

A graph and optimization based crew scheduling framework for urban rail transit*

Aadesh Jain¹, Rupesh A. Sonawane¹, Sarvar Abbas¹, Ashutosh Mahajan¹,
Narayan Rangaraj¹, Madhu N. Belur^{1,2*}, Ashish K. Anmol³, Rajnish K. Rana³,
Amit K. Jain³

¹Dept. of Industrial Engineering & Operations Research, Indian Institute of Technology
Bombay.

^{2*}Dept. of Electrical Engineering, Indian Institute of Technology Bombay.

³Delhi Metro Rail Corporation, India.

Authors-emails: amahajan@iitb.ac.in; narayan.rangaraj@iitb.ac.in; *belur@iitb.ac.in;
ashish.anmol@dmrc.org; rajnish.rana@dmrc.org; jainakirts@yahoo.com;

Abstract

This paper presents a graph-based framework for solving the crew pairing problem for the Delhi Metro Pink Line main loop operations. Train services are modeled as nodes in a directed service network graph, where feasible connections are established using operational constraints such as station continuity, rake compatibility, duty duration limits, driving time restrictions, break rules, and jurisdiction conditions. Feasible duty paths are generated using stack-based depth-first search with early pruning strategies to eliminate infeasible partial duties. To improve computational efficiency, parallel processing is employed during feasible duty generation. The generated feasible duties are formulated as a ‘set partitioning optimization problem’ and solved using the Minotaur optimization toolkit. Solutions for intra-day planning of the operations are achieved in reasonable time by reducing search space through graph modeling, early pruning, parallel processing for generating duties and using specialized options for a linear programming based branch-and-bound. The results show that the proposed methodology provides a scalable and efficient solution for large-scale real-world metro crew scheduling problems.

Keywords: Crew scheduling, graph-based optimization, set partitioning problem, parallel processing, feasible duty generation

1 Introduction

Crew pairing in railways, bus, airlines or other service systems refers to planning the number of crew required to undertake a given set of services and designing duties for the crew: also called as duty-chart. It is different from crew-rostering where individual employees are assigned to duties. In the context of railways, crew pairing is an important component of planning activities along with line planning, timetabling, platforming, crew-rostering etc. [1]. In fact, the inputs for crew pairing, are obtained once a timetable of the services and the assignment of rakes to these services and platforms has been obtained.

In this work, we describe a crew-pairing system that we developed for one of the Metro lines in Delhi, India. The system is generic and can be easily deployed on other lines and possibly other transportation

*This work was funded in part by Delhi Metro Rail Corporation.

systems as well. Although Kolkata was the first city in India to introduce a metro service, Delhi was the first city to have large scale metro system in India; the Delhi Metro Rail Corporation currently operates 10 metro-lines, having a network length of more than 350 km, running more than 4500 services a day, and carrying more than 6 million passengers daily; these numbers are increasing due to upgradation of infrastructure and the resulting increase in passenger-counts. The Pink Line of Delhi Metro, which we use as a case-study in this work, has about 75km of track length, about 45 stations/stops, and more than 400 (to and fro) train services daily: these services serving more than 830 thousand daily commuter trips.

Our starting point of our system is the timetable of services announced by the operator. This timetable is intra-day and usually remains in effect on all working days of the week. Weekends, holidays and other special days have different timetables. The daily operations start early morning and end in the night. A typical line has hundreds of services running daily and may require more than 100 crew members to operate all services. This timetable along with other operational inputs about rakes etc is used to first create an input for a mathematical model for optimizing the crew pairing. We describe this system in Section 3. The optimization model is then solved, and the results are translated into a usable format for the planning team. To our knowledge, this system may be the first of its kind developed in India.

Following the large body of crew-pairing in literature [2, 3], we model the required services as nodes of a directed acyclic graph. A directed edge from a node, say i to another node, j is drawn if the service j can potentially be served by the same crew immediately after completing the service i . A duty, which is a collection of services that can be done by a single crew in one shift is then a path in this graph. Additional constraints like maximum duty hours and rules for breaks in between a duty etc forbid selecting many of these paths. One way of finding an optimal set of duties (paths) to cover each service (node of the graph) is to model it as a set-partitioning problem using binary variables. Each variable in the set partitioning model represents a feasible duty and is allowed to take a value 1 (if selected) or 0 (if not). Crew-pairing and also the set-partitioning problems are both NP-hard [4], which means that exact approaches to solve them may require time that grows exponentially in the size of the input. Set-partitioning models are often solved using integer optimization approaches that rely on solving a series of linear programming relaxations in a branch-and-cut framework. These models can be solved by currently available solvers in a reasonable time for a certain number of possible duties. This approach in which an optimization model is created and given to an off-the-shelf solver is quite straightforward to implement.

The number of possible duties increases rapidly with the number of services, often making the model sizes too large for any typical computer to store. Column generation based approaches have been developed to mitigate this issue. These approaches are useful when very-few variables take non-zero values in the optimal solution. Column generation [5], in its simplest form, is a method to solve linear programs by working with only a small set of variables without creating all the other variables. Additional variables are added to the model only when required, thus keeping the memory requirements low. This method is quite fast for solving the linear programming relaxations of the set-partitioning model, but the solution may have non-integer values. Column generation is then embedded in a branch-and-cut framework to find the optimal 0-1 values for the set-partitioning model. Such a branch-and-cut framework in which new variables may be generated at various stages of solving is called branch-and-price [6, 7]. Column generation based approaches have been successfully used in crew-scheduling for a large number of services [8]. However, branch-and-price frameworks are not that straightforward to implement. The algorithms for creating and adding new variables for the linear programming relaxation are model specific and need to be modified significantly if the duty rules change. Branch-and-price frameworks are also not easily implemented with state-of-the-art solvers. Several call-back routines for solving the LP relaxation, branching rules, heuristics, and cutting planes may have to be developed by the user in order for the procedure to be effective.

