SCHEDULING OF TRACK INSPECTION AND MAINTENANCE ACTIVITIES IN RAILROAD NETWORKS

BY

FAN PENG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Civil Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Doctoral Committee:

Assistant Professor Yanfeng Ouyang, Chair
Dharma Acharya, Ph.D., CSX Transportation
Professor Christopher P.L. Barkan
Professor Rahim F. Benekohal

ABSTRACT

The U.S. railroad companies spend billions of dollars every year on railroad track maintenance in order to ensure safety and operational efficiency of their railroad networks. Besides maintenance costs, other costs such as train accident costs, train and shipment delay costs and rolling stock maintenance costs are also closely related to track maintenance activities. Optimizing the track maintenance process on the extensive railroad networks is a very complex problem with major cost implications. Currently, the decision making process for track maintenance planning is largely manual and primarily relies on the knowledge and judgment of experts. There is considerable potential to improve the process by using operations research techniques to develop solutions to the optimization problems on track maintenance. In this dissertation study, we propose a range of mathematical models and solution algorithms for three network-level scheduling problems on track maintenance: track inspection scheduling problem (TISP), production team scheduling problem (PTSP) and job-to-project clustering problem (JTPCP).

TISP involves a set of inspection teams which travel over the railroad network to identify track defects. It is a large-scale routing and scheduling problem where thousands of tasks are to be scheduled subject to many difficult side constraints such as periodicity constraints and discrete working time constraints. A vehicle routing problem formulation was proposed for TISP, and a customized heuristic algorithm was developed to solve the model. The algorithm iteratively applies a constructive heuristic and a local search algorithm in an incremental scheduling horizon framework. The proposed model and algorithm have been adopted by a Class I railroad in its decision making process. Real-world case studies show the proposed approach outperforms the manual approach in short-term scheduling and can be used to conduct long-term what-if analyses to yield managerial insights.

PTSP schedules capital track maintenance projects, which are the largest track maintenance activities and account for the majority of railroad capital spending. A time-space network model was proposed to formulate PTSP. More than ten types of side constraints were considered in the model, including very complex constraints such as mutual exclusion constraints and consecution constraints. A multiple neighborhood search algorithm, including a decomposition and restriction search and a block-interchange search, was developed to solve the model. Various performance enhancement techniques, such as data reduction, augmented cost function and subproblem prioritization, were developed to improve the algorithm. The proposed approach has been adopted by a Class I railroad for two years. Our numerical results show the model solutions are able to satisfy all hard constraints and most soft constraints. Compared with the existing manual procedure, the proposed approach is able to bring significant cost savings and operational efficiency improvement.

JTPCP is an intermediate problem between TISP and PTSP. It focuses on clustering thousands of capital track maintenance jobs (based on the defects identified in track inspection) into projects so that the projects can be scheduled in PTSP. A vehicle routing problem based model and a multiple-step heuristic algorithm were developed to solve this problem. Various side constraints such as mutual exclusion constraints and rounding constraints were considered. The proposed approach has been applied in practice and has shown good performance in both solution quality and efficiency.

ACKNOWLEDGMENT

I owe my deepest gratitude to my advisor, Professor Yanfeng Ouyang. It was he who led me into the field of operations research and helped me tremendously throughout my graduate study. He has not only taught me the scientific knowledge, but also influenced me with his passion and scientific rigor. I feel very fortunate to study under his guidance.

I would also like to thank the three other committee members, Dr. Dharma Acharya, Professor Christopher P.L. Barkan and Professor Rahim F. Benekohal, for spending their time reviewing this work and providing valuable comments and suggestions. I am grateful for the great opportunity I had to conduct research in the field of railroad operations and engineering. I would also like to express my thanks to all UIUC professors from whom I have taken courses and learned knowledge.

I would like to thank the colleagues from CSX Transportation who have collaborated with me on this dissertation research. I would like to particularly thank Kamalesh Somani. We worked very closely on this research in the past two and half years. I have learned much railroad engineering knowledge from him.

I would like to thank all my friends at the University of Illinois, including Xiaopeng Li, Xiying Mi, Wei Xu, Yun Bai, Seyed Mohammad Nourbakhsh and Taesung Hwang, who have helped me in both research and life. Special thanks go to Xiaopeng Li, who collaborated with me on a number of research and course projects. I also appreciate the help for this research from Seungmo Kang, who is now a faculty member at Korea University, and Seyed Mohammad Nourbakhsh.

Finally, I would like to thank my grandparents and parents. They are my most important people, and it is their love that supports me to finish my doctoral study.

This work is financially supported by CSX Transportation, Inc. The Illinois Department of Transportation and the National Science Foundation have also partially supported the first two years of my graduate study at the University of Illinois.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1 INTRODUCTION

1.1 Background

According to the Association of American Railroads (AAR), the U.S. Class I railroads (i.e., the freight railroad companies with 2009 operating revenue of $378.8 million or more) operated 160,781 miles of rail tracks in 2009 (AAR, 2010), including 62,067 miles of high-density "A" tracks (i.e., tracks with a freight density of at least 20 million gross ton-miles per track mile per year). The extensive railroad network carried 42.7% of the U.S. freight revenue ton-miles in 2007 (AAR, 2010). In order to ensure operational efficiency and safety, railroad companies spend billions of dollars every year[1] on track maintenance (Tolliver and Benson 2010) to (i) recover the serviceability of tracks from defects and damages, (ii) prevent further wear-out, and (iii) eliminate potential safety hazards.

Track maintenance has significant impacts on train accidents and train delay. Track defects, if not identified and repaired in time, may damage railcars and locomotives (leading to higher rolling stock maintenance costs) or even cause train accidents. According to Federal Railroad Administration (FRA) Office of Safety Analysis (2010), track defects have become the leading cause of train accidents in the United States. Among the 1,890 train accidents happened in 2009, 658 or 34.81% of them were caused by track defects, resulting in a total reportable damage of $108.7 million. On the other hand, track maintenance activities may disturb train operations and cause delays to both trains and shipments (or passengers). Schafer and Barkan (2008) and Schlake et al. (2009) estimated that train delay costs alone can be as high as $200 to $300 per hour per train, and shipment or passenger delay costs can make the actual costs even much higher.

[1] In 2008, the Class I railroads alone spent 7.52 billion dollars on track maintenance.

Given the huge costs related to track maintenance, even a small percentage of cost reduction implies a large saving in absolute value. One way to achieve such savings is to optimize the track maintenance decision-making process. Figure 1 illustrates some of the optimization problems on track maintenance in one of the Class I railroads[2], which fit into four categories: (i) the track inspection scheduling problem (TISP), (ii) the capital track maintenance scheduling problem (CTMSP), (iii) the routine track maintenance scheduling problem (RTMSP), and (iv) the track maintenance logistics problem (TMLP).

[2] Other railroads may have slightly different procedures but the general framework should be similar.

