Rescheduling algorithm using operational patterns for rolling stock operation at train depots

T. Tomiyama[1], T. Sato[1], K. Okada[2] & T. Wakamiya[2]
[1]*Yokohama Laboratory, Hitachi, Ltd., Japan*
[2]*Infrastructure Systems Company, Hitachi, Ltd., Japan*

Abstract

Railway companies have to create schedules for their rolling stock to operate a large number of trains by defining which route each train runs on and when maintenance is done. This scheduling task requires a lot of time and highly-skilled operators, called rolling stock managers, because of numerous complications. We developed an algorithm to help them that automatically creates a schedule for rolling stock during disruptions. We applied the Dijkstra method, which is a mathematical algorithm for searching efficient paths from a network model like the routes taken by sales staff. Our algorithm takes into consideration 1) maintenance cycles where railway companies are required by law to periodically inspect the condition of all trains. It schedules maintenance to continue these cycles. It also takes into account 2) operational patterns through scheduling patterns to create schedules that can compete with manual schedules. These patterns can represent experiences by rolling stock managers like the rotations of train routes. We carried out numerical experiments using real data (small and large numbers) at two train depots of a Japanese railway company. The small ones included ten train sets and a bi-monthly schedule. The large ones included 67 train sets and a monthly schedule. We generated experimental data on virtual disruptions. Our algorithm could mostly generate feasible schedules both from the small and large numbers of data, between 0.02 and 173.25 s, and in 632.47 s. Slight deviations in maintenance cycles were involved in some of the large ones. These results indicate that our algorithm could be feasible for real situations, even though satisfaction with constraints and calculation time should be improved to achieve highly interactive rescheduling operations.
Keywords: rolling stock operation, scheduling, Dijkstra method.

WIT Transactions on The Built Environment, Vol 127, © 2012 WIT Press
www.witpress.com, ISSN 1743-3509 (on-line)
doi:10.2495/CR120471

1 Introduction

1.1 Rolling stock operations at railway companies

Japanese laws require railway companies to regularly inspect their rolling stock at stated intervals to ensure safe transportation, which are based on running distances and inspection periods. Each railway company manages their rolling stock by creating schedules of rolling stock operations and checking the results. The schedules for rolling stock operations define the assignments of cars to trains and the dates of inspections.

The schedules for rolling stock operations are based on train timetables. Once transport disorders occur, the schedules are changed. As a result, the running distances of individual trains differ from those that are planned, and some trains finish operations at unscheduled stations. Railway companies have to recreate their schedules in these cases for the days after transport disorders occurred.

Rescheduling requires a lot of time because these schedules are manually prepared to ensure consistency with relevant departments such as departments for inspection, those for planning train timetables, and those for crew management. The rate of change in schedules has to be low to reduce the time and labor spent in rescheduling.

We propose an algorithm for rescheduling rolling stock operations in this paper that takes into consideration operational patterns to reduce the number of periods with changes.

1.2 Existing research

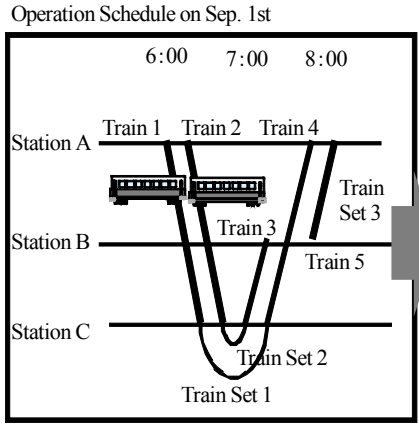
There has been some research regarding algorithms for scheduling rolling stock operations for passenger or freight trains. This research has supported the optimization of schedules such as minimizing both the number of seats and the number of inspections [1–3]. They have proposed several algorithms to solve scheduling problems such as hybrid methods of column generation and Lagrangian relaxation [1, 3–5], Benders relaxation [6], heuristics methods like a mixture of greedy and backtrack [2], and meta-heuristics [7]. These researchers have aimed at automatically calculating optimal or feasible solutions. Our aim was to calculate a feasible solution with operational patterns to reduce the number of differences from original schedules.

