This is to certify that the

thesis entitled

Optimal Routing

on a Railway Network

presented by

Hyun-Duk John

has been accepted towards fulfillment
of the requirements for

Master's degree in Computer Science

Major professor

Date 4/10/91

OPTIMAL ROUTING
ON A RAILWAY NETWORK

BY

Hyun-Duk John

A THESIS

Submitted to

Michigan State University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

Department of Computer Science

1991

ABSTRACT

OPTIMAL ROUTING
ON A RAILWAY NETWORK

By

Hyun-Duk John

Finding an optimal route for a train on a railway network with the given time constraints is a priority based optimal routing problem. This thesis states the problem for a local railway network and proposes two methods as its solutions. Both methods are based on A*/Branch-and-Bound algorithm. The first algorithm "Time-Constraint-Propagation" propagates whole time constraint information to each node on a search graph. The second algorithm "Incremental-Branch" describes a way of minimizing the number of nodes on a search graph using backtracking. Even though the complexity of the both algorithms are same, the second algorithm usually shows better performance than the first algorithm according to the experimental results. These algorithms can be used as techniques for finding a local optimum in a total railway network problem. These algorithms can also be applied to other applications with high branching factor and/or time constraints. A solution to a simple shortest problem on a railway is described, which is also used as an estimation function in our two algorithms.

To Daewoo Engineering Company

ACKNOWLEDGMENTS

I wish to express my gratitude and appreciation to my major professor, Dr. Moon Jung Chung, for his guidance and encouragement during preparation of my thesis.

I would also like to thank to Dr. Carl Page, Dr. Wen-Jing Hsu, and Dr. Bill Punch for participating as members of my graduate committee.

My sincerest thanks goes to the staffs of Daewoo Engineering Company for their support and love. Finally, I must express my heartful thanks to my deceased mother for her sincere prayer and endless love for my life.

Contents

List of Figures

Chapter 1

Introduction

There has been a lot of work on railway network problems: network modeling, scheduling on railway networks, delay analysis, etc. But most of the work deals with the problems of *wide area railway networks* [Ass80, Che72, HP74].

Nowadays, as the industrial automation is becoming more complicated and interrelated between unit shops or plants, the need for the automation of the transportation systems is increasing significantly. Therefore, the control automation of those systems becomes crucial for the successful implementation of the industry-wide CIM (Computer Integrated Manufacturing) systems.

Railway is an important transportation system in large-scale industries, and the railway systems usually control the major production flows. But it turns out to be much more difficult to be automated than other transportation systems such as conveyor belt systems or unmanned vehicle systems. The major difficulties in the software design for railway systems arise from the nondeterministic characteristics of transportation requests.

This thesis deals with the optimal routing problem in a local railway network which is a subproblem of the total automation problem. A railway network in a steelmill company

can be a good example. Railway is one of the major means of transportation in a steelmill company. It carries raw material into the steelmill and also carries final products (e.g. hot coil, cold coil, etc.) out of the steelmill to the clients. Furthermore, most of the modern steelmill companies rely on the railway to move the molten iron from the blast furnace to the steel making units. Certain kinds of intermediate products (e.g. casted iron) are also moved by the railway.

The non-determinism of the transportation problem in a steelmill company can be illustrated as follows : We can estimate the average production rate of molten iron in a blast furnace, but it's very hard to estimate the production rate at a moment. So it is also difficult to decide which vehicle is chosen and when is the vehicle ready to move. Sometimes, the destination of a vehicle cannot be decided in advance. For example, it is always recommended to have a vehicle cleaned after every transit of molten iron. But if there are not enough vehicles in the blast furnace area, the need arises to send the vehicle directly to the blast furnace without cleaning.

To understand the position of our routing problem in the total automation of a railway control, we can divide the control function into three subfunctions.

1. Real hardware control and monitor : This subfunction deals with signalling, point switching, controlling powered vehicles, monitoring each field sensor, etc.

2. Scheduling : A railway control system receives the *transportation requests* from other plants. Based on its currently available vehicle information, the control system generates *moverequests*. Each moverequest specifies a powered vehicle id, operation time, type of the work, and the destination.

3. Routing : The route for each moverequest is found.

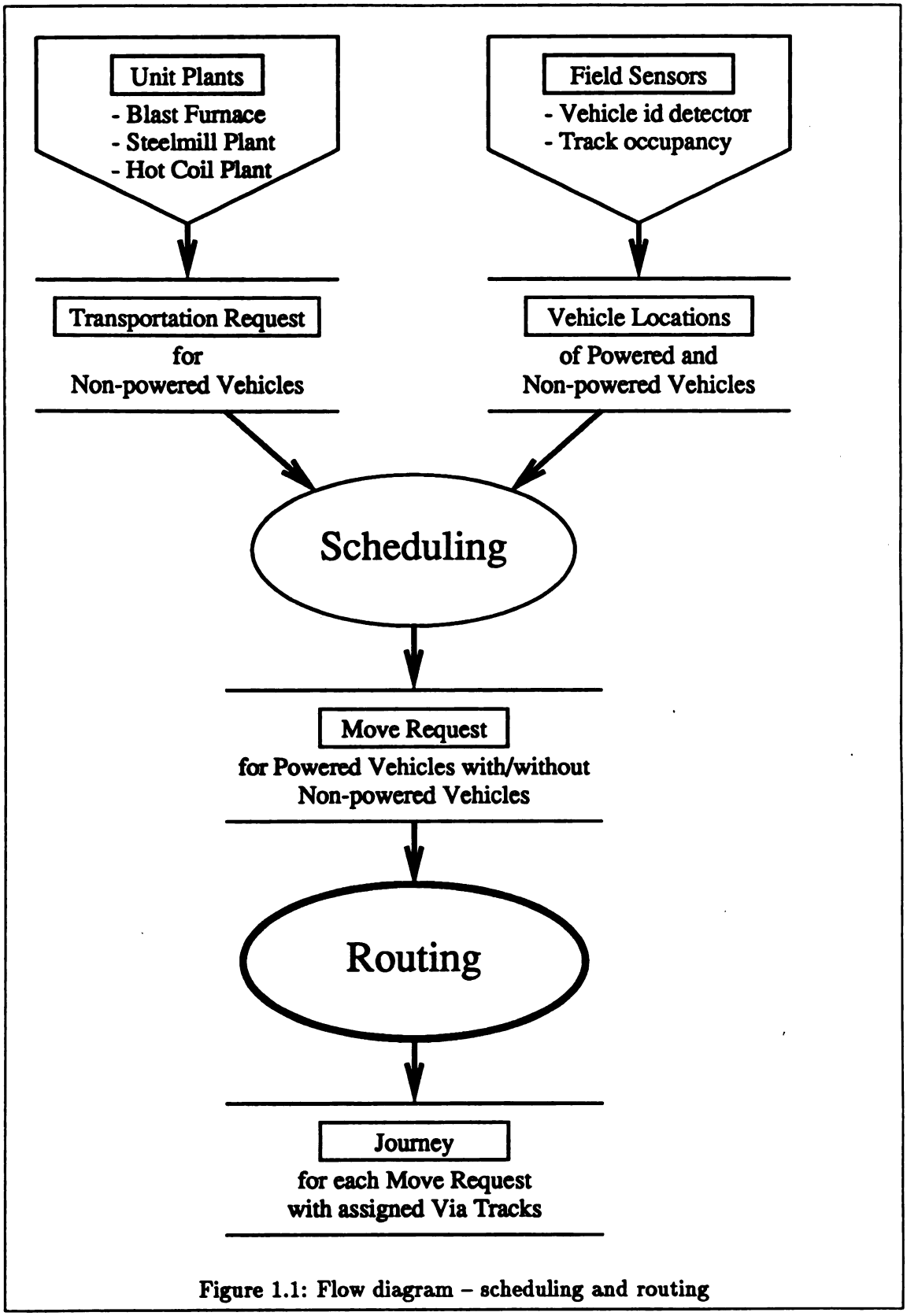


Figure 1.1: Flow diagram – scheduling and routing

Figure 1.1 shows an example for scheduling and routing functions in a steelmill company. Each unit plant sends a transportation request to the railway control system. The scheduling function invokes moverequests using the vehicle location information for the transportation requests. Then the routing function computes the optimal route for each moverequest. We call the moverequest with an assigned route a *journey*. A formal definition of the *journey* will be given in Chapter 2.

The following simple scenario will help to understand the required steps in scheduling and routing functions. Figure 1.2 shows a part of the railway network in a steelmill company. A powered vehicle, locomotive #A, is located on track *tr-5* and hot coils are being loaded onto the wagons in the hot coil plant on track *tr-1*.

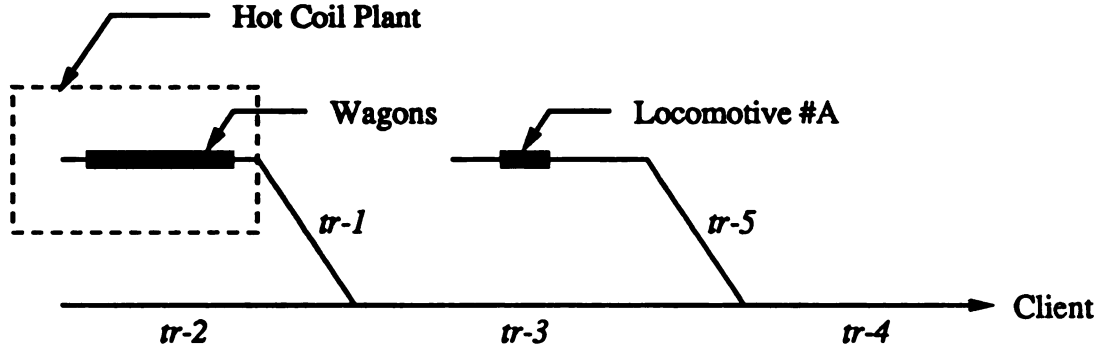


Figure 1.2: An example – scheduling and routing

1. **Transportation Request** from the Hot Coil Plant.
   "The wagons are fully loaded with hot coils and are ready to depart. Please take the wagons out and bring them to the client."
2. **Move Requests** generated.
   (a) "Move locomotive #A to the Hot Coil Plant."
   (b) "Let locomotive #A pull the wagons and bring them to the client."
3. **Journeys** : Move Requests with assigned track sequences.

(a) Move-request : tr-5 → tr-4 → tr-3 → tr-1.
(b) Move-request : tr-1 → tr-3 → tr-4 → Client.

A *routing problem* is to find journeys for the given moverequests on a railway network. The routing problem can be modeled in several ways and those problems are described in Chapter 2. In this thesis, we will concentrate on the optimal routing problem which asks an optimal journey for a moverequest with a set of already scheduled journeys of other trains.

The remaining part of this thesis is organized as follows. Chapter 2 describes the basic terminologies we need to define the problems and to describe the solutions. It also deals with some modelling work for the mathematical formulations. The simple shortest path problem on a railway network is solved in Chapter 3. The problem is not only interesting in itself but can also be applied as an estimation function for our proposed algorithms described in the next chapter. In the beginning of Chapter 4, the characteristics of a routing problem on a railway network are analyzed and A* algorithm is introduced which will be used as the basis of our proposed approaches. Later in the chapter, our two approaches, *Time-Constraint-Propagation* and *Incremental-Branch*, are described with a sample problem and the solutions. The first algorithm propagates the whole time constraints to each point on a railway network. The second algorithm minimizes the number of nodes generated with a single time gap for each node in the search graph. Chapter 5 shows a performance analysis of the two algorithms based on experimental results. The conclusion part, Chapter 6, summarizes the research and describes some topics for further studies.

Chapter 2

Problems on a Railway Network

In this chapter, we describe several routing problems on a railway network. First section describes basic terminologies and the optimal routing problem for which this thesis will propose two techniques to solve the problem. Second section describes other problems and related works.

2.1 Optimal Routing Problem

A *railway network R(V,E)* is a structure which consists of a set of finite number of *points* $V = \{v_1, v_2, \ldots, v_m\}$ and a set of finite number of *tracks* $E = \{e_1, e_2, \ldots, e_n\}$; each track e is incident to the elements of an unordered pair of points $\{u, v\}$, which are necessarily distinct[1], and we write $e = (u, v)$ or $e = (v, u)$. Each track e has its own length of positive value $l(e)$. For example, consider the railway network given in Figure 2.1.

Here $V = \{v_1, v_2, \ldots, v_8\}, E = \{e_1, e_2, \ldots, e_8\}$. The track e_7 is incident to v_6 and v_7 which are called its *endpoints*. The tracks e_2 and e_3 have the same endpoints and are

[1]For some railway networks, we can see a self-loop. But for the simplicity of the analysis, we do not consider the existence of self-loops.

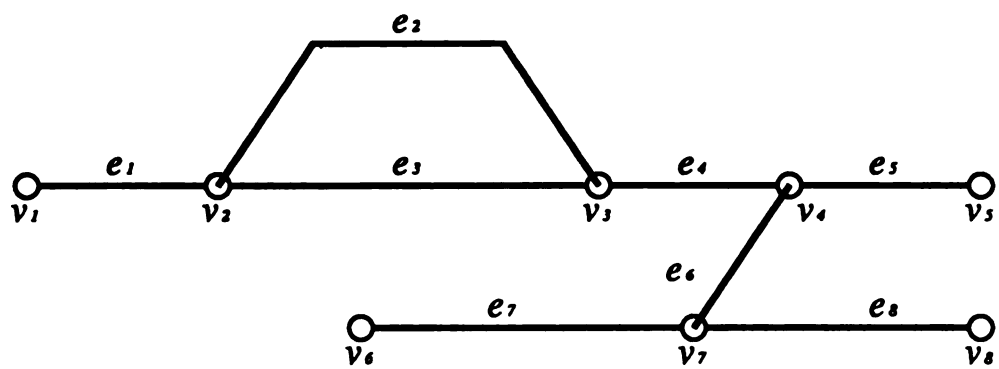


Figure 2.1: A sample railway network

therefore called *parallel* tracks. The *degree* of a point v, $d(v)$, is the number of times v is used as endpoints of the tracks. In a railway network, the degree of a point can be at most 3 and at least 1. In our example, $d(v_2) = 3, d(v_4) = 3$, and $d(v_6) = 1$. A point of degree 1 is called a *terminal*, so v_1, v_5, v_6 and v_8 are terminals. The length of a train a is denoted as $l(a)$ and the *location* of a train is an ordered triple (e, v, d); the train a is located on track e, and its middle position is d apart from v (See Figure 2.2). Note that the location (e, v, d) in Figure 2.2 can also be represented as $(e, w, l(e) - d)$.