The metro systems we consider are too big for modeling and solving using off-the-shelf solvers but at the same time are not too large that require a sophisticated implementation of branch-and-price framework. We develop a novel middle path in which we solve a large set partitioning model after generating

all the variables. Instead of using the default simplex-based methods of existing solvers, we instead use a crashing technique recently developed in the CLP solver [9] and analyzed by [10]. By suitably embedding this technique in an existing open-source branch-and-cut solver [11], we are able to solve the model much faster than off-the-shelf solvers and without using any sophisticated branch-and-price implementations. We provide details of the model and the algorithmic approach in Section 3.

2 Literature review

Heil [12] offers a comprehensive review of advancements in railway crew scheduling. It traces the evolution of the field from its origins in airline and bus industries to its current state, highlighting the increasing complexity and scale of scheduling problems. The authors categorize over 130 studies published since 2000, focusing on three primary model formulations: set covering, set packing, and network flow models. These models aim to optimize various objectives, including schedule efficiency, robustness, and employee satisfaction. The paper also discusses the integration of operational constraints, such as legal regulations and practical operational rules, into scheduling models.

Crew pairing problems are typically formulated as mathematical models, often using either set covering or set partitioning, and are subsequently addressed using exact solution techniques or heuristics. Desrochers and Soumis [13] proposed one of the early column generation approaches for urban transit crew scheduling, demonstrating the effectiveness of decomposition-based optimization methods for large-scale scheduling problems. Similarly, Barnhart et al. [6] presented branch-and-price techniques for solving huge integer programming problems, which later became highly influential in transportation scheduling applications. Borndörfer et al. [14] applied set partitioning and column generation approaches to airline crew scheduling, while Lusby et al. [15] developed a column generation-based heuristic incorporating work patterns for ground crew rostering.

Crew planning for metro operations, including both crew pairing and crew rostering are studied by Zhou et al. [16] for Hefei Metro in China, [17] for Chengdu Metro in China. These planning approaches are somewhat simpler than planning for long distance services that involve overnight stays, rest periods while away from base depots and deadheading. (‘Deadheading’ is also called ‘travel as passenger’ (TAP), in which a crew-member travels while on duty to another location so that he/she can take up the next driving task from the next location.) The planning periods for long distance services typically span a week or sometimes even a month, as compared to a day for metro services.

Several researchers have also explored integrated and large-scale railway scheduling frameworks. Caprara et al. [18] discussed models and algorithms for railway crew management, focusing on combinatorial optimization techniques and operational feasibility in railway systems. Huisman et al. [19] highlighted the growing role of Operations Research techniques in passenger railway transportation, particularly for crew scheduling, vehicle scheduling, and timetable optimization. Freling et al. [20] further investigated integrated vehicle and crew scheduling models, demonstrating the advantages of jointly considering multiple operational planning components.

The work on the Taipei Mass Rapid Transit System [21] demonstrated the effectiveness of constraint programming approaches in handling complex operational rules and crew assignment restrictions. Such methods are particularly useful when scheduling problems involve numerous logical and operational constraints that are difficult to represent directly in traditional mathematical programming formulations. Abbink et al. [22] describe an Operations Research framework for railway crew scheduling at Netherlands Railways (NS Reizigers). The model includes more than 6,500 crew drivers and conductors. Schedule rules obtained through the model increased crew task diversity while decreasing personnel expenses and enhancing operational effectiveness. In another recent development, Leibchen et al. [23] devise and deploy a series of mathematical optimization models for crew rostering and scheduling tasks at S-Bahn Berlin GmbH. The models have been used to produce yearly crew rosters since 2024 and daily crew duty

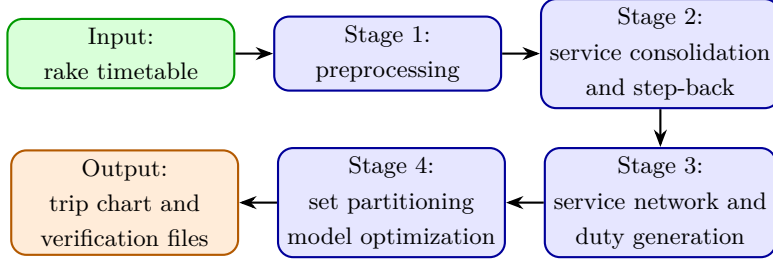


Fig. 3.1 Four stage crew scheduling framework adopted for crew pairing

Table 3.1 Sample (indicative) excerpt of the input rake timetable used for preprocessing

Station	701	710	716	720
MKPD	05:40	05:50	06:00	06:10
MKPR	06:00	06:10	06:20	06:30
AZU	06:03	06:13	06:23	06:33
SMBG	06:06	06:16	06:26	06:36
SAKP	06:11	06:21	06:31	06:41
PVGW	06:14	06:24	06:34	06:44
DDSC	06:32	06:42	06:52	07:02
KKDA	07:20	07:30	07:40	07:50
MUPR	07:35	07:45	07:55	08:05

assignments since May 2025, illustrating practical implementations of optimization-based crew planning in urban rail transit systems. For intercity passenger rail transit operations, Bansal et al. [24] develop a heuristic technique to produce recurring crew schedules considering operational and crew assignment policies. The proposed framework considered crew task connectivities under cyclic train timetables and claimed up to 12% reduction in overall crew tasks. Such examples clearly show that crew planning challenges differ considerably depending on whether the problem concerns national railways, urban transit rails, or intercity railways. Consequently, crew planning frameworks must consider operational issues, labor regulations, crew preferences, and efficient resource utilization constraints.