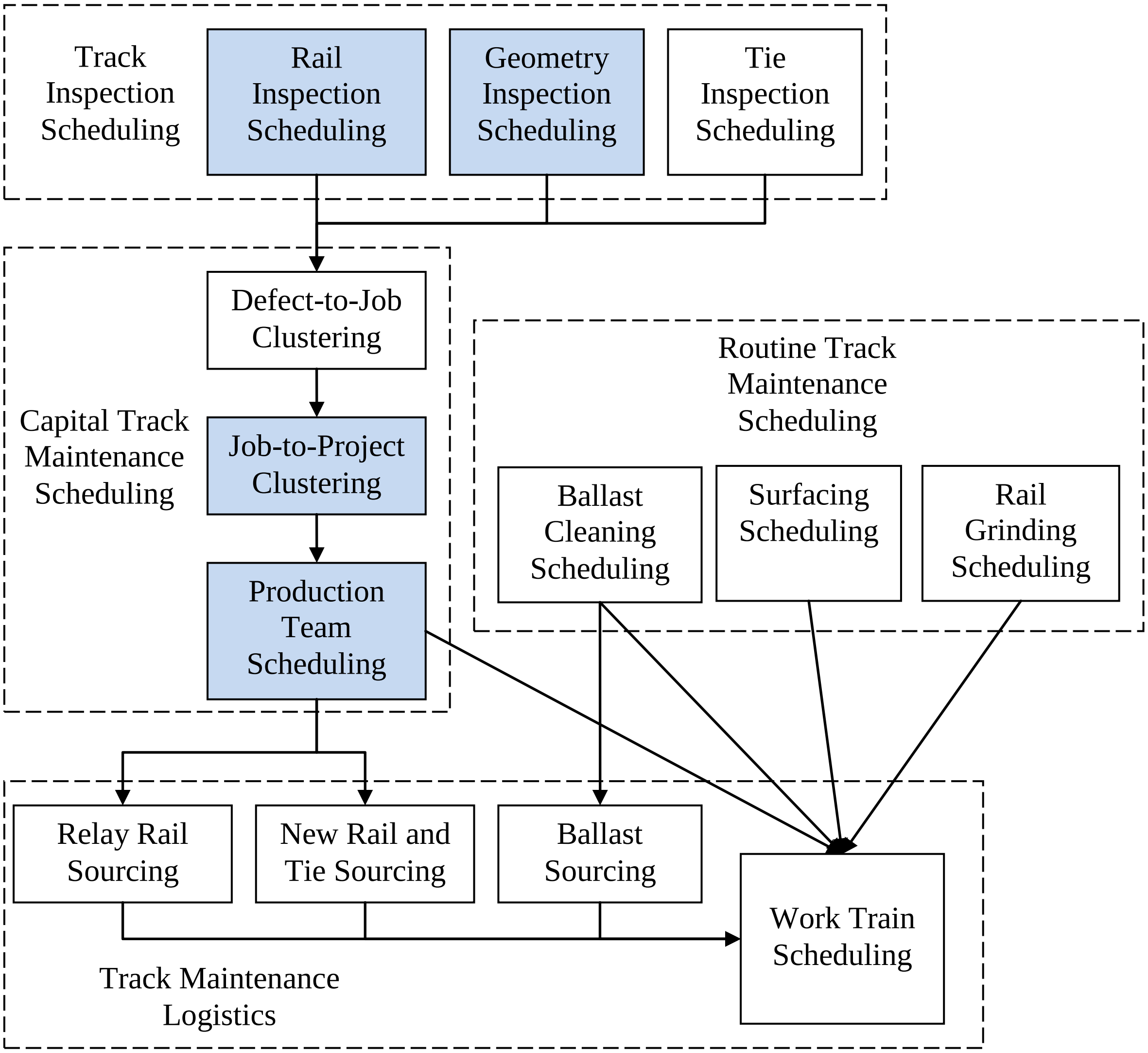

Track Inspection Scheduling
Rail Inspection Scheduling
Geometry Inspection Scheduling
Tie Inspection Scheduling
Defect-to-Job Clustering
Capital Track Maintenance Scheduling
Job-to-Project Clustering
Production Team Scheduling
Routine Track Maintenance Scheduling
Ballast Cleaning Scheduling
Surfacing Scheduling
Rail Grinding Scheduling
Relay Rail Sourcing
New Rail and Tie Sourcing
Ballast Sourcing
Work Train Scheduling
Track Maintenance Logistics

Figure 1. Optimization problems on track maintenance in a Class I railroad company.

Railroad tracks are inspected periodically to identify the defects in order to avoid train damage or accidents. The first category of track maintenance problems, TISP, schedules such track inspection activities (for simplicity, track inspection activities will be referred to as “tasks” from now on). Track inspection includes rail inspection, tie inspection and geometry inspection, which are performed by corresponding inspection teams. A team is usually composed of an inspection vehicle and its crews (some inspections are conducted manually by inspectors walking along the track; in this study we only focus on those automated inspections by inspection vehicles). Rail inspection teams examine rail tracks for both external and internal rail defects (such as kidney defects, wheelburn defect, head checking and squats) by visual inspection and technologies such as induction and ultrasonic devices (Cannon et al., 2003). Geometry inspection teams measure geometric parameters of the track, such as gauge, curvature and alignment (FRA, 2002), by technologies such as laser measurement systems and accelerometers. Tie inspection teams identify deteriorated ties using technologies such as the Gage Restraint Measurement System, which is installed on a train and is able to find deteriorated ties while the train travels on the track (FRA, 1999). For every inspection task, the TISP determines its start time and assigns an inspection team to it, in a way that certain business requirements (e.g., track should be inspected at certain inspection frequencies) are satisfied and the operating costs (e.g., for inspection teams) are minimized.

Track maintenance activities can be divided into three categories: corrective maintenance, capital maintenance and routine maintenance. Corrective maintenance activities are usually performed by local maintenance teams to fix the most severe and urgent defects identified from track inspection. Capital maintenance activities (i.e., “projects”) involve the replacement of large quantities of components with large maintenance teams (i.e., production teams) and expensive machines. They are normally charged to capital expenditures and accounted for the majority of the total capital spending (e.g., 67% in 2002) in U.S. railroads (Grimes and Barkan, 2006). Projects are carefully identified and planned every year with sophisticated decision making process (Gorman and Kanet, 2010), partially based on recorded track inspection data (Acharya et al., 1990; Simson et al., 1999). Projects are identified and scheduled in the capital track maintenance scheduling problem (CTMSP) in three steps. First, maintenance jobs (referred to as “jobs” from now on) need to be created based on the recorded track defects. Then, the job-to-project clustering problem (JTPCP) clusters jobs into projects such that the duration of every project spans an integer number of weeks. Finally, the projects are scheduled by the production team scheduling problem (PTSP). Production teams (also called production gangs in some railroads) are composed of a large number of crews equipped with highly specialized machines (such as ballast regulators, tampers, spike inserters, spike pullers, and tie extractors/inserters). There are multiple categories of production teams which are specialized at different categories of projects; e.g., rail teams repair and replace defected rail tracks, and timber and surfacing (T&S) teams replace deteriorated ties and tamp the ballast.

Routine maintenance activities include ballast cleaning, surfacing and rail grinding, and are performed by specialized routine maintenance teams (which are typically smaller than production teams). These routine maintenance activities are scheduled in the routine track maintenance scheduling problem (RTMSP). A ballast cleaning team uses a ballast cleaner to replace worn and unusable ballast in order to improve track drainage and ballast load-bearing elasticity. A surfacing team is equipped with machines such as ballast regulator and tamper to correct the track alignment and make the track more durable. A rail grinding team uses a rail grinder to restore the profile of worn rails and remove their irregularities. Different from projects, routine maintenance activities are performed at regular frequencies, independent of track inspection results. Note that different railroads may have different classifications of capital and routine maintenance activities. For example, ballast cleaning activities are classified and scheduled as capital maintenance projects in some railroad companies.

The track maintenance logistics problem (TMLP) decides where to obtain maintenance supply materials and how to transport the materials and maintenance teams to maintenance sites with work trains. The sources of materials, including new rail, relay rail (i.e., used rail that can be reused), ties and ballast, are determined so as to satisfy the demand from maintenance activities. Then, the schedule of work trains is determined to transport the materials and maintenance teams to maintenance sites. The work train scheduling problem covers the scheduling of work trains' locomotives, cars, crews and end-of-train devices.

The solutions to those track maintenance optimization problems have significant impacts on railroad performance and safety. For example, if a segment of track is not inspected in time, its potential defects will not be identified and train accident may occur. If the maintenance activities are not well scheduled, they may incur high maintenance costs (e.g., if they are scheduled during heavy snow), disturb train operations, and cause shipment delay.

However, track maintenance optimization problems are usually very large-scale and complex. Hundreds or thousands of maintenance activities, tens of maintenance teams and thousands of business constraints are usually involved in one single problem. Due to lack of systematic solution techniques, current practice in the railroad industry mostly relies on the knowledge, experience and judgment of experts. The solution process may take a long time but the obtained solution may not be satisfying. For example, in one of the Class I railroads, JTPCP and PTSP used to be solved manually. It took a team of experts at least one week to solve each of these problems; yet, many constraints could not be accommodated in the solution (some may even be overlooked). Whenever a tradeoff among conflicting constraints has to be made, it is very difficult for an expert to judge which constraints are more important. Sometimes data errors are overlooked by human, and they lead to incorrect results. It is also difficult to document the experts' experience and knowledge in a sustainable way so that the problems can be solved across generations. An automated solution approach that is general, efficient, and practically implementable will help the railroad industry gain significant cost savings and operational benefits in the long run.

As a final remark, we note that the flowchart in Figure 1 summarizes the current practice, and hence it may not be the optimum procedure. Ideally, all optimization problems should be solved in an integrated fashion, but that would make the problem too huge and too complex to solve. Hence, we still solve the problems separately. However, we bear in mind that the flowchart could possibly be improved, if better solution techniques are developed in the future. For example, relay rail sourcing could be incorporated into PTSP, or ballast sourcing could be solved with ballast work train scheduling.

1.2 Objectives and Contributions

This study addresses four major optimization problems on track maintenance, namely, the rail inspection scheduling problem (RISP), the geometry inspection scheduling problem (GISP), JTPCP and PTSP. These problems are highlighted in Figure 1. Since RISP and GISP are very similar, we study them together as one generalized TISP.

This study formulates innovative mathematical models for these interrelated yet different complex problems and develops customized algorithms to solve them. Few previous studies have been conducted on similar topics, and even fewer have been implemented in practice. The proposed models are complex and realistic enough to accurately reflect the business goals and constraints of the railroad companies. Many difficult constraints are identified from the industry practice and formulated for the first time. For example, in the conventional periodic vehicle routing problems, the periodicity constraints are imposed through a set of given candidate schedules. The TISP in this study, however, enforces the periodicity constraints via penalty costs associated with inspection interval lengths; as such, the inspection schedules could be much more flexible. Other new formulations include the PTSP's split project constraints, which require two projects to be performed either consecutively by the same team or simultaneously by multiple teams, and JTPCP's core model, which minimizes the total project duration after clustering jobs and rounding (up or down) the duration of each project.