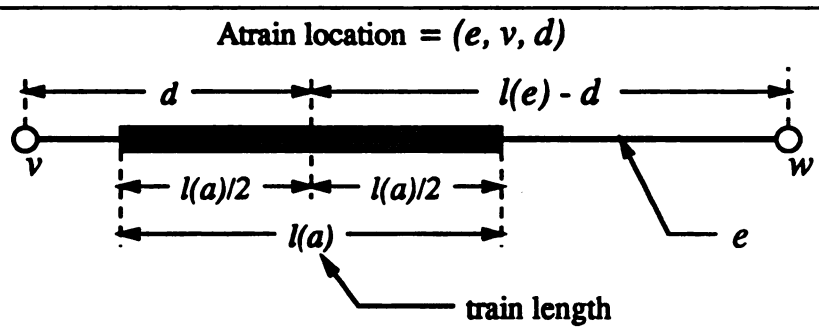


Figure 2.2: Location of a train

A *path* from a location $s = (e_s, v_s, d_s)$ to another location $d = (e_d, v_d, d_d)$ is a sequence $P = (s, v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v_i, d)$, such that, for $1 \leq j < i$, e_j is incident to v_j and v_{j+1}, where

1. either $v_j = v_{j+1}$ or $e_j = (v_j, v_{j+1})$,

2. v_1 is one endpoint of e_s, and

3. v_i is one endpoint of e_d.

We may sometimes denote $P = (e_s, v_1, e_1, v_2, e_2, \ldots, e_{i-1}, v_i, e_d)$ when the exact location of a train is of little importance. Any subsequence (e_j, v_j, e_{j+1}) of a path P is called a *step*, and we denote it $e_j \rightarrow v_j \rightarrow e_{j+1}$.

Let $P = (e_1, v_1, e_2, v_2, \ldots, e_{i-1}, v_{i-1}, e_i)$, then for any $1 \leq j < i$, e_j and e_{j+1} are two distinct tracks incident to v_j. So a point of degree 1 cannot appear on a path and a point of degree of at least 2 will appear on a path. If a point is of degree 3, certain steps are prohibited between the two tracks among the three incident tracks. In our example in Figure 2.1, consider the possible steps around the point v_2. The two steps like $e_2 \rightarrow v_2 \rightarrow e_3$ and $e_3 \rightarrow v_2 \rightarrow e_2$ are prohibited while all the other steps, i.e., $e_1 \rightarrow v_2 \rightarrow e_2$, $e_2 \rightarrow v_2 \rightarrow e_1$, $e_1 \rightarrow v_2 \rightarrow e_3$, and $e_3 \rightarrow v_2 \rightarrow e_1$ are allowed.

Any prohibited steps are called *illegal* steps, while the others are called *legal* steps or simply *steps*. If any step in a path P is legal, then the path is called a *legal* path or simply a *path*, else the path is called *illegal*. A step or a path will always be legal unless otherwise specified.

A *moverequest* is an ordered quadruple (a, s, d, t_s); a train a, an originating location s, a destination d, and the earliest permissible departure time t_s from s.

Let $P = (e_1, v_1, e_2, v_2, \ldots, e_{i-1}, v_{i-1}, e_i)$, then we can associate a *timetable* (sequence of time intervals) $T(P) = ([t_1^1, t_1^2], [t_2^1, t_2^2], \ldots, [t_i^1, t_i^2])$ to P, such that t_j^1 is the arrival time to the track e_j for all j, $1 < j \leq i$, and t_j^2 is the departure time from track e_j for all j, $1 \leq j < i$. t_1^1 and t_i^2 are the starting time from the starting location and the arrival time to the destination location, respectively. A *journey* of a train a, $J(a)$ is an ordered pair of a path P and a timetable of the path $T(P)$, $(P, T(P))$; a specification of its path on the

network and its time schedule.

Consider a railway network involving train movements, where a train can move back and forth freely. Assume that

1. the train speed is invariant when it moves, say

$$\text{train speed} = 1 \text{ unit length} / 1 \text{ unit time}$$

We will not use dimensional units in describing time, length, speed, etc.

2. A train can stop anywhere on a track. But it cannot stop when its body is occupying a point.

3. When a moving train reverses its direction, it takes a definite time interval, and that time interval is called as *stopping penalty* or simply *penalty*. Since changing direction works based on a sequence of two operations, stop and restart in reverse direction, a train cannot change its moving direction while it is moving between two adjacent tracks, i.e., when its body is occupying a point.

4. Two or more trains cannot be on the same track at any moment.

5. Let $E = \{e_1, e_2, \ldots\}$ be the tracks in a railway network, then for any train a and track e_i, $l(a) < l(e_i)$. And let $q(a) = (e, v, d)$ be either the start location or the destination location, then $l(a)/2 < d$ and $l(a)/2 < l(e) - d$.

If a train a enters track e at time t_1 from point v_1 and leaves the track e at time t_2 through point v_2, then the track is *closed* for other train movements during the period $[t_1, t_2]$. From the above assumptions we have either

$$t_2 - t_1 \geq \quad 2 * l(a) + \textit{penalty} \quad \text{if } v_1 = v_2 \text{ or}$$

$$t_2 - t_1 \geq \qquad l(a) + l(e) \qquad \text{if } v_1 \neq v_2 \tag{2.1}$$

Now, assume that a number of trains are moving in the network according to well defined journeys. For each track, this situation results in a set of time intervals during which a train is not allowed to enter the track.

Let $R = (V, E)$ be a railway network, and $e = (u, v)$ be a track in E. It is known that if a planned journey includes traveling from u to v (or v to u, u to u, v to v) via track e, then the traveling time in the track should be within one of the time intervals $[\alpha_m^e, \beta_m^e]$, $m = 1, 2, \ldots,$ $M^e$and M^e is a finite number. Let $U_e = \{[\alpha_m^e, \beta_m^e]; m = 1, 2, \ldots, M^e\}$ be a set of permissible traveling times on track e. Let the traveling time of a train on track e be $[t_1, t_2]$, then $[t_1, t_2]$ should be in a permissible time slice such that $t_1 \geq \alpha_m^e$ and $t_2 \leq \beta_m^e$ for some m, $1 \leq m \leq M^e$. We denote it as $[t_1, t_2] \in U_e$.

The journey of a movement in a non-permissible time, $[t_1, t_2] \notin U_e$, on a physical railway network will violate the basic assumptions, since there will be a time period in which two trains exist in the same track e. We note that the move in a non-permissible time is prohibited since it would be either infeasible (trains will be blocked, if they move in the opposite directions) or unfavorable for safety considerations. We also note that if a new train is now scheduled to move on track e, then the permissible traveling times U_e for track e should be modified accordingly, such that

$$\text{new } U_e = U_e - [t_1, t_2].$$

This updating of the restrictions will not be necessary in our problem when only one additional train is scheduled. However, it will be unavoidable when two or more additional trains should be sent for journeys on a network [Che72].

We consider a journey of a train a, $J(P,T(P))$, where $P = (e_1, v_1, e_2, \ldots, e_{n-1}, v_{n-1}, e_n)$, and $T = \{[t_1^1, t_1,^2], [t_2^1, t_2^2], \ldots, [t_n^1, t_n^2]\}$. We assumed that a train cannot stop on the border between two tracks. So the equality relation

$$t_{i+1}^1 = t_i^2 - l(a) \tag{2.2}$$

holds for all $1 \leq i \leq n-1$. Our concern here is to minimize t_n^2, the arrival time to the destination. We note here that t_1^1 is the earliest time the train can start a journey and t_n^2 is the arrival time of the train at its destination. But e_1 will be closed during $[0, t_1^2]$ (not $[t_1^1, t_1^2]$), and e_n will be closed during $[t_n^1, \infty]$ (not $[t_n^1, t_n^2]$), since a train is assumed to be on the originating track at the beginning of the journey and to occupy the destination track forever after its arrival to the track.

Any solution to the problem of a moverequest (a, s, d, t_s) sending a train from the origin s to the destination d is represented as a journey $(P,T(P))$, namely, the specification of a path from s to d and a time table for traveling periods along the path. A journey is feasible if and only if it does not violate the restrictions placed on movements along the tracks. In other words, a journey $(P,T(P))$ is feasible if and only if

1. For the first track e_1 and last track e_n in P, $[0, t_1^2] \in U_{e_1}$ and $[t_n^1, \infty] \in U_{e_n}$, respectively, and for every other track e_i in P, $[t_i^1, t_i^2] \in U_{e_i}$, and

2. Equation 2.1 holds for all $i (2 \leq i \leq n-1)$. And for the first track e_1 with the originating location $s = (e_1, v, d_s)$,

$$t_1^2 - t_1^1 \geq \quad l(a)/2 + d_s \qquad \text{if } v_1 = v \text{ or}$$

$$t_1^2 - t_1^1 \geq \quad l(a)/2 + l(e_1) - d_s \quad \text{if } v_1 \neq v \tag{2.3}$$

3. Equation 2.2 holds for all $i (1 \leq i \leq n-1)$.

Let $\mathbf{P}$ be the set of paths from the origin s to the destination d. For every $P \in \mathbf{P}$, there are in general an infinite number of feasible timetables $T(P)$. Let $\mathbf{T}(P)$ be the set of all such feasible time schedules for a given path P. Note that an empty $\mathbf{T}(P)$ is possible for some $P \in \mathbf{P}$. The set of all feasible solutions is the set of pairs, $(P, T(P))$, such that $P \in \mathbf{P}$ and $T(P) \in \mathbf{T}(P)$. An optimal solution is one which minimizes t_d^2 over a set of feasible solutions. The problem can be stated as below.

Problem 1 Given a railway network $R = (V, E)$, a set of permissible traveling time intervals U_e for each track $e \in E$, and a moverequest (a, s, d, t_s), find an optimal journey $(P, T(P))$.

Note that under some conditions, no feasible solution may exist, e.g., if a train is initially located on track e and all its neighbor tracks are found to be blocked during the time period $[0, \infty]$, then no feasible solution exists for any requested move. If there is a feasible solution, the existence of an optimal solution is guaranteed since the non-negativity of the track lengths gives the lower bound of the earliest arrival time.

2.2 Genaral Problems

Suppose we deal with a routing problem in which we would like to find a set of journeys for the set of moverequests in such a way that the sum of the time taken by each train is

minimized. The problem can be stated below using the terminologies defined in the previous section.

Problem General-Routing Given a railway network $R = (V, E)$ and a set of moverequests, find a set of journey which minimizes the total time of the train movements.

In dealing with multiple moverequest problems, a simple and natural question arises. Is there a solution at all ? The problem can be stated below.

Problem Existence-of-a solution Given a railway network $R = (V, E)$ and a set of moverequests, does there exist a set of journey which satisfy the given moverequests ?

We can generalize the routing problem as shown in the next section.

2.3 Traveling Stars Problem

A network routing problem can be simplified and/or modified to more generalized problems. In this section, one of the generalized version of the network routing problem named "Traveling Stars" is described. The famous problem "8-puzzle" can be viewed as an instance of the traveling stars problem. Following subsections describe the problem as two decision problems and an example is given for understanding the problem.

Problem Description

- Given an undirected graph $G = (V, E)$ and m stars, where $m \leq |V|$).
- A star occupies a vertex and can be moved to an adjacent vertex one by one. (Only one star can be moved at a time)
- No two stars can share a vertex at any moment.
- Initially, each star i is located in its starting vertex s_i, and is destined to its terminal vertex t_i.

Questions :

1. Can all the stars be moved from their starting vertices to terminal vertices ?

2. Is there a way of moving stars from their starting vertices to terminal vertices in less than or equal to k steps ?

   (A movement of a single star from a vertex to its adjacent vertex is called a step.)

A Formal Expression of the Problem

For more formal approach, we define some terminologies.

- A *state* is a sequence of vertices $(v_1, v_2, \ldots, v_m)$ such that $\forall i \neq j, v_i \neq v_j$.

- A *transition* is an ordered pair of states (W, X) where $W = (w_1, w_2, \ldots, w_m)$ and $X = (x_1, x_2, \ldots, x_m)$ in which there exists one and only one i, such that $\forall j \neq i, w_j = x_j$ and $(w_i, x_i) \in E$.

- A *move* of size k is a sequence of states $(W_0, W_1, W_2, \ldots, W_k)$ where every pair (W_i, W_{i+1}) is a transition.

  We denote the move as $W_0 \xrightarrow{*} W_k$.

Problem Traveling-Stars

- **Instance :**
  - Undirected graph $G = (V, E)$.
  - Two states S and T of size m.
  - An integer k.
- **Question :**
  1. Is there a move $S \xrightarrow{*} T$?
  2. Is there a move $S \xrightarrow{*} T$ of size $\leq k$?

Example

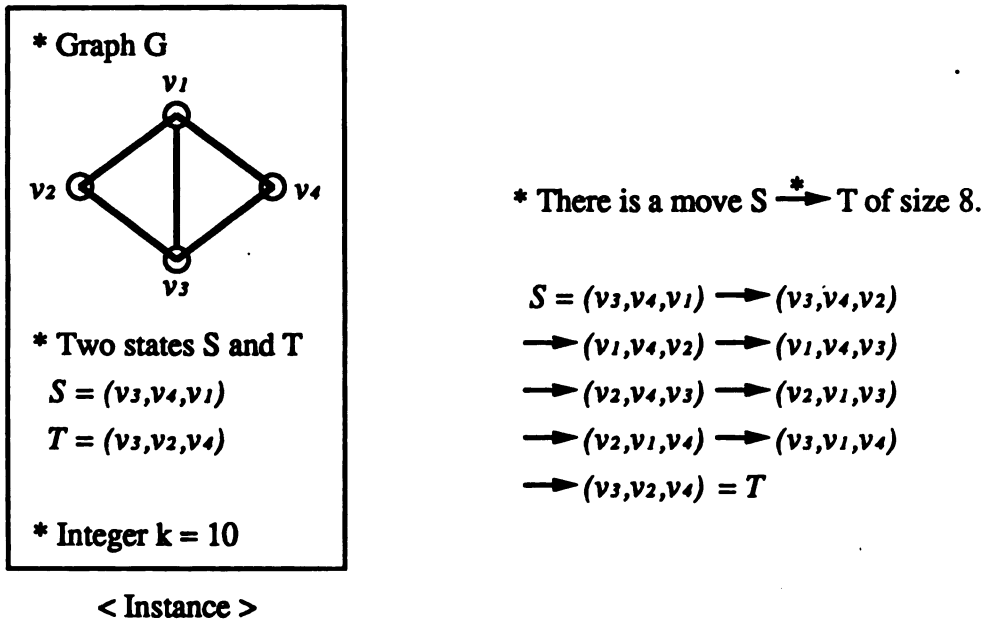


Figure 2.3: An instance and sample solution

Chapter 3

Shortest Path Problem on a Railway Network

Before trying to solve problem 1 given in Chapter 2, let's consider a simpler case: How to find an optimal solution, if there are no previous journeys scheduled for other trains? This problem is, in other words, how to find a shortest path in a railway network. Using the terminologies given in Chapter 2 the shortest path problem can be stated as shown below.