3 System design

The crew pairing system developed here consists of four sequential stages beginning with timetable preprocessing and ending with trip-chart generation and verification. The output of each stage serves as the input to the subsequent stage (see Figure 3.1).

3.1 Stage 1: preprocessing

The first stage converts the rake timetable, available in a tabular format, into a collection of basic services that serve as the fundamental units for crew pairing. A service denotes movement of a particular rake between two important locations on the line along with the starting and ending time of the movement. Each service requires a loco-pilot, and the loco-pilot can serve multiple services provided they do not have time conflicts. For creating a list of services that need crew, all stops of a line do not need to be considered. Crew of a train can change rakes only on selected stations. Also the rakes may be taken out of service during off-peak hours only at selected ‘terminal stations’ or depots. Similarly, rakes can start operating only from places where they were previously ‘stabled’ or parked. The schedule is thus broken down into service segments defined between crew-changing stations and these terminal points only. For each service, attributes such as the start station, destination station, departure time, arrival

Table 3.2 Snapshot of elementary services (indicative sample input) generated after Stage 1: timetable Pre-processing

Rake No.	Start Station	Start Time	End Station	End Time	Dir.	Service Time (min)
717	MUPR DN SDG	05:30	MUPR	06:20	UP	50
701	MKPD	05:40	MKPR	06:00	DN	20
703	VND	05:40	IPE	06:00	UP	20
702	VND	05:40	IPE	06:00	DN	20
709	IPE	05:45	PVGW UP	06:57	UP	72
715	MVPO DN	05:45	IPE	06:36	DN	51
704	MKPR	05:50	PVGW DN	06:03	DN	13
710	MKPD	05:50	MKPR	06:10	DN	20
708	MUPR	06:00	KKDA UP	06:16	UP	16
707	SVVR	06:00	MUPR	06:10	UP	10

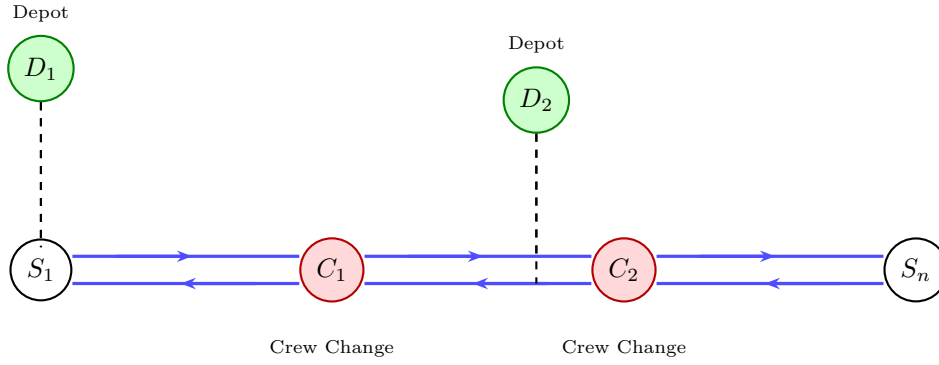


Fig. 3.2 Schematic representation of a metro corridor with terminal stations, crew-changing stations, and depots. Depot D_2 is connected to the line between crew-changing stations C_1 and C_2

time, rake number, and direction are extracted and stored. The output of this stage is a service database containing all basic train movements together with step-back information, which subsequently forms the basis for duty construction. Figure 3.2 is an illustration of a single metro line. Services are created for rake movements between segments like $D_1 - S_1$, $S_1 - C_1$, $D_2 - C_1$, $D_2 - C_2$, $C_1 - C_2$, $C_2 - S_n$, $S_n - C_2$, $C_2 - C_1$, $C_1 - S_1$ etc. Any pair of services created in this process for the same rake should not overlap in time and space. A representative snapshot of the initial services generated after Stage 1 is presented in Table 3.2.

3.2 Stage 2: service consolidation and step-back processing

Following the decomposition of the timetable into basic train movements, consecutive service segments are merged to form larger services whose start and/or end points coincide with designated crew-changing stations or depots. Step-back operations (for reversal of train at a station for next trip in the opposite direction) are incorporated during this stage, and information regarding jurisdiction compatibility, rake transitions, and constituent train movements is maintained. A representative snapshot of the consolidated services generated after this stage is presented in Table 3.3. The resulting chronologically ordered list of consolidated services, together with their operational attributes, serves as input for the duty generation stage.