2 Requirements of schedules for rolling stock operations

2.1 Schedules for rolling stock operations

The schedules for rolling stock operations are normally created in train units. A train unit means a set of merged cars, which are not divided except in special cases like those in breakdowns. The first step in assigning train units is to define sets of trains that are assigned the same train unit (operation schedule). After that, train units are respectively assigned to the sets of trains, and dates of

inspections are defined in regular cycles (car schedules). While operation schedules are created each day, car schedules are created bi-monthly or monthly. Figure 1 shows an example of schedules for rolling stock operations.



Car Schedule in September

	Sept. 1st	Sept. 2nd	Sept. 31th
Train Unit A	Train Set 1	Train Set 2	Train Set 3
Train Unit B	Train Set 2	Train Set 3	Train Set 1
Train Unit C	Train Set 3	Train Set 1	Train Set 2

Figure 1: Example of Schedules for rolling stock operations.

Our target was to recreate a car schedule after a transport disorder had occurred.

2.2 Constraints

There are three constraints that should be considered in car schedules.

(1) Connection constraint
This constraint limits sequences of a set of trains assigned to the same train unit. It restricts the sequences to ones that ensure places are connected between dates that are next to each other. Violation of this constraint means the train unit starts to run from a station that is different from the station it arrived at on the previous day. In such cases, a deadhead train is needed to complete the schedule. A deadhead train causes changes in relevant schedules with train timetables and planning for crews and it wastes additional electricity. To avoid additional deadhead trains, car schedules should retain the sequences of places.

(2) Inspection constraint
This constraint keeps cycles of inspections consistent. There are several kinds of inspections, and each inspection has a different cycle. Although we deal with monthly inspections and daily inspections in this paper, this does not mean that our algorithm is limited to these inspections. Daily inspections are the simplest of all our inspections. The main parts of cars, such as pantographs and axletrees, are visually confirmed at car depots. The cycle is normally defined to be between three and five days. Monthly inspections include e.g., strict visual confirmations,

inspections under the floor, the grinding of axletrees, and the exchange of consumables. The cycle is normally defined within 30 days.

(3) Fewer differences from original schedule
This constraint is considered to reduce the range of influences affecting rescheduling. Maintenance, such as daily inspections and work on shunting, are planned on the basis of car schedules. Reducing the number of differences from the original schedule leads to reducing the range of influences on this task.

3 Model and algorithm

3.1 Model

We represent a car schedule as a network model, where nodes represent a set of trains included in a target period that are on the schedule and arks represent connection constraints, which means arks are generated between nodes that can be linked under connection constraints. In addition, if a train unit is stocked at a car depot for a whole day, this operation is represented as a node, which is the same as a set of trains. Each node has four attributes of the date when the set of trains will run, the first station where the set of train will start to run, the last station where the set of trains will finish running for a whole day, and the possibility of daily inspections. The connection constraint, which was mentioned in Section 2, is considered to represent arks of the network model. Therefore, the assignment of a train unit can be represented as a path of the network model. Figure 2 shows an example of this network model.

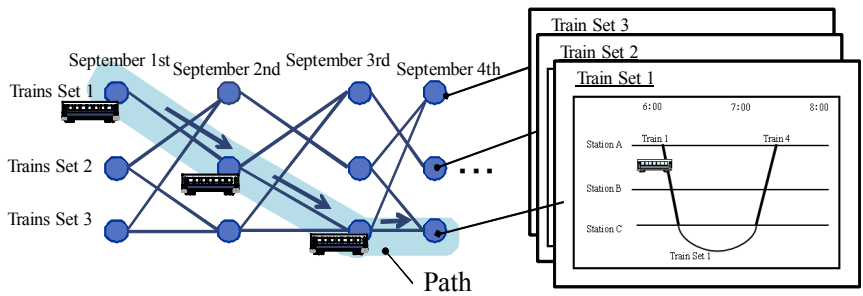


Figure 2: Network model for car schedule.

Preparing a car schedule is a problem that extracts a set of paths from the network model. Our algorithm takes into account three constraints when solving this problem, where the constraints are defined on the basis of the constraints mentioned in Section 2 and the characteristics of a car schedule.