Problem 2 Given a railway network $R = (V, E)$, where $\forall e \in E$, the set of permissible traveling times $U_e = \{[0, \infty]\}$, find an optimal solution $(P, T(P))$ for a moverequest (a, s, d, t_s).

There are various kinds of shortest path problems and many algorithms have been suggested for solving those problems. A good survey of those algorithms are given in [Pie75]. As we shall see later in this chapter, the shortest path problem on a railway network can be viewed as a shortest path problem with turn penalties. Intersection between a freeway and a surface street using entrance and exit ramps is an example of the network with turn penalties.

First, we will review shortest path algorithms with/without turn penalties, then we will analyze the railway network as a kind of a network with turn penalties and describe an algorithm for the shortest path problem in a railway network.

3.1 Shortest Path Problem with Turn Penalties

```
Algorithm Single_Source_Shortest_paths (G, v);
Input : G = (V, E) (a weighted directed graph), and v (the source vertex).
Output : for each vertex w, w.SP is the length of the shortest path from v to w.
  { all lengths are assumed to be nonnegative. }

begin
  for all vertices w do
    w.mark := false;
    w.SP := ∞;
  v.SP := 0;
  while there exists an unmarked vertex do
    let w be an unmarked vertex such that w.SP is minimal;
    w.mark := true;
      for all edges (w, z) such that z is unmarked do
        if w.SP + length(w, z) < z.SP then
          z.SP := w.SP + length(w, z)
end
```

With reference to a network with turn penalties, we will call a network *simple*, if there are no turn penalties in the network. There are several famous shortest path algorithms for simple networks (e.g. Moore's algorithm, Dijkstra's algorithm, Ford's algorithm, etc.). Our shortest path problem is a single-source single-destination problem with weighted tracks on a sparse[1] network. So Dijkstra's algorithm will be the good starting point of our discussion. Dijkstra's algorithm finds the shortest path lengths from a single point to all other remaining

[1] A network $R(V, E)$ is said to be sparse, if $|E| = O(|V|)$. In a railway network, there are at most 3 tracks incident to a point. So

$$2|E| = \sum_{v \in V} d(v) \leq 3|v|$$

$\Rightarrow |E| = O(|V|)$.

points in a network, and hence called as *Single_Source_Shortest_Paths.*

The algorithm is exhibited in [Man89, pp. 206–207].

Shortest path problem with turn penalties and/or movement prohibitions is an issue in general transportation networks and several approaches have been proposed. (e.g. [KP69], [Eas85]) Before describing our approach, it is useful to discuss some of the methods proposed in general transportation networks.

Here we consider three methods described in [Eas85]: modification of simple shortest path algorithms, modification of the object networks, and finally multi-labeling method.

Some early transportation researchers proposed a method to account for the movement prohibitions by modifying Dijkstra's algorithm. (See [Eas85]) The method is almost the same as Dijkstra's algorithm except that it does not propagate to the next point if the movement is prohibited to avoid the sequence of tracks containing the prohibited movements. Unfortunately, this method does not assure to find a shortest path.

Another method called *network modification* was proposed by some transportation researchers and was found to have some weak points to be used directly in real transportation networks. (See [Eas85])

Multi-labeling method suggested by Easa in [Eas85] is a successful and at the same time an efficient algorithm to solve the shortest path problem with turn penalties. In a sense, the multi-labeling method can be viewed as a generalization of the network modification method even though the multi-labeling method does not explicitly show the modified network. It is almost impossible to draw a modified network for a general traffic network using the multi-labeling method, since the modified network would look too complicated.

A railway network connection is much simpler than that of a general traffic network, and a network modification method for railway network is presented in Section 3.2. We can

easily show that the multi-labeling method will result in exactly the same modification.

3.2 Shortest Path on a Railway Network

To solve the shortest problem (Problem 2) defined in the beginning of this chapter, this section gives an algorithm based on the network modification scheme introduced in the previors section. The network modification scheme converts a railway network into a simple directed network, with which we can apply the plain shortest path algorithm directly.

3.2.1 Network Modification

Recall that the origin and the destination locations are given as a certain location on a track rather than a point, but it would be easier to understand if we can view those locations as points. So, we introduce two new points p and q virtually for the origin and the destination. Suppose that we are moving from a certain location on e_2 to a certain location on e_4 in the sample railway network $R(V, E)$ shown in Figure 2.1. We can construct a new network $R'(V', E')$ as shown in Figure 3.1 using the given network $R(V, E)$, such that $V' = V \cup \{p, q\}$ and $E' = (E - \{e_2, e_4\}) \cup \{e_{21}, e_{22}, e_{41}, e_{42}\}$. On the modified network R', our problem can be changed to finding a shortest path between two points p and q.

Before we describe the second step of the network modification, it's worth while to analyze the vehicle movements on a railway network. Recall the conventions of the legal steps and the illegal steps defined in Chapter 2.

Figure 3.2 shows the possible movements from p, which includes some of the possible journey paths from p to q. In Figure 3.2, we can find that starting from a track, a train can move through any of its endpoints except for the terminals. But starting from a point, a train can move only in one of the two possible directions. For example, from v_2, we can

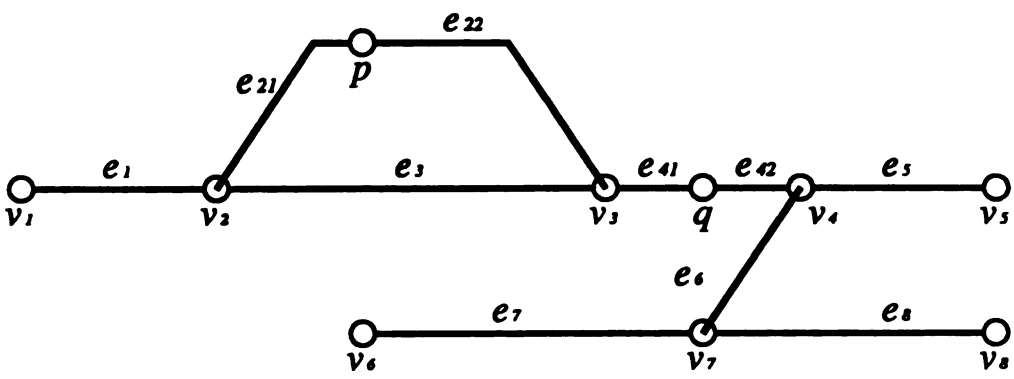


Figure 3.1: Network modification

move only to e_1 if we are moving from e_2 or e_3, else we can move only to e_2 or e_3 if we are moving from e_1.

We can generalize this concept(duality of moving directions from a point) as *pseudopoints* as defined below. Consider a point v of degree 3, and a track e which is incident to v. If a train can move from e through v to any of its two neighbor tracks, then e is called the *supertrack*[2] of v, and those neighbor tracks are called the *non-supertracks* of v. For the points of degree 2, we arbitrarily select an incident track as the supertrack to generalize our approach. Thus we can define pseudopoints v_1 and v_2 for every point v, such that v_1 represents the traversing from a non-supertrack to the supertrack, and v_2 represents supertrack $\rightarrow$ non-supertrack traversing. (See Figure 3.3) We denote a pseudopoint of a point v as $\tilde{v}$. So $\tilde{v}$ is either v_1 or v_2.

Now, let's consider a train moving from a non-supertrack of a point v to another non-supertrack of the point v. As was described earlier, direct traversing between two non-supertracks is prohibited. Starting from a non-supertrack, a train must move to the supertrack of the point v, then change the direction of movement and move to the other non-supertrack. Figure 3.4 shows the train movement between two non-supertracks with-

[2]The similar concept was introduced in [Chu88] as *super arc*.

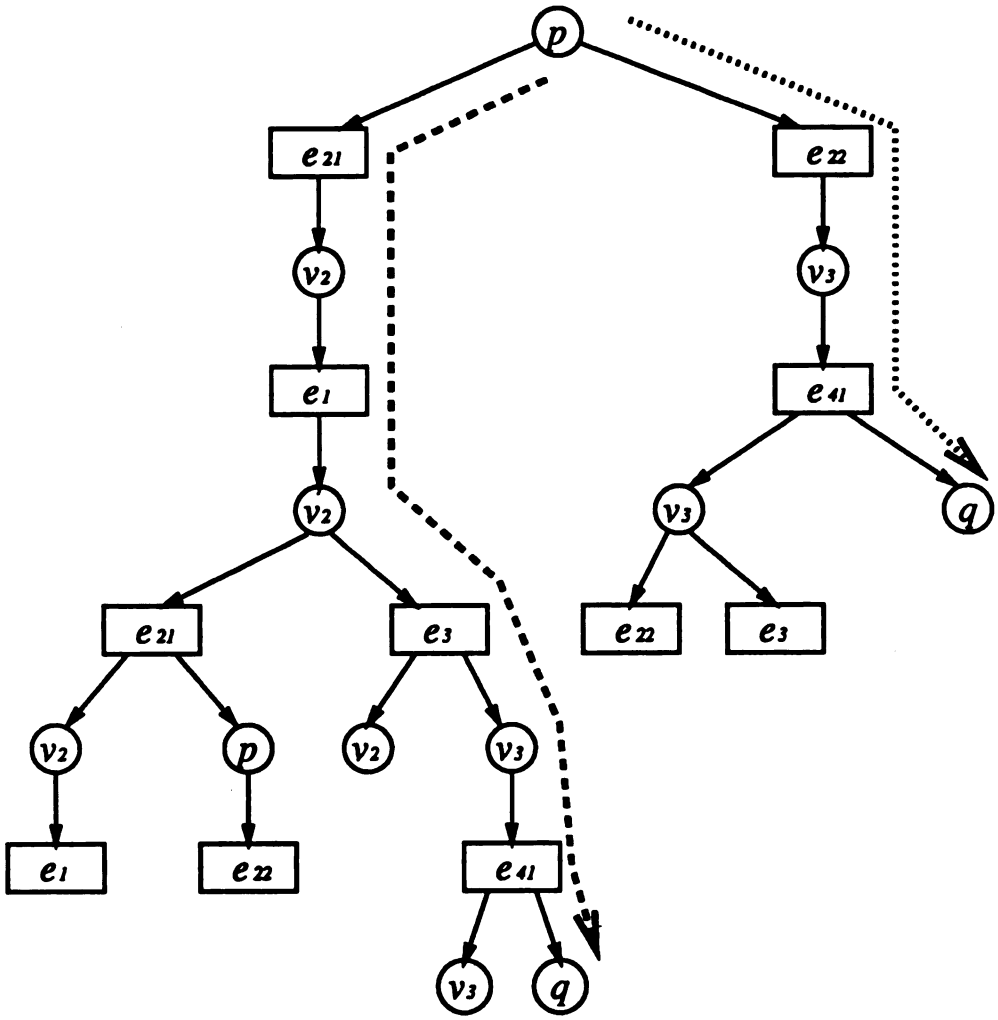


Figure 3.2: Example of possible movements

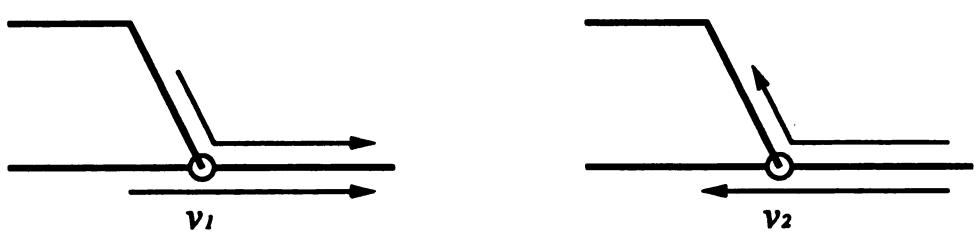


Figure 3.3: Traversing directions and pseudopoints

out any additional delay except for the penalty of the direction change.