A variety of innovative algorithms have been developed to solve the proposed models. The models are quite complex, and the practical problem instances are all very large-scale. No existing algorithms or solvers can be directly applied. Therefore, we developed customized algorithms to take advantage of the characteristics of those problems. The TISP algorithm includes two parts: (i) a constructive heuristic which tries to first schedule the most delayed tasks, and (ii) a local search algorithm which incorporates various performance enhancement techniques, such as prioritization and preliminary checks of local search moves, and elimination of duplicated moves and excessive calculation. The algorithm dynamically generates variables and constraints to improve the solution speed. The algorithm is implemented in an incremental scheduling horizon framework, which utilizes the feature of the optimal solution (e.g., tasks should generally be scheduled around their due times) to help the local search from being trapped in local optima.

The PTSP algorithm first solves a simplified scheduling model to obtain an initial solution which is able to satisfy a subset of side constraints. Then it applies a multiple neighborhood search to improve the solution. Two types of neighborhood structures are explored. The first one is decomposition and restriction. Restricted subproblems are solved by a mixed integer programming (MIP) algorithm, while various embedded techniques such as MIP cuts, augmented cost functions, subproblem tightening and subproblem prioritization, are used. The second neighborhood structure is block interchange, and the solution techniques include elimination of excessive calculation, preliminary checks of moves, and parallel computing. The iterative exploration of different neighborhood structures has been shown to be able to efficiently improve the solution. Other solution enhancement methods, such as project splitting and data reductions, are also incorporated into the algorithm.

The JTPCP algorithm utilizes features of the optimal solution to develop a constructive heuristic that addresses some of the dominating costs and constraints. It then iteratively applies a neighborhood search, which includes a job interchange and a project interchange, and a two-step feasibility heuristic, to attack the problem from different directions. Solution techniques, such as prioritization of moves, constraint relaxations, and solving subroutine problems, are embedded in the proposed algorithm.

The proposed models and solution approaches are shown to be very effective, efficient, and suitable for full-scale practical use. They have been adopted by a Class I railroad to help with its track maintenance decision making process in the past two years. Comparisons between our model solutions and previous manual solutions show that our models are able to obtain much better solutions and could bring significant cost savings and efficiency improvements. Although the studied problems are applied to only one Class I railroad, the proposed models and algorithms can be easily adapted to address similar track maintenance problems in other railroad companies. It shall also be possible to apply similar modeling and algorithm development approaches to other problems in different network scheduling contexts, such as railroad transportation scheduling, highway transportation scheduling and airline scheduling.

1.3 Outline

The remainder of this dissertation is organized as follows.

Chapter 2 reviews previous studies on routing and scheduling problems related to TISP and PTSP. Related general mathematical models, such as vehicle routing problem (VRP) and time-space network (TSN) models, and suitable solution algorithms are also reviewed.

Chapter 3 presents a VRP-based model for TISP. Various types of side constraints, such as periodicity constraints and discrete working time constraints, are addressed. A heuristic algorithm is proposed for the model, where a constructive heuristic and a local search heuristic are iteratively applied within an incremental horizon framework. The proposed approach is applied to real-world TISP instances and shown to solve both short-term and long-term planning instances efficiently.

Chapter 4 presents a TSN-based model for PTSP, including multiple types of side constraints (such as mutual exclusion constraints and split project constraints). A scheduling model is formulated to obtain an initial solution. An iterative multiple neighborhood search algorithm is developed to improve the solution. The search algorithm explores two different neighborhood structures: decomposition and restriction, where the MIP algorithms are applied, and block interchange. Other solution techniques, such as project splitting and data reductions, are used to improve the algorithm. Case studies have been conducted with real-world data and it is shown that the proposed approach significantly outperforms the state-of-the-art process used by the railroad.

Chapter 5 presents a VRP-based model for JTPCP. Side constraints such as limit constraints and project duration constraints are included in the formulation. A multiple-step heuristic algorithm is proposed to solve the model. The algorithm is composed of a constructive heuristic, a local search algorithm including two types of neighborhood structures, and a two-step feasibility heuristic. Case studies with real-world data have shown that the proposed approach is able to bring improvements to both solution quality and efficiency.

Chapter 6 summarizes this dissertation and discusses future research directions.

CHAPTER 2 LITERATURE REVIEW

This chapter reviews a few existing studies on the track inspection scheduling problem (TISP) and the production team scheduling problem (PTSP)[3]. The problem definition and solution approaches will be briefly introduced. This chapter also reviews two most commonly-used optimization models for network routing and scheduling problems: vehicle routing problem (VRP) model and time-space network (TSN) model, and the algorithms that could be used to solve these problems.

2.1 Track Maintenance Problem Review

2.1.1 Track Inspection Scheduling Problem

One important characteristic of TISP is the inspection frequency requirement. The railroad network can be divided into many segments, each of which requires periodic inspection. If a segment is not inspected for a long time, its risk of having track defects increases and accidents may occur. Therefore, a railroad company usually has a fleet of inspection vehicles (teams) that travel through the railroad network and inspect the track continuously.

Morales et al. (2008) studied a geometry inspection scheduling problem (GISP) which is one of the TISPs. Besides the inspection frequency requirement, some constraints for rail-bound inspection vehicles were also considered. These constraints include crew change point constraints and track restrictions such as directionality, sharp-turn constraints and multiple tracks. The inspection territories of vehicles were pre-determined as input data, and the model scheduled

[3] The job-to-project clustering problem (JTPCP) is a new problem with no prior research.

one vehicle at a time. The model enumerated all possible day routes as decision variables, and could usually be solved by a commercial integer programming solver within 12 hours. What-if analysis was used to reassign territories among inspection teams in order to balance workload and improve the solution.

Routine track maintenance scheduling, such as rail grinding scheduling, are subject to track inspection frequency requirements (which is similar to TISP). Unlike PTSP, the routine track maintenance scheduling problem (RTMSP) does not have a set of pre-planned one-time maintenance activities (i.e., projects) as input. Instead, their input is a set of track segments which require periodic maintenance. Therefore, RTMSP can usually be solved in a similar way as TISP but not as PTSP. Derinkuyu et al. (2010) recently studied a rail grinding scheduling problem in a Class I railroad and developed an optimization model whose objective was to minimize the deviations of grinding activities from the given set of desired maintenance frequencies. A heuristic algorithm was developed to solve the problem. Retharekar and Mobasher (2010) studied a preventive maintenance scheduling problem. Constraints related to various factors such as train schedules and desired maintenance frequencies are considered. They also used a heuristic algorithm to solve the problem.

2.1.2 Production Team Scheduling Problem

Many studies have investigated the problems of highway infrastructure maintenance planning in either long-term or short-term horizons (see a comprehensive review in Ng et al., 2009). The impacts of maintenance activities on traffic operations are usually represented by traffic delay and congestion. Long-term planning problems mostly focus on developing strategies to schedule maintenance projects with or without traffic rerouting options such that the long-run operational cost is minimized under limited investment budget (e.g., Golabi and Pereira, 2003; Ouyang and Madanat, 2004, 2006; Ouyang 2007; Durango-Cohen and Madanat, 2008; Unnikrishnan et al., 2009; Li et al., 2010). Short-term planning mostly focuses on mitigating instant traffic delays (Cheu et al., 2004; Ma et al., 2004; Jiang and Adeli, 2004). Studies have also been conducted on the maintenance of other types of transportation networks, such as water pipe networks (Dridi et al., 2008), where disruption of “traffic” should also be considered.

The maintenance scheduling problems of railroad track are usually very different from their highway counterparts. For example, railroad maintenance projects normally involve the transportation of large machinery over a large spatial area (e.g., the entire railroad network), so the travel costs of the maintenance teams are no longer negligible. Another difference is that trains are normally considered as discrete objects with very few rerouting options, and thus the impacts of maintenance activities on traffic operations are usually addressed by specific business constraints such as mutual exclusion constraints (e.g. no more than one maintenance activities in neighboring track sections should be carried out simultaneously). The railroad companies usually have many such rules which turn out to significantly increase the complexity and computational difficulty of the problem. These unique properties postulate the need for different model formulations and solution techniques.

Track maintenance scheduling problems (TMSP) have been studied only in recent years, and most existing studies focused on project-level TMSP on a single railroad track segment. Higgins (1998) and Higgins et al. (1999) developed a mathematical model to determine the assignment and schedules of maintenance teams so as to minimize the disruption of train operations. The model considered budget constraints, train schedules, crew travel times, and various interrelations among maintenance activities. The model was applied to a short-term project-level scheduling problem (a project lasting for several days on a track segment) and was solved by tabu search. Lake et al. (2001) modified the model to minimize the total maintenance costs, while team set-up and take-down times are also incorporated. Cheung et al. (1999) solved a resource-allocation problem for a railroad segment to optimize the assignment of maintenance activity requests based on priorities. Budai et al. (2006) discussed a preventive maintenance scheduling problem in which short-time routine activities and long-time unique projects are formulated differently. Zante-de Fokkert et al. (2007) developed a model to improve workers' safety by minimizing the number of night maintenance activities.

