A Network-based Representation of Track Layouts in Railway Stations

Vishnu Narayanan[*] Narayan Rangaraj[†]

March 15, 2005

Abstract

A general representation scheme for the track layout at a railway station is proposed. The representation is a graph-based one with a list of conflicting track segments. Any feasible path on the network corresponds to a unique train movement on the actual track configuration. The representation is useful in simulation models for operations planning and capacity estimation at railway terminals. We also show that the representation can be derived from the original track configuration in polynomial time.

1 Introduction

Railways form an important mode of transportation system of many countries including India. The cost of infrastructure and the time it takes for infrastructural changes makes infrastructure planning and capacity estimation an important problem to study for rail planners. Also, delay of a train at a station propagates all along its remaining route and sometimes a small delay at a station tends to cause a very large delay when the train arrives at its destination. It has been shown in Malde (2001) and elsewhere that even simple versions of the Railway Scheduling problem with prohibited time intervals are NP complete.

In this paper we consider the problem of operations management at railway stations. This problem is important for the following reasons. Stations

[*]Department of Mechanical Engineering, Indian Institute of Technology Bombay (Now at the University of California, Berkeley). Email: `vishnun@berkeley.edu`

[†]Associate Professor, Industrial Engineering and Operations Research, Indian Institute of Technology-Bombay. Email: `narayan.rangaraj@iitb.ac.in` Address: IIT Bombay, Mumbai–400 076, INDIA. Tel.: +91-22-2576-7882. **Corresponding Author**

are places where several operations (like reversals, overtaking, maintenance) take place. Hence, stations need to be able to accommodate a large number of trains. Operations like reversals and shunting necessitate occupancy of track and platform resources for significant amounts of time, which makes it necessary to model such operations in detail for the study of the performance of railway stations. In addition, there are many routing options in stations. The choice of a particular route for a train may affect the schedules of other trains.

As a first step in solving this problem, we need a scheme for proper representation of the layout of complicated railway stations in a tractable manner. The requirements of such a representation are: (a) all train movements on the actual track configuration should be representable using the scheme, (b) a movement generated from the scheme should correspond to a unique train movement on the actual track configuration, and (c) simultaneous conflicting movements should be prohibited. We propose a network based model, along with a set of conflicting arcs, for the purpose, so that the trains can be scheduled by finding paths on a time-space graph generated from this network. In the next section we outline some models in literature that deal with routing in stations. In section 3 we present an initial representation that fails for some cases. In section 4 we specify the basic building blocks of our model. Section 5 talks about conflicts between track segments. Our procedure is described in detail in sections 6 and 7.

2 Models in literature

Literature is not readily available for the problem of routing trains in stations, although models for scheduling trains on lines are abundant. The problem of routing trains is posed as a node-packing problem in Zwaneveld et. al.(1997,2001) and some methods to reduce the problem size are outlined. However, the model does not deal with the representation of station layouts although it speaks of checking conflicts for every (*route, train*) pair. In Bourachot (1986), we come across a quadratic integer programming model of the routing problem, where the variables correspond to (*route, train*) pairs. This model also does not deal explicitly with the representation of station layout. A model for the evaluation of the infrastructure at terminals is given in Powell and Wong (2000). The model defines the state of a station as a bit vector and tries to find the maximum cycles in the state transition graph. All operations are assumed to be of unit time duration, and the possibility of multiple routes is not considered in the model. Another integer programming model can be found in Carey (1994). The model presented in Malde (2001) gives a model of station representation which is the same as the initial representation outlined in section 3. This model, which represents the track layout in the form of a graph, is good enough for representing simple train movements. However, when one starts considering movements like reversals, this model has limitations.

We propose a generalised representation scheme which can handle complicated movements like reversals also. This representation scheme uses a graph along with a conflict list. Then, to get the schedule of a train, we find a path on a time-space network constructed using the proposed representation. A simulation model based on finding paths on sequentially constructed time-space graphs, which is used for operations scheduling, timetabling and capacity estimation in railway terminals is found in Vishnu (2003). The model was used to investigate the possibility of desired headways at a terminus. The significance of this issue in practice on Indian Railways is described in Rangaraj and Vishnu (2002).