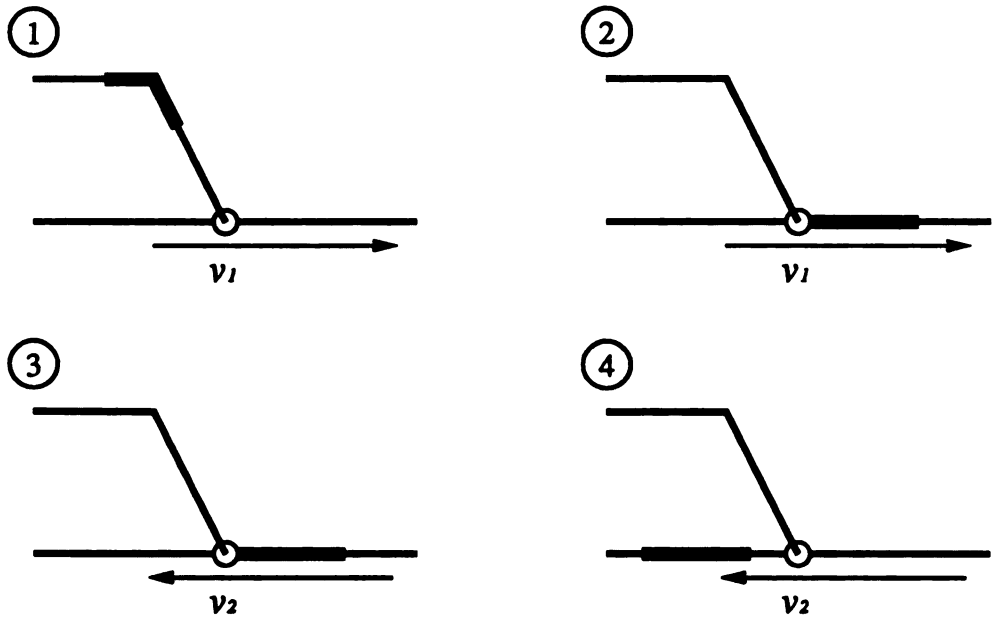
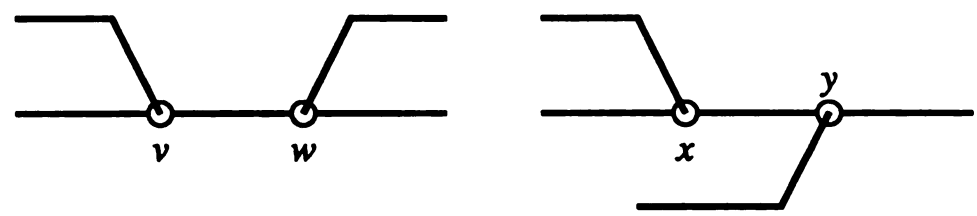


Figure 3.4: Train movement between two non-supertracks

In the fig. 3.4, 1) and 2) show the movement from a non-supertrack to a supertrack, so the pseudopoint traversed is v_1. And 3), 4) show the movement from a supertrack to a non-supertrack, so the pseudopoint traversed is v_2. As was described in Chapter 2, there is a penalty for changing the direction of movement from v_1 to v_2. Let t_r be the total time needed to reverse the direction. Since the middle point of the train represents the train movements, we can easily see that

$$t_r = l(a)/2 + penalty + l(a)/2 = l(a) + penalty.$$

Actually t_r can be viewed as the turn penalty in a network with penalty described in the previous section. When we modify the network using pseudopoints, t_r can be used as the weight between two pseudopoints of the same point.



pseudo-tracks : (v-1,w-2), (w-1,v-2), (x-1, y-1), (y-2,x-2)

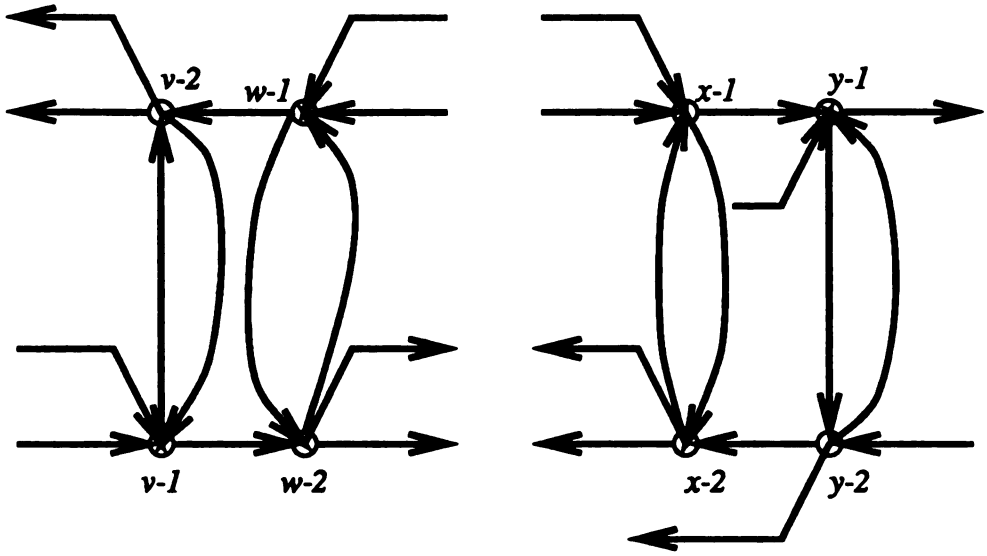


Figure 3.5: Modified graph G''

Using the concept of pseudo-point and penalties, we can construct a directed graph G'' which is equivalent to G' constructed above. Two samples are shown in Figure 3.5. Once we construct G'', the problem becomes the simple shortest problem on a directed graph. We call the directed arcs in G'' as **pseudo-tracks**. Here we note that the weight of each pseudo-track between two pseudo-points of a point is t_r.

3.2.2 Algorithm

The following algorithm *Rail_Shortest_path* modifies the given network G into G'' and finds out a shortest path from the pseudopoints of the starting position to one of the pseudopoints of the destination using Dijkstra's algorithm.

Algorithm Rail_Shortest_path(*G,S,D,l*)
Input : $G = (V, E)$ (weighted non-directed graph), S (starting position) D (destination) and l (train length).
Output : $min(d_1.SP, d_2.SP)$ is the length of the shortest path from s to d, where s and d are the new points created from S and D when we modify the network G into G'.
For each pseudo-points w_i in the path, $w_i.from$ is the pseudo-point from which the path is incident to point w.

```
begin
  Create G' using G, S, and D.
  Create G'' using G'.
  {See preceding section for G' and G''.}
  for all points w in G'' do
  {w1 and w2 are pseudo-points matched to w.}
    w1.mark := w2.mark := false;
    w1.SP := w2.SP := ∞;
    w1.from := w2.from := NULL;
  s1.SP := s2.SP := 0;
  while d1 and d2 is unmarked do
    let wi be unmarked such that wi.SP is minimal;
    wi.mark := true;
    if w ≠ d then
      if j ≠ i and wj is unmarked then
        if wi.SP + penalty < wj.SP then
          wj.SP := wi.SP + penalty
          wj.from := wi
      for all pseudo-tracks (wi, zj) such that zi is unmarked do
        if wi.SP + length(w, z) < zj.SP then
          zj.SP := wi.SP + length(w, z)
          zj.from := wi
end
```

To find a minimum among a set of path lengths, as was suggested by Udi Manber in [Man89, pp.206], a heap structure is adopted. And we follow the methods described in [BB88] for the implementation of the heap.

Complexity Creating G' takes a constant time. In creating G'', it takes $O(|V'|) = O(|V|)$ time to create pseudopoints and pseudotracks between them. Creating two pseudotracks per each track takes $O(|E'|) = O(|E|)$ time. So, creating G'' takes $O(|V| + |E|)$ time. In the main loop, updating the length of a path takes $O(\log m)$ comparisons, where m is the size of the heap. So m is bounded by $|V''|$. Since each pseudotrack can cause at most one update, there are at most $(|E''|)$ updates, leading to $O(|E''| \log |V''|) = O(|E| \log |V|)$ comparisons in the heap. There are also at most $|V''|$ iterations which lead to $|V''| = |V|$ deletions from the heap. So, the main loop takes $O((|E| + |V|) \log |V|)$ time. Hence the total running time is $O(|V| + |V| + |E| + (|E| + |V|) \log |V|) = O((|E| + |V|) \log |V|)$ time. Since a railway network is sparse, $O(|E|) = O(|V|)$. So the time complexity is $O(|V| \log |V|)$.

Chapter 4

Proposed Techniques

To solve our problem (Problem 1 defined in Section 2.1), two techniques are proposed in this chapter. Both techniques, *Time – Constraint – Propagation* and *Incremental – Branch*, are based on A*/Branch-and-Bound algorithm. The first section in this chapter will describe some general aspects for solving the problem, and also describes the A* algorithm and its properties briefly. Proposed two techniques are described in later sections.

4.1 General Approaches

Before we describe our techniques, we need to define a new terminology, *hop*. The movement of a train from point u to one of its adjacent points or to u itself via no other points is called a hop. So a hop is analogous to a step defined in Chapter 2. A hop can be represented as an ordered triple, (u, e, v), where of a point(from-point), a track(via-track), and a point(to-point) (u, e, v). Then a path can be viewed as a sequence of hops. The previous example of possible movements in Figure 3.2 can be expressed using hops as shown in Figure 4.1.

Since there are moving directions on the points in a hop, a hop can also be represented as an ordered triple of a pseudopoint, a track and a pseudopoint $(\tilde{u}, e, \tilde{v})$. We call $\tilde{u}$ the

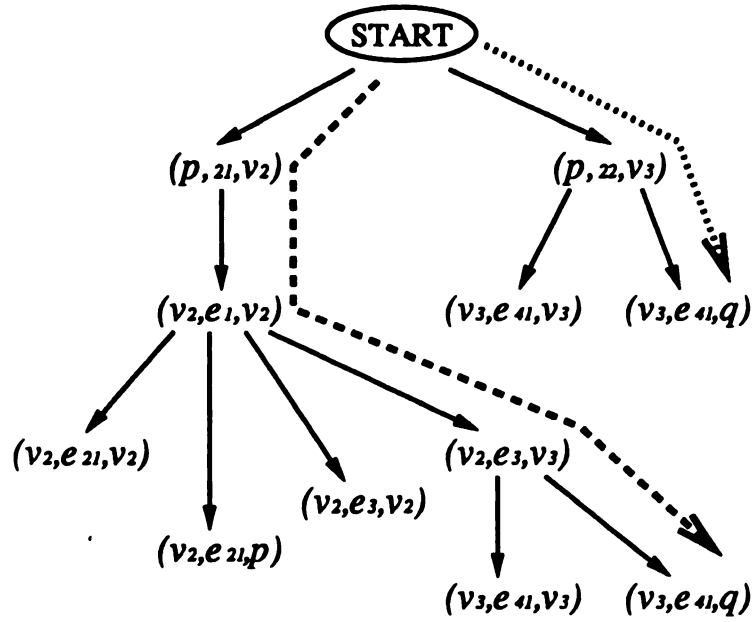


Figure 4.1: Sample movements represented by hops

entrance pseudopoint and $\tilde{v}$ the exit pseudopoint of the hop h. Here u and v are called simply the entrance point and the exit point respectively.

We can view our problem as a search problem on a state space, where each state corresponds to a hop with time information. The state space will vary based on the scheme, how we relate time information to a hop.

The simplest idea is to define a state as the triple of a hop h, an entrance time t_1, and an exit time t_2, (h, t_1, t_2). Here t_1 and t_2 designate the time at which the middle point of the train enters and leaves the hop respectively. In the search graph, the entrance time of a successor node is the exit time of the parent node, since a train is not allowed to stop while traversing a point.

If we are dealing with the simple shortest path problem, i.e. if there exist no previously scheduled journeys for the other trains, there will be no extra delay other than the penalty defined in Chapter 2, so the times t_1 and t_2 represent the earliest arrival time and the

earliest exit time of the hop following the path in the search graph.

Let a be the train under consideration. Since those times t_1 and t_2 represent the movements of the middle position of a train, t_2 can be easily computed in terms of the entrance time t_1, track length $l(e)$, train length $l(a)$, and the penalty as shown in Eq. 4.1.

$$\begin{aligned} t_2 = \quad & t_1 + l(e) \quad && \text{if } u \neq v \\ t_2 = \quad & t_1 + l(a) + \textit{penalty} \quad && \text{if } u = v \end{aligned} \tag{4.1}$$

Actually this approach will give a shortest path information which will lead us to the same result as was shown in Chapter 3. But this approach cannot be used to solve our problem directly, since in the presence of the time constraints on each track, optimal solution may include some delay on some tracks. So we must consider all the possible delays, but allowing delay means infinite number of possible exit times, hence infinite number of successor nodes in the search tree. What makes the situation worse is that there is no way of enumerating the successors.

Let's consider another approach of defining a state. We define a new terminology *timegap* and introduces a new time information t_3–latest exit time from the hop. Timegap is the triple (t_1 =entrance time, t_2 =earliest exit time, t_3 =latest exit time) and the pair (h =hop, t_g =timegap) will be our new definition of the state.

In our problem, a train is supposed to stay on the destination track forever after its arrival. So, a hop (u, e, v) is called a *last hop* in a path if

1. v is the destination, and

2. $t_3 = \infty$ for the associated timegap (t_1, t_2, t_3).

Recall that each track e is associated with a set of permissible traveling times U_e. Let a and (h, t_g) be the train and the state under consideration respectively, and let $h = (u, e, v)$. The timegap is called feasible, if and only if there exists a time interval $[t_i^1, t_i^2] \in U_e$ such that $t_1 \geq t_i^1 - l(a)/2$ and $t_2 \leq t_i^2 - l(a)/2$, where Equation 4.1 holds for t_1 and t_2. For a feasible timegap, the latest exit time t_3 is computed as

$$t_3 = t_i^2 - l(a)/2.$$

A* algorithm and its properties

Our approaches will be described on the basis of the so called Branch-and-Bound/A* algorithm. We will briefly review the previous works to understand the foundation of our algorithm.

Depth-first search and *Breadth-first* search are the commonly known strategies for exploring search trees. *Best-first* search is a way of combining the advantages of both depth-first and breadth-first search into a single method. In many applications, same node can be generated from different parent nodes. So the resulting search space is a directed graph rather than a tree. *A** algorithm is a way to implement best-first search of a problem graph avoiding the pursuing duplicate paths.

The A* algorithm is first described in [HNR68, HNR72], and Rich gives a good introduction to the algorithm [Ric83]. A* algorithm maintains two sets called *OPEN* and *CLOSED*. OPEN contains the nodes generated but not expanded, and CLOSED contains the nodes already expanded. In the A* algorithm, each node in the problem graph contains a value

$$f' = g + h',$$

where g is the exactly known cost of getting from the initial state to the current node and h' is an estimate of the additional cost of getting from the current node to a goal state. The combined function f', then, represents an estimate of the cost of getting from the initial state to a goal state along the path that generated the current node. If more than one path generated the node, then the algorithm will record the best one.

The A* algorithm operates as follows. It proceeds in steps, expanding one node at each step, until it generates a node that corresponds to a goal state. At each step, it picks the most promising of the nodes that have so far been generated but not expanded. It then generates the successors of the chosen node checking if any of them have been generated before, to guarantee that each node only appears only once in the graph. Then the next step begins again and repeats until it generates a goal node.

Here we note that the A* algorithm is *admissible*, if h' never overestimates the real distance of a goal node h. A search algorithm is said to be admissible, if it guarantees to find an optimal path to a goal node, if there exists any. Furthermore, if

$$h'(m) - h'(n) \leq c(m,n),$$

where $c(m,n)$ is the weight of the arc (m,n) in the problem graph, then the estimate function h' is called consistent. And if the estimate function is consistent, it is guaranteed that A* will never reopen a closed node(See [Mo84]).

The shortest path algorithms presented in Chapter 3 can be viewed as a variation of the A* algorithm in which the estimate function h' is always zero. Thus it never overestimates the real distance h, and hence is admissible. It is also consistent, because $h'(m) - h'(n)$ is always zero and never overestimates an arc weight in the search graph.

Successor generation

To apply a search strategy to a specific application, we must give a scheme to generate successor nodes in the search graph and an estimate function. Consider the following questions for our problem.