< Decision Variables >

$$x_{ij} = \begin{cases} 1 & : \text{ Node } j \text{ is included in path } i. \\ 0 & : \text{ Others} \end{cases}$$

Seq_{ij} : Index number of node that is j th node included in path i.

< Invariables >

$$a_{jk} = \begin{cases} 1 & : \text{ Inspection } k \text{ is executable at node } j \\ 0 & : \text{ Others} \end{cases}$$

N : Total number of train units
T : Total number of nodes
C_k : Cycle of inspection k

(1) Each node is included in only one path.
Each set of trains is defined on the assumption that only one train unit is assigned to one set. Paths cannot include the same nodes between each other to deal with this assumption. When some train units are merged to assign one train together, as many nodes that represent the same train are added to the network model as the number of merged train units.

$$\sum_{i \in N} x_{ij} = 1. \tag{1}$$

(2) Each node is included in one path.
Even if a train unit does not run as trains for a whole day, train companies plan this case as an operation "reserve". For example, a train set is stocked at a train depot when the train set is planned for inspection the next day, or to equalize mileage. Therefore, all train units are assigned to trains or reserve operations. This means each train unit is definitely assigned to one path.

$$\sum_{j \in T} a_{ij} \geq 1. \tag{2}$$

(3) All paths include more than one inspection node in a consistent cycle.
If a set of trains includes a sufficient time interval to execute inspection, the node representing the set of trains has an attribute that means the inspection is practical. Our algorithm makes each path include such nodes within the inspection cycle.

$$\sum_{m=0}^{m=C_k} a_{seq_{i\,j-m}k} \geq 1. \tag{3}$$

3.2 Algorithm

We describe the process of extracting a set of paths from a network model that satisfies constraints in this section.

The connection and inspection constraints when creating a car schedule need to be considered as mentioned in Section 2. Reducing the number of changes, which was mentioned as the third constraint in Section 2, is considered to be the objective of our algorithm because this constraint is less important than the other two. Considering the two constraints, i.e., the connection and inspection constraints simultaneously may require more time than considering each constraint individually. In addition, daily inspections, which are the most common, can be completed at unplanned dates in emergency situations like those when transport disorders occur. Our algorithm takes the two constraints into account individually giving priority to the calculation time to obtain a feasible solution. First, our algorithm calculates a feasible solution under the connection constraint. After that, it modifies the solution under the inspection constraint as effectively as possible. Figure 3 outlines the flow to extract a set of paths.

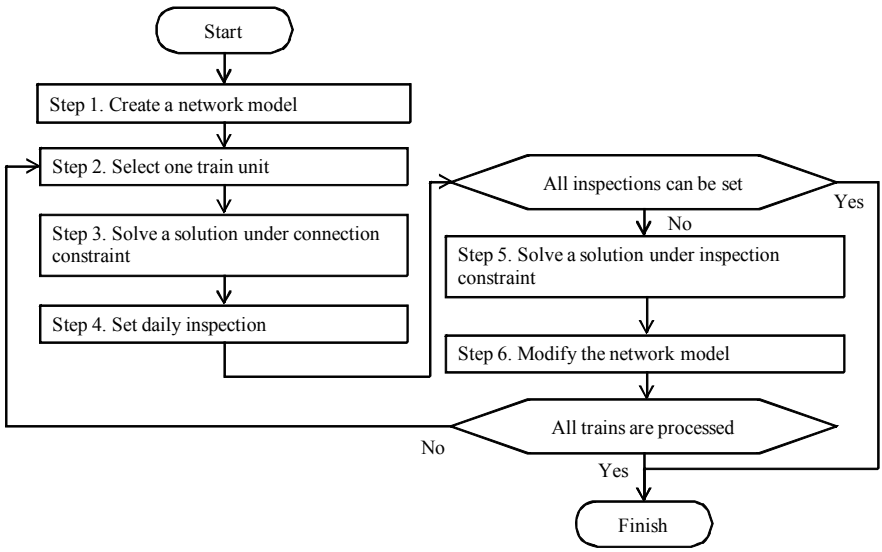


Figure 3: Flowchart for searching route sets from network model.