3 An initial representation

An initial model for the track configuration is the following. For every track segment i on the actual track layout, add two nodes and a directed arc between them. Let the segment i be denoted by the (directed) arc (u_i, v_i). For all track segments j reachable (in the actual track configuration) immediately after leaving segment i, add (directed) arcs (v_i, u_j). All arcs of the form (u_i, v_i) are associated with a real number t_i which denotes the travel time of a train on the track segment i. Arcs of the form (v_i, u_j) are not associated with a travel time, or they have zero travel time (by definition). Every feasible movement on the actual track network is represented by paths on the representation. Now consider the following example. The track layout is shown on the left while the representation is shown on the right side.

Now, consider a train entering segment 2 and then going to segment 5. One can see that it takes the route $2 \rightarrow 3 \rightarrow 4 \rightarrow 5$. The time taken by the train on the actual track configuration is $t_2 + t_3 + 2t_4 + t_5$. This is because the train traverses segment 4 twice: once while entering it from segment 3, and then while leaving it to enter segment 5. In the representation, although the train takes the same path to get to segment 5 from segment 2, the total time taken is only $t_2 + t_3 + t_4 + t_5$. Hence, we can see that the initial model fails here.

In the network in Figure 1, movement of trains from 2 to 3, for example,

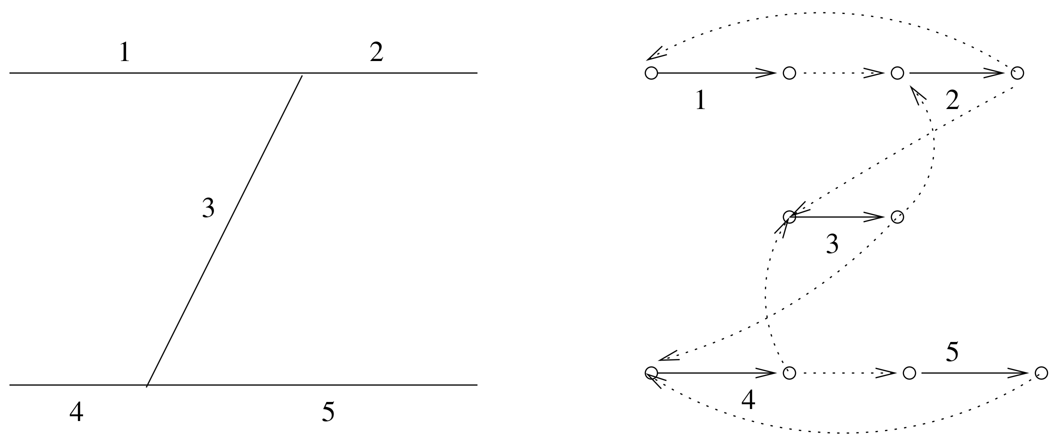


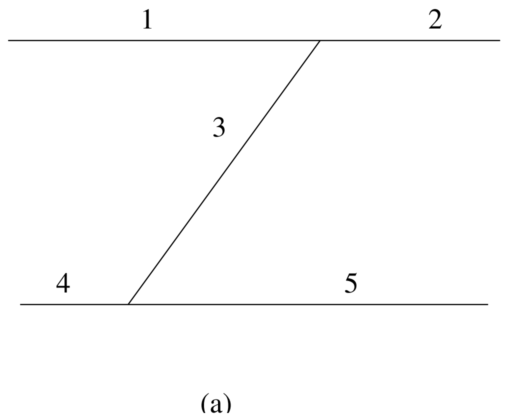
Figure 1: The initial representation for an example track network

involves a crossover, which would be at low speed, and the traversal time on link 2 would be high, as compared to the traversal time on link 2 during the movement $2 \rightarrow 1$. In general, since we determine the route of a train only after a shortest path problem is solved, we cannot determine the traversal times on some of the arcs in this initial model. This is another place where the initial model fails. Keeping this in mind, we now give a representation scheme, such that all feasible movements and only feasible movements are modelled by the representation.