3.3 Stage 3: service network and duty generation

In this stage we generate all possible feasible duties by combining the services collected in the previous stage. To model feasible connections between services, a directed graph is constructed in which each node

Table 3.3 Snapshot of consolidated services generated after stage 2 (service consolidation and step-back)

Rake No.	Start Station	Start Time	End Station	End Time	Dir.	Time (min)	Juris.	Step-Back Rake	Merged Service 1	Merged Service 2
717	MUPR DN SDG	05:30	MUPR	06:20	UP	50	Yes	–	26	–
701	MKPD	05:40	PVGW DN	06:14	DN	34	Yes	–	131	132
703	VND	05:40	PVGW UP	06:59	UP	79	No	–	5	6
702	VND	05:40	KKDA DN	06:07	DN	27	Yes	–	39	40
709	IPE	05:45	PVGW UP	06:57	UP	72	No	–	3	–
715	MVPO DN	05:45	KKDA DN	06:42	DN	57	Yes	–	72	73
712	SAKP 3RD	05:45	KKDA DN	06:59	DN	74	No	–	94	95
710	MKPD	05:50	PVGW DN	06:24	DN	34	Yes	–	151	152
713	VND	05:53	PVGW UP	07:12	UP	79	No	–	8	9
708	MUPR	06:00	KKDA UP	06:16	UP	16	Yes	–	12	–

represents a service. An arc from the node for service s_i to that of service s_j is added to the graph if it is feasible for a crew to serve s_2 after serving s_1 without serving any other service. For example, if Service s_i and s_j overlap in time, one can not have arcs $(s_i, s_j), (s_i, s_j)$. Other reasons which forbid having an arc (s_i, s_j) come from operational requirements of the company, e.g., the end station of s_i should be same as the start station of s_j (no deadheading), the idle period between two consecutive services served by a crew should not be more than a certain limit, etc. Two dummy nodes, S, T , are also introduced in the graph to ease modeling. They represent the start and end of all duties.

Algorithm 1: Construction of Feasible Service Network

Input: Set of services S

Output: Feasible service graph $G = (V, E)$

Initialize directed graph G ;

Add all services, source node s_{start} , and sink node s_{end} to G ;

Connect s_{start} to all services;

foreach pair of services (s_i, s_j) where $i \neq j$ **do**

 Compute connection gap between s_i and s_j ;

if station and time constraints are satisfied **then**

if same-rake or valid rake-change conditions hold **then**

 Add edge (s_i, s_j) to G ;

Connect all services to s_{end} ;

Return G ;

The graph construction implicitly models continuity between two consecutive services that a crew can possibly serve. It enforces several requirements like station continuity, maximum connection gap limits, rake consistency, and rake-change feasibility constraints. This graph is also the starting point of the optimization model in the next stage. This procedure runs in $O(n^2)$ time where n is the number of services or nodes. For some inputs with upto 900 services, a python implementation of this step does not take more than a second on a desktop computer. A small network with 7 services is shown in Figure 3.3 for illustration.

Once this graph has been created, one can construct possible duties for crew. A duty is a collection of services that a crew is assigned to for a given day. In our scheme, all allowed duties for a crew are first enumerated from the service network. An optimization model described in the next section then selects the duties to cover all services. It is assumed that all crew are capable of providing all services, i.e. there are no services that only some specialized crew can perform.

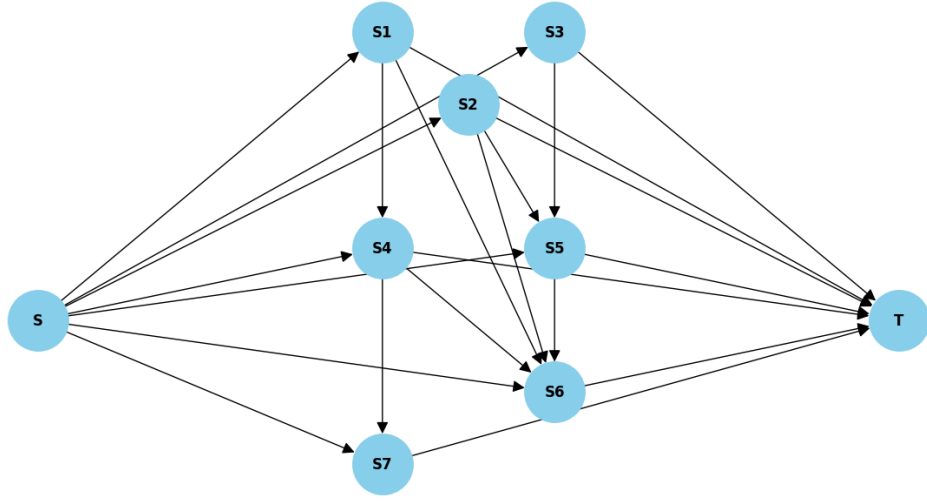


Fig. 3.3 A small service network with 7 services

Table 3.4 Graph characteristics

Parameter	Value
Number of nodes	944
Number of edges	23,234
Number of rakes (Mainloop)	51
Number of stations (Mainloop)	11
Time required to build the graph	0.5 s

Crew duties in metro and other services are governed by the “Hours of Work and Periods of Rest (HOER)” rules that are formulated by the Indian Railways for loco-pilots. These rules are subject to periodic updates. These have been adapted for metro-rail duties also. The key rules for duty considered in this study are:

1. Service continuity: The end station of a service must be the same as the start station of the following service in any duty.
2. Time feasibility: The end time of a service must not exceed the start time of the subsequent service.
3. Jurisdiction constraint: A duty must begin and end in the same jurisdiction for the crew.
4. Rake change constraint: Rake changes for crew are allowed only at particular stations. The gap between rake changes without a break in duty should be within a certain time limit.
5. Maximum duty hours: The total time of a duty must not exceed a given time limit. Time of duty is the difference between the start time of the first service and end time of the last service of a duty. For morning duties (starting before 6:00 AM) and evening duties (ending after 23:30 PM), the maximum duty hours are less as compared to other duties.
6. Maximum driving time: The cumulative driving duration within a duty must not exceed a given limit. The driving duration is calculated as the sum of service times and any gap between consecutive services that is less than 30 minutes:
7. Continuous driving limit: No more than 180 minutes of continuous driving is allowed without at least a 30-minute break.
8. Break conditions: A duty must contain at least one break at one of the break stations. At least one of these breaks must occur within the jurisdiction of the first service’s start station. If a driver operates continuously for more than 120 minutes, the following break must be at least 50 minutes long. The

total duration of all breaks must be less than 120 minutes. Further, every duty should at least one break of 50 minutes or more.