Very limited research has been done for TMSP in a large-scale railroad network. Network-level problems often involve hundreds of projects and complex relationships among them, and thus pose a much larger number of side constraints. For example, in order to reduce the impact on traffic operations, different railroad subdivisions (i.e., a section of tracks defined by railroads) incident to a junction (i.e., a place where multiple rail routes converge or diverge) or in a corridor (i.e., a route with heavy traffic traversing multiple subdivisions) are not allowed to have ongoing maintenance activities at the same time. PTSP is one of such network-level TMSP. The prohibiting complexity associated with large-scale PTSP prevents existing mixed integer programming (MIP) solvers, e.g. CPLEX (ILOG, 2006), from obtaining an optimal or even feasible solution.

Li et al. (2009) studied a version of PTSP with three types of costs: fixed team costs, travel costs and preference costs, and three types of side constraints: simultaneity, non-simultaneity and precedence constraints. In their problem, the number of teams is not fixed and is endogenously determined by the model. The problem was formulated into a TSN model. In order to solve the large-scale problem instance with MIP solver efficiently, both network and side constraints were significantly consolidated to reduce the number of variables and constraints. After preprocessing, the input data contained about 200 maintenance jobs, 300 side constraints, 9 network vertices and 72 network edges. An incremental solution strategy was also applied and was reported to be able to greatly reduce the solution time. However, the details about the algorithm have not been disclosed so far.

Gorman and Kanet (2010) studied a similar version of PTSP as the one studied by Li et al. (2009). In their problem, a production team is allowed to stay idle without doing any work, and the duration of a project is related to its start time. Two different models, TSN model and job scheduling model, were used to formulate the problem. The TSN formulation was solved with MIP solver, and the job scheduling formulation was solved with both constraint programming and genetic algorithm. Similar to Li et al. (2009), an approximation was applied to geographic and time definitions of TSN formulation in order to reduce its scale. A small problem instance with 16 projects and 7 side constraints was used to test and compare the three algorithms, and TSN formulation with MIP algorithm was recommended for large-scale problems. They reported that acceptable solutions to the real-world large-scale problems can be obtained in 6 hours by commercial MIP solver, but did not disclose the exact problem instance scale.

Bog et al. (2010) studied a different version of PTSP from another Class I railroad. Mutual exclusion, time window and precedence constraints were considered. A *k*-weekly incremental algorithm with backtracking was developed to solve the problem. The formulated problem was solved with MIP solver incrementally. Every time the problem was solved, only *k* weeks are considered and projects are added to the schedule to fill the *k* weeks. If no side constraints are violated, the model fixed the newly-added projects. Otherwise the model fixed a gradually decreasing number of projects and re-solved the *k* weeks. Travel costs were only considered between the last fixed projects and the first unfixed projects. In that way, the number of variables was greatly reduced at the cost of the loss of optimality. The model was applied to large-scale problem instances. It was reported that the model was not able to satisfy all side constraints, but the numbers of side constraints were not reported.

Nemani et al. (2010) studied the same version of PTSP as that of Bog et al. (2010). They presented four solution approaches: TSN model, duty generation model (DGM), column generation model and decomposition-based heuristics with DGM. In decomposed DGM, DGM was first used to obtain an initial solution without considering travel costs. Then the problem was decomposed week-wise, and subproblems were solved separately. All four approaches were applied to the same problem instances used by Bog et al. (2010), and the results were compared with those obtained by *k*-weekly progressive algorithm with backtracking proposed by Bog et al. (2010). Only decomposed DGM was able to obtain better solution, but it was still not able to satisfy all side constraints.

Peng et al. (2010) developed a TSN model for a PTSP which is similar to the one studied by Nemani et al. (2010) and Bog et al. (2010). The difference was that the exact travel costs and one more type of side constraint, preference constraints, were considered. A clustering model was used to obtain an initial solution. Then multiple neighborhood search was used to improve the solution. Two types of searches, decomposition and restriction, and project interchange, were used. Their proposed model and algorithm were adopted by a Class I railroad company.

2.2 Optimization Models and Algorithms

Scheduling (or timetabling) problem is a very broad class of optimization problems (Leung, 2004), including shop scheduling problems (Pinedo, 1995; Lee et al., 1997) and staff scheduling problems (Tien and Kamiyama, 1982; Balakrishnan and Wong, 1990). Generally, a scheduling problem takes the following input information: (i) a set of activities (or jobs, duties) which are to be performed, (ii) a set of teams (or machines, employees) which are used to perform the activities, and (iii) a scheduling time horizon. The output includes a schedule with two dimensions: team dimension and time dimension. Every activity is assigned to a point in the schedule with both team coordinate and time coordinate. In other words, every activity is assigned to a team and scheduled with a start time. Scheduling problems usually have some side constraints related to different activities and teams. Such side constraints may include (i) time window constraints, which require an activity to be performed within certain time windows, (ii) preference constraints, which require an activity to be performed by certain teams, and (iii) more complex constraints interrelating multiple activities and/or teams, such as precedence constraints which require an activity to be performed before another one. Those side constraints greatly increase the difficulty of finding the optimal schedule (or even a feasible schedule). In practice, because different scheduling problems may have different side constraints, different customized algorithms are usually developed to solve them efficiently.

Both TISP and PTSP are scheduling problems. They take as input a set of activities (inspection tasks or maintenance projects) and a set of teams (inspection teams or production teams) to perform these activities. However, they are different from the conventional scheduling problems because their activities are spatially-distributed in the railroad network. After finishing an activity, the team cannot start working on the next activity immediately. It must first travel a distance from the location of the finished activity to the location of the next activity. Such traveling has two possible impacts on the problems: 1) the time spent traveling may need to be considered, which postpones the start time of the next activity; and 2) the costs spent traveling may need to be considered, which add an additional term of travel costs to the objective function of the model. Problems considering such travel time and/or travel costs between spatial points are usually called routing problem, which is also a very broad class of optimization problems. Typical routing problems include the traveling salesman problem (TSP), where there is only one team, and the vehicle routing problem (VRP), where there are multiple teams (Dantzig and Ramser, 1959; Toth and Vigo, 2001).

Scheduling problems and routing problems have some common features (i.e., a set of activities and a set of teams), and there is no distinct boundary between them. Travel time/costs in routing problems can be considered as transition time/costs between activities in scheduling problems, and side constraints such as time window and precedence constraints in scheduling problems can also be imposed on routing problems (Selensky, 2001; Beck et al., 2002). Generally, if a problem mostly focuses on variables and constraints in the spatial dimension, i.e., those related to a pair of consecutively performed activities, then it is often classified as a routing problem; on the other hand, if it focuses on the temporal dimension, e.g., precedence constraints, then it is often classified as a scheduling problem. Due to such difference, routing problems and scheduling problems are usually solved using different algorithms (Beck et al., 2003). Selensky (2001) have tested scheduling algorithms on some benchmark routing problem instances, and also routing algorithms on some benchmark scheduling problem instances. It was found that the performances of algorithms were greatly degraded in both cases.

TISP and PTSP should be considered as both routing and scheduling problems, because variables and constraints in both spatial and temporal dimensions are equally important. Such problems in the literature are usually formulated as routing problems, where variables and constraints in the temporal dimension are considered in side constraints (e.g., VRP with time windows; see Soloman, 1987; and more complex problems; see Li et al., 2009 and Bog et al., 2010). In our research, TISP and PTSP are also formulated as routing problems with side constraints. JTPCP does not involve the time dimension. Therefore, it can be naturally formulated as a routing model with side constraints, and its core model (i.e., the routing problem model) is similar to those of TISP and PTSP.

Ignoring the side constraints, the core part of a routing problem can be described as follows: Let I' be a set of spatially-distributed activities representing tasks, projects, etc., each of which corresponds to a spatial location. We use index $i \in I'$ to represent both an activity and its location. Let K be a set of teams which are used to perform those activities. Let i_{start} and i_{end} be respectively the start location and end location of all teams (they can be considered as “dummy” activities). Let $I = I' \cup \{i_{\text{start}}\} \cup \{i_{\text{end}}\}$. The movement of team $k \in K$ from location $i \in I$ to location $j \in I$ incurs c_{ijk} and t_{ijk} , which respectively represent the cost and time needed to complete activity $i \in I$ and travel between $i, j \in I$. For simplicity, we assume $t_{ijk} = 0$ when $i = i_{\text{start}}$ or $i = i_{\text{end}}$. The typical objective is to determine the schedule of activities so that all activities are completed within a time horizon and the total costs are minimized.

The routing (or scheduling) problems are mostly solved with either a mixed integer programming (MIP) algorithm or a heuristic algorithm. If an MIP algorithm is used, the problem must be first formulated into an MIP model. In the following subsections, we will first review two conventional MIP models for scheduling and routing problems (i.e., VRP model and TSN model) and the corresponding MIP algorithms. Then we will review some well-known heuristic algorithms.

2.2.1 Vehicle Routing Problem Model

VRP model has been widely applied to real world problems; e.g., in the context of distribution systems between depots and customers. Let $\mathbf{x} = \{x_{ijk}\}$ be a set of binary variables so that $x_{ijk} = 1$ if team $k \in K$ perform activity $j \in I$ right after activity $i \in I$ (i.e., team $k \in K$ moves directly from location $i \in I$ to $j \in I$), and $x_{ijk} = 0$ otherwise. So $\mathbf{x}$ determines the routes of all teams. Let decision variables $\mathbf{u} = \{u_i\}$ be the start time for activity $i \in I$ to be performed by some team (without losing generality, we can enforce $u_{i_{\text{start}}} = 0$).

