

An Exact Solution Procedure for Determining the Optimal Dispatching Times for Complex Rail Networks

Maged M. Dessouky*

Daniel J. Epstein Department of Industrial and Systems Engineering
University of Southern California
Los Angeles, CA 90089-0193
Phone: (213) 740-4891 Fax: (213) 740-1120
maged@usc.edu

Quan Lu

Oracle Corporation
Redwood Shores, CA 94065

Jiamin Zhao

Oracle Corporation
Redwood Shores, CA 94065

Robert C. Leachman

Department of Industrial Engineering and Operations Research
University of California, Berkeley
Berkeley, CA 94720
Phone: (510) 642-7054
leachman@ieor.berkeley.edu

**corresponding author*

An Exact Solution Procedure for Determining the Optimal Dispatching Times for Complex Rail Networks

Abstract: Trains operating in densely populated metropolitan areas typically encounter complex trackage configurations. To make optimal use of the available rail capacity, some portions of the rail network may consist of single-track lines while other locations may consist of double- or triple-track lines. This paper develops a branch-and-bound procedure for determining the optimal dispatching times for trains traveling in complex rail networks. We demonstrate the efficiency of our branch-and-bound algorithm by comparing it to CPLEX, a commercially available integer program solver, on an actual rail network in Los Angeles County.

1 Introduction

Rail companies are striving to maintain regular and reliable train service in order to compete with alternative modes of freight transportation. To achieve this, a common and expensive strategy is to increase the physical capability of the rail network. A more cost effective alternative is to improve the quality of the train routing and scheduling.

The train routing and scheduling problem is generally decomposed into the following three levels.

- (1) Freight train routing problem. It includes assigning an origin-destination pair of demands into *cars*, assembling and disassembling cars into *blocks*, grouping and ungrouping blocks into trains, and determining the routing and frequency of trains.
- (2) Scheduling problem. It addresses the problem of developing the operational timetables for both the freight trains created from the train routing problem and the passenger trains.
- (3) Dispatching problem. Although train timetabling addresses the temporal dimension of railroad operations from a tactical point of view, the detail movements of freight and passenger trains on the lines of the physical railway network still need to be precisely determined for real-time operations. Given a train timetable, the train dispatching problem is to determine a feasible plan of

meets and overtakes that minimizes train delays from the scheduled timetables while satisfying all the operational constraints. (Cordeau, Toth, and Vigo 1998)

Recently, the train dispatching problem has received attention in the research community. The reason is partially due to the fact that faster trains, increasing traffic density, and the availability of real-time information on train position and velocity have made the tasks of the dispatchers too complex and demanding to perform manually. A deficient dispatching strategy may cause a decrease of the railroad line capacity and service reliability and the increase of energy consumption and pollution to the environment, or in the most severe case lead to deadlock. Therefore, building a robust methodology to handle train dispatching in an optimal way is an important research issue in rail freight transportation. In this paper, we propose a new optimization approach to dispatch trains for a general railway trackage configuration, commonly found in urban networks.

The mathematical modeling of train dispatching operations dates back to the 1970's. Szpigel (1973) presents an integer-programming model, which seeks an optimal planning of meets and passes for trains on a single-track railway with fixed velocities.

Sauder and Westerman (1983) describe a partial enumeration scheme that generates the optimal meet-pass schedule for a single-track rail line. The model is proven to be very effective in practice and estimates to place a \$3 million annual savings for the Norfolk-Southern Railroad company.

Karry, Harker and Chen (1991) present a model that minimizes the total fuel consumption while maintaining the satisfaction of departure and arrival time windows. This is achieved by permitting trains to travel at a pacing speed less than the maximum velocity over each dispatcher's territory. An exact solution procedure is proposed for solving this model as well as alternative heuristics.

Jovanovic and Harker (1991) describe the conceptual underpinnings and the associated algorithms of an interactive freight traffic scheduling system: SCAN (Schedule Analysis) system. Their algorithm can generate a good feasible schedule based upon an existing feasible/infeasible schedule.

Carey and Lockwood (1995) describe a 0-1 mixed integer program model for train dispatching. The model considers multiple trains traveling at different speeds and in the same direction of a line. Hence, the main issue is the determination of the occurrence of the overtaking. The authors propose a heuristic that improves an initial solution iteratively by possibly re-dispatching individual trains. In a follow-up paper, Carey (1994a, 1994b) extends the original model by introducing line choices and two-way tracks.

Caprara, Fischetti and Toth (2002) consider the problem of determining the timetable for trains in a single track between two major stations, with a number of intermediate stations in between. Each train may have to stop at some intermediate stations and the meets and bypasses can only occur at these stations. They derive an integer linear programming model that is relaxed in a Lagrangian way. The relaxation is embedded within a heuristic algorithm, which makes extensive use of the dual information associated with the Lagrangian multipliers.

For a more comprehensive review of the literature dealing with the optimization models for train dispatching, refer to an excellent review article by Cordeau, Toth, and Vigo (1998).

In general, the train dispatching problem involves a huge number of precedence conditions. Solving it optimally using mathematical programming may require the introduction of many logic variables (e.g. Higgins et al., 1995) and makes the real problem computationally intractable. Thus, some authors resort to using discrete event simulation modeling to schedule the trains. In a discrete event simulation system, at each time instant, we can only dispatch the trains based on the current system state since we do not know the event that will occur beyond this time instant. Dorfman and Medanic (2004) use a discrete event simulation model to solve the train scheduling problem. The model uses a capacity check algorithm to avoid the deadlocks and defines some special engagement rules to improve the dispatching efficiency in some predefined situations. The approach can quickly handle perturbations in the schedule and is shown to perform well on three time-preference criteria. Lu, Dessouky, and Leachman (2004) develop an efficient deadlock free dispatching heuristic embedded in a simulation environment that can model general rail networks.

The main issue in train dispatching is how to dispatch trains while maximally utilizing the trackage capacity and avoiding deadlocks. There are two different types of deadlock-eliminating approaches: deadlock prevention and deadlock avoidance. The deadlock prevention approach statically establishes a control policy. Once established, it guarantees that no deadlock can occur by ensuring that at least one necessary condition for forming deadlock can never hold. The deadlock avoidance approach dynamically identifies whether a state that the system can evolve at this point is safe or not. The deadlock avoidance approach will not let the system enter into any not safe state at any point. A safe state is a state from which the system can start to end without encountering any deadlock. For example, the difference between the deadlock prevention and deadlock avoidance approaches is similar to the difference between a traffic light and a police officer directing traffic. In this paper, we focus on the design of a deadlock prevention approach.

All the above literature except for the paper by Lu, Dessouky, and Leachman (2004) consider generating dispatching schedules on a single-track or double-track line without crossovers, because most of the United States' and world's freight rail network consists of only these types of trackage. However, in some metropolitan areas, the rail network can have a complex configuration with some triple-track segments and complex junction intersections. For example, in the Southern California area, to meet the increasing freight traffic originating from the Ports of Los Angeles and Long Beach, it is not uncommon to find segments containing multiple mainline trackage. In this paper, our purpose is to design an efficient train dispatching algorithm that minimizes the total travel time of trains while satisfying the timetable and assuring no deadlock occurs for a general railway trackage. As opposed to the earlier work by Lu, Dessouky, and Leachman (2004), which focused on a simulation based approach, this paper focuses on the development of an exact solution procedure.