1. How to generate the successors ?

2. How to compute g and h' ?

The second question is easier to answer in our problem. The exit time t_2 will be the cost g, since we are only concerned with the time taken to arrive at a node. The simplest way for computing h' is to set $h' = 0$ for all nodes. Since the estimate function $h' = 0$ never overestimates h and is consistent, it guarantees to find an optimal solution in an efficient way. We can consider another estimate function which will reduce the fair amount of search space. The shortest path algorithm suggested in Chapter 3 gives a good estimation. We can compute the shortest path length from the destination to every other pseudopoint following the algorithm in Section 3.2. Since we are moving from the origin to the destination, the shortest length computed for a pseudopoint v_1, is the estimate of the other pseudopoint v_2 of the same point v, and vice versa. The estimate h' of a state (h_p, t_p) is the shortest path length from the destination to the exit pseudopoint of hop h_p. Here, we note that the shortest path length is never greater than the cost to the destination, and hence searching with this estimate function is admissible. Moreover consistency is guaranteed, for there is no negative delay.

Now consider the first question. Let (t_1^p, t_2^p, t_3^p) be the timegap of the node under consideration. Then for each possible next hop [1] (u, e, v), we do the following.

[1] A hop (u_2, e_2, v_2) is a next hop of (u_1, e_1, v_1), if $v_1 = u_2$ and $e_1 \neq e_2$.

1. Compute the traversing time t_t of the next hop.

$$\begin{aligned} t_t = \quad & l(e) \quad && \text{if } u \neq v \\ t_t = \quad & l(a) + penalty && \text{if } u = v \end{aligned} \tag{4.2}$$

2. $\forall (t_1, t_2) \in U_e$,

   (a) Compute

$$\begin{aligned} t_1^s = \quad & t_2^p \quad && \text{if } t_1 \leq t_2^p - l(a)/2 \\ t_1^s = \quad & t_1 + l(a)/2 && \text{if } t_1 > t_2^p - l(a)/2 \end{aligned} \tag{4.3}$$

   (b) If $t_1^s + t_t \leq t_2$ then set $t_2^s = t_1^s + t_t$, and $t_3^s = t_2 - l(a)/2$. Then generate SUCCESSOR with the state (h_s, t_s), where h_s is the next hop (u, e, v), and t_s is the associated timegap (t_1^s, t_2^s, t_3^s).

In exploring a search graph, whenever a new node is generated, we must check whether the node is previously generated. A timegap (t_1^1, t_2^1, t_3^1) is called a *subtimegap* of a timegap (t_1^2, t_2^2, t_3^2), if and only if $t_1^1 \geq t_1^2$ and $t_3^1 \leq t_3^2$. In out state space, if two nodes have same hop and the timegap of one node is a subtimegap of the other, one node should not be expanded any more.

4.2 Time Constraint Propagation

In our problem, the number of permissible time intervals of a track e, $|U_e|$ is expected to be relatively large, hence, strict application of the approach suggested in the previous section

may suffer from the huge number of nodes caused by the high branching factor. Reducing the number nodes is highly required in real applications.

Halpern and Priess suggested a *time-constraint-propagation* method for solving the optimal routing problem in a wide area railway network model [HP74]. The same idea can be applied to solve our problem. In solving our problem with the time-constraint-propagation model, a state of the problem graph can be defined as an ordered pair of a hop h, and a set of timegaps E_h, (h, E_h). Then there will be only one successor node for each possible set of major attributes, so the branching factor is restricted to the maximum number of the possible sets of major attributes of the successors of a node in the search graph.

The problem solving procedure in the time constraint propagation model can be described as follows. As was stated in the problem $\forall e \in E$, a set of permissible traveling time U_e is given as input. In exploring the search graph, we denote the set of nodes already examined as H_1 and set of the nodes generated but not examined as H_2. For each member of H_1 and H_2, we know

1. the set of timegaps explored so far, and
2. the earliest feasible departure time from the hop.

   The earliest departure time t_h from the hop h is the minimum of the exit times in the set of associated timegaps. i.e.

$$t_h = min[t_2|(t_1, t_2, t_3) \in E_h].$$

Initially $H_1 = \phi$ and H_2 contains two hops (p, e, u) and (p, e, v), where p is the starting point in the modified network R'(See Section 3.2.1). Among the members of H_2, a hop with the earliest departure time is selected. The algorithm terminates if the selected member is

a last hop. If the selected member is not a last hop, put it into H_1 and generate all the possible next hops with set of timegaps.

Let h be the hop selected from H_2 and $h' = (u', e', v')$ be one of the next hop. Then for computing the set of timegaps associated with h', we define a function f such that

$$E_{h'} = f(h, E_h, U_{e'}).$$

The function f can be defined as the union of all the timegaps computed by the procedure in Section 4.1 for each timegap in E_h.

Let $E'_{h'}$ be the set of timegaps already assigned to the hop h'. Without loss of generality, we set $E'_{h'} = 0$ initially for all hops h. If $E_{h'} \not\subseteq E'_{h'}$ and h' was not previously in H_2, put h' into H_2. Every time a successor h' is generated, we compute

$$newE_{h'} = E_{h'} \cup E'_{h'},$$

and modify path information if the next hop is already generated.

The algorithm allows however multiple traveling of a same hop before arriving the destination: i.e. a hop already in H_1 with different set of timegaps can be a candidate for a new member in H_2.

We can formally present the algorithm as shown below.

Algorithm Time_Constraint_Propagate
Input : $R = (V, E)$ (railway network), a (train),
$U = \{U_e | e \in E\}$ (set of the sets of the permissible
travelling times of all the track, and
(a, s, d, t_s) (moverequest), where $s = (e_s, v_s, d_s)$ and $d = (e_d, v_d, d_d)$.
Output : The earliest arrival time.

begin
Step 1: Modify the given network and introduce new points p and q,
which are the locations of origin and destination, respectively.
Step 2: {Initialize two sets : H_1 (already examined), and H_2 (to be examined) }
$H_1 := \phi$; H_2 =two initial hops;
Step 3: **if** $H_2 = \phi$ **then** {No feasible solution exists}
STOP
else select an element h in H_2 with the earliest departure time.
end if
if the selected element is a last hop **then**
STOP {We found a solution}
else
$H_2 := H_2 - \{h\}$;
$H_1 := H_1 \cup \{h\}$;
for all next hops $h' = (\tilde{u'}, e', \tilde{v'})$ **do**
$E_{h'} := f(h, E_h, U_{e'})$;
if h' is generated for the first time **then**
$H_2 := H_2 \cup \{h'\}$;
else if $E_{h'} \not\subseteq E'_{h'}$ **then**
$H_2 := H_2 \cup \{h'\}$;
$newE_{h'} = E_{h'} \cup E'_{h'}$;
end if
go to Step 3.
end if
end

Properties of the algorithm.

The algorithm is guaranteed to find an optimal solution, if there exists any.

Lemma 4.1 *Time-Constraint-Propagation algorithm is admissible.*

Proof : The algorithm may terminate when, in the third step, the set H_2 is found to be empty. The construction of H_2 is such that it contains all nodes which are reachable by

only one hop from a node in H_1 and it is feasible to get there from the source via a path with all its nodes contained in H_1. If the set H_2 is empty, it follows that the hops in nodes in H_1 are physically or in the time dimension disconnected from other hops in the given network. So, if there exist feasible solutions, the algorithm will not fail to find one.

The admissibility of the algorithm immediately follows the admissibility of A* algorithm. Since the earliest arrival time to the exit pseudopoint of a node represents the cost from the root node to the current node, it can be noted as g of A* algorithm. Then the estimate function h' is always zero in the Time-Constraint-Propagation algorithm, hence never overestimates the real cost to destination. □

On the Complexity of Time-Constraint-Propagation

Let T_m be the time after which no train is scheduled to move. We should note that if the requested departure time t_s of the current train is not less than T_m, our problem can be reduced to a simple shortest path problem with an additional condition that all the destination tracks of previously scheduled trains are blocked. Let L_t be the minimum length of a train and M be the maximum number of permissible time intervals assigned to a track. Then

$$M = \lceil T_m / L_t \rceil$$

.

Let a denote the current train and without loss of generality, assume that the requested departure time of the current train, t_s, is zero. Let t_k be the value of the earliest departure time, which is obtained in Step3 at the k_{th} iteration. Then

Lemma 4.2 *If a feasible solution exists, then for any iteration, k, there exists an integer,*

v, and a positive real number d, such that $t_{k+v} - t_k \geq d$.

Proof : Let $d = min\{l(a) + penalty, l(e) | e \in E\}$ and let v be the number of the members in H_2 at the k_{th} iteration. Following the algorithm, the earliest departure time of newly generated nodes should be greater than or equal to $t_k + d$, since t_k is the minimum in H_2 and it takes at least d time to traverse a hop. So, we have maximum v number of nodes of value not greater than t_k in H_2 to be selected, hence, $t_{k+v} \geq t_k + d$. □.

Here we note that $L_t < d$, hence our algorithm will terminate before $v * M$ iterations or will be reduced to the simple shortest path problem after that number of iterations. So, the complexity is bounded by the complexity of $v * M$ iterations. In each iteration, it executes a certain number of heap operations. The number of the heap operation is bounded by a constant 5 (One deletion and maximum 4 insertions, for maximum number of next hops is 4). Each heap operation takes $O(\log v)$ time.

Lemma 4.3 *Number of timegaps associated to a hops is not greater than the number of permissible traveling time intervals associated to the hop.*

Proof : Suppose otherwise, there exist two timegaps associated to a hop, such that the two timegaps are in the same permissible traveling time interval. Since the latest exit time is decided by the traveling time interval, one of the two timegaps must be a subtimegap of the other. A subtimegap of a timegap in the same hop is not considered a different timegap. Contradiction. □.

A successor generation takes $O(M)$ time, since

1. number of timegaps associated to a hop is also bounded by M as shown in above lemma, and

2. each operation $E_{h'} := f(h, E_h, U_{e'})$, and $newE_{h'} = E_{h'} \cup E'_{h'}$ can be considered as a linear merging of two sorted lists.

In each iteration a finite number of successor generation function is performed for each next hop, hence an iteration takes $O(M)$ time. So the total complexity is $O(vM(M + \log v)$. Since v is less than the number of possible hops which is bounded by $O(|V|)$, the time complexity is $O(M|V|(M + \log|V|)$.

Improvement to the algorithm

Using an appropriate estimate function will reduce the number of nodes generated. The shortest path algorithm described in Section 3.2 can be a good estimate function. The next example will show the computation of the estimate function.

Sample problem :

- Railway network R in Figure 4.2.

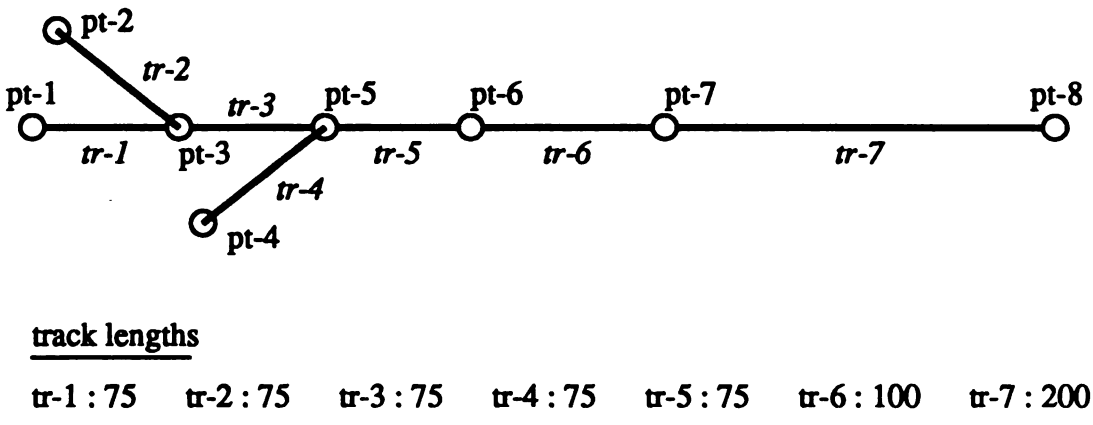


Figure 4.2: Problem network

- Train #a is already scheduled for a moverequest $(a, s_a, d_a, 0)$, where s_a = (tr-7, pt-7,

175) and d_a = (tr-2, pt-3, 25). Train #a is scheduled without any delay. There is no other move scheduled other than train #a. Figure 4.3 shows the resulting set of permissible traveling times U_e for each track e.

- Given a new moverequest $(b, s_b, d_b, 25)$ for train #b, where s_b = (tr-1, pt-3, 25), and d_a = (tr-7, pt-7, 50). Find an optimal solution $(P, T(P))$.

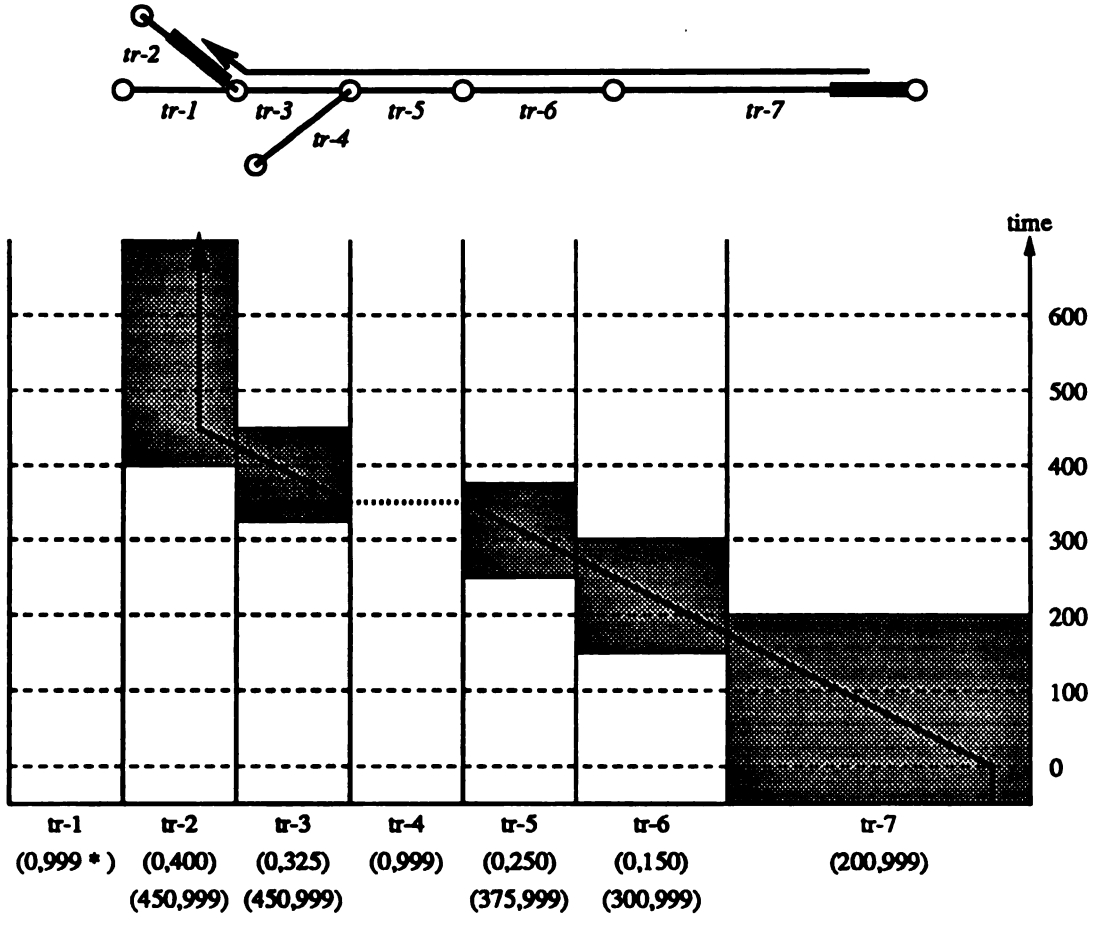


Figure 4.3: Permissible traveling times for each track

Applying Time-Constraint-Propagation to the problem.