4 Primitives

We define a *tracknet* to be a set of tracks that form a track configuration in some actual railway station. This means that any possible arrangement of tracks that is physically possible is a tracknet. We use a set of *primitives* to conveniently represent large networks in a modular fashion. Each primitive is a small track network of at most two lines and a crossover between them. For each primitive the corresponding representation can be derived and validated and used when representation of a large track network is finally desired. This makes the modelling convenient and error-free. The simplest primitive is shown in figure 2(b). A general primitive that is useful to define is the one that is shown in figure 2(a).

We now present a representation scheme for these primitive tracknets and show that they can be used as building blocks to get track layout representation for cases with more complicated layouts. For example, consider



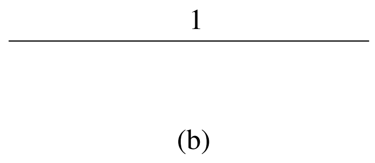


Figure 2: The most common primitive tracknets

the tracknet primitive in figure 2 (a). This is quite a general primitive since it allows traffic in both directions in all segments. This primitive tracknet can be modelled using the following network.

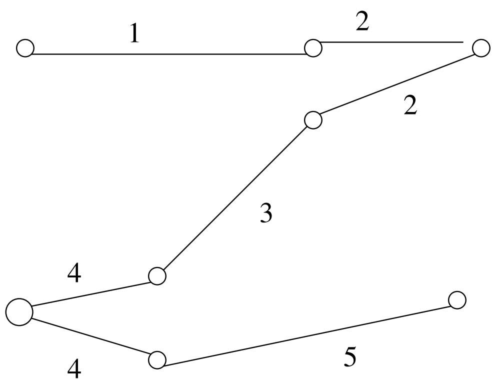


Figure 3: Representation of the primitive in figure 2 (a)

Now, it can be seen that if we seek a path from section 2 to section 5, we get the total traversal time as $t_2 + t_3 + 2t_4 + t_5$, which allows us to capture variable traversal times depending on the path actually used. It is verified by an exhaustive enumeration, that all feasible movements on the tracknet corresponds to paths on the above graph and all paths on the above graph corresponds to some feasible movement on the actual tracknet. For all paths on the actual track configuration, we show a unique corresponding path on the representation. Let us consider all paths that start at an endpoint of the tracknet and end at another. The paths and their corresponding representations are:

- i $1 \rightarrow 2$ corresponds to $a \rightarrow b$
- ii $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ corresponds to $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e$
- iii $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ corresponds to $a \rightarrow b \rightarrow c \rightarrow d \rightarrow e \rightarrow f \rightarrow g$
- iv $2 \rightarrow 3 \rightarrow 4$ is already a part of (ii)
- v $2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ is already a part of (iii)
- vi $4 \rightarrow 5$ is a part of (iii) and corresponds to $f \rightarrow g$
- vii All other paths that were not specified are mirror images of the above ones

Note that the path $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ and other similar paths contain reversals. To model reversals successfully in a graph based framework, we need to use a representation of our kind. It should be clear that the initial model fails in the case of reversals.

Similarly, we can argue conversely that for an arc on the representation of the primitive, we have a unique block corresponding to it. Now consider a path on the representation. We can argue (by presenting an enumeration as above) that it corresponds to a unique feasible train movement on the tracknet. Hence, we have a system that represents all primitive tracknets in a satisfactory manner. Other simpler primitives can be modelled in a similar manner.

We now can find a feasible train schedule (on a primitive) as follows. Construct a timespace graph corresponding to the representation of the primitive. Each node of the timespace graph corresponds to a node of the representation *at a particular time.* If we find a path on this timespace graph (given the train entry and exit points and the entry time), we get a feasible train schedule *if there wer only one train.* After we find a timespace path for a train, if we try to find a feasible path for another train after we delete the arcs on the timespace graph corresponding to the path of the previous train, there still remain possibilities of infeasible schedules. They are outlined in the next section.