We first give a formal definition of the problem in Section 2. Then, we propose a branch-and-bound based algorithm in Section 3 that optimally solves the problem. Section 4 provides computational experiments and Section 5 presents the conclusions.

2 Problem Description

In this section, we provide a formal description of the train dispatching problem. Given a railway network that consists of main tracks, sidings, junctions, and platforms, it can be converted to a general network $G = (N, A)$, where N is the node set and A is the arc set. (see e.g. Lu, Dessouky, and Leachman 2004). Each node $j \in N, j = 1, 2, \dots, |N|$ contains a set of segment resources and a set of junction resources and denote L_j to represent the length in miles of the node. Each arc $a \in A$ contains no segment or junction resources and the length of the arc is always equal to zero. Define $R = \{R_1, R_2, \dots, R_{|N|}\}$ the set of all resources, where $R_j \in R, j = 1, 2, \dots, |N|$, represents all the segment and junction resources at node j . Without loss of generality, we assume that $R_j \cap R_{i \neq j} = \phi$, for all $R_i, R_j \in R$. In Figure 1, we show an example of how to translate an actual track configuration into network G .

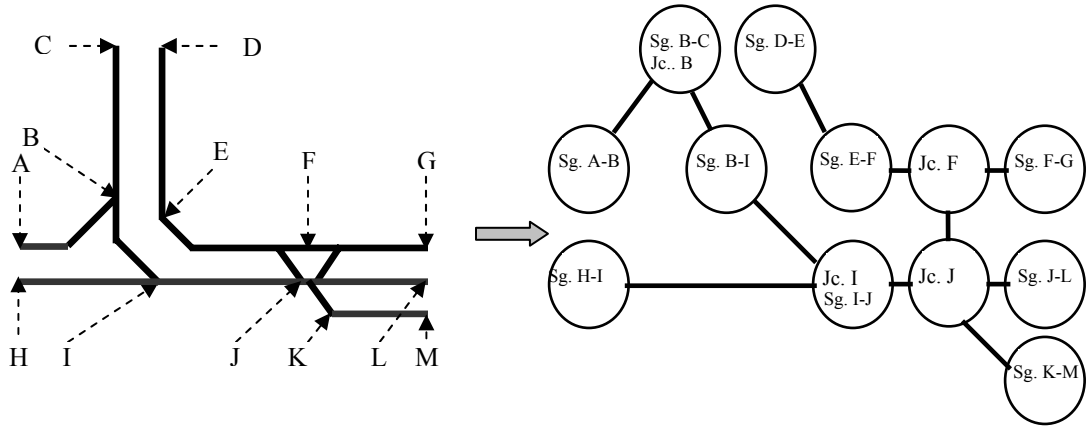


Figure 1. Transfer Trackage to General Network

Let set Q denote all the trains to be scheduled. For each train $q \in Q, q = 1, 2, \dots, |Q|$, let S_q be the length of train q and P_q be the path from train q 's origin node, node n_q^o , to train q 's destination node, node n_q^d . All the nodes train q encounters in sequence from node n_q^o to n_q^d in path P_q are: $\{n_{q,1}, n_{q,2}, \dots, n_{q,|P_q|}\}$, where $n_{q,1} = n_q^o$ and $n_{q,|P_q|} = n_q^d$. Hence, all the resources train q encounters in sequence in path P_q are $R_{n_{q,1}}, R_{n_{q,2}}, \dots, R_{n_{q,|P_q|}}$.

Define $\underline{T}_{q,i}$ the earliest time that train q 's head can arrive at the i^{th} node (node $n_{q,i}$) in its path P_q and $\overline{T}_{q,i}$ the latest time that train q 's tail has to leave node $n_{q,i}$ to assure that train q can arrive at its destination before the latest arrival time in the timetable. Furthermore, define $B_{q,i}^1$ the minimal travel time between train q 's head entering into node $n_{q,i}$ and train q 's head leaving from node $n_{q,i}$ to node $n_{q,i+1}$. Define $B_{q,i}^2$ the minimal travel time between train q 's head entering into node $n_{q,i}$ and train q 's tail leaving node $n_{q,i}$. In Figure 2, we show the relationship between these four variables for a simple example.

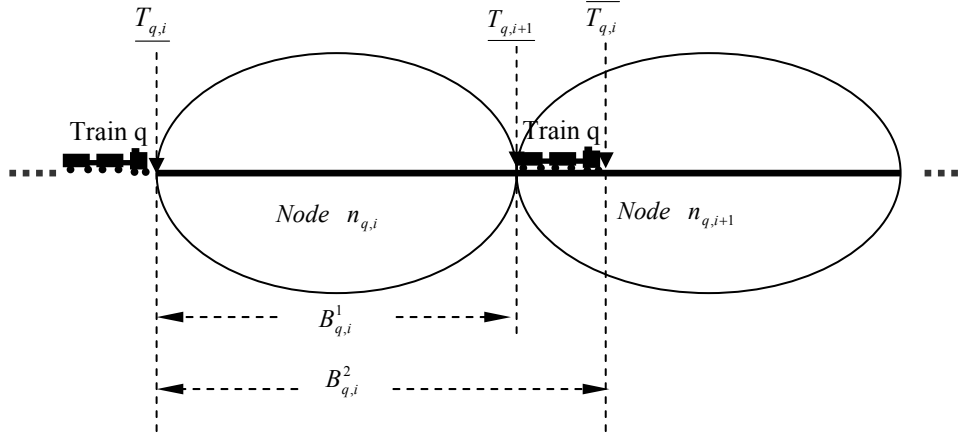


Figure 2. Relationship between Variables

Let \underline{T}_q be the scheduled earliest time for train q to start from its origin node n_q^o and \overline{T}_q be the scheduled latest time for train q to arrive and stop at its destination node n_q^d . Constants \underline{T}_q and \overline{T}_q are predetermined by the timetables. Since the difference between \underline{T}_q and $\underline{T}_{q,1}$ and the difference between \overline{T}_q and $\overline{T}_{q,|P_q|}$ are both constants, we assume $\underline{T}_{q,1} = \underline{T}_q$ and $\overline{T}_{q,|P_q|} = \overline{T}_q$ for all trains $q \in Q$ to simplify the formulation.

Define $t_{q,i}^a$ as the time that train q 's head arrives at the i^{th} node in its path P_q , and $t_{q,i}^d$ as the time that train q 's tail leaves from the i^{th} node in its path P_q . A zero-one decision variable $x_{q_1,q_2,k}$ is defined to represent the sequence of two trains q_1 and q_2 passing the same node k , $k \in N$.

$$x_{q_1,q_2,k} = \begin{cases} 1 & \text{if train } q_1 \text{ passes node } k \text{ before train } q_2 \\ 0 & \text{if train } q_2 \text{ passes node } k \text{ before train } q_1 \end{cases}$$

We use μ to represent the minimal safety headway between two consecutive trains and M is a big number set as $M = \max\{\overline{T}_q \mid q \in Q\}$. Then the problem can be described as a 0-1 mixed integer programming problem as follows:

$$\min \sum_{q \in Q} t_{q,|P_q|}^a \quad (1)$$

s.t.