Step 1. Create a network model: A network model is created on the basis of a timetable and a car schedule. The timetable reflects a transport disorder.
Step 2. Select one train unit: One train unit is selected from unprocessed train units.
Step 3. Find a solution under the connection constraint: A path is explored under the constraints mentioned in Section 3 as constraints (1) and (2). This exploring process uses the Dijkstra method that extracts a path from a network model minimizing the total weight of arks. To reduce the number of changes from the original schedule, arks between the same nodes with those of the original schedule are given smaller weights, which take into consideration constraint (3) mentioned in Section 3.

Step 4. Set daily inspections: The path explored in Step 3 is verified as to whether inspections can be set within consistent cycles.
Step 5. Solve a solution under the inspection constraint: If no inspection can be set in Step 4, the path for the solution to the train unit is explored again. First, another path is explored from the present network model. The previous solution is canceled when no solution is available. This means that nodes included in the solution of another train unit, which was previously calculated, are added to the present network model (these nodes were excluded from the network model in Step 6). After that, the solution is explored again. The number of cancelations is limited to a few times each train unit to avoid excessively long calculation times.
Step 6. Modify the network model: Nodes included in the path explored in Step 5 are excluded from the present network model.

The process from Steps 2 to 5 is repeated until solutions to all train units are calculated or there are no more nodes in the network model.

4 Operational patterns

4.1 Categories of operational patterns

We extended the approach mentioned in Section 3 to reflect operational patterns in the car schedule. The operational patterns provide regularity to the trains assigned to each train unit. These patterns are used in common practice when a car schedule is manually prepared. Our algorithm takes into account three patterns that appear in actual car schedules.

(1) Sequence pattern: This pattern defines sequences of sets of trains that are continuously assigned to the same train unit. Figure 4-(1) shows an example of this pattern, where sequence pattern means that train sets 1, 2, and 3 were continuously assigned to the same train unit on September 1, September 2, and September 3.

(2) Exchange pattern: This pattern defines combinations of sets of trains that exchange train units assigned to them. Figure 4-(2) shows an example of this pattern, where exchange pattern means the sequence from train set 1 to 2 and the sequence from train set 3 to 1 can be exchanged. The sequence from train set 1 to 2 is changed to the sequence from train set 1 to 1 by following this pattern, and the sequence from train set 3 to 1 is changed to the sequence from train set 3 to 2.

(3) Cutting pattern: This pattern defines combinations of sets of trains that exchange train units in one day. Figure 4-(3) shows an example of this pattern, where train sets 1 and 2 exchange trains included in each other. Train set 1 in the original schedule include trains 1 and 4, and train set 2 include trains 2 and 3. Train sets 1 and 2 exchange trains 4 and 3 by following this pattern.

While the other patterns do not change combinations of trains in one day, this pattern changes them. This leads to increased candidates for connection, and the possibility of success for exploration is increased.