9. Morning first break rule: If a duty starts from a non-crew-controlled station, the crew must drive continuously for at least 90 minutes before taking the first valid break.

The number of possible duties that satisfy the above rules can be quite large. For example, one network of about 900 services that we considered had close to 600,000 possible duties. Of course, this large number includes subsets of valid duties, some of which could be very suboptimal: pruning these would decrease the number only marginally, and not substantially. Thus, a fast procedure for generating and verifying these duties is required. Since each duty is a time-ordered set of services that a single crew has to be performed, a duty is a path starting from the dummy node S and ending in the dummy node T in the service network. A stack-based depth-first search [25] was implemented to enumerate paths in the service network starting from dummy node S and ending at node T . The depth-first search ensures that all paths are created sequentially.

Some of the HOER constraints like ‘service continuity’ and ‘time feasibility’ listed above are already implicitly enforced at the time of adding arcs in the service network. These constraints need not be checked during the duty enumeration process. Some constraints like the ‘maximum driving time constraint, the ‘maximum duty hours’ constraint (with special limits for morning and evening duties), and the ‘continuous driving limit’ are checked every time a node is included in a path during the depth-first-search. This step helps to prune invalid duties early in the enumeration process. Finally, once a path reaches the end dummy node T and is a candidate for a complete duty, the final set of constraints is evaluated. These constraints depend on the whole path and cannot be checked in intermediate stages. These include the ‘jurisdiction constraint’, which requires the duty to begin and end within the same jurisdiction, and the detailed break conditions. The break rules verify that valid breaks occur only at designated break stations, that at least one break lies within the jurisdiction of the starting station, and that the cumulative break duration remains within the allowed limits. Additional structural requirements are also checked, including the 120-minute continuous driving threshold rule, which requires a long break after extended continuous driving, and the ‘morning first break rule’, which requires duties starting from non-crew-controlled stations to drive at least 90 minutes before the first break. A path is accepted as a valid duty only if it satisfies all of these final conditions.

To make the enumeration even faster, we perform the depth-first-search in parallel by utilizing multiple cores available in a computer. The procedure starts by creating K copies of the service network, where K is the number of processing threads available on a shared memory architecture. Enumeration is divided into K partitions based on the first service of a duty. All paths with the same first service are enumerated by a single thread. A thread is allotted several starting service-nodes from which it finds all the paths. A dynamic task allocation mechanism balanced the workload among processors, preventing idle workers when some starting nodes produced significantly more feasible paths than others. Each thread stored its generated duties in a separate output file. Individual duty files were concatenated to obtain the final duty pool.

The proposed parallel implementation significantly reduced the computational time for duty generation. Table 3.7 shows the number of duties created and the time taken by various modes of our approach. If we disable the early pruning rules of our duty generation and check all rules when the full path is created in the network, it takes several hours. Early pruning reduces it to about 24 minutes. Running with 8 cores instead of 1 further reduces the time to about 4 minutes, showing the scalability and effectiveness of the approach.

Table 3.5 Example of a Valid Duty with Two Breaks

ID	Start	End	From	To	Svc. Time	Gap	Break	Rake Change
18	06:15	06:21	VND	KKDA	6	0	–	–
583	06:21	06:36	KKDA	MUPR	15	4	–	–
586	06:40	06:55	MUPR	KKDA	15	0	–	–
28	06:55	08:01	KKDA	PVGW	66	45	45 min	Yes
70	08:46	09:52	PVGW	KKDA	66	51	51 min	Yes
119	10:43	11:50	KKDA	PVGW	67	0	–	–
442	11:50	12:25	PVGW	PVGW	35	0	–	–
159	12:25	13:31	PVGW	KKDA	66	–	–	–

Table 3.6 Constraint Summary for Valid Duty

Constraint	Condition	Status
Total Driving Time	$340 \leq 360$	OK
Cumulative Breaks	$96 \leq 120$	OK
Duty Duration	$436 \leq 445$	OK
Max Continuous Drive	$168 \leq 180$	OK
Jurisdiction Match	$2 = 2$	OK
Rake Changes	2	PVGW after 28, KKDA after 70

Table 3.7 Comparison of different modes of the enumeration scheme

	Serial computing without early pruning	Serial computing with early pruning	Parallel computing with 4 processors	Parallel computing with 8 processors
Number of feasible duties enumerated	647,509	647,509	647,509	647,509
Time taken(s)	> 5 hours	1432 seconds	422 seconds	215 seconds