$$t_{q,i+1}^a - t_{q,i}^a \geq B_{q,i}^1, \quad \text{for all } q \in Q \text{ and } 1 \leq i \leq |P_q| - 1 \quad (2)$$

$$t_{q,i}^d - t_{q,i+1}^a \geq B_{q,i}^2 - B_{q,i}^1, \quad \text{for all } q \in Q \text{ and } 1 \leq i \leq |P_q| - 1 \quad (3)$$

$$t_{q,|P_q|}^d - t_{q,|P_q|}^a \geq B_{q,|P_q|}^2, \quad \text{for all } q \in Q$$

$$x_{q_1,q_2,k} M + t_{q_1,i}^a \geq t_{q_2,j}^d + \mu, \quad \text{for all } q_1, q_2 \in Q \text{ and node } k = n_{q_1,i} = n_{q_2,j} \quad (4)$$

$$(1 - x_{q_1,q_2,k}) M + t_{q_2,j}^a \geq t_{q_1,i}^d + \mu, \quad \text{for all } q_1, q_2 \in Q \text{ and node } k = n_{q_1,i} = n_{q_2,j} \quad (5)$$

$$x_{q_1,q_2,k} \in \{0,1\}, \quad \text{for all } q_1, q_2 \in Q \text{ and } 1 \leq k \leq |N| \quad (6)$$

Objective function (1) minimizes the sum of all the trains' arrival times to their destination nodes. Constraint (2) ensures that the arrival time for each train at each node in its path must not be less than the corresponding minimal travel time interval in the same portion of the path. Constraint (3) ensures the minimum time that a train needs to completely leave the previous node, once it begins to enter the next node. Constraints (4) and (5) enforce the rule that no two trains can simultaneously occupy the same node.

Thus the actual headway between two trains running along the same path in the same direction is forced to be $\mu + B_{q,i}^1$ if train q is the latter one to pass node $n_{q,i}$. To make the model plausible, we have to assume that the minimum length of all nodes is no less than the maximum length of all the trains. This assumption guarantees that a train can only occupy at most two nodes at a time. This implies that the departure time at any node is later only than the arrival time at the immediate next node but is earlier than the arrival time at any other successive nodes in the route.

3 Branch-and-Bound Algorithm

In this section, we describe a branch-and-bound procedure that optimally solves the train dispatching problem. First, denote set $\Pi_j = \{q \mid q \in Q \text{ and } j \in P_q\}$ for each node $j, j \in N$. That is, Π_j includes all the trains that have node j in their paths. The basic idea behind our branch-and-bound scheme is the “sequence-fixing” of all the trains in set Π_j at each node j . Since a schedule is a set of orderings of passing trains at each node, a natural way to compute it is to fix the sequence of all the trains passing the same node step by step. To make a distinction between a node in the rail network and in the branch-and-bound tree, we refer to a node in the branch-and-bound tree as a *search node*. Each search node h branches w_h child search nodes. Each of h 's child search nodes further fixes the trains' sequence at a node $i, i \in N$, where the trains' sequence at node i is not uniquely determined in the parent search node h .

3.1 Rule and Sequence Graph

We assume that the nodes are not sharable (i.e., the same node cannot be simultaneously occupied by multiple trains). For example, if two trains q_1 and q_2 travel the same node $j, j \in N$, then either train q_1 passes node j before train q_2 , denoting it as $q_1 \rightarrow q_2$, or train q_2 passes node j before train q_1 , denoting it as $q_2 \rightarrow q_1$. We call $q_1 \rightarrow q_2$ or $q_2 \rightarrow q_1$ a rule at node j . Denote W_j the set of all the rules currently at node j . There are two ways to add a new rule: (1) deduce the new rule from some existing rule or (2) create it from a domain specific constraint in the rail network. For example, there may be a

constraint that gives priority to one train type over another when passing a junction. In this paper, we will not consider any domain specific constraint. We note that domain specific constraints may be able to reduce the size of the search tree and dramatically accelerate the solution time of the algorithm. For any two trains q_1 and q_2 passing node j , if rule $q_1 \rightarrow q_2 \in W_j$ and there exist two or more other rules in W_j that can also induce $q_1 \rightarrow q_2$, then rule $q_1 \rightarrow q_2$ is called a *redundant* rule for W_j . We assume that W_j contains no redundant rules.

We use an acyclic graph called sequence graph $GW_j = (NW_j, AW_j)$ to represent the rule set W_j at each node $j \in N$, where NW_j and AW_j is the node set and the arc set of GW_j , respectively. For each train q , $q \in \Pi_j$, define two nodes $m_{q,j}^a$ and $m_{q,j}^d$, and add a directed arc from node $m_{q,j}^a$ to node $m_{q,j}^d$, where $m_{q,j}^a$ represents the arrival event of train q 's head to node j and $m_{q,j}^d$ represents the departure event of train q 's tail from node j . Therefore, the graph GW_j has $2|\Pi_j|$ number of nodes. If there is a non-redundant rule $q_1 \rightarrow q_2 \in W_j$, then in graph GW_j there exists a corresponding arc from node $m_{q_1,j}^d$ to node $m_{q_2,j}^a$, and vice versa. Figure 3 shows an example of a sequence graph. The graph in Figure 3 represents the partial sequence of six trains passing node j . This partial sequence is determined by the four rules shown in Figure 3.

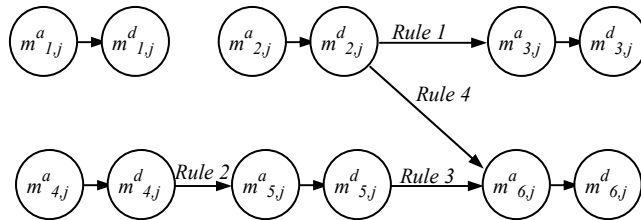


Figure 3. Sequence Graph GW_j at Node j with Six Trains and Four Rules

3.2 Propagation of Rules

In order to efficiently prune the search space, as many as possible new rules need to be generated at each node of the search tree. We say that the total rule set W is *safe* iff there exists a solution that satisfies all the rules in W and can complete all the trains' movements without a deadlock. Otherwise, W is *unsafe*. W is *completely safe* iff W is safe and adding one or more rules to any $W_j \in W$, which are not conflicting with any existing rule in W_j , will not make W unsafe. The characteristic of a completely safe rule set is very important regarding to the measurement of the rule propagation's quality. If a rule set W is safe but not completely safe, there exists at least a rule $q_1 \rightarrow q_2 \in W_j$ and $W_j \in W$, that adding $q_1 \rightarrow q_2$ to W will make W unsafe. Therefore, rule $q_2 \rightarrow q_1$ is a rule that can ideally be generated and added to W .

Proposition 1. Given a rule set W , it is NP-hard to determine the safety of W .

Proof. Lawley and Reveliotis (2001) prove that the Single-Unit Safe Problem (SU-SAFE) is NP-complete. We now show that SU-SAFE is polynomial-time reducible to our rule set safety problem. The SU-SAFE problem is defined as follows: for a finite set of resources and processes and each process requires a sequence of resources, is it possible to complete every process from a given set of partially completed processes?

Given a set of partially completed processes, we can generate the rules to represent all the completed portion for each process in polynomial time. Then, if the SU-SAFE problem is safe, there exists a sequence of resource allocations to complete all the processes; from the definition, the corresponding rule set problem is safe. If the SU-SAFE problem is unsafe, then it is impossible to complete all the processes and the corresponding rule set problem is unsafe. \square

Proposition 2. For a safe rule set W , there does not exist a polynomial-time algorithm that can make it completely safe.