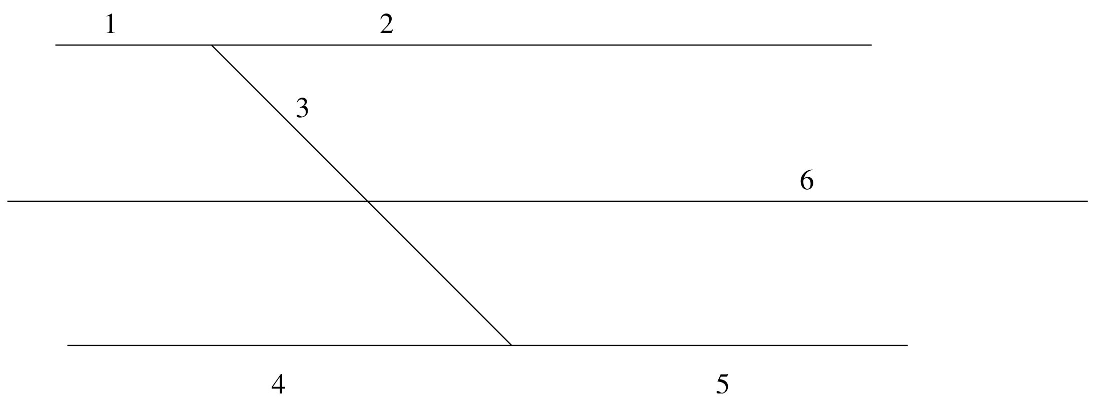


Figure 4: The diamond crossover

5 Conflicts between arcs

Consider the following layout.

The crossing between 3 and 6 cannot be modelled as a graph. Such a crossing is called a diamond crossover (in Indian Railways terminology). The above diamond consists of two primitives, as follows.

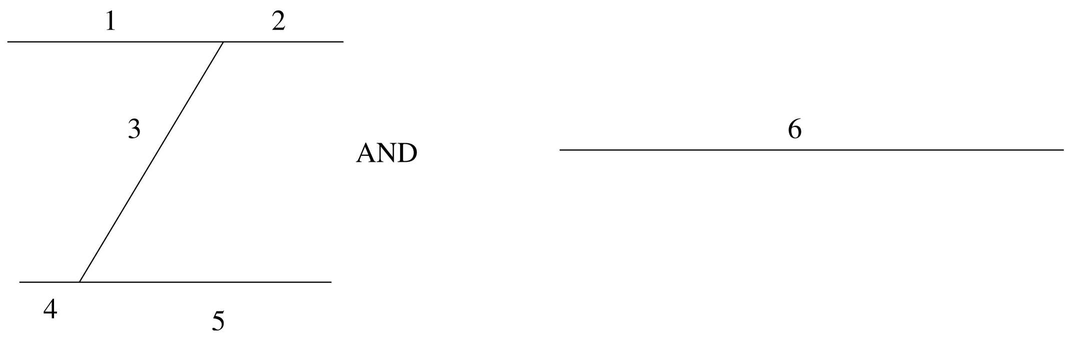


Figure 5: Diamond crossover primitives

The separate representations for the above two primitives can easily be drawn, but when we try to combine the two to find a representation for the "diamond", the model permits simaultaneous trains on segments 3 and 6, which is not physically possible. Suppose we find a path for a train as 1 $\rightarrow$ 3 $\rightarrow$ 5. We then reserve the three segments. But for another train coming after this train, it is quite okay to go along segment 6 *while the first train is moving along 3.* This situation cannot be modelled as a network/graph. For this purpose we introduce the concept of conflict arcs. So, along with our station representation we provide a list of arcs that are in conflicts with each other, so that trains cannot be scheduled simultaneously on them. Hence, a diamond can be represented as follows.

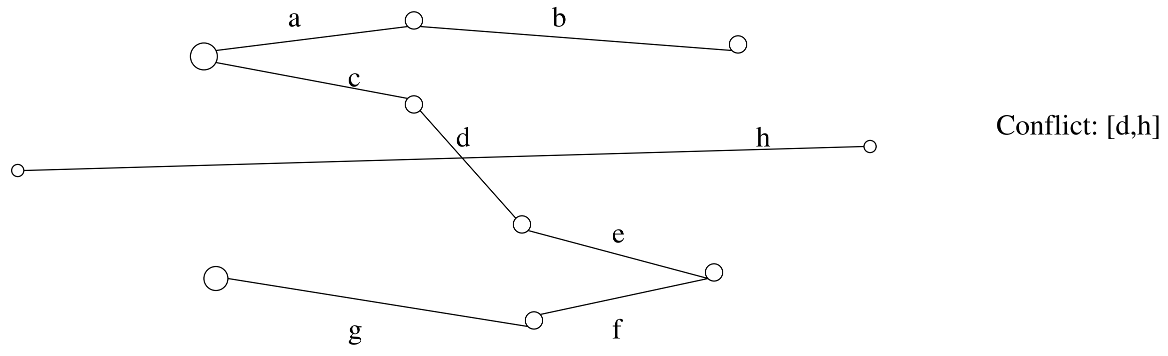


Figure 6: Representation of the diamond crossover

Similar is the case of a scissor crossover, shown in Figure 7:

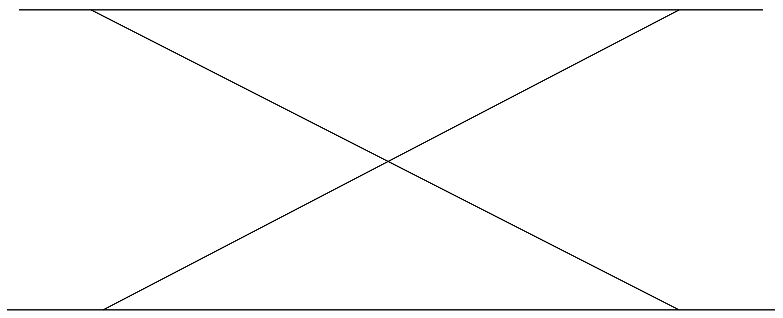


Figure 7: The scissor crossover

Consider the case of two-way tracks, which are used in both directions. When a train starts using it, it remains reserved for that train and no other train can get onto the track. But, if we are not careful in this case, there is a possibility of a train in the *opposite* direction getting on to this track leading to a head-on collision. This can be demonstrated as follows. Suppose a two-way track is as shown in figure 8.

To schedule a train on this track, consider a time-space graph of the two-way track. Since the track is two-way, there will be arcs in both directions($2 \to 1$, $1 \to 2$ etc.) and the corresponding timespace graph is as shown in figure 9. The time instances t_i on the time-space graph are such

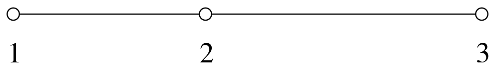


Figure 8: A two-way track

that $t_{i+1} - t_i$ corresponds to the minimum possible traversal time of any block on the track network.

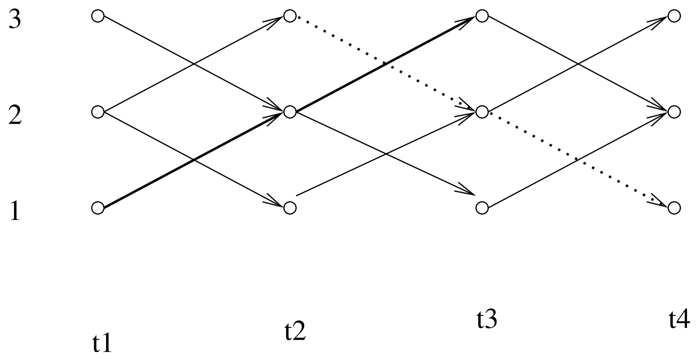


Figure 9: Time-space graph of the layout in figure 8

Suppose a train takes the thick path as shown. Still, there is a possibility of a train taking the thick dotted path. This is physically impossible since the track is the one and the same. If in the timespace graph, we remove the arcs $(2, t_1) \rightarrow (1, t_2)$ and $(3, t_2) \rightarrow (2, t_3)$, such a situation will not arise. This can be easily implemented by introducing another conflict: if (i, j) and (j, i) are arcs in the representation, add a conflict between the two. The addition of such a conflict guarantees a feasible schedule in case of a two-way track.

Similarly, in cases where the same track segment is represented by more than one arc (in case of crossovers), these arcs must be put in conflict with each other so that there will not arise a possibility of two trains on the same block at any time. Now, we are ready with a framework for the station reprensentation. We now outline the rest of the process where we describe how a tracknet is split into primitives and how the representations of the individual primitives are combined to get a representation for the entire station.