3.4 Stage 4: Set partitioning model and optimization

Once the complete pool of all feasible duties has been obtained, the crew scheduling problem is formulated as a Set Partitioning Problem (SPP). The objective is to select the minimum number of duties in a way that every service is covered exactly once. All duties are not same, in the sense that some involving work in early morning or late evening hours may incur more cost for the operator. So, instead of minimizing the number of duties, we can also minimize the sum of costs of the selected duties. Let P be the set or the pool of all duties enumerated in Stage 3. The mathematical model has one binary variable x_p for each duty $p \in P$ which should take value 1 if the duty is selected in the plan and 0 otherwise. For each service ($i \in S$) in the system, exactly one selected duty should contain it. For a particular service, say s , this requirement is modeled as

$$\sum_{p \in P} a_{sp} x_p = 1,$$

where a_{sp} is a parameter with value 1 if duty p includes service s , and 0 otherwise. The complete model is:

$$\begin{aligned} & \min \sum_{p \in P} c_p x_p \\ & \text{subject to } \sum_{p \in P} a_{ip} x_p = 1, \text{ for each } i \in S, \end{aligned}$$

$$x_p \in \{0, 1\}, \text{ for each } p \in P.$$

The above model has parameters c_p that model the cost of each duty to the system. These costs have to be provided by the operator. If we want to minimize the total number of duties, then $c_p = 1$ for all $p \in P$.

The above model can be solved using an off-the-shelf open-source integer linear optimization solver like CBC [26], SCIP [27], Highs [28] or any commercially available solver. These solvers use a linear programming based branch-and-cut algorithm along with heuristic search, presolving, conflict analysis, symmetry exploitation etc to solve general mixed-integer linear optimization models. The solvers give optimal solutions in reasonable time up to a certain number of services and duties, beyond which they can provide a feasible solution and a bound on the cost. For the SPP model we have, the number of possible duties is of the order of 10^5 , and as result, these solvers take a long time to solve. The main bottleneck in these models is solving the linear optimization relaxation, where the constraints $x_p \in \{0, 1\}$ are relaxed to allow any value in the interval $[0, 1]$. This relaxation is solved repeatedly with small modifications so that the search converges to an optimal solution. Simplex method usually works best for most models because it can warm-start from the previous solution after some constraints or bounds have changed.

For our model, we observe that an advanced crashing method called ‘Idiot Crash’ [9, 10] implemented in the CLP linear optimization solver is particularly suited for set partitioning models. We further note that the crashing method is faster even when the linear relaxation is modified and re-solved. That means re-solving the linear relaxation with simplex is still slower than using the crashing method from scratch. Based on this observation, we customize a branch-and-cut method to use the Idiot Crash repeatedly throughout the branch-and-cut tree completely ignoring the dual-simplex method that most solvers use by default.

Set partitioning models have been used in branch-and-price framework to great effectiveness [7], but requires good implementation that is tied to a particular solver. The proposed approach, on the other hand, is simple to implement. The rules for enumerating duties can be implemented independently of the solver. The drawback is that it is suitable for medium sized models only, as is the case of daily services of a metro line.

A csv file containing a list of all feasible duties is first read by a Python program, and a mathematical optimization model is created using Pyomo [29], an open-source tool for modeling. The model is then given to a solver. This approach is modular and any solver for integer linear optimization can be used. We have implemented our specialized routines for large-scale set partitioning Minotaur [11], an open-source solver. The linear relaxations are solved using CLP [9], which includes the Idiot Crash implementation. At each iteration of branch-and-bound, Idiot Crash is invoked to solve the linear programming (LP) relaxation. Pseudocost branching is used to select branching variables because strong branching and reliability branching are both expensive for this model, and the LP relaxation already gives a solution value very close to the optimal. Depth first search is used as a node-selection rule to avoid excessive memory use. Among the two children of a node in the branch-and-bound tree, the one where a variable is fixed to value 1 is selected first, because setting a variable to 1 in a set partitioning model sets many other variables to 0, while setting a variable to 0 has little impact.

For computational testing, we compare the proposed approach to off-the-shelf solvers with no customization or problem specific options. We selected open-source solvers SCIP [27], CBC [26], HiGHS [28] and a state-of-the-art commercial solver. This comparison is not to benchmark the solvers but to only understand the effects of the proposed approach on set-partitioning models arising from crew-planning only. Table 3.8 gives a summary of time taken on a Linux computer with Intel(R) Xeon(R) Gold 6130 2.10GHz CPU with 128GB RAM. All solvers were run with default configuration with number of threads 1 on a single instance based on a timetable provided by Delhi Metro. HiGHS, SCIP and CBC all spent more than one hour in processing the root only because the dual-simplex they use after adding cutting planes takes a long time. Minotaur-default is also very slow. It solves a few nodes more than other solvers

Table 3.8 Comparison of off-the-shelf solvers to the customized approach on the set partitioning model in crew pairing.

Solver	Time (s)	Obj Value	Nodes	LPs solved	Gap
CBC	>3600	–	1	1	–
HiGHS	>3600	–	1	46	–
SCIP	>3600	–	1	1	–
Commercial	>3600	121	1431	–	9.09%
Default Minotaur	>3600	–	254	734	–
Customized Minotaur	314	114	78	78	3.66%

Table 3.9 Snapshot of the trip chart generated.

Rake	Induction	Induction	PVGW		KKDA		MUPR		Return
No.	Station	Time	Duty	Time	Duty	Time	Duty	Time	Reversal
704	MKPR	05:50	503	06:03	503	07:09	503	07:24	MUPR
701	MKPD	05:40	501	06:14	501	07:20	501	07:35	MUPR
710	MKPD	05:50	504	06:24	504	07:30	218	07:45	MUPR
716	MKPD	06:00	505	06:34	505	07:40	505	07:55	MUPR
720	MKPD	06:10	507	06:44	507	07:50	507	08:05	MUPR
721	MKPD	06:20	508	06:54	508	08:00	224	08:15	MUPR

Table 3.10 Snapshot of the crew duty-chart generated after Stage 5.