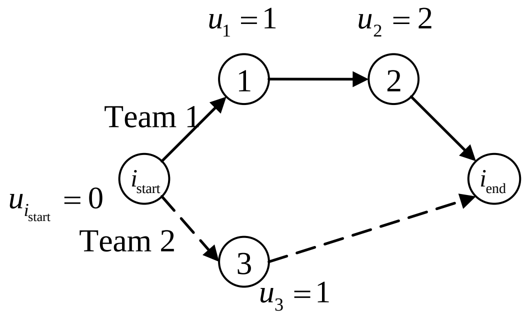


Figure 2. Illustration of vehicle routing problem model.

Figure 2 illustrates a simple problem instance formulated by the VRP model. A feasible schedule is also shown. Circles represent activities (and locations). There are five locations (including i_{start} and i_{end}), and three tasks at locations 1, 2 and 3. Suppose $t_{ijk} = 2$ for activity $i = 3$, and $t_{ijk} = 1$ for $i = 1, 2$, $\forall j \in I, k \in K$. There are two teams in the problem instance. Solid arrows represent the route of team 1, and dashed arrows represent that of team 2. It can be seen that both teams leave start location i_{start} at time $u_{i_{\text{start}}} = 0$. Then team 1 consecutively visits locations 1 and

2 respectively at time $u_1 = 1$ and $u_2 = 2$, and finishes its travel at end location i_{end}. Team 2 visits location 3 at time $u_3 = 1$, and then travels to i_{end}.

The VRP model is formulated as:

(VRP)
$$\min \sum_{i \in I} \sum_{j \in I} \sum_{k \in K} c_{ijk} x_{ijk} \,, \tag{2.1}$$

s.t.

$$\sum_{j \in I} x_{i_{\text{start}} jk} = 1, \qquad \forall k \in K, \tag{2.2}$$

$$\sum_{j \in I} x_{jik} - \sum_{j \in I} x_{ijk} = 0, \qquad \forall i \in I', k \in K, \tag{2.3}$$

$$\sum_{j \in I} \sum_{k \in K} x_{ijk} = 1, \qquad \forall i \in I', \tag{2.4}$$

$$u_i + t_{ijk} + U(x_{ijk} - 1) \le u_j \,, \qquad \forall i, j \in I, k \in K, \tag{2.5}$$

$$x_{ijk} \in \{0,1\}, \qquad \forall i, j \in I, k \in K. \tag{2.6}$$

Here U is a number no smaller than the maximum possible value of $u_{i_{\text{end}}}$. Objective (2.1) minimizes the total costs. Constraints (2.2) and (2.3) are flow conservation constraints, ensuring a team starts from location i_{start}, performs activities at a sequence of locations, and arrives at location i_{end}. Constraints (2.4) require each activity (except i_{start} and i_{end}) is performed exactly once. Constraints (2.5) ensure that if any two activities are performed by a team consecutively, the finish time of the preceding one must be earlier than the start time of the succeeding one. These constraints are necessary even if the time dimension does not need to be considered in the problem (i.e., there is no costs or constraints related to the time dimension), because they also eliminate subtours (i.e., some infeasible solutions). Constraints (2.6) are binary constraints.

Although the value of U does not impact the correctness of the model (as long as it is sufficiently large), a small U is preferred because it could generally improve the solution speed by MIP algorithms. A loose lower bound of U can be calculated as $\max_{k \in K} \sum_{i \in I} \max_{j \in I} t_{ijk}$, because $U \geq u_{i_{\text{end}}} \geq \max_{k \in K} \sum_{i \in I} \max_{j \in I} t_{ijk}$. However, if the scheduling horizon length U is predetermined, a constraint should be added to the model:

$$u_{i_{\text{end}}} = U \ , \tag{2.7}$$

which requires that all teams arrive at i_{end} by time U. This constraint can also tighten the model.

Conventional VRP allows a team to wait after performing an activity, but in track maintenance problems, a team may be required to immediately move to the next activity after performing an activity. In that case, another set of constraints should be added to the model:

$$u_j + U(x_{ijk} - 1) \leq u_i + t_{ijk} \ , \qquad \forall i, j \in I \setminus \{i_{\text{end}}\}, k \in K \ . \tag{2.8}$$

With these constraints, the start time of all activities are determined once the routes of teams are determined, i.e., $\mathbf{u} = \mathbf{u}(\mathbf{x})$.

In conventional VRP, an activity is generally assumed to be located at a single spatial point. But in our context, an activity is usually defined continuously along a track segment; i.e., a team starts from one endpoint of the segment, performs the activity while moving along the track, and finishes the activity at the other endpoint. Because such track segment may be as long as a few hundred miles, it can hardly be assumed as a single point; otherwise the calculations of travel costs and travel time will have large errors. Therefore, when we formulate the optimization models for such track maintenance problems, the direction of working on an activity (i.e., which endpoint of the segment the team starts from) will be addressed.

Various side constraints are usually added to the VRP model to describe additional practical issues such as vehicle capacity constraints, route length constraints, pickup and delivery constraints, and time window constraints (Toth and Vigo, 2001). The time window constraints are the most relevant to railroad track maintenance problems. Typically, the time window constraints require an activity $i \in I'$ is performed within a given time interval $[u_{i,\min}, u_{i,\max}]$ (Solomon, 1988), as follows:

$$u_{i,\min} \le u_i \le u_{i,\max}, \qquad \forall i \in I'. \tag{2.9}$$

Many algorithms have been developed to solve VRP with time windows (VRPTW), including both MIP and heuristic algorithms (Laporte, 1992; Toth and Vigo, 2001; Kallehauge et al., 2005; Kallehauge, 2008). Among them, MIP algorithms include column generation and Dantzig-Wolfe decomposition (Desrochers, et al., 1991), Lagrangian decomposition (Halse, 1992), Lagrangian relaxation (Kohl and Madsen, 1997) and other extensions, variants and hybrids.

De Jong et al. (1996) extended VRPTW to VRPMTW (i.e., VRP with multiple time windows), where an activity is allowed to be performed within multiple time intervals. Ibaraki et al. (2005) studied a more general version, vehicle routing problem with general time windows (VRPGTW), where time window constraints are allowed to be violated at some penalty costs, and the penalty function of the start time of an activity can be of any piecewise linear form. These more general time window constraints are non-convex, and are difficult to solve using MIP algorithms. Therefore, heuristics such as local search algorithms were used to solve them (De Jong et al., 1996; Ibaraki et al., 2005). In track maintenance problems, time window constraints can be piecewise linear (similar to those in VRPGTW), and there are even more difficult side constraints.

Periodic vehicle routing problem (PVRP) (Beltrami and Bodin, 1974; Christofides and Beasley, 1984; Francis et al., 2008) is a variant of VRP. In PVRP, activities are performed periodically at a spatial location, which is similar to the cases in TISP and RTMSP. In PVRP, every location has a fixed frequency at which the activities are performed, and a set of candidate schedules, which is defined as a set of time points when activities are performed. For example, for a location that needs service three times a week, two possible schedules are MWF (performing activities on Monday, Wednesday and Friday) and TRS (Tuesday, Thursday and Saturday) (Beltrami and Bodin, 1974). Francis et al. (2006) extended the PVRP to PVRP with service choice (PVRPSC), where the service frequency of a location is no longer fixed, and candidate schedules with different frequencies for a location are allowed. The model was solved using Lagrangian relaxation combined with branch and bound procedure. Other studies on PVRP include Alonso et al. (2008) and Coene et al. (2010).

Despite some similarities with regard to periodic scheduling and routing, PVRP (including PVRPSC) are fundamentally different from the track maintenance problems (such as TISP). First, the inspection or maintenance periods in track maintenance problems are usually as long as one month or one year, and the periods are required to be flexible. For example, suppose the inspection frequency of a track segment is 60 days and the flexibility of schedule is +/-15 days.

Assume the segment has been inspected on the 0th day, i.e., right before the beginning of the scheduling horizon, then the first inspection activity in the scheduling horizon could be scheduled on any days between the 45th day and the 75th day. Each of these 31 possible dates for the first inspection satisfies the inspection requirement. Once the first inspection date is determined, the date of the second inspection activity on this segment also has 31 choices. This leads to $31\times31\approx900$ candidate schedules for the first two inspections. If the scheduling horizon is one year, there will be about 6 inspection activities on this segment during the year, and the total number of candidate schedules will be about $31^6\approx9\times10^8$. It is obviously impractical to formulate this problem into PVRP and enumerate all candidate schedules as the model input. Furthermore, PVRP assumes that an activity is always performed within one day. In track maintenance problems, however, it is very common that an activity is performed across multiple days: an activity may be interrupted at the end of a day and be resumed at the beginning of the next day, or its duration is longer than one day and must be performed on multiple days. This makes the predetermination of all candidate schedules even more impractical. Therefore, conventional PVRP cannot be applied to periodic track maintenance problems directly.