Proof. We prove this statement by contradiction. Assume that there exists a polynomial-time algorithm Θ that can make any safe rule set completely safe. Apply Θ to a given rule set W and obtain a new rule set W^l . W^l is either unsafe or completely safe. For any

two trains $q_1 \in \Pi_j$ and $q_2 \in \Pi_j$, and neither $q_1 \rightarrow q_2$ nor $q_2 \rightarrow q_1$ is a rule or a redundant rule to W_j^1 , add rule $q_1 \rightarrow q_2$ or $q_2 \rightarrow q_1$ to W_j^1 and obtain a rule set W^2 . Apply Θ to W^2 to get rule set W^3 . If W^1 is completely safe, from the definition of the completely safe, W^2 is safe. Thus, W^3 is completely safe. Repeat the above operations by finding another pair of trains, which pass the same node and their sequence of passing this node has not been determined, until the sequence of all the trains at each node is fixed. If there are n nodes and m trains in the rail network, at most $nm^2/2$ number of rules is needed to fix the sequence of all the trains at each node. Thus, the whole procedure is polynomial-time and this polynomial-time procedure can be used to check the safety of rule set W , which is a contradiction of Proposition 1. Hence, there does not exist a polynomial-time algorithm that can ensure that a rule set is completely safe. \square

It would clearly be more efficient if we could identify and prune all the created unsafe child search nodes and generate as many as possible rules for all the created safe child search nodes. However, Propositions 1 and 2 tell us that this cannot be done in polynomial-time. Hence, we resort to heuristics to partially identify unsafe child search nodes and to find all the rules that can make a safe child search node completely safe. Applying these procedure speed up the search for the optimal solution and we discuss them next.

3.3 Adjacent Propagation

Proposition 3. Let node j and k be two adjacent nodes in graph G . Trains q_1 and q_2 belong to the set $\Pi_j \cap \Pi_k$. If $q_1 \rightarrow q_2 \in W_j$, then $q_1 \rightarrow q_2 \in W_k$ or $q_1 \rightarrow q_2$ is a redundant rule for W_k .

Proof. If train q_1 and q_2 run in the same direction, without loss of generality, we assume that they run from node j to node k , then we have $q_1 \rightarrow q_2 \in W_j \Rightarrow t_{q_1, n_k}^a \leq t_{q_1, n_j}^d < t_{q_2, n_j}^a \leq t_{q_2, n_k}^a \Rightarrow t_{q_1, n_k}^a < t_{q_2, n_k}^a \Rightarrow q_1 \rightarrow q_2$ must be satisfied at node k .

If train q_1 and q_2 run in the opposite direction, we assume that train q_1 runs from node j to node k and train q_2 runs from node k to node j , then we have $q_1 \rightarrow q_2 \in W_j \Rightarrow t_{q_1, n_k}^a \leq t_{q_1, n_j}^d < t_{q_2, n_j}^a \leq t_{q_2, n_k}^d \Rightarrow t_{q_1, n_k}^a < t_{q_2, n_k}^d \Rightarrow q_1 \rightarrow q_2$ must be satisfied at node k . \square

Adjacent propagation is simple and straightforward and it can be used to quickly propagate rules.

3.4 Feasibility Propagation

Feasibility Propagation is based on a graph called Total Rule Graph TGW . TGW is generated by connecting all the rule graphs at each node j , GW_j , $j=1, 2, \dots, |N|$, with directed arcs based on the following rules. First, for each train q , we sort its arrival and departure events at each node along its path according to the sequence of their occurrence. Then, for every two adjacent events in the sorting list of each train q , connect the corresponding nodes in GW_j by a directed arc. For example, if train q arrives first to node 2 and then leaves node 1, add a directed arc from node $m_{q,1}^a$ in GW_1 to node $m_{q,2}^a$ in GW_2 .

For any node k and two trains $q_1, q_2 \in \Pi_k$, if $q_1 \rightarrow q_2$ is a *foreign* rule of W_k in Graph TGW , the following conditions must hold.

- (1) There exists a directed path from node $m_{q_1, k}^a$ or $m_{q_1, k}^d$ to node $m_{q_2, k}^a$ or $m_{q_2, k}^d$.
- (2) This path contains arcs that represent rules not belonging in W_k ,
- (3) $q_1 \rightarrow q_2 \notin W_k$ and $q_1 \rightarrow q_2$ is not a redundant rule of W_k .

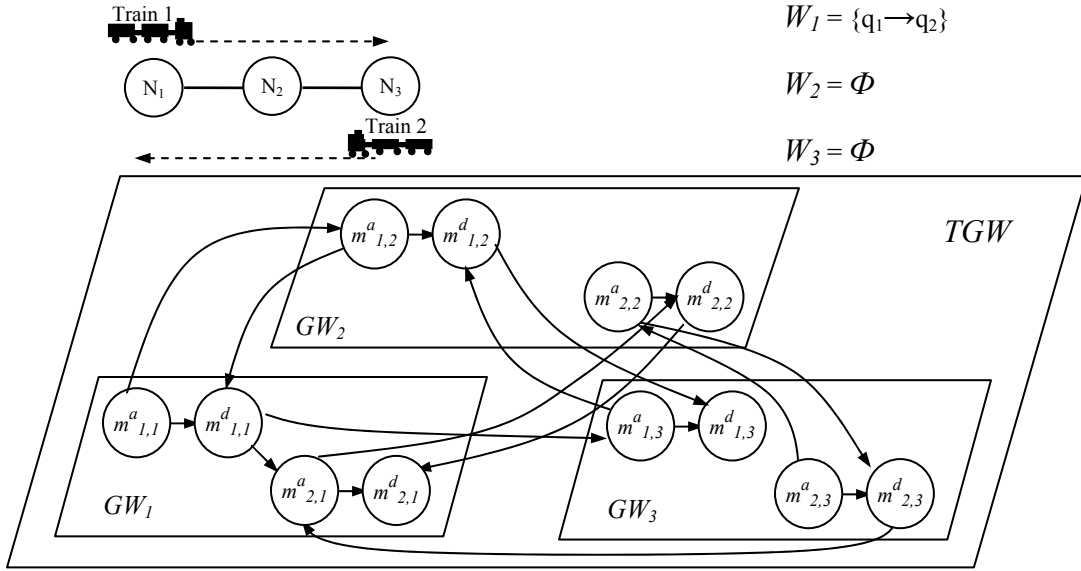


Figure 4. TGW and Sequence Graphs

The purpose of identifying foreign rules is that they allow us to generate other rules that can be added to rule set W to reduce the search space. For example, Figure 4 shows a problem with two trains q_1 and q_2 and three nodes. Train q_1 leaves from node 1 to node 3 through node 2, and train q_2 leaves from node 3 to node 1 through node 2. Assume that $W_1 = \{q_1 \rightarrow q_2\}$, $W_2 = W_3 = \Phi$, the total rule graph TGW and sequence graph GW_1 , GW_2 , and GW_3 are shown in Figure 4, and path $m_{1,2}^a, m_{1,1}^d, m_{2,1}^a, m_{2,2}^d$ is a foreign rule for W_2 . From this foreign rule, we can deduce and add another rule $q_1 \rightarrow q_2$ to W_2 .

The basic idea behind feasible propagation is: when a rule is added into W , we eliminate all the new created foreign rules by finding and adding some new rules to W . These new rules can be considered as the rules propagated from the foreign rules. The whole feasible propagation procedure of adding a rule λ_0 into rule set W is as follows.