Duty	On	Off	Rake	From	To	Dep.	Arr.	Dur.	Brk.	SB	Jur.
201	MUPR	VND	717	MUPR DN SDG	MUPR	05:30	06:20	50	0	–	Y
			717	MUPR	KKDA UP	06:20	06:35	15	0	–	
			717	KKDA UP	PVGW UP	06:35	07:41	66	58	–	
			712	PVGW UP	PVGW DN	08:39	09:13	34	0	–	
			712	KKDA DN	KKDA DN	09:13	10:18	65	34	–	
501	MKPD	PVGW	701	MKPD	PVGW DN	05:40	06:14	34	0	–	Y
			701	PVGW DN	KKDA DN	06:14	07:20	66	0	–	
			701	KKDA DN	KKDA UP	07:20	07:57	37	83	726	
			717	KKDA DN	KKDA UP	09:20	09:58	38	0	732	
			732	KKDA UP	PVGW UP	09:58	11:05	67	31	–	

because it does not have any cut generation for set partitioning models. The custom algorithm, on the other hand, was able to provide a solution within 5% of the optimal in about 6 minutes. It used about 4 seconds per LP (with more than 600,000 variables), which is close to the performance of a commercial solver. Customizing these techniques for crew-planning and other applications that use set partitioning models is a topic of our future research.

3.5 Output generation: trip chart and verification files

The selected duties obtained from the optimization model are converted into operational outputs that the planning team requires. Post-processing generates a trip chart (see Table 3.9) which lists the service that covers each rake movement and the duty in which that service appears. The other output is a duty chart (see Table 3.10) that lists the services and rake movements covered by each duty.

Table 3.11 Sample Service Data

Serv.	Rake	Start Stn	Start	End Stn	End	Dir.	Time	Juris.
577	717	MUPR DN	05:30	MUPR	06:20	UP	50	Yes
11	712	SAKP	05:45	KKDA DN	06:59	DN	74	No
570	709	IPE	05:45	PVGW UP	06:57	UP	72	No

Table 4.1 Computational results of the proposed graph-based crew scheduling framework on a sample timetable.

Parameter	Value
Metro Network	Delhi Metro Pink Line Main Loop
Total Number of Services	944
Graph Representation	Directed Service Network
Path Generation Method	Stack-Based DFS
Number of Processing Cores Used	8
Total Feasible Duties Generated	647,509
Feasible Duty Generation Time	4 Minutes
Optimization Model	Set Partitioning Problem
Optimization Solver	Minotaur
Optimal Number of Duties	114
Optimization Time	7.78 Minutes
Service Coverage	100%

4 Conclusions

Before the adoption of computational approaches, crew scheduling in the Delhi Metro system was largely performed manually using complex spreadsheet manipulations. Manual creation of crew-duties and trip-charts needs timetable-controller personnel to examine service timetables, rake movements, break requirements, and operational rules individually while constructing feasible crew duties. Due to the large number of train services and complex operational constraints, the manual process was highly time-consuming, labor-intensive, and dependent on human expertise. In addition, manual scheduling often suffered from limited scalability, difficulty in handling large datasets, inconsistent duty quality, and a higher possibility of human errors or constraint violations. Generating and validating feasible duties manually for large metro lines required significant operational effort and was not suitable for rapid schedule revisions or large-scale optimization.

This work presented a graph-based and optimization-driven framework for solving the crew scheduling problem for the Delhi Metro Pink Line main loop operations. A directed service network was constructed to model feasible connections between train services while incorporating operational constraints such as station continuity, rake feasibility, duty duration limits, driving time restrictions, break conditions, and jurisdiction requirements. Using this graph representation, feasible duty paths were efficiently generated through stack-based depth-first search along with constraint-based pruning strategies.

Overall, the proposed methodology provides a scalable, efficient, and practically applicable solution for automated metro crew scheduling. The framework reduces manual scheduling effort, improves computational performance, and ensures operational feasibility, making it suitable for deployment in large urban rail transit systems. Future work may focus on integrating additional operational objectives such as fairness, robustness under disruptions, multi-day duty-chart preparation, and real-time schedule adaptation.