6 Breaking a tracknet into primitives

We now know how to represent primitive tracknets in an effective manner. In this section we describe how to decompose a tracknet into primitives. Consider the layout shown in figure 10. If one cuts the layout at the places shown, we get a set of primitives that can be easily represented. The three primitives so generated are denoted by $\{A_i\}$, $\{B_i\}$, and $\{C_i\}$, with a conflict between C_1 and B_3.

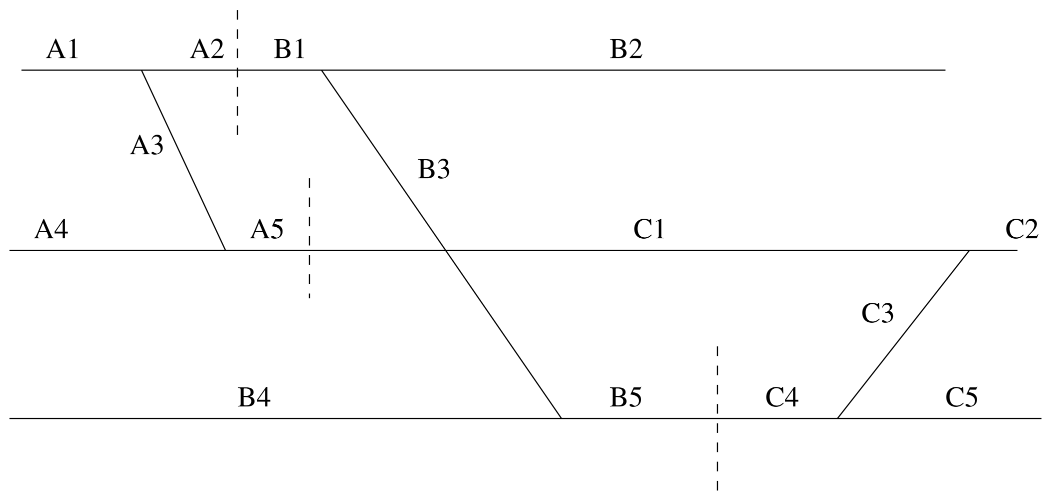


Figure 10: An example of a larger network decomposed using primitives

To get the representation of the entire station, we need to combine the representations of the individual primitive tracknets. Hence, for every primitive, we need to keep track of the place where the different lines were cut to seperate out this tracknet (i.e., the primitive tracknet to which it is to be connected) and the segments (of other primitive tracknets) that are in conflict with some segments of the primitive tracknet under consideration. For this, we propose the following data structure (which we call *primitive* for a primitive tracknet.

(a) *line1* from length l_1 to l_2

(b) *line2* from length l'_1 to l'_2

(c) *crossover* from L_1 (*line1*) to L_2 (*line2*) cuts lines $m_1, m_2, \ldots$; crossovers $c_1, c_2, \ldots$.

The data structure contains the two lines (along with the starting and ending lengths) from which the primitive is cut. If some primitive P_1 doesnot need two lines, then $P_1.\mathit{line2}$ will be set as NULL. The crossover contains two lists, one containing the list of lines that are in conflict with the crossover, and one containing the list of other crossovers that are in conflict with this crossover.

For the purpose of splitting the tracknet into primitives, we assume the following information to be given. For each crossover c, we have the following information.

(a) L_1, the first line of the crossover

(b) L_2, the second line of the crossover

(c) l_1. the coordinate where the crossover c meets line L_1

(d) l_2, the coordinate where the crossover c meets line L_2

(e) the list of all crossovers conflicting with c

(f) the list of all lines conflicting with c

The inputs that we require for the procedure below are the above information along with a list for each line: the list contains the position of wach crossover on the line in increasing order of distance from the start of the line.

We also define two lists for each line: $list1_i$ and $list2_i$. The purpose of these two lists will be made clear in the breaking up procedure. We can now describe the procedure for breaking up a tracknet into primitives as follows.