2.2.2 Time-Space Network Model

In the special case that the scheduling horizon can be discretized into a set of time points W (e.g., t_{ijk} is an integer, or it can be approximated into integers by scaling the time unit), the routing and scheduling problems can also be formulated into a time-space network (TSN) model. Let $\mathbf{x}'=\{x'_{ijkw}\}$ be a set of binary variables so that $x'_{ijkw}=1$ if team $k\in K$ moves directly from location $i\in I$ to location $j\in I$ at time point $w\in W$, and $x'_{ijkw}=0$ otherwise. So $\mathbf{x}'$ determine the routes of all teams. Let $w_{\text{start}} \in W$ be the time when all teams leave location i_{start}. Without losing generality, we can assume $w_{\text{start}} = 0$.

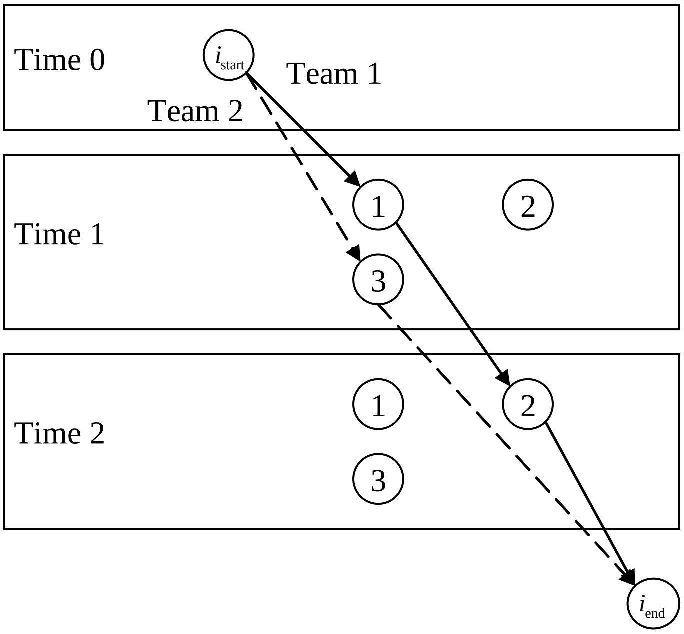


Figure 3. Illustration of time-space network model.

Figure 3 illustrates a feasible schedule for a simple problem instance in the form of a TSN model. Both the problem instance and the schedule are exactly the same as those in Figure 2. The only difference is that the formulation is changed from VRP model to TSN model. It can be seen that the spatial locations (i.e., location 1, 2 and 3) are duplicated at every discrete time point. So a circle now represents both a spatial location and a time point. The teams start from location i_{start} at time $w_{\text{start}} = 0$, and then they respectively go to locations 1 and 3 at time 1. After visiting location 1, team 1 continues its travel and arrives at location 2 at time 2. This figure illustrates how variables u_i are removed from the model by extending variables x_{ijk} into multiple duplicates x'_{ijkw}.

The TSN model can be formulated as:

(TSN) $$\min \sum_{w\in W}\sum_{k\in K}\sum_{i\in I}\sum_{j\in I} c_{ijk} x'_{ijkw} , \tag{2.10}$$

s.t.

$$\sum_{j\in I} x'_{i_{\text{start}} jkw_{\text{start}}} = 1 , \qquad \forall k \in K , \tag{2.11}$$

$$\sum_{j\in I} x'_{jik(w-t_{ijk})} - \sum_{j\in I} x'_{ijkw} = 0 , \qquad \forall i \in I', k \in K, w \in W , \tag{2.12}$$

$$\sum_{w\in W}\sum_{k\in K}\sum_{j\in I\setminus\{i\}} x'_{ijkw} = 1 , \qquad \forall i \in I' , \tag{2.13}$$

$$x'_{ijkw} \in \{0,1\} , \qquad \forall i, j \in I, k \in K, w \in W . \tag{2.14}$$

Objective (2.10) minimizes the total costs. Constraints (2.11) and (2.12) are the flow conservation constraints, ensuring a team starts from location i_{start} at time point w_{start} , performs activities at a sequence of locations, and arrives at location i_{end} . Constraints (2.13) require each activity (except i_{start} and i_{end}) are performed exactly once. Constraints (2.14) are binary constraints. If there are no side constraints besides this core model, there is always an optimal solution with binary flows even if (2.14) is removed. However, if there are additional side constraints, the binary constraints (2.14) may impact the optimal objective value.

If the team must immediately move to the next activity after performing an activity, another set of constraints should be added to the model:

$$x'_{iikw} = 0 , \qquad \forall i \in I, k \in K, w \in W . \tag{2.15}$$

VRP model and TSN model are similar but have a few differences. VRP model uses continuous variables $\mathbf{u}$ to represent the start times of activities, while TSN model uses an additional subscript $w \in W$. VRP model uses U as the scheduling horizon length, while TSN model uses the largest element in W, i.e., $\max_{w \in W} w \approx U$ ("$\approx$" is used because there may be approximation during the discretization of scheduling horizon). TSN model can be viewed as an approximation of VRP model: the discretization of the continuous scheduling horizon may cause loss of accuracy, but a set of discrete points can always be found in a continuous horizon, which means TSN model can always be translated into VRP model exactly, but the reverse is not true.

TSN model has been applied to many real-world problems with difficult side constraints, such as railroad track maintenance scheduling (Li et al., 2009; Gorman and Kanet, 2010; Peng et al., 2010), railroad transportation scheduling (Florian et al., 1976; Kwon et al., 1998; Ahuja et al., 2005), airline scheduling (Jarrah et al., 1993; Hane et al., 1995; Thengvall et al., 2000), bus scheduling (Kliewer et al., 2006; Steinzen et al., 2010) and military convoy scheduling (Chardaire et al., 2005). These problems are usually solved using MIP algorithms. For example, Hane et al. (1995), Ahuja et al. (2005), Kliewer et al. (2006), Li et al. (2009), Gorman and Kanet (2010), Peng et al. (2010) and Steinzen et al. (2010) all used some MIP software in their algorithms. Some customized MIP methods may also be applied, such as interior-point algorithm, dual steepest edge simplex and cost perturbation (Hane et al., 1995), dynamically adding strong inequalities (Li et al., 2009), and Lagrangian relaxation and column generation (Steinzen et al., 2010).

Generally, VRP models are more focused on side constraints in the spatial dimension (i.e., more routing oriented), and TSN models are more focused on those in the temporal dimension (i.e., more scheduling oriented). The performance of the VRP model or the TSN model largely depends on the specific problem structure. Clearly, TSN model cannot be used if the time horizon cannot be discretized. For example, in the case that activities can be performed at any time within the scheduling horizon, the horizon is usually discretized into equal intervals. The unit of such interval should be small enough to represent the smallest t_{ijk} in order to avoid losing accuracy. Then the number of discrete time points, $|W|$, approximately equals to the scheduling horizon length U divided by the length of discretized interval. If some t_{ijk} is much smaller than U, the interval length will be much smaller than U, and $|W|$ will be very large. Because the number of variables x_{ijkw} is proportional to $|W|$, a large $|W|$ will require many variables, making the model very difficult to solve. In that case, the TSN model is not appropriate.

Some past research has focused on comparing the performances of the TSN model and the VRP model. Steinzen et al. (2010) found that the TSN model outperformed the VRP model for their bus scheduling problem. The reason is that the bus departure/arrival time is fixed in their problem, so it is easy to discretize the time horizon without introducing a large number of variables. Reversely, the multi-resource routing problem (MRRP) with flexible tasks is usually formulated into VRP model but not TSN model, because the latter involves more intensive computation (Francis et al., 2007). Peng et al. (2010) formulated a PTSP using both VRP model and TSN model, and tested them on some moderate-scale problem instances using MIP solver CPLEX. They found that CPLEX failed to solve VRP model within a reasonable time, but it was able to solve the corresponding TSN model.

In general, scheduling and routing problems with conventional VRP-type side constraints, such as vehicle capacity and route length constraints, are usually formulated into VRP model and solved with VRP algorithms. For problems with other types of side constraints (e.g., the railroad scheduling and airline scheduling problems reviewed in this subsection), the TSN model is usually more suitable when the temporal discretization is possible, and the model can often be solved using MIP algorithms. Intuitively, the TSN model has an underlying flow network structure and its constraint coefficient matrix is closer to being totally unimodular --- this feature favors MIP algorithms. Reversely, the VRP model typically includes constraints (2.5), which correlate binary variables $\mathbf{x}$ to real variables $\mathbf{u}$ using a large number U . Such constraints are generally difficult to solve using general MIP algorithms.