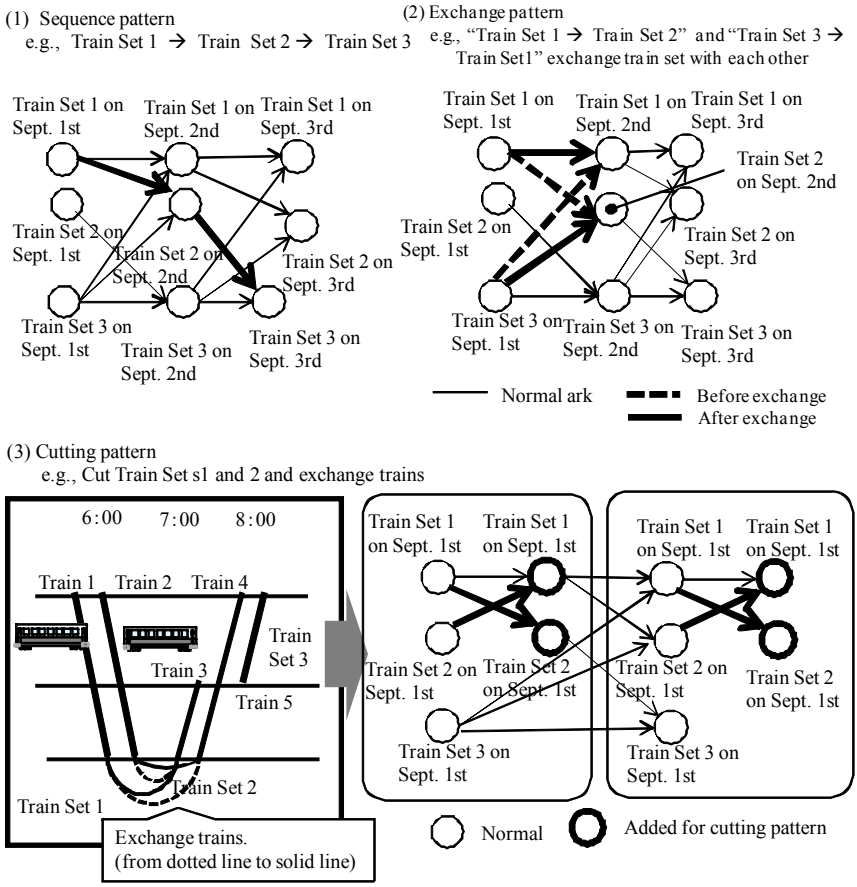


Figure 4: Operational patterns for car schedule.

4.2 Model with operational patterns

Our algorithm takes into consideration operational patterns as weights of arks. We set smaller weights to arks that link nodes included in patterns.

If all three operational patterns are simultaneously considered, the ark weights of some patterns are mixed, and no patterns are reflected in the calculated results. To avoid this, we assumed that each category of patterns would be considered individually.

5 Numerical experiments

We verified the effect of operational patterns and the ability of our algorithms to satisfy constraints. The test data were based on two actual data.

(1) Small depot: including ten train units with target period of 15 days.
(2) Large depot: including 67 train units with target period of 29 days.

We used a Pentium4 computer (3.2 GHz and 2 GB). The programming language was VC++6.

5.1 Results from experiments

5.1.1 Satisfaction with constraints

(1) Connection constraint

We created virtual transport disorders and created car schedules that included violations of the connection constraint using small numbers of test data. Table 1 summarizes the results obtained with our algorithm by applying this test data. Data 1 and 2 were created by assuming large and smaller disorders.

Table 1: Results from small data test.

	Rates of change from original schedule (%)	Calculation time (s)
Data 1	15.3	173.25
Data 2	0.03	0.109

Our algorithm could prepare a car schedule without violating the connection constraint. The range of changes was about nine days, where the schedule was changed in nine days. The calculation times ranged from 0.03 to 173.25 s, which is sufficient for use in actual situations.

(2) Inspection constraint

We verified how many violations of the inspection constraint could be solved with our algorithm by using the test data from the large depot that had not yet been included in the inspections. We verified these using two data obtained bi-monthly and monthly. Table 2 lists the results.

Table 2: Results from large data test.

(1) Bi-monthly (15 days)

	Violations of inspections		Rate of change from original schedule (%)	Calculation time (s)
	No. of violations	Solved rates (%)		
Original	1476	-	-	-
Our results	15	98.98	84.50	310.77

(2) Monthly (29 days)

	Violations of inspections		Rate of change from original schedule (%)	Calculation time (s)
	No. of violations	Solved rates (%)		
Original	1539	-	-	-
Our results	90	94.15	87.84	632.47

The rate of solved violations was high for both bi-monthly data and monthly data, from approximately 94 to 98%. The calculation time ranged approximately from five to nine minutes, which we think is in a feasible range.

5.1.2 Effect of operational patterns

We extracted operational patterns from an actual schedule of a large depot. We verified the effect of operational patterns by using these patterns. The test data were the same as those in Section 5.1.1 (1). The results are presented in what follows.

(1) Sequence pattern

We considered a sequential train set's numbers as sequence patterns, e.g., train sets 1, 2, and 3 in Figure 4. This is the same as is done manually. Table 3 summarizes the results for the test of sequence patterns.

Table 3: Results for sequence patterns.

	Calculation time (s)	No. of patterns per length of pattern* *No. of sequential nos.					Solved rate for inspection violations (%)	Rate of change from original schedule (%)
		2	3	4	5	Total		
Original	-	127	13	60	42	242	-	-
Our results	162.22	285	38	84	51	458	96.14	63.83

Our algorithm could solve about 96% of inspection violations. The resulting solutions included more patterns than the original, and the rate of change from the original schedule was improved. The calculation time was about three minutes, which we believe is in a feasible range.