Figure 4.4 shows the length from the destination to each pseudo point as the estimate function h'. In the figure, a pair of numbers (n_1, n_2) is assigned to each point v, where n_1

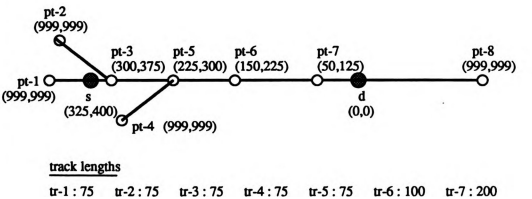


Figure 4.4: An estimate function h' using shortest path lengths

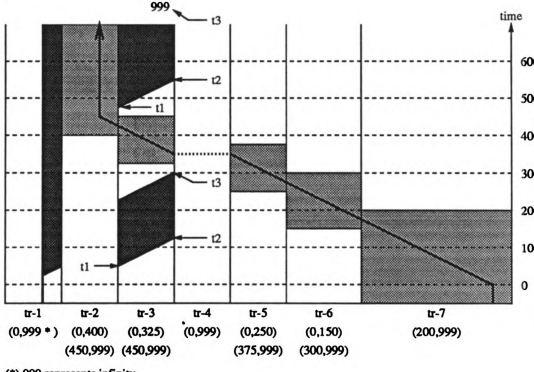


Figure 4.5: An example of timegaps associated with a hop

and n_2 represent the estimate function of each pseudo point $h'(v_1)$ and $h'(v_2)$ respectively. Figure 4.5 shows an example of timegaps. Each set of the time values t_1, t_2, t_3 represents a timegap associated with the hop (pt-3, tr-3, pt-4). In the figure, we see that the number of timegaps associated with the hop is 2. Figure 4.6 shows the search graph using the shortest path length as the estimate function h'. Each node in the search graph graph contains the information of the hop, timegaps and the estimated total cost f'. The numbers above each node shows the order in which the node was generated. Figure 4.7 shows the resulting journey with the journey of previously scheduled train.

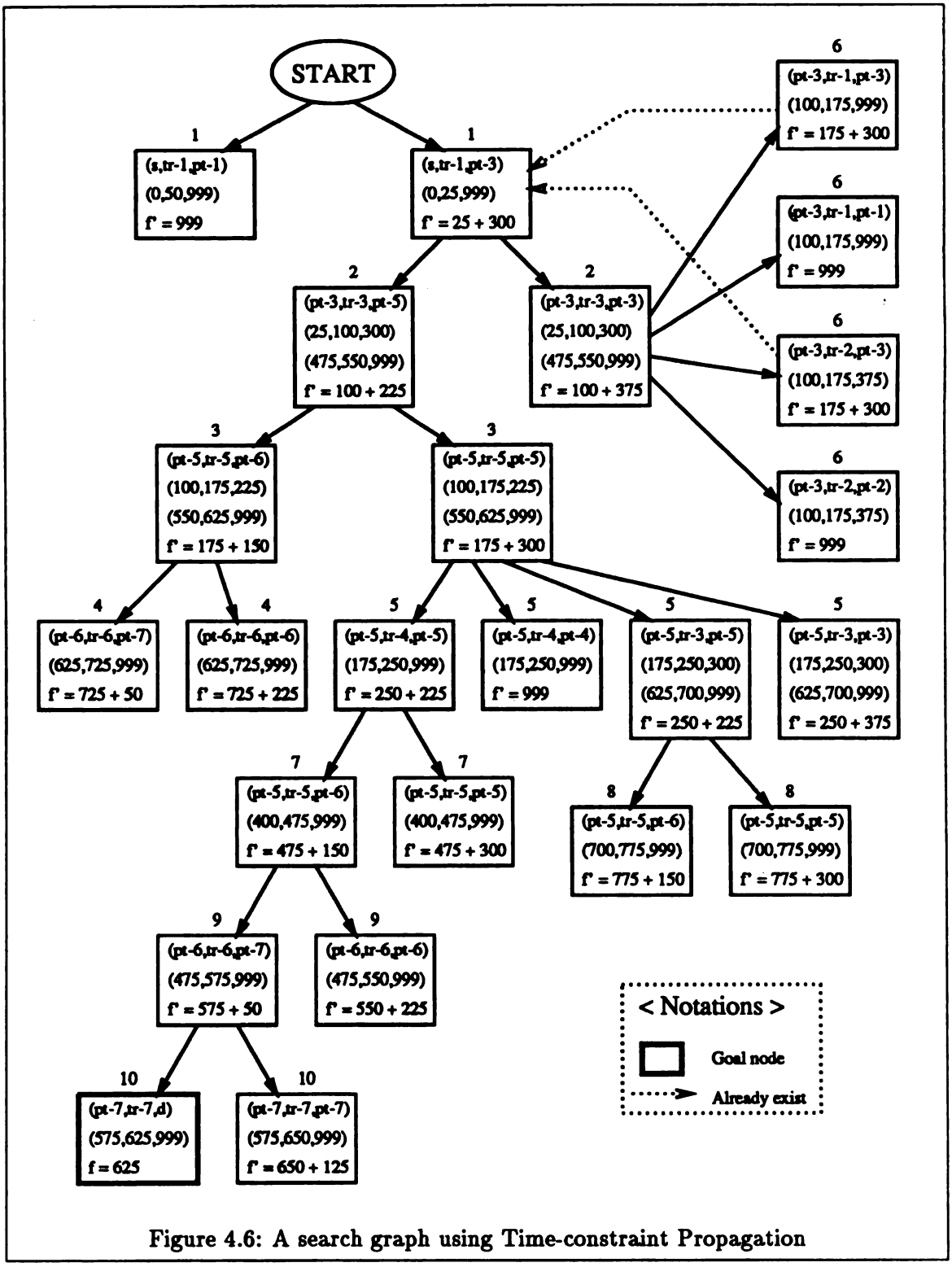


Figure 4.6: A search graph using Time-constraint Propagation

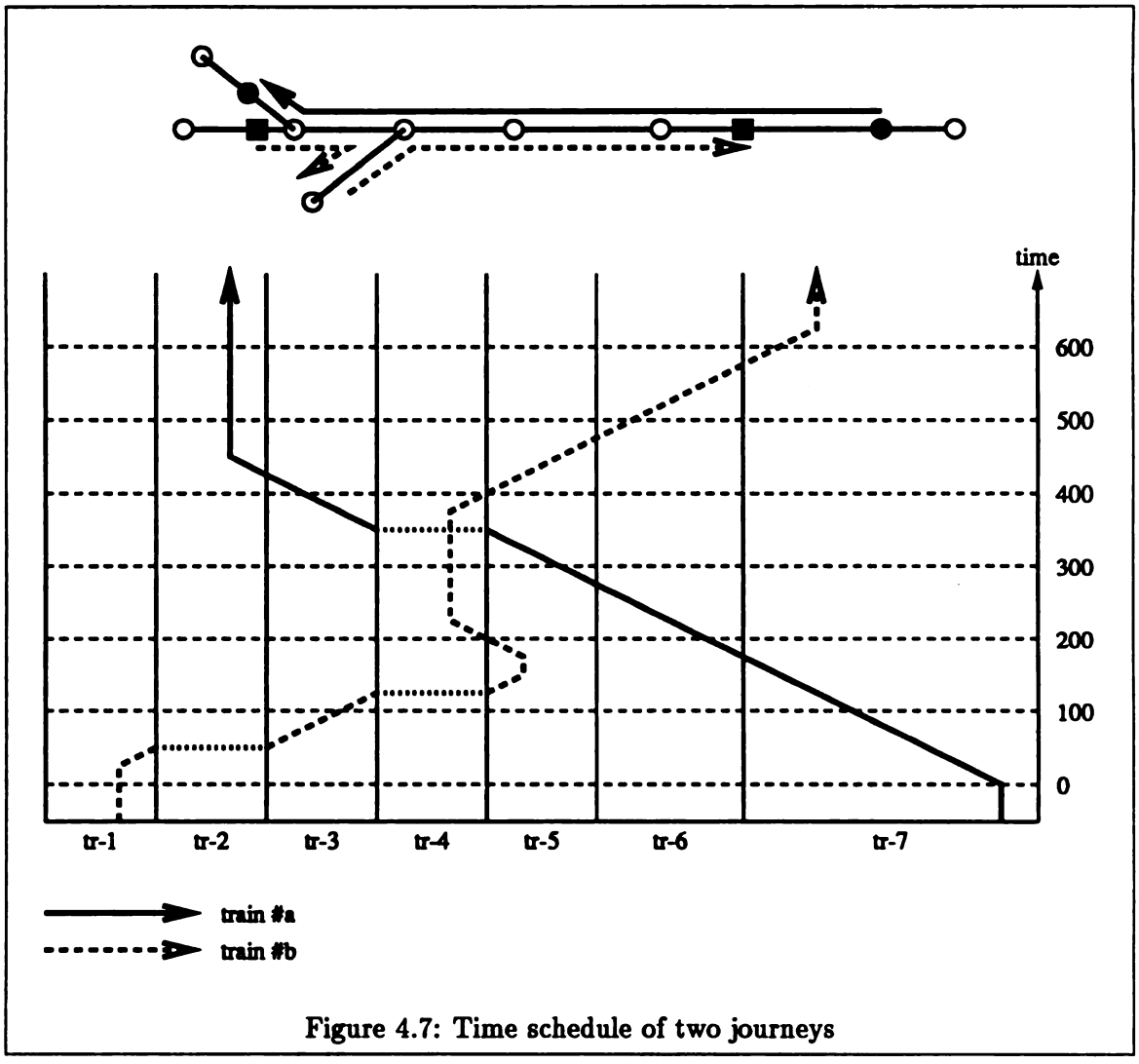


Figure 4.7: Time schedule of two journeys

4.3 Incremental-Branch

Now, let us go back to our old definition of a state in which a hop is associated with a single timegap. Still, there is a way we can reduce the number of nodes generated, and this section describes the approach named Incremental-Branch, in which we do not generate a successor until an explicit request is invoked.

There have been many variations and improvements to the basic A* algorithm [Geo83, Kor85, Mo84, NKK84]. But through all the variation and improvements, the procedure of the node expansion, in which all possible successor nodes are generated at a time, remains unchanged.

Our approach is based on the observation that the attributes of a state in a state space can be divided into *major* attributes and *minor* attributes in our problem, and the number of successor nodes with different major attributes is always finite. A state in this approach is defined as a pair (h, t_g) where h is a hop and t_g is a timegap associated to the hop. The hop h will be the major attribute and the timegap t_g will be the minor attributes in our problem.

The basic operation of Incremental-Branch is almost same as A* algorithm except for the differences shown below.

1. Incremental-Branch generates, from the current node, only one node for each different set of next major attributes with minimum costs, while A* algorithm generates all the successor nodes at a time.

2. If there exists a set of next major attributes a_m with feasible states not generated yet and none of them is reachable from the current node, propagate the unreachability information back to its predecessor recursively until it generates a state which may

have a path to a state with the set of major attributes a_m. Call this procedure *incremental branching.*

The algorithm is described in detail below.

Algorithm Incremental-Branch

1. Start with OPEN containing only the initial nodes. Set CLOSED to the empty list.
2. Until a goal node is found, repeat the following procedure : If there are no nodes on OPEN, report failure. Otherwise, pick the node on OPEN with the lowest f' value. Call it BESTNODE. Remove it from OPEN. Place it on CLOSED. See if BESTNODE is a goal node. If so, exit and report a solution.
3. Otherwise, set *sibling_flag* = 0, and
   for each set of possible next major attributes of the BESTNODE, do the following
   (a) If there exist feasible states reachable from BESTNODE, generate a SUCCESSOR with the lowest f' value among those states. If SUCCESSOR has same properties of any node on OPEN or CLOSED, do relevant works depending on the applications. Otherwise, put it on OPEN.
   (b) If there exists a feasible state which is not reachable from BESTNODE and has not been generated yet,
   set *sibling_flag* = 1.
4. If *sibling_flag* = 0, CALL **Create-Sibling**(BESTNODE).

End Incremental-Branch

Algorithm Create-Sibling(BESTNODE)

1. If BESTNODE is an initial node, return.
2. If there already has been a request to create a sibling of BESTNODE, return.
3. If there exists no feasible state with the same major attributes of BESTNODE, return.
4. Pick the immediate predecessor of BESTNODE in CLOSED. Call it PARENT.
   (a) If there exist feasible states with same major attributes of BESTNODE, and are reachable from PARENT generate a SUCCESSOR of PARENT among those states with the lowest f' value. Let's call it SIBLING of BESTNODE. Put it on OPEN.
   (b) If there exist feasible states with same major attributes of BESTNODE, which are not reachable from PARENT,
   CALL **Create-Sibling**(PARENT) recursively.

End Create-Sibling

Properties of Incremental-Branch

Let's denote the path from the root to the goal node on the search graph given by the algorithm as the sequence of nodes, $P = (n_1, n_2, \ldots, n_m)$, and n_1 is the root node and n_m is the goal node.