Step 0: Let set $S = \{\lambda_0\}$.

Step 1: If $S = \text{NULL}$, stop. All foreign rules have been eliminated. Otherwise, arbitrarily choose one rule λ_j from set S .

- Step 2: Find all the foreign rules generated by adding rule λ_j and store them into set F .
- Step 3. If $F = \text{NULL}$, go to Step 1. Otherwise, arbitrarily choose one foreign rule from F and eliminate it by adding a rule λ_k to W . If adding rule λ_k creates a cycle in graph TGW , stop. The rule set W is unsafe.
- Step 4. Add λ_k to set S . Go to Step 1.

Property 3. Feasibility Propagation does not change the safety of the rule set.

Proof: Given a rule set W , assume that the rule set after performing the foreign rule elimination on W is set W' . If W is unsafe, then W' is unsafe because adding more rules to an unsafe rule set W will not make it safe.

To prove that if W is safe then W' is safe, we only need to show that adding any new rule to eliminate an existing foreign rule $q_1 \rightarrow q_2$ in GW_j will not change W from safe to unsafe. This holds because a schedule δ with proper arrival and departure times for each train at each node that completes all the train's movements can be found since W is safe. Assume that train q_1 and q_2 both need to pass node j , we have either $m_{q_1,k}^a < m_{q_1,k}^d < m_{q_2,k}^a < m_{q_2,k}^d$ or $m_{q_2,k}^a < m_{q_2,k}^d < m_{q_1,k}^a < m_{q_1,k}^d$ in schedule δ . On the other hand, since $q_1 \rightarrow q_2$ is a foreign rule in GW_j , from the definition of a foreign rule, we have $m_{q_1,k}^a < m_{q_2,k}^a$ or $m_{q_1,k}^a < m_{q_2,k}^d$ or $m_{q_1,k}^d < m_{q_2,k}^a$ or $m_{q_1,k}^d < m_{q_2,k}^d$. Hence, in schedule δ , $m_{q_1,k}^a < m_{q_1,k}^d < m_{q_2,k}^a < m_{q_2,k}^d$. Thus, schedule δ is a feasible schedule for the rule set obtained by adding rule $q_1 \rightarrow q_2$. Therefore, a safe rule set remains safe by adding any rule to eliminate existing foreign rules. \square

Property 3 tells us that applying Feasibility Propagation to a rule set will not change the safety of the rule set. Note that the previous Adjacent Propagation can be included in Feasibility Propagation. However, since the procedure of Adjacent Propagation is very simple and quick, we use it to accelerate the total propagation procedure before applying Feasibility Propagation.

Next we give an example of using Adjacent Propagation and Feasibility Propagation. Assume that there is a triangle like trackage configuration (a similar trackage configuration can be found in a railway network near the West Colton yard in Los

Angeles) with three trains, train 1 is moving from node 1 to node 3; train 2 is moving from node 2 to node 1; and train 3 is moving from node 3 to node 2. Assume that the length of each train is shorter than the length of each single node. The current rule set W and the rule set W' after the propagation are shown in Figure 5, respectively.

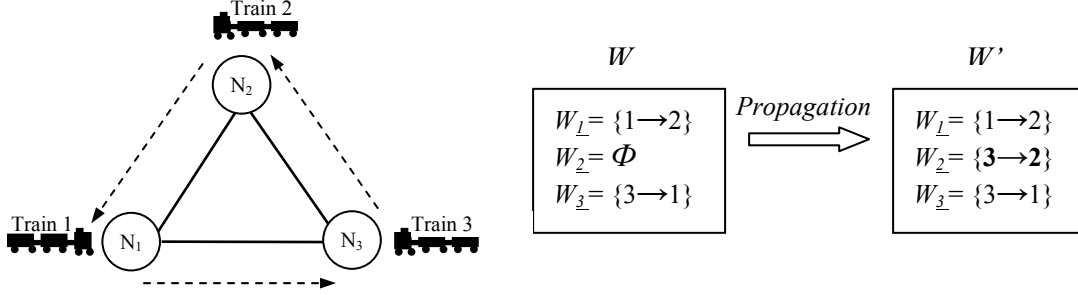


Figure 5. Generate New Rules by Propagation

3.5 Time Window Adjustment

Assume that a new rule $q_1 \rightarrow q_2$ is added at node k , where node k is the i^{th} node in train q_1 's path and the j^{th} node in train q_2 's path. Besides propagating new rules, adding rule $q_1 \rightarrow q_2$ may also change the earliest arrival time and the latest leaving time of some trains at some of the nodes. Next, we discuss how to efficiently address these changes.

First, adding rule $q_1 \rightarrow q_2$ may increase train q_2 's earliest arrival time at node k . That is, $\underline{T}_{q_2,j}' = \max(\underline{T}_{q_2,j}, \underline{T}_{q_1,i} + B_{q_1,i}^2 + \mu)$ and may decrease train q_1 's latest leaving time at node k . That is, $\overline{T}_{q_1,i}' = \min(\overline{T}_{q_1,i}, \overline{T}_{q_2,j} - B_{q_2,j}^2 - \mu)$. If $\underline{T}_{q_2,j}' \neq \underline{T}_{q_2,j}$, we adjust the earliest arrival time for the trains at the nodes according to the following equations:

$$\begin{aligned} \underline{T}_{q_2,j+1}' &= \max(\underline{T}_{q_2,j+1}, \underline{T}_{q_2,j} + B_{q_2,j}^1), & \text{for any train } q_k \text{ that node } n_{q_2,j} \\ & & \text{is the } l^{\text{th}} \text{ node in its path and} \\ \underline{T}_{q_k,l}' &= \max(\underline{T}_{q_k,l}, \underline{T}_{q_2,j} + B_{q_2,j}^2 + \mu), & q_2 \rightarrow q_k \text{ is a rule at node } n_{q_2,j}. \end{aligned} \quad (7)$$

Similarly, if $\overline{T}_{q_1,i}' \neq \overline{T}_{q_1,i}$, we adjust the latest leaving time for the trains at the nodes according to the following equations:

$$\begin{aligned}
\overline{T_{q_1, j-1}}' &= \min(\overline{T_{q_1, j-1}}, \overline{T_{q_1, j}} - B_{q_1, j}^1), & \text{for any train } q_k \text{ that node } n_{q_1, j} \\
& & \text{is the } l^{\text{th}} \text{ node in its path and} \\
\overline{T_{q_k, l}}' &= \min(\overline{T_{q_k, l}}, \overline{T_{q_1, j}} - B_{q_1, j}^2 - \mu), & q_k \rightarrow q_1 \text{ is a rule at node } n_{q_1, j}.
\end{aligned} \tag{8}$$

Repeat applying the above equations (7) and (8) until there are no more changes in the earliest arrival time or the latest leaving time for any train at any node. If there exists a train q , whose earliest arrival time is greater than the latest leaving time at the j^{th} node in its path (i.e., $\underline{T_{q, j}} + B_{q, j}^2 \geq \overline{T_{q, j}}$) then the rule set W is unsafe (infeasible) regarding to the time window.

3.6 Overall Solution Algorithm

First, we need to set the minimal traveling time $B_{q, i}^1$ and $B_{q, i}^2$ for each train q at the i^{th} node along its path. If the maximal speed of train q is V_q , then $B_{q, i}^1 = L_{n_{q, i}} / V_q$ and $B_{q, i}^2 = (L_{n_{q, i}} + S_q) / V_q$.