(2) Exchange pattern

We extracted four patterns that appeared more than twice in the original schedule. Each pattern had a length of four, which means each consisted of four sets of trains. Table 4 summarizes the results.

Table 4: Results for exchange patterns.

	Calculation time (s)	No. of patterns in each pattern length			Solved rate for inspection violations (%)	Rate of change from original schedule (%)
		2 (partial)	3 (partial)	4 (full)		
Original	-	27	2	4	-	-
Our results	558.89	51	8	5	92.68	42.02

Our algorithm solved about 92% of inspection violations. More patterns were included than in the original, and the rate of change was improved. The calculation time was about ten minutes, which again is in a feasible range.

(3) Cutting pattern
We created test data that included connection violations by eliminating cutting patterns from the original schedule. Table 5 lists the results.

Table 5: Results for cutting patterns.

	Calculation time (s)	No. of dates with changed schedules	No. of patterns	Rate of change from original schedule (%)
Original	-	-	10	-
Our results	36.86	1	8	0.00

We could create a schedule with no changes from the original schedule, and the number of dates with changed schedules was only one day. This meant the car schedule was changed for only the day that transport disorder occurred.

6 Conclusions

We proposed a method of making a car schedule with operational patterns. We confirmed that our algorithm could calculate feasible solutions within a feasible calculation time. While our algorithm could solve connection constraints, some inspection constraints were not solved. However, our algorithm could solve more than 94% of bi-monthly inspection violations. Unsolved monthly inspections could be planned by partly fixing and recalculating train unit schedules with monthly inspections. Unsolved daily inspections could be manually arranged to be carried out at unplanned stations on unplanned dates in light of actual operations.

Our algorithm could decrease the number of changes from the original schedule by setting ark weights based on operational patterns. However, the

results included many partial patterns and adversely affected other meaningful weights for patterns consisting of more than three train sets like those in exchange patterns. Therefore, we have to consider which patterns should be applied. Our algorithm could solve solutions without connection violations within short periods by using cutting patterns. However, because these patterns extended the search space, we should consider limiting situations where cutting patterns are used, e.g., cases where there are no solutions because of short scheduling periods.

References

[1] V. Cacciani, A. Caprara, and P. Toth, "Solving a Real-World Train Unit Assignment Problem". In Proceedings 7th Workshop on Algorithmic Approaches for Transportation Modeling, Optimization, and Systems, pp. 79–95, 2007.
[2] T. Otsuki, H. Aisu, and T. Tanaka, "The Solution for Constraint Satisfaction Problems of Railway Rolling Stock Allocation". Transactions of the Operations Research Society of Japan, Vol. 53, pp. 30–55, 2010.
[3] K. Sato and N. Fukumura, "Freight Train Locomotive Rescheduling Problem after Disruptions". IPSJ TOM, Vol. 2, No. 3, pp. 97–109, 2009.
[4] D. Huisman, R. Freling, and A. P. M. Wagelmans, "Multiple-Depot Integrated Vehicle and Crew Scheduling". Transportation Science, Vol. 39, No. 4, pp. 491–502, 2005.
[5] T. Sato, T. Tomiyama, T. Morita, and T. Murata, "A Lagrangian Relaxation Method for Railway Crew and Vehicle Rescheduling of Railway Passenger Transportation and its Application". IEEJ Transactions on Electronics, Information and Systems, Vol. 132, No. 2, pp. 260–268, 2012.
[6] J. F. Cordeau, F. Soumis, and J. Desrosiers, "A Benders Decomposition Approach for the Locomotive and Car Assignment Problem". Transportation Science, Vol. 34, No. 2, pp. 133–149, 2000.
[7] Y. Tsuji, M. Kuroda, Y. Imoto, and E. Kondo, "Rolling Stock Planning for Passenger Trains Based on Ant Colony Optimization", Trans. Japan Society Mechanical Engineers, Vol. C-76, No. 762, pp. 171–180, 2010.