Let n_1 and n_2 be any two arbitrary states in the state space with same major attributes where the path costs from the root to those nodes hold the relation $g(n_2) > g(n_1)$. Let m_1 be a successor node of n_1 and m_2 be a successor node of n_2. If it is guaranteed that

1. $g(m_2) \geq g(m_1)$, whenever the major attributes of m_1 and m_2 are same, and
2. the estimate function h' is a function of only major attributes and consistent, then

Lemma 4.4 *there exists no path* $P' = (n'_1, n'_2, \ldots, n'_m)$ *such that* $\forall i (1 \leq i \leq m)$, *major attributes of* n_i *and* n'_i *is same and the cost of the path* $g(n'_m) < g(n_m)$.

Proof : Suppose not. Let k be the first index in the path P' such that $\forall i \geq k$, $g(n'_i) < g(n_i)$ (hence $f'(n'_i < f'(n_i)$, since h' is the function of major attributes). Following the selection rule in the algorithm, n'_i is selected before n_i is selected for all $i \geq k$. So the algorithm will report n'_m before n_m. Contradiction. □

Lemma 4.5 *the Incremental-Branch is admissible.*

Proof : Suppose not. Then there must exist a path $P' = (n'_1, n'_2, \ldots, n'_{m'})$, such that $n'_{m'}$ is a goal node, so the major attributes of n'_m and n_m are same and $g(n'_{m'} < n_m)$. Following the similar argument above, a contradiction occurs. □.

Complexity in the Optimal Routing Problem

Generally speaking, Incremental-Branch algorithm is of exponential complexity. But the optimal routing problem we are dealing has some restrictions. Based on the finiteness of scheduling time and the concept of the minimum length of a train, we assume a maximum number of timegaps a hop can have, which is denoted as M. We also know that the number of possible set of major attributes A is bounded by $O(|V|)$. So the total number of possible states is not greater than $M * A$. Following the algorithm, SIBLING of a node is never generated before the node is selected from OPEN. So the maximum number of nodes in OPEN is bounded by A.

Suppose a node is selected from OPEN. The selection takes $O(\log A)$ time. Then it tries to generate a constant number of successors. Without considering the backtracking procedure, successor generation takes $O(M)$ time, if we use a linear list structure for storing the permissible traveling time intervals. If we an array implementation and binary search, a successor generation will take $O(\log M)$ time. The maximum number of times, a node is called by the *Create-Sibling* procedure in backtracking operation, is bounded by the maximum number of successors, $O(M)$. The *Create-Sibling* procedure takes constant time, except for the first time it is called, which takes $O(\log M)$ times based on array structure and binary search. So the backtracking takes $O(M + \log M) = O(M)$ time per node.

The number of nodes is bounded by the total number of states, so $O(MA)$. For each node, we cosider selection, successor generation, and backtracking. So the total time required is $O(MA(\log A + \log M + M) = O(MA(\log A + M)$. $A = O(|V|)$. Hence the complexity is $O(M|V|(\log|V| + M)$.

Sample problem and solution.

If the estimate function h' never overestimates the real cost h, we note that Incremental-Branch algorithm is admissible for our problem. Let n_1 and n_2 be any two states with same

exit pseudopoint. Suppose $g(n_2) > g(n_1)$. In our problem, g is the earliest exit time t_2 in timegap of the state. Then following the successor generation procedure in Section 4.1, we can easily find that $g(m_2) \geq g(m_1)$, where m_1 and m_2 are successors of n_1 and n_2 respectively and of same major attributes. So it is admissible following the lemmas.

Let's apply Incremental-Branch algorithm to the sample problem described in Section 4.2.

Every time a new node is generated, we must test whether that node has already been generated. Let $n_1 = ((\tilde{u}, e, \tilde{v}), (t_1, t_2, t_3))$ be the newly generated node. If there already exists a node $n_2 = ((\tilde{u'}, e', \tilde{v'}), (t_1', t_2', t_3'))$ in OPEN or CLOSED such that $\tilde{v} = \tilde{v'}$, $t_2 \geq t_2'$ and $t_3 \leq t_3'$, then we don't need to explore n_1 any more. So we simply do not put n_1 into OPEN. Conversely, if $\tilde{v} = \tilde{v'}$, $t_2 \leq t_2'$ and $t_3 \geq t_3'$, then n_2 is not required to be expanded anymore. If n_2 is in OPEN, we simply remove n_2 from OPEN and put n_1 into OPEN. If n_2 is in CLOSED, we remove the successor of node n_2, either from OPEN or from CLOSED, if there exists any. Then, put n_1 into CLOSED and remove n_2 from CLOSED.

Figure 4.8 shows the search graph for the problem described in 4.2 using the shortest path length from the destination as the estimate function h'. The figure explains the operation of incremental branching clearly.

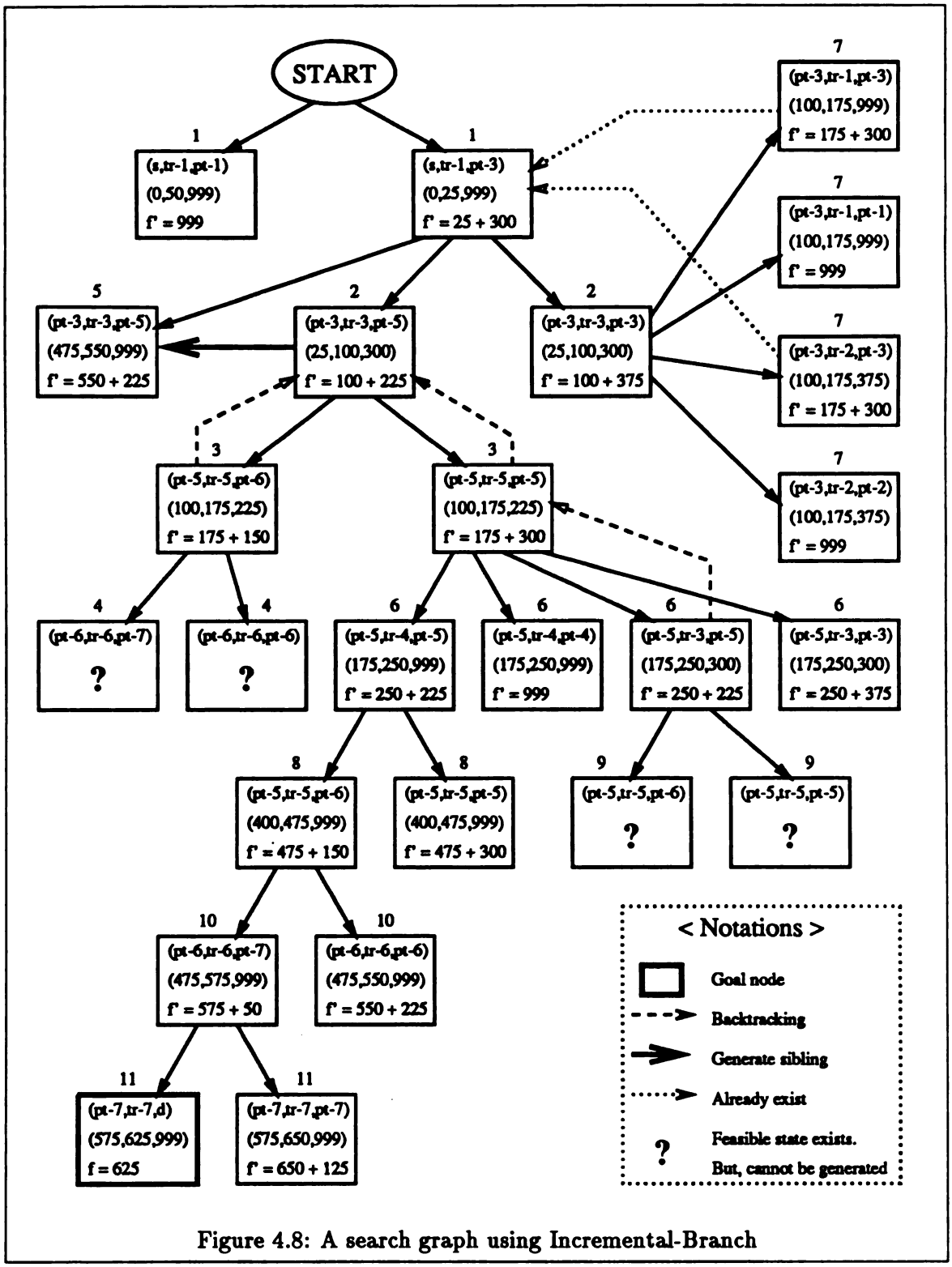


Figure 4.8: A search graph using Incremental-Branch

Chapter 5

Implementation and Experimental results

The proposed algorithms with and without improvements were programmed in C for a SUN4 sparc station. Also a graphic animation was implemented using Xwindow utilities to visualize the movement of the scheduled trains.

To ensure the applicability of the algorithms, test data were carefully chosen to resemble real-world problems. The railway network used in the experiment has the same configuration as the real one designed for the transportation of molten iron in Pohang Steelmill Company in Korea. in a steelmill company. The network consists of 79 points and 90 tracks. Figure 5.1 is the screendump of a window which shows train movements on the railway network. Permissible traveling time intervals are assigned to each track using random number generation. With the given network, as was shown in the complexity analysis, number of time intervals and time limit of the movement of the previously scheduled trains will be two components which will affect the solution time. So various combination of the number of time intervals and the schedule time were tested for a same moverequest.

Figure 5.2 shows the CPU time required for each case using the four different methods : "Time-Constraint-propagation", "Incremental-Branch", and their improved versions. As shown in the Figure, infeasible combinations are excluded. The maximum CPU time required among all the test data and applied methods is 1.1 sec, which is fairly acceptable in real applications. In both algorithms, improved version shows better performance for most of the time. Computing shortest paths only takes 0.01 - 0.02 sec in the experiments, so improved versions are more recommended than the original ones. We also note that "Incremental-Branch" is always better than "Time-Constraint-Propagation" : 3 - 4 times faster in average. So we can conclude that propagating whole time-constraints is more expensive than backtracking overhead, hence "Incremental-Branch" is recommended.

The test result shows an expected feature, which is shown in Figure 5.3 and Figure 5.4. In the figures, we see that the solution time increases as the number of time intervals increases to a certain point and it decreases after that point for both methods. The increment of the solution time can easily be expected from the complexity analysis. The decrement of the solution time results from the fact that as the the number of time intervals increases above a certain value, the resulting number of timegaps decreases rapidly as the time-constraint propagates to each point on the network. We also expect this result, since many number of time intervals means small size of the intervals, and a train movement needs a certain size of time interval to traverse a track, hence small number of resulting timegaps.

Figure 5.5 shows another behavior of the algorithms. With a fixed number of time gaps, as the scheduled time increases, the solution time increases to a certain point and then saturates. The figure shows the behavior of the Time-Constraint-Propagation methods, and the Incremental-Branch shows similar behavior.

Mathematical analysis of the solution time behaviors will be a topic for further re-

searches.