1. Set $l = 1$.
2. Scan for line l, the list $list1_l$. The list $list1_l$ contains the list of all intervals (x_l, y_l) such that crossovers from lines $l' < l$ to line l (which lie in these intervals have been separated out. Suppose the list contains elements $(x_l^1, y_l^1), \ldots, (x_l^k, y_l^k)$. This means that all points on line l that lie in these intervals have been scanned. From this, we know that the intervals $(0, x_l^1), \ldots, (y_l^i, x_l^{i+1}), \ldots, (x_l^k, B_l)$ are not yet scanned (B_l is the length of the line l). Add these intervals to $list2_l$.
3. Repeat the following till all intervals in $list2_l$ are exhausted

- Scan the interval. When a crossover c is encountered, create a new instance of a primitive P. Cut line l at a point which is at a distance greater than that of the crossover c, but less than that of the next crossover. Set $P.\mathit{line1} = l$ and $P.\mathit{crossover} = c$.
- Find the other end of the crossover c, located at line l', and cut that line on both sides of the end of the crossover c (if any one side of the crossover has no other crossovers, take the entire part of the line as part of the crossover. Set $P.\mathit{line2} = l'$. If l' is cut at points $x_{l'}$ and $y_{l'}$, add $(x_{l'}, y_{l'})$ to $list1_{l'}$.

4. If no crossover is encountered, create a new primitive P with $P.\mathit{line1} = l$, $P.\mathit{line2} =$ NULL, $P.\mathit{crossover} =$ NULL.
5. If the line l is entirely scanned, $l = l + 1$.
6. If $l = n + 1$, where n is the total number of lines, stop. Else go to step 2.

7 Combining representations to get the station representation

From the above procedure for breaking up tracknets into primitives and the data structure for a primitive, we know the following.

i The (network) representation of the primitive along with some conflict arcs;

ii Which tracknet primitive(s) to join this primitive to. We can also find the node (of the representation of the primitive) which is to be connected to a node of the primitive under consideration.

The connection can be done as follows. Let P_1 and P_2 be two tracknet representations (not necessarily of primitives) to be combined. The combination can be done as follows.

1. Suppose node x of P_1 is to be connected to node y of P_2. We define $N_p(x) = \{i | (i, x) \in p \text{ or } (x, i) \in p\}$. So we have two sets $N_{P_1}(x)$ and $N_{P_2}(y)$. We now add a new node z, and make connections such that, $N_{P_1}(x) \subseteq neighbours(z)$, $N_{P_2}(y) \subseteq neighbours(z)$, and $N_{P_1}(x) \cup N_{P_2}(y) = neighbours(z)$. Then delete nodes x and y.
2. If there are any arcs in P_1 and P_2 that conflict each other, add the corresponding arcs to conflict list

8 Computational Complexity

Now we consider the time complexity of the above procedure. We first consider the complexity of the breaking up process and then we estimate the complexity of the combining process.

8.1 The breaking up process

Suppose we have n lines and c crossovers. Suppose all the crossovers are contained in $k \leq n$ lines and the rest $(n-k)$ lines do not have any *endpoint* of any crossover on them (some crossovers could cut them though). Considering the k lines that contain the c crossovers, one can prove that $c \geq k-1$. Also, one can prove that there will be at least two lines (among these k) that are not being *cut* by any crossover. Hence in the worst case each crossover cuts $n-2$ lines and $c-1$ crossovers. For the $n-k$ lines that do not contain any crossovers, we require $Q_1(n-k)$ steps, where Q_1 is a constant. For the remaining k lines, there are c crossovers. If we consider the procedure as a whole, we need to scan a total of $\sum_{u=1}^{k} p_u$ intervals, where p_u is the number of intervals in line u among the k lines that contain crossovers. Also, we need to seperate out c crossovers, which will take $Q_2 c$ steps (Q_2 is a constant). Now, the total number of intervals in the "$list2$" of all lines taken together will be $O(c)$. For the scanning process, we need to search in an ordered list which contains at most c elements, and hence, for scaning the intervals we need $O(c)\log c$ steps. Hence, the total number of steps required is $O(n + c\log c)$ (since k is $O(n)$).

We now consider the number of primitives formed by the breaking up operation. The scenario is the same: n lines, c crossovers having endpoints on $k \leq n$ lines and $n-k$ lines having no endpoints of any crossover. For every crossover, a primitive is formed, and also, there will be $n-k$ tracknet primitives for the "crossover-free" lines. Hence the number of primitives formed equals $c+n-k$. We require this number to evaluate the performance of the "recombination procedure" which is specified in the next section.

