, v) \in R^*(p)$, there is a non-interfering trajectory $\tau$ that reaches $p$ at time $t$ and velocity $v$. Let $p'$ be the vertex preceding $p$ in $\tau$, $I'$ the signal interval under which $\tau$ enters block $(p', p)$, and $(t', v')$ the time-velocity of $\tau$ at $p'$. By the choice of tuple $(t, v)$, we have $(t', v') \in R_k(p')$. But $t' < t \leq t_k$; so the atomic profile $\Pi'$ at $p'$ containing $(t', v')$ must have been expanded before iteration $k$. This means that at that stage $(p, I'(\Pi'))$ must have been added to $H$. Thus tuple $(t, v)$ at vertex $p$ must be in $H \cup F$ at the start of iteration $k$, which is a contradiction. ■

Since TRAIN-PATH always terminates in finite time, Lemma 7 implies the correctness of this algorithm.