References

- [1] Caprara, A., Kroon, L., Monaci, M., Peeters, M., Toth, P.: Passenger railway optimization. In: Barnhart, C., Laporte, G. (eds.) *Transportation. Handbooks in Operations Research and Management Science*, vol. 14, pp. 129–187. Elsevier, Amsterdam (2007)
- [2] Arabeyre, J.P., Fearnley, J., Steiger, F.C., Teather, W.: The airline crew scheduling problem: A survey. *Transportation Science* **3**(2), 140–163 (1969)
- [3] Hoffman, K.L., Padberg, M.: Solving airline crew scheduling problems by branch-and-cut. *Management Science* **39**(6), 657–682 (1993)
- [4] Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, USA (1979)
- [5] Gilmore, P.C., Gomory, R.E.: A linear programming approach to the cutting-stock problem. *Operations Research* **9**, 849–859 (1961)
- [6] Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., Vance, P.: Branch-and-price: Column generation for solving huge integer programs. *Operations Research* **46**(3), 316–329 (1998)
- [7] Desrosiers, J., Lübbecke, M., Desaulniers, G., Gauthier, J.B.: *Branch-and-Price*. Springer, Switzerland (2026)
- [8] Jütte, S., Thonemann, U.W.: Divide-and-price: A decomposition algorithm for solving large railway crew scheduling problems. *European Journal of Operational Research* **219**(2), 545–556 (2012)
- [9] Forrest, J., Vigerske, S., Ralphs, T., Hafer, L., Fasano, J.P., Santos, H.G., Jan-Willem, Saltzman, M., Andre, A., Kristjansson, B., Cullmann, C., Gassmann, H.I., King, A., Arevall, Mart, B., Bonami, P., Luies, R., Brito, S.: COIN-OR/Clp: Release 1.17.11. Zenodo. <https://github.com/coin-or/Clp> (2026)
- [10] Galabova, I.L., Hall, J.A.J.: The ‘idiot’ crash quadratic penalty algorithm for linear programming and its application to linearizations of quadratic assignment problems. *Optimization Methods and Software* **35**(3), 488–501 (2020)
- [11] Mahajan, A., Leyffer, S., Linderoth, J., Luedtke, J., Munson, T.: Minotaur: A mixed-integer nonlinear optimization toolkit. *Mathematical Programming Computation* **13**(1), 1–47 (2021)
- [12] Heil, J., Hoffmann, K., Büscher, U.: Railway crew scheduling: Models, methods and applications. *European Journal of Operational Research* **283**(2), 405–425 (2020)
- [13] Desrochers, M., Soumis, F.: A column generation approach to the urban transit crew scheduling problem. *Transportation Science* **23**(1) (1989)
- [14] Borndörfer, R., Schelten, U., Schlechte, T., Weider, S.: A column generation approach to airline crew scheduling. In: Haasis, H.-D., Kopfer, H., Schönberger, J. (eds.) *Operations Research Proceedings 2005*, pp. 343–348. Springer, Berlin, Heidelberg (2006)
- [15] Lusby, R.M., Dohn, A., Range, T.M., Larsen, J.: A column generation-based heuristic for rostering with work patterns. *Journal of the Operational Research Society*, 63(2): 261–277 (2012)

- [16] Zhou, J., Xu, X., Long, J., Ding, J.: Metro crew planning with day-off pattern, duty type, and rostering scheme considerations. *Transportation Research Part C: Emerging Technologies* **143**, 103832 (2022)
- [17] Xue, F., Zhang, X., Hu, P., Ma, X.: Metro crew planning with heterogeneous duty paths and period-cycle pattern considerations. *Computers & Industrial Engineering* **182**, 109354 (2023)
- [18] Caprara, A., Fischetti, M., Toth, P.: Models and algorithms for railway crew management. *Mathematical Programming* **79**, 125–141 (1997)
- [19] Huisman, D., Kroon, L., Lentink, R., Vromans, M.: Operations Research in Passenger Railway Transportation. *Statistica Neerlandica*, 59(4) (2005)
- [20] Freling, R., Huisman, D., Wagelmans, A.P.M.: Models and algorithms for integration of vehicle and crew scheduling. *Journal of Scheduling* **6**(1), 63–85 (2003)
- [21] Han, A., Li, E.: A constraint programming-based approach to the crew scheduling problem of the Taipei mass rapid transit system. *Annals of Operations Research* **223** (2014)
- [22] Abbink, E., Fischetti, M., Kroon, L., Timmer, G., Vromans, M.: Reinventing crew scheduling at Netherlands railways. *Interfaces* **35**(5), 393–401 (2005)
- [23] Liebchen, C., Türk, C., Vornhusen, B.: Railway crew rostering: Tailored optimization solutions in daily use in berlin. *Urban Rail Transit* (2026)
- [24] Bansal, A., Anoop, K.P., Rangaraj, N.: A heuristic for railway crew scheduling with connectivity of schedules. *Transportation Research Record* **2678**(3), 873–887 (2024)
- [25] Dasgupta, S., Papadimitriou, C.H., Vazirani, U.V.: *Algorithms*. McGraw-Hill Education, Berkeley (2006)
- [26] Forrest, J., Santos, H.G., Ralphs, T., Vigerske, S., Hafer, L., Kristjansson, B., Jan-Willem, Straver, E., Lubin, M., Lougee, R., Brito, S., Gassmann, H.I., Cullmann, C., Cristina, Saltzman, M., Pitrus, B., Panov, D., Matsushima, F., Vossler, P.: COIN-OR/Cbc: Release v2.933. Zenodo. <https://github.com/coin-or/Cbc> (2026)
- [27] Hojny, C., Besançon, M., Bestuzheva, K., Borst, S., Chmiela, A., Dionísio, J., Eifler, L., Ghannam, M., Gleixner, A., Göß, A., Hoen, A., Hulst, R., Kamp, D., Koch, T., Kofler, K., Lentz, J., Maher, S.J., Mexi, G., Mühmer, E., Pfetsch, M.E., Pokutta, S., Serrano, F., Shinano, Y., Turner, M., Vigerske, S., Walter, M., Weninger, D., Xu, L.: The SCIP Optimization Suite 10.0. Technical report, Optimization Online (November 2025)
- [28] Huangfu, Q., Hall, J.A.J.: Parallelizing the dual revised simplex method. *Mathematical Programming Computation* **10** (2018)
- [29] Bynum, M.L., Hackebeil, G.A., Hart, W.E., Laird, C.D., Nicholson, B.L., Sirola, J.D., Watson, J.-P., Woodruff, D.L.: *Pyomo—optimization modeling in Python*. Springer (2021)