8.2 The combining process

Now, suppose we join two tracknets P_1 and P_2 using the above procedure. Suppose the node x_1 of P_1 was to be combined with node x_2 of P_2, it means that to get into tracknet P_2 from P_1, we need to follow the path $x_1 \rightarrow x_2$. Once we reach x_2, we know that any path in P_2 gives a feasible train path. Similar is the case of P_1. Hence, we can conclude that if we find a path in the combination of P_1 and P_2, we get a feasible train path. Hence, when all primitives are combined appropriately, we get a feasible representation for the entire station.

We now look at the complexity of the combination process. From the discussion in the previous section, the maximum number of conflicts for a certain crossover is given by $(c-1)+(n-2)$. Given a primitive P_o, the maximum number of conflicts it has with another primitive P_1 is 3. So let S_o represent a primitive. Define $S_1, S_2, \ldots, S_{c+n-k-1}$ to be the sequence of tracknet representations formed by combining at the i^{th} step one primitive with S_{i-1}. Let us look at the number of operations needed to go from S_{i-1} to S_i. For the connection purpose, the nodes x and y will have two neigbours at most. Hence step 1 will take at most four steps. For step 2, the number of conflicts between the primitive chosen at step 2 with S_{i-1} is at most $3i$. Hence the total number of operations involved are $4(c+n-k-1)+\sum_{i=1}^{n+c-k-1} 3i$, which is $O\{(n+c)^2\}$. Hence, from the previous section, the entire process of generating the tracknet representation is $O\{(n+c)^2\}$.

9 Conclusions

We have presented a network-based representation scheme for general track layouts. The representation scheme is such that it can represent all train movements in an unambiguous way and paths on the network give a unique train movement. We have also shown that the representation can be derived in polynomial time from the track layout information. Details of simulation experiments using the station representation proposed in this paper are presented in Vishnu (2003). One of the interesting questions answered using this representation was to evaluate alternate layouts to see whether a repetitive stream of traffic could be handled at a commuter terminus (Mumbai CST station). The analysis established a trade-off between an achievable headway between trains and the platform occupancy time (for discharge/boarding of passengers and crew change, among other things). Further studies on reliable schedules were also conducted, which considered different arrival patterns of trains. In summary, the model is flexible enough to consider details of routing, conflicts and sequence-dependant resource usage, and therefore permits a nearly-exact evaluation of infrastructure possibilities at railway terminals.

Acknowledgements

The authors would like to thank Hrishikesh Karmarkar (Comp. Sci., IIT Bombay), Abhiram Ranade (Comp. Sci., IIT Bombay) and N. Hemachandra (IEOR, IIT Bombay) for discussions and comments. The authors are also thankful to M. Sahai (Central Railways, Mumbai) for providing them with information and data regarding yard operations and terminal movements.

References

Zwaneveld, P. J., Kroon, L. G. and van Hoesel S. P. M. (2001), Routing trains through a railway station based on a node packing model, *EJOR* **128** pp. 14–33.

Zwaneveld, P. J., Kroon, L. G. and Romeijn, H. E. (1997) Routing trains through railway stations: complexity issues, *EJOR* **98** pp. 485–498.

Bourachot, J. (1986) Computer aided planning of traffic in large railway stations by means of the AFAIG model, *Rail International* pp. 2–18.

Powell, S. and Wong, H. Y. (2000) A deterministic approach to evaluating transport infrastructure at a terminus, *Transportation Research-A* **34A** pp. 287–302.

Carey, M.(1994) A model and strategy for train pathing with choice of lines, platforms and routes, *Transportation Research-B* **28B** pp. 333–353.

Malde, S. B.(2001) Railway timetable scheduling. Bachelors' thesis, Department of Computer Science and Engineering, Indian Institute of Technology Bombay.

Vishnu B. N.(2003) A simulation model for operations management and capacity evaulation at railway stations. Masters' Thesis, Department of Mechanical Engineering, Indian Institute of Technology-Bombay.

Rangaraj, N. and Vishnu B. N.(2002) Node capacity and terminal management on Indian Railways. *Invited paper at Vision 2025, Railway Staff College, Vadodara.*