Then the initial earliest arrival and latest leaving time for each train q in the i^{th} node along its path are:

$$\underline{T_{q, i}} = \underline{T_q} + \sum_{j=1}^{i-1} B_{q, j}^1 \tag{9}$$

$$\overline{T_{q, i}} = \overline{T_q} - \sum_{j=i+1}^{|P_q|} B_{q, j}^1 + B_{q, i}^2$$

(10)

We create the initial rule set W , if $\underline{T_{q_1, i}} + B_{q_1, i}^2 + \mu > \overline{T_{q_2, j}}$ by adding rule $q_2 \rightarrow q_1$ to W_k for any two trains q_1 and q_2 passing node k , where node k is the i^{th} node in train q_1 's path and the j^{th} node in train q_2 's path. If $\underline{T_{q_2, i}} + B_{q_2, i}^2 + \mu > \overline{T_{q_1, j}}$, we add rule $q_1 \rightarrow q_2$ to W_k . We next apply Algorithm 3.1 to propagate new rules and tighten the time windows for rule set W , while setting set $C_r = W$ and set $C_t = \emptyset$.

Algorithm 3.1

While $C_r \neq \emptyset$ and $C_l \neq \emptyset$

 If $C_r \neq \emptyset$

 Get a rule λ , $\lambda \in C_r$.

 Apply adjacent propagation for rule λ and add all the propagated rules to set C_a .

 For each rule λ_a , $\lambda_a \in C_a$

 Add λ_a to rule set W . If this operation causes a cycle in the total rule graph TGW , W is unsafe and stop.

 Add λ_a to C_r . If adding λ_a causes a change in the earliest arrival time or the latest leaving time, add the new times to set C_l .

 Apply feasible propagation for rule λ and add all the propagated rules to set C_a .

 For each rule λ_a , $\lambda_a \in C_a$

 Add λ_a to rule set W . If this operation causes a cycle in the total rule graph TGW , W is unsafe and stop.

 Add λ_a to C_r . If adding λ_a causes a change in the earliest arrival time or the latest leaving time, add the new times to set C_l .

 If $C_l \neq \emptyset$

 Get a changing time η , $\eta \in C_l$.

 If η is a change in the earliest arrival time, use equation (6) to change all the earliest arrival time and add them to set C_e .

 For each changing time $\underline{T}_{q_1,i}$, $\overline{T}_{q_1,i} \in C_e$

 If $\underline{T}_{q_1,i} + B_{q_1,i}^2 \geq \overline{T}_{q_1,i}$, stop. W is unsafe.

 If there exists a train q_2 and $\underline{T}_{q_1,i} + B_{q_1,i}^2 + \mu > \overline{T}_{q_2,j}$, add a rule $q_2 \rightarrow q_1$ to C_r .

 If η is a change in the latest leaving time, use equation (7) to change all the latest leaving time and add them to set C_r .

 For each changing time $\overline{T}_{q_1,i}$, $\underline{T}_{q_1,i} \in C_l$

 If $\overline{T}_{q_1,i} + B_{q_1,i}^2 \geq \underline{T}_{q_1,i}$, stop. W is unsafe.

 If there exist a train q_2 and $\overline{T}_{q_1,i} + B_{q_1,i}^2 + \mu > \underline{T}_{q_2,j}$, add a rule $q_1 \rightarrow q_2$ to C_r .

At each search node, the upper bound is set as the best known feasible schedule. The lower bound for rule set W at the current search node is equal to the sum of the earliest leaving time of all the trains at their destination node, $\sum_{q \in Q} (\underline{T}_{q,|P_q|} + B_{q,|P_q|}^2)$.

Whenever some new rules are added to the rule set W (e.g. creating the child node in the search tree), Algorithm 3.1 is called to perform the propagation and time window adjustments.

Finally, we briefly describe how to create the actual schedule if the sequence of all the trains at each node is fixed. For each train q_l at the j^{th} node in its path, there are two times need to be decided: $t_{q_l,i}^a$ the time train q_l 's head arrives to node $n_{q_l,i}$ and $t_{q_l,i}^d$ the time train q_l 's tail leaves from node $n_{q_l,i}$. They are determined using equations (11) and (12). In these equations, q_2 is the immediate predecessor train passing node k before train q_l , if train q_l is not the first train to pass node k . Assume that node k is the i^{th} node in train q_l 's path and the j^{th} node in train q_2 's path. Train q_l 's tail leaves node k when train q_l 's head is in the i' node in its path, $i' > i$ (i' does not necessarily have to equal $i+l$).

$$t_{q_l,n_{q_l,i}}^a = \begin{cases} \overline{T}_q & \text{if } i = 1 \text{ and } q_2 = \phi \\ t_{q_l,n_{q_l,i-1}}^a + B_{q_l,n_{q_l,i-1}}^1 & \text{if } i > 1 \text{ and } q_2 = \phi \\ \max\{t_{q_l,n_{q_l,i-1}}^a + B_{q_l,n_{q_l,i-1}}^1, t_{q_2,n_{q_2,j}}^d + \mu\} & \text{if } i > 1 \text{ and } q_2 \neq \phi \end{cases} \quad (11)$$

$$t_{q_l,n_{q_l,i}}^d = t_{q_l,n_{q_l,i'}}^a + B_{q_l,n_{q_l,i}}^2 - \sum_{h=i}^{i'-1} B_{q_l,n_{q_l,h}}^1 \quad (12)$$

If node k is not the origin node in train q_l 's path and train q_l is not the first train to pass node k , from equation (11), $t_{q_l,i}^a$ is determined by the maximal time between train q_l 's earliest arrival time to node k from the predecessor node in its path and train q_2 's leaving time from node k plus the minimal safety headway, where train q_2 is the immediate predecessor train passing node k . Equations (11) and (12) are repeatedly applied until all the arrival and leaving times are determined.

4 Computational Results

In this section, we compare our proposed branch-and-bound algorithm against CPLEX 8.0 on a portion of the railway network near downtown Los Angeles. The

network is shown on Figure 6. Note that the network is a mixture of double and triple track.