2.2.3 Heuristic Algorithms

Practical routing and scheduling problems usually have a large number of variables and complex side constraints. MIP algorithms, such as linear relaxation, Lagrangian relaxation (Fisher, 1981), column generation (Ford and Fulkerson, 1958; Dantzig and Wolfe, 1960) and Benders' decomposition (Benders, 1962), are usually not able to solve such large-scale problem instances efficiently. For example, Nemani et al. (2010) found that column generation does not perform well on a PTSP. Some other algorithms, such as direct tree search methods and dynamic programming (Laporte, 1992), are also developed for routing and scheduling problems. But they are not widely-used, because their solution time for large scale problems is usually prohibitive. Gorman and Kanet (2010) found that constraint programming is very slow for a PTSP. Therefore, most research has focused on developing various heuristic algorithms for large-scale routing and scheduling problems. For heuristics algorithms, it is not necessary to formulate the problem into a mathematical model, although a mathematical formulation may be helpful to describe the problem.

Heuristic methods can be divided into two categories: constructive heuristics and improvement heuristics. Constructive heuristics build a solution from scratch, while improvement heuristics improve an existing solution. In some implementations, constructive heuristics are used as the first step to obtain an initial solution, and then improvement heuristics are applied to make improvements. In some so-called “incremental algorithms”, constructive and improvement heuristics are applied iteratively. Heuristics may also be combined with other solution approaches. For example, constructive heuristic can provide an initial solution for MIP algorithms (such as Lagrangian relaxation), or a relaxed MIP model can be solved to provide an initial solution for improvement heuristics.

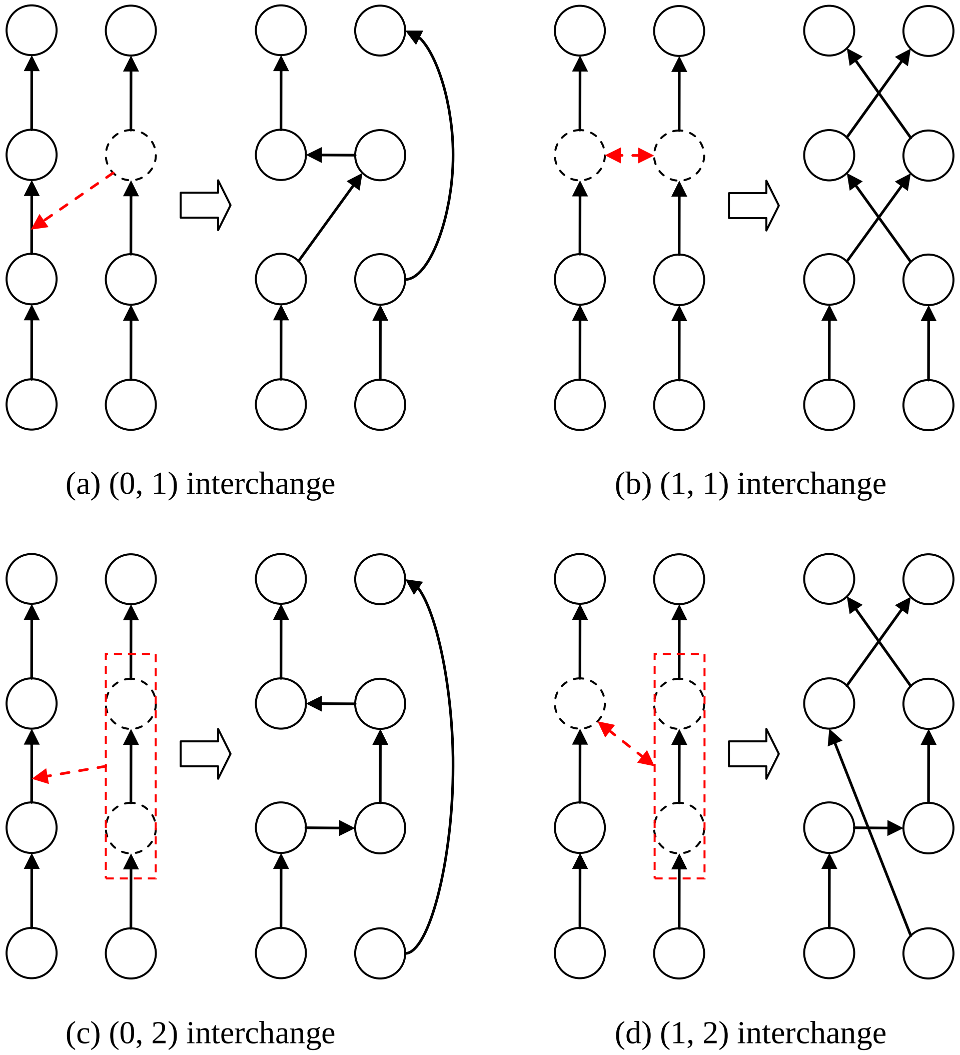
Constructive heuristics proposed for VRPTW includes “I1” by Solomon (1987), where every route is initialized with a “seed” activity and the remaining unscheduled activities are added to the route until its total duration reaches the scheduling horizon. A parallel version of I1 by Potvin and Rousseau (1993) initializes all routes at once and then adds the remaining unscheduled activities one by one. Another I1-based algorithm by Ioannou et al. (2001) inserts activities in a way that the impact on all customers is minimized. A comprehensive review can be found in Braysy and Gendreau (2005). For large-scale routing and scheduling problems with many side constraints, constructive heuristics may be more customized and complex. Examples include the incremental algorithm with backtracking for a PTSP (Bog et al., 2010).

Improvement heuristics can be further divided into two categories: single-solution based heuristics and population-based heuristics (Talbi, 2009). Single-solution based heuristics only keep one “current solution” in memory. Examples include local search, simulated annealing, and tabu search. Population-based heuristics keep a population of candidate solutions. Examples include genetic algorithms and ant colony algorithms. A categorized bibliography of improvement heuristics applied in VRP-type problems can be found in Gendreau (2008). Most studies on railroad routing and scheduling problems are based on single-solution based improvement heuristics. Gorman and Kanet (2010) tried a genetic algorithm on a PTSP, but its performance was not as good as MIP algorithm (with the TSN model). In this study we will focus on the development and implementation of single-solution based improvement heuristics.

The most commonly-used single-solution based improvement heuristics are local search (or neighborhood search) algorithms, which are often found to be the most efficient and effective way to solve large-scale routing and scheduling problems (Funke et al., 2005). A local search algorithm defines a neighborhood structure for the problem, so that every solution to the problem has a set of neighbors in the solution space. Local search starts from an initial solution as the current solution and searches among its neighbors. If a neighbor satisfies certain criteria (e.g., it has lower costs than the current solution), the algorithm sets that neighbor as the current solution and starts searching the neighborhood of this new solution. Such an update to the current solution is called a "move". Local search is also the most fundamental type of single-solution based improvement heuristic. Many more complex heuristics, such as simulated annealing and tabu search (Osman, 1993), are based on local search.

For routing and scheduling problems, there are two categories of most commonly-used neighborhood structures: node interchange and edge interchange. Use I_k to denote the locations visited by team $k \in K$. Use $i_{k1}, i_{k2}, \dots, i_{k|I_k|}$ to denote the route of team $k \in K$, i.e., the team visits locations $i_{k1}, i_{k2}, \dots, i_{k|I_k|} \in I_k$ in the order of $i_{k1}, i_{k2}, \dots, i_{k|I_k|}$. This route can be viewed as a path composed of nodes and directed edges, respectively representing locations, i.e., i_{ko}, $o = 1, \dots, |I_k|$, and travels between locations, i.e., $(i_{ko}, i_{k(o+1)})$, $o = 1, \dots, |I_k| - 1$. Define a "block" as a sequence of consecutive nodes on the route, i.e., $i_{ko}, i_{k(o+1)}, \ldots, i_{k(a+o-1)}$, where a is the number of nodes in the block. Define the “position” of block $i_{ko}, i_{k(o+1)}, \ldots, i_{k(a+o-1)}$ as $< k, o >$, i.e., both its team and its position in the route. A block can contain 0 nodes. In that case it only represents a position $< k, o >$.

In a node interchange, n blocks (containing $a_1, \ldots, a_n$ nodes, respectively) interchange their positions. Let $< k_m, o_m >$ be the position of block $m = 1, \ldots, n$. One possible interchange scheme is to let block m take block $(m+1)$'s position $< k_{m+1}, o_{m+1} >$ for $m = 1, \ldots, n-1$, and block n take block 1’s position $< k_1, o_1 >$. Ignoring the detailed interchange scheme, a node interchange can be represented by n integer numbers $(a_1, \ldots, a_n)$. Some special cases of node interchange are widely used. For example, the (0, 1) interchange is also called “insertion” or “relocation” (Savelsbergh, 1992), as illustrated in Figure 4(a). It moves a single node from one route to another or some other position of the same route. The (1, 1) interchange is also called “exchange” or “swap” (Savelsbergh, 1992), as illustrated in Figure 4(b). It swaps the positions of two nodes. Figures 4(c) and (d) illustrate node interchanges with blocks containing more than one node. For an (a_1, a_2) interchange, if both a_1 and a_2 are no larger than a constant λ, it is called an λ-interchange (Osman, 1993). For example, Figures 4(a)-(d) are all 2-interchanges, but only (a) and (b) are 1-interchanges. If both a_1 and a_2 are larger than 0, it is called CROSS-exchange (Taillard et al., 1997). Figures 4(b) and (d) are CROSS-exchanges but (a) and (c) are not. An $(1, \ldots, 1)$ interchange with arbitrary n is called an ejection chain (Glover, 1992); an example with $n = 2$ is given in Figure 4(b).