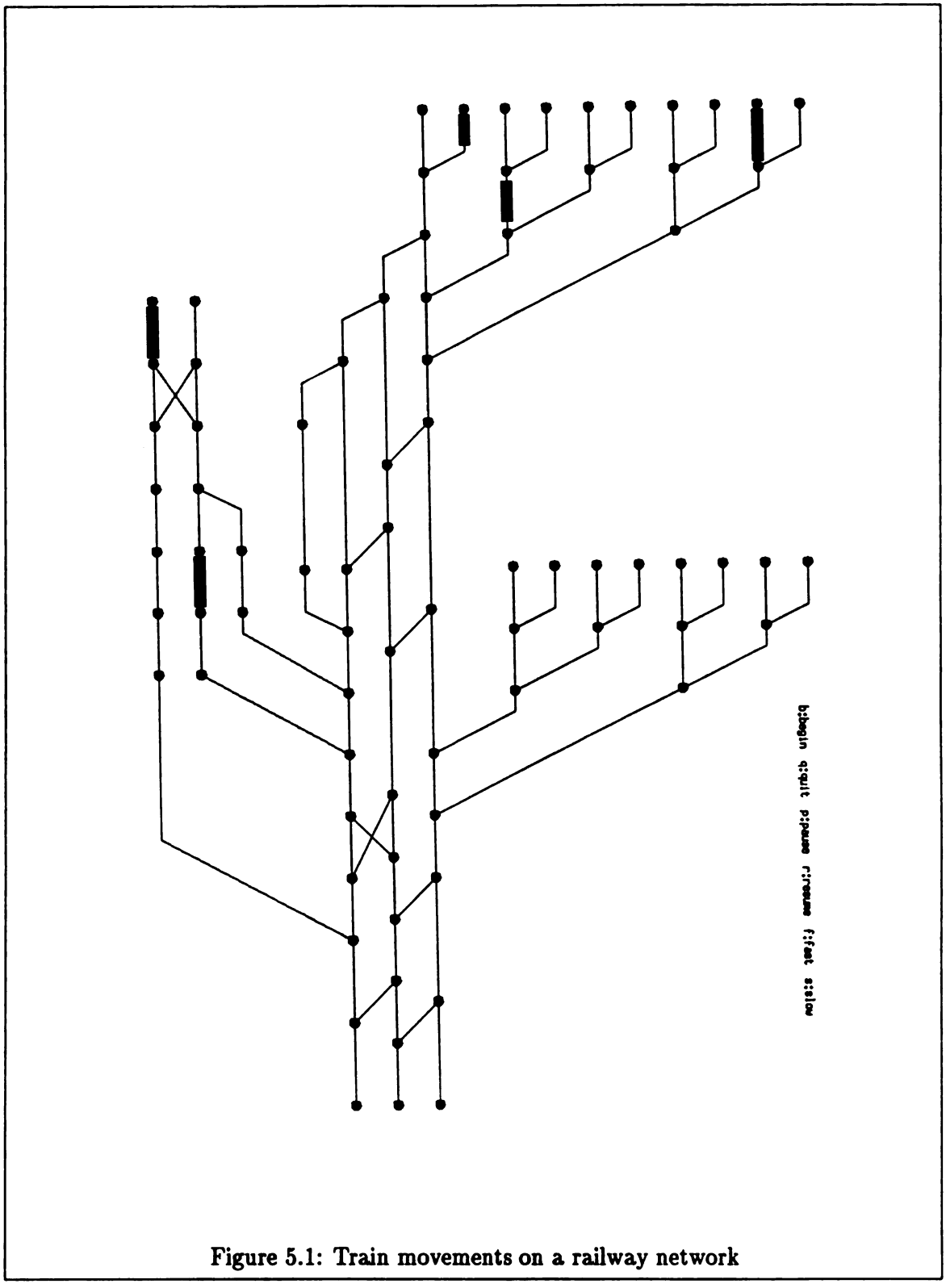


Figure 5.1: Train movements on a railway network

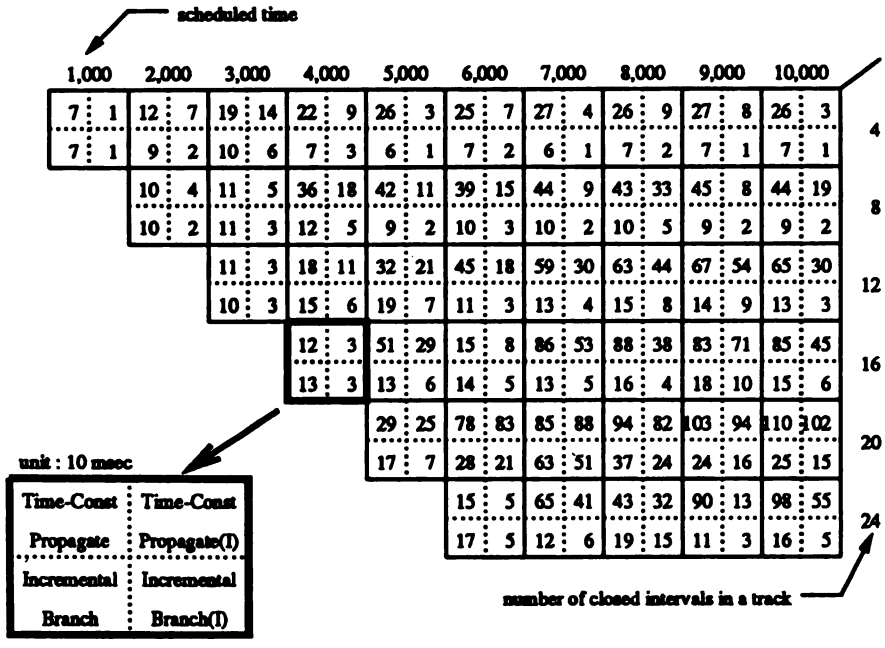


Figure 5.2: Execution time table

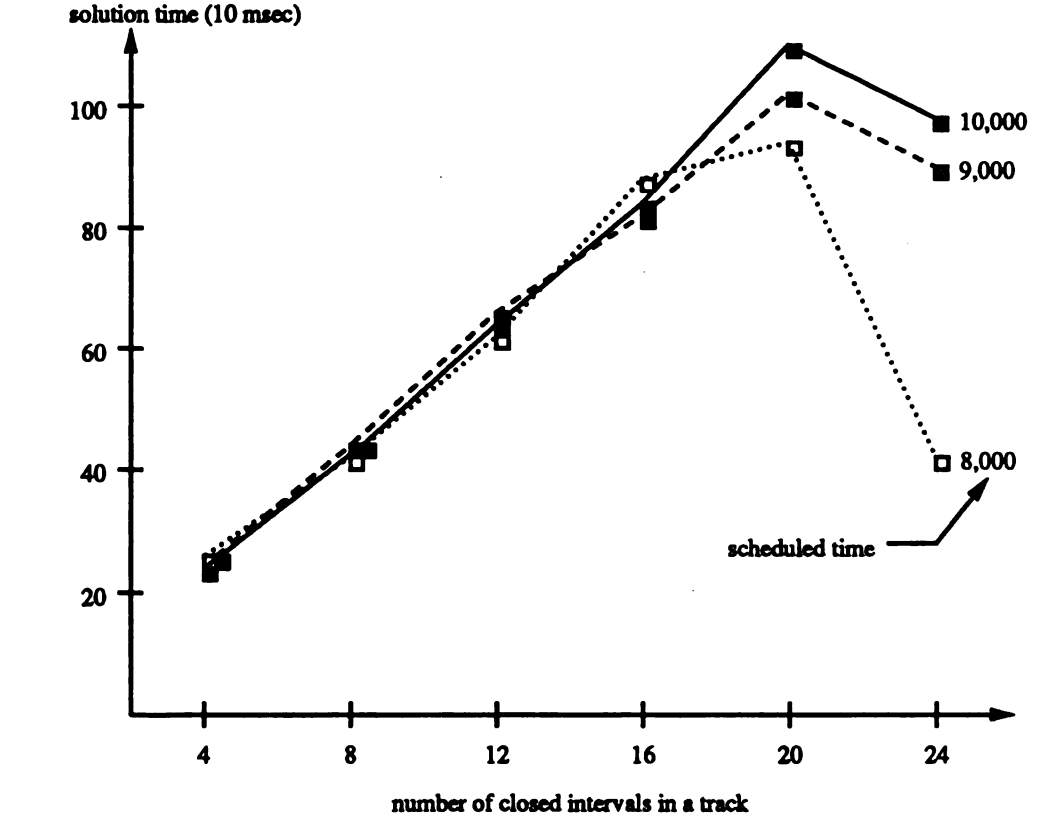


Figure 5.3: Solution time analysis of Time-Constraint-Propagation method (1)

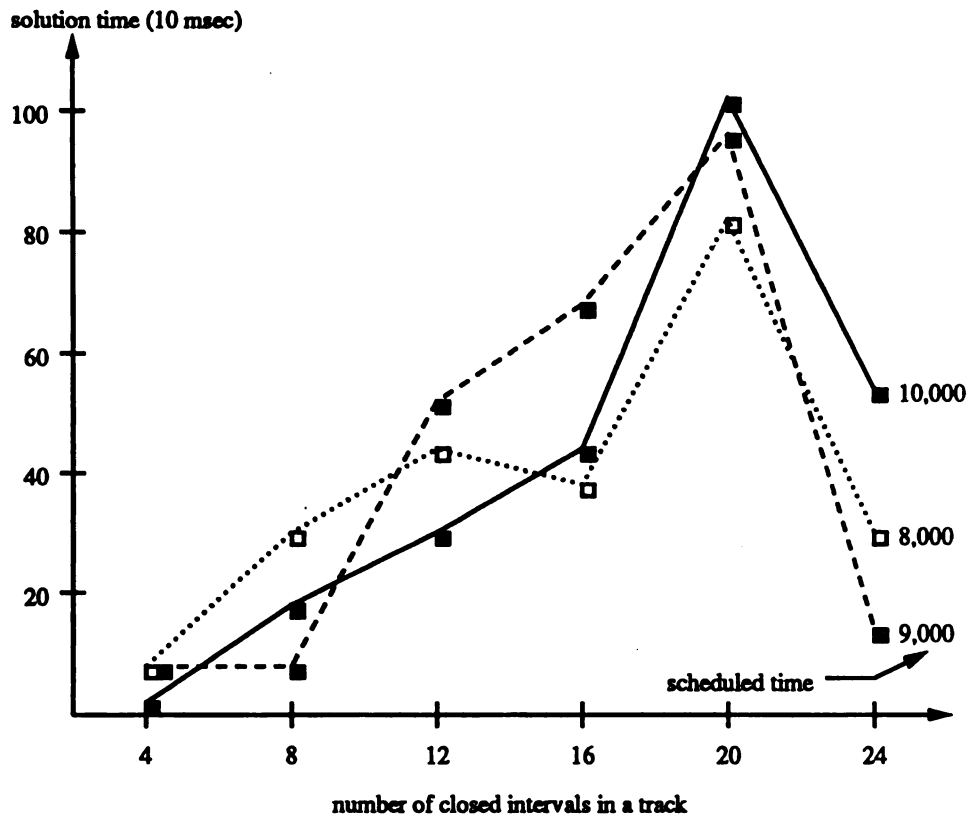


Figure 5.4: Solution time analysis of Incremental-Branch method (1)

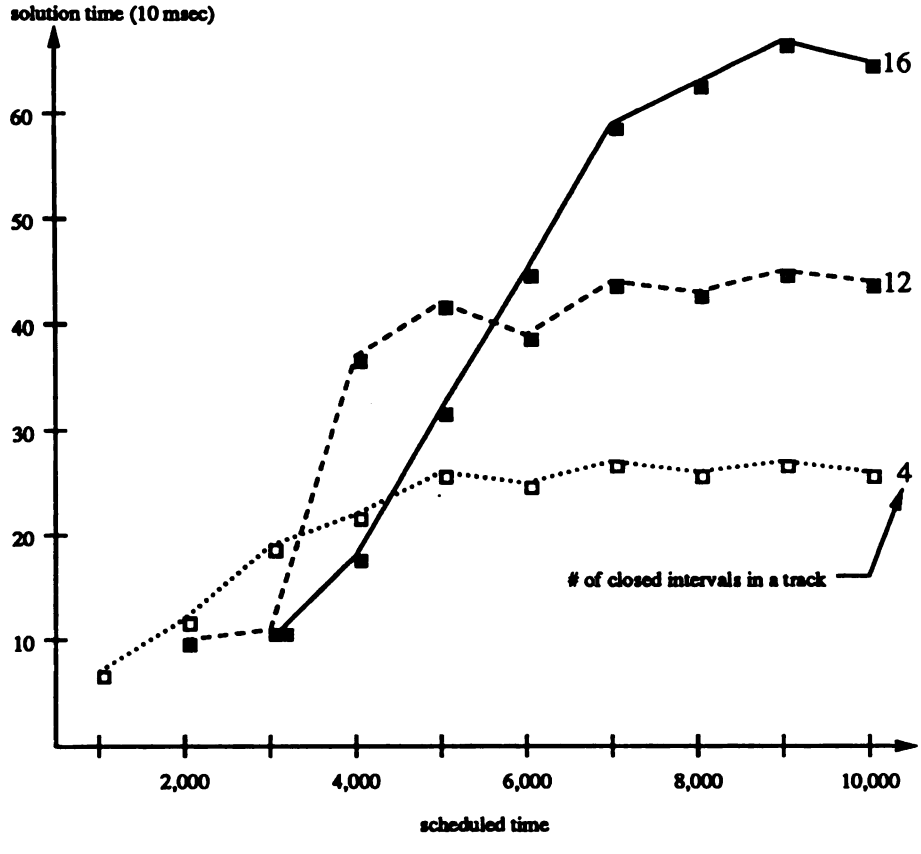


Figure 5.5: Solution time analysis of Time-Constraint-Propagation method (2)

Chapter 6

Concluding Remarks

We have defined the optimal routing problem on a railway network and proposed two algorithms "Time-Constraint-Propagation" and "Incremental-Branch". An algorithm for solving the simple shortest path problem on a railway network is also proposed, which is used as the estimate function h' in the improved version of the two algorithms for the optimal routing problem. The experimental results show that the both algorithms as well as their improved versions are fairly acceptable. Based on the experimental results, "Incremental-Branch" algorithm shows better performance than that of "Time-Constraint-Propagation" algorithm. The experimental results also shows some solution time behaviors, mathematical analysis of which will be a topic for further researches.

What we have dealt in this thesis is one of many possible problems which can be defined on a railway network. Based on the requirement of each railway network, we may consider other problems. Cherniavsky [Che72] defined *Timetable compilation problem* and proposed a *Look-ahead* method to solve the problem. The problem is finding an optimal solution, given a set of regularly repetative transportation requests.

Parallelization will be another topic for further researshes. In A*-like algorithms, heap

access is usually the bottleneck in the performance improvement of parallelization. There are several heap access schemes proposed recently : Centralized Access [Qui87], Concurrent-heap [RK88], Distributed Access [FK88], etc. Implementation and performance analysis of each parallelization schemes for our proposed algorithms will be very helpful not only to the railway network application itself, but also to the general research of heap parallelization, since there have not been many real world applications for the parallelization schemes.

Bibliography

[Ass80] Arang A. Assad. Modelling of rail networks: Toward a routing/makeup model. *Transportation Research-B*, 14:101–114, 1980.

[BB88] Gilles Brassard and Paul Bratley. *Algorithmics. Theory & Practice.* Prentice-Hall, 1988.

[Che72] A.L. Cherniavsky. A program for timetable compilation by a look-ahead method. *Artificial Intelligence*, 3(1):61–76, 1972.

[Chu88] Kwang-Hyung Chung. Shortest path problem. algorithmic approach. 1988.

[Eas85] Said M. Easa. Shortest route algorithm with movement prohibitions. *Transportation Research-B*, 19:197–208, 1985.

[FK88] Chris Ferguson and Richard E. Korf. Distributed tree search and its application to alpha-beta pruning. In *Proc. Seventh National Conference on Artificial Intelligence*, pages 128–132, August 1988.

[Geo83] M.P. Georgeff. Strategies in heuristic search. *Artificial Intelligence*, 20:393–425, 1983.

[HNR68] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[HNR72] P.E. Hart, N.J. Nilsson, and B. Raphael. Correction to 'a formal basis for the heuristic determination of minimum cost paths'. *SIGART Newssletter*, 37, 1972.

[HP74] J. Halpern and I. Priess. Shortest path with time constraints on movement and parking. *Networks*, 4:241–253, 1974.

[Kor85] R.E. Korf. Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–107, 1985.

[KP69] Ronald F. Kirby and Renfrey B. Potts. The minimum route problem for networks with turn penalties and prohibitions. *Transportation Research*, 3:397–408, 1969.

[Man89] Udi Manber. *Introduction to Algorithms. A Creative Approach.* Addison-Wesley, 1989.

[Mo84] László Mérõ. A heuristic search algorithm with modifiable estimate. *Artificial Intelligence*, 23:13–27, 1984.

[NKK84] Dana S. Nau, Vipin Kumar, and Laveen Kanal. General branch and bound, and its relation to a* and ao*. *Artificial Intelligence*, 23:29–58, 1984.

[Pie75] A. R. Pierce. Bibliography on algorithms for shortest path, shortest spanning tree, and related circuit routing problems (1956-1974). *Networks*, 5:129–149, 1975.

[Qui87] M.Q. Quinn. *Designing Efficient Algorithms for Parallel Computers.* McGraw-Hill, 1987.

[Ric83] Elaine Rich. *Artificial Intelligence.* McGraw-Hill, New York, 1983.

[RK88] V. Nageshwara Rao and V. Kumar. Concurrent insertions and deletions in a priority queue. In *Proc. 1988 Parallel Processing Conference*, 1988.