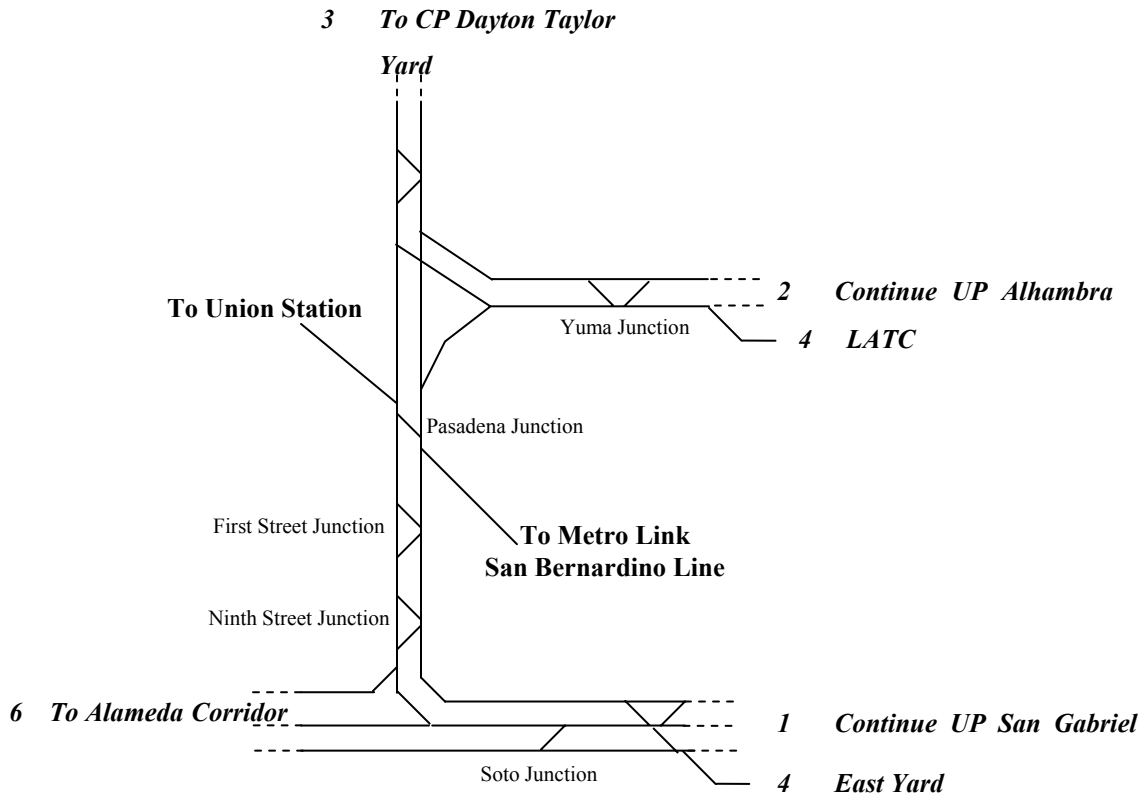


Figure 6. A Portion of the Rail Network Near Downtown Los Angeles

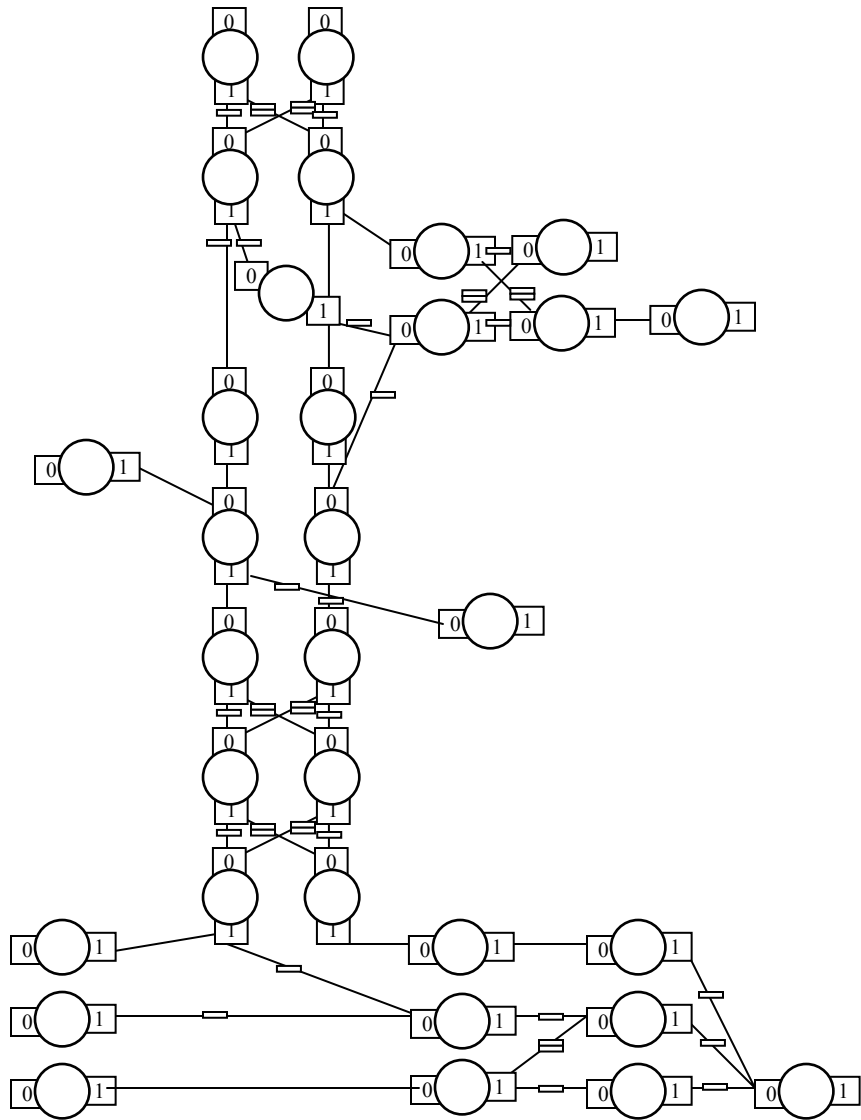


Figure 7. Network Translation for the Rail Network Shown in Figure 6

The network translation for the above rail network is shown in Figure 7. Note that the translated network consists of 33 nodes. We consider dispatching freight trains along routes specified in Table 1, one train per route. All trains are assumed ready at their origins at time zero. For each route, we specify the origin and destination and the number of the nodes in the route, the length of the route, the train length and the train speed. The train speed varies in some routes since each node has a different speed limit and the values in the table represent the speed limit range for the nodes in the route.

Route ID	Origin	Destination	Number of Nodes	Length of Route	Train Length	Train Speed (mph)
1, 13	CP Dayton Taylor Yard	Alameda Corridor	7	11.16 mi	1.17 mi	10 ~ 30
2,14	Alameda Corridor	CP Dayton Taylor Yard	7	11.16 mi	1.17 mi	10 ~ 30
3	LATC	Alameda Corridor	7	11.02 mi	1.17 mi	10 ~ 30
4	Alameda Corridor	LATC	7	11.02 mi	1.17 mi	10 ~ 30
5	CP Dayton Taylor Yard	LATC	3	4.56 mi	1.00 mi	10
6	LATC	CP Dayton Taylor Yard	3	4.56 mi	1.00 mi	10
7	Union Station	Metro Link San Bernardino Line	2	3.04 mi	0.92 mi	10
8	Metro Link San Bernardino Line	Union Station	2	3.04 mi	0.92 mi	10
9	CP Dayton Taylor Yard	East Yard	7	11.08 mi	0.67 mi	10 ~ 30
10	East Yard	CP Dayton Taylor Yard	6	9.56 mi	0.83 mi	10 ~ 30
11	East Yard	Alameda Corridor	4	6.20 mi	0.67 mi	10 ~ 20
12	Alameda Corridor	East Yard	4	6.20 mi	0.83 mi	10 ~ 20

Table 1. Route Data Specification I

Our experiments were conducted on a Linux server with a 3.06GHz Intel® Xeon™ CPU. We use a stopping rule of 2 CPU hours. We first solve the optimal dispatching problem with the first six routes. Then we increase the problem size by including the seventh route, and so on. The largest problem that was solvable within the 2 hours CPU limit is with 14 routes using our branch-and-bound algorithm. We also solve the problems using CPLEX 8.0 and compare the performance of the two approaches. The CPLEX settings that were used are summarized in Table 2.