(a) (0, 1) interchange

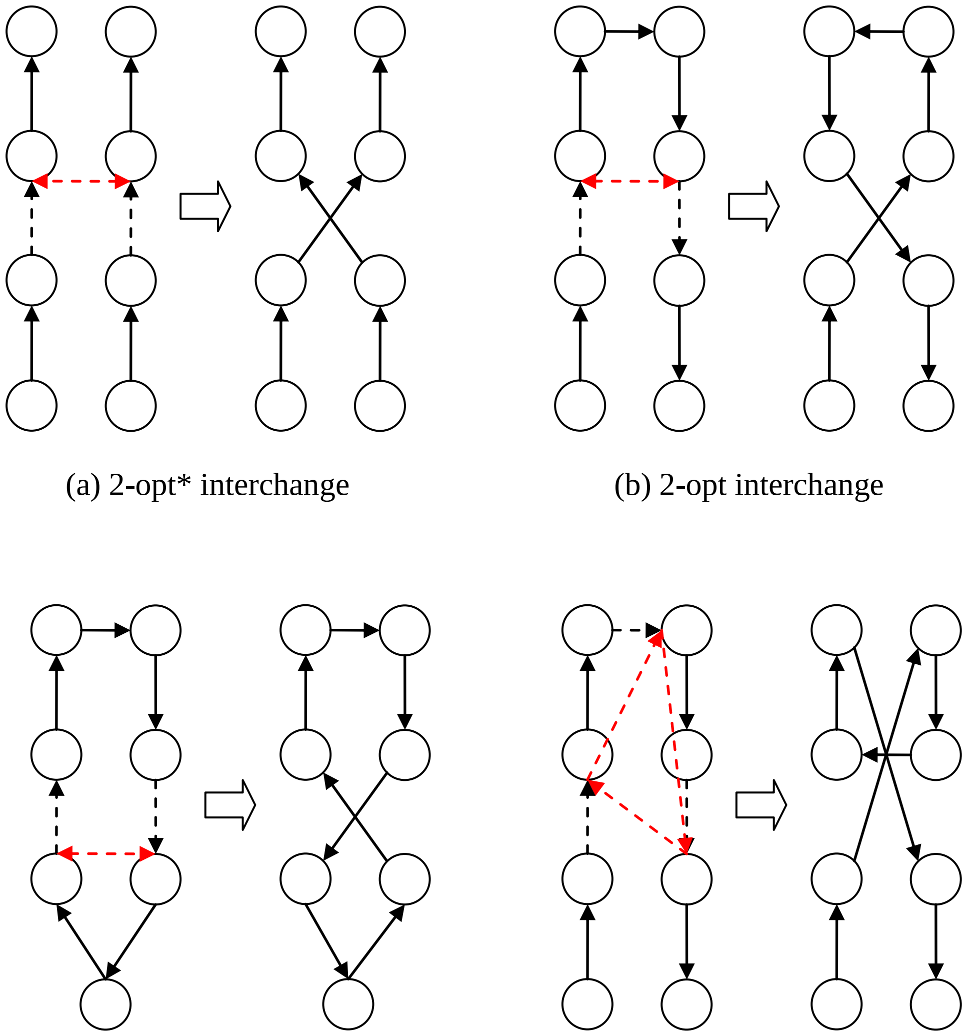
(b) (1, 1) interchange

(c) (0, 2) interchange

(d) (1, 2) interchange

Figure 4. Examples of node interchange.

In an edge interchange, n edges in the current solution are replaced by another set of n edges. If these edges are from different routes, such move is called n-opt* (Potvin and Rousseau, 1995). Figure 5(a) illustrates a 2-opt* exchange. If these edges are from the same route, such move is called n-opt (Russell, 1977). Figure 5(b) illustrates a 2-opt interchange, and (c) illustrates another possible 2-opt interchange when the start node and the end node of the route are the same (i.e., the start and end locations of teams are the same). In n-opt, the direction of part of the route may be reversed, which will change the start time of activities dramatically and is likely to cause violations to side constraints related to the time dimension. In 2-opt, such reversion cannot be avoided. In 3-opt, there is only one possible move which does not cause reversion on edges which are not exchanged, as illustrated in Figure 5(d). This 3-opt is also called the Or-opt (Or, 1976).



(a) 2-opt* interchange

(b) 2-opt interchange

(c) Another 2-opt interchange

(d) Or-opt interchange

Figure 5. Examples of edge interchange.

Besides the above-mentioned neighborhood structures, many other moves are also developed, such as GENI-exchange (Gendreau et al., 1992) and cyclic transfers (Thompson and Psaraftis, 1993). A comprehensive review can be found in Braysy and Gendreau (2005) and Funke et al. (2005).

The implementation of a heuristic usually requires customized algorithm design based on the characteristics of the specific problem, and its computational performance highly depends on the problem structure and even the input data. Therefore, although the algorithms developed by previous studies can serve as useful references, it is very important to adapt and further improve them for the railroad track maintenance problems.

Glossary of Symbols

c_{ijk}: Costs for team $k \in K$ to perform activity $i \in I$ and to move from location $i \in I$ to location $j \in I$,

I: Set of locations; $I = I' \cup \{i_{\text{start}}\} \cup \{i_{\text{end}}\}$,

I': Set of activities (or locations with an activity),

i_{end}: End location of all teams,

I_k: Set of locations visited by team $k \in K$,

i_{ko}: The o th location visited by team $k \in K$,

i_{start}: Start location of all teams,

K: Set of teams,

t_{ijk}: Time for team $k \in K$ to perform activity $i \in I$ and to move from location $i \in I$ to location $j \in I$,

$\mathbf{u} = \{u_i\}$: Time when some team arrives at location $i \in I$,

$u_{i,\max}$: The latest time when activity $i \in I$ can be performed,

$u_{i,\min}$: The earliest time when activity $i \in I$ can be performed,

U: Scheduling horizon length,

W: Set of discrete time points,

w_{start}: Time when all teams leave location i_{start},

$\mathbf{x} = \{x_{ijk}\}$: Binary variables indicating if team $k \in K$ moves directly from location $i \in I$ to location $j \in I$,

$\mathbf{x}' = \{x'_{ijkw}\}$: Binary variables indicating if team $k \in K$ moves directly from location $i \in I$ to location $j \in I$ at time point $w \in W$.

CHAPTER 3 TRACK INSPECTION SCHEDULING

3.1 Introduction

Railroads use a fleet of track inspection vehicles (and associated crews) to examine the status of tracks all over the railroad network, so that the track defects can be identified and repaired in time and potential train accidents can be avoided. This study considers two categories of track inspection activities: rail inspection and geometry inspection. In rail inspection, rail tracks are inspected for defects from bending and shear stresses, wheel/rail contact stresses, and thermal stresses, etc. Geometry inspection inspects geometric parameters such as alignment, curvature, crosslevel, gauge, and rail profile.

Rail inspection and geometry inspection are performed by rail inspection teams and geometry inspection teams respectively. Railroad network is divided into hundreds of segments in order to schedule the inspection activities (i.e., “tasks”). Each segment should be inspected periodically at a certain frequency (which generally varies from once a month to once a year) to ensure the safety of train operations. A track inspection schedule includes the assignment of the tasks to teams as well as the start times of the tasks such that the required inspection frequencies are satisfied for the segments. It should also satisfy a variety of other business constraints such as geographic preference, non-simultaneity and time window constraints. The scheduling horizon is normally in the short term (e.g., a few weeks long), while the schedule is updated more frequently (sometimes on a daily basis) to address unexpected events (e.g., the delay of a task or the breakdown of a vehicle). In additional to short-term scheduling, long-term planning is also needed in order to determine the optimal number of inspection teams and to balance the workload through the year.

Currently, track inspection scheduling is mostly manually performed in the railroad companies. Since the rail inspection scheduling problem (RISP) and the geometry inspection scheduling problem (GISP) share some similar features, this chapter proposes a mathematical model for the general track inspection scheduling problems (TISP) and develops a customized algorithm to efficiently solve the model. This general TISP model can be used to solve both RISP and GISP.

3.2 Model and Algorithm Selection

TISP is a routing and scheduling problem. There are generally three approaches to solve such problems: (i) vehicle routing problem (VRP) model with mixed integer programming (MIP) algorithms, (ii) time-space network (TSN) model with MIP algorithms, and (iii) heuristic algorithms. As discussed in Chapter 2, VRP model with MIP algorithm has been applied to many routing and scheduling problems with certain side constraints, such as VRP with time windows (VRPTW) and periodic vehicle routing problem (PVRP). However, the computational performance of these algorithms is not generally good for large scale problems with difficult side constraints. A typical TISP instance contains thousands of tasks and many difficult side constraints such as periodicity constraints and non-simultaneity constraints. The problem scale and complexity far exceed the capability of existing MIP algorithms for VRP models. Therefore, VRP model with MIP algorithms is not suitable for TISP.

TSN model with MIP