Name	Value	Meaning
Mip.Emphasis	0	Balance optimality and integer feasibility
Mip.Cuts.All	0	Automatically choose an appropriate algorithm to find cuts
Mip.Strategy.Subalgorithm	2	Use dual simplex to solve sub-problems
Strategy.Backtrack	.9999	A value close to one makes it more likely to move deeper in the search tree than to backtrack
Strategy.Nodeselect	1	Use the “best-bound” search when selecting the next node when backtracking
Strategy.Variableselect	0	CPLEX determines which variable to branch
Timelimit	7200	Stop after 2 hours if no optimal solution is found

Table 2. CPLEX Settings

The results are shown in Table 3. In the table our branch-and-bound algorithm is referred to as FP. The first two columns describe the problem size (the number of involved routes and used nodes); columns 3 to 5 describe the MIP problem size (the number of integer variables (IV), continuous variables (CV), and constraints (Cnstr)); columns 6 to 9 compare the algorithm performance in terms of the number of explored branch-and-bound nodes. The last two columns show the final objective value obtained for each procedure. For the 14 route problem, CPLEX was not able to find the optimal solution within the two hours CPU limit. In this case we list the lower and upper bounds given by CPLEX at termination.

# of Routes	# of Nodes	# of IV	# of CV	# of Cnstr	# of Explored B&B Nodes		Objective Value	
					FP	CPLEX	FP	CPLEX
6	14	36	68	134	75	117	5.1037	5.1037
7	16	36	72	137	75	122	5.2557	5.2557
8	18	38	76	144	151	301	5.8348	5.8348
9	19	46	90	173	539	1333	7.0747	7.0747
10	20	58	102	208	1921	9000	8.3839	8.3839
11	22	64	110	227	4115	10863	8.9531	8.9531
12	24	72	118	250	13883	24999	9.8912	9.8912
13	24	99	132	317	139597	389015	12.1375	12.1375
14	24	132	146	396	321700	10190783	14.4985	13.6215 (LB) 14.4985 (UB)

Table 3. Performance Comparison on Complex Network

From Table 3, we can see that our proposed algorithm can save at least half the number of explored branch-and-bound nodes in most situations.

Next, we select a single-track route from the Los Angeles railway network that is shared by multiple trains. The railway segment is shown in Figure 8. The number of nodes along the shared segment is 32; the total number of railway nodes in this portion of the network is 52. In Table 4, we give the data setup for every route and we assume all trains are ready at time zero.



Figure 8. Single-Track Route That Is Shared by Multiple Trains

Route ID	Origin	Destination	Length of Route	Train Length	Max Train Speed (mph)
1, 5, 9	Cajon	East Yard	74.43 mi	1.14 mi	50
2, 6, 10	East Yard	Cajon	76.77 mi	1.14 mi	50
3, 7	Yuma	East Yard	71.77 mi	1.52 mi	50
4, 8	East Yard	Yuma	74.11 mi	1.52 mi	50

Table 4. Route Data Specification II

We start from a problem with the first two routes, then the first three routes, and so on. The largest problem that we can solve within the 2 hours CPU limit is with 10 routes. Again, we compare our computational results with CPLEX 8.0 in Table 5. The comparison shows that the adjacent and feasibility propagation rules are powerful tools to reduce the number of branch-and-bound nodes to be expanded.

# of Routes	# of IV	# of CV	# of Cnstr	# of Explored B&B Nodes		Objective Value	
				FP	CPLEX	FP	CPLEX
2	38	176	250	1	33	5.0670	5.0670
3	113	260	483	5	162	7.0573	7.0574
4	227	348	798	13	5055	10.5299	10.5299
5	383	434	1195	35	39091	13.4557	13.4557
6	581	524	1680	182	2036093	17.7685	16.1684 (LB) 18.5966 (UB)
7	810	608	2221	422	2225503	20.2947	16.3142 (LB) 60.6056 (UB)
8	1082	696	2852	3917	1716145	25.1782	18.6431 (LB) 79.2441 (UB)
9	1394	782	3561	21259	1292106	29.2464	20.4468 (LB) 98.5978 (UB)
10	1752	872	4366	158479	961461	34.7491	22.8350 (LB) 122.7128 (UB)

Table 5. Performance Comparison on Single Track Network

5 Conclusion

Trains operating in densely populated metropolitan areas typically encounter complex trackage configurations. In this paper, we develop a branch-and-bound procedure for determining the optimal dispatching times for trains traveling in complex rail networks. Prior research in this area focused on single-track or double-track segments. The model

developed in this paper can be applied to any type of trackage configuration including triple-track segments which are commonly found in the Southern California region.

We tested the proposed branch-and-bound algorithm against a commercially available integer programming solver, CPLEX 8.0. On actual rail data set from Los Angeles County, we demonstrated the effectiveness of the adjacent and feasibility propagation rules in reducing the number of explored nodes in the search tree. Since exact solution procedures are limited in the size of the problem that they can optimally solve, future research can focus on developing effective heuristics for solving the dispatching problem for complex track configurations.

Acknowledgments

The research reported in this paper was partially supported by the National Science Foundation under grant NSF DMI-0223479.

Reference

- CAPRARA A., M. FISCHETTI, and P. TOTH. 2002, Modeling and Solving the Train Timetabling Problem. *Operations Research* 50, 851-861.
- CARRY M. 1994(a), A Model and Strategy for Train Pathing with Choice of Lines, Platforms and Routes. *Transportation Research* 28B, 333-353.
- CARRY M. 1994(b), Extending a Train Pathing Model from One-Way to Two-Way Track. *Transportation Research* 28B, 395-400.
- CARRY M. and D. LOCKWOOD. 1995, A Model, Algorithms and Strategy for Train Pathing. *Journal of Operations Research Society* 46, 988-1005.
- CORDEAU, J.-F., P. TOTH, and D. VIGO. 1998, A Survey of Optimization Models for Train Routing and Scheduling. *Transportation Science* 32, 380-404.

- DORFMAN, M.J. and J. MEDANIC. 2004, Scheduling Trains on a Railway Network using a Discrete Event Model of Railway Traffic. *Transportation Research Part B* 38, 81-98.
- HIGGINS, A., L. FERREIRA, and E. KOZAN. 1995, Modeling Single-line Train Operations. *Transportation Research Record* 1489, 9-16.
- JOVANOVIC, D. and P.T. HARKER. 1991, Tactical Scheduling of Rail Operations: the SCAN I System. *Transportation Science* 25, 46-64.
- KARRY, D., P.K. HARKER, and B. CHEN. 1991, Optimal Pacing of Trains in Freight Railroads: Model Formulation and Solution. *Operations Research* 39, 82-99.
- LAWLEY, M. and S. REVELIOTIS. 2001, Deadlock Avoidance for Sequential Resource Allocation System: Hard and Easy Cases. *International Journal of Flexible Manufacturing Systems* 4, 385-404.
- LU, Q., M.M. DESSOUKY, and R.C. Leachman. 2004, Modeling Train Movements through Complex Rail Networks. *ACM Transactions on Modeling and Computer Simulation* 14, 48-75.
- SAUDER, R.L. and W.M. WESTERMAN. 1983, Computer Aided Train Dispatching: Decision Support Through Optimization. *Interfaces* 13, 24-37.
- SZPIGEL, B. 1973, Optimal Train Scheduling on a Single Track Railway. In *Proceedings of the IFORS Conference Operational Research '72*; Amsterdam, North-Holland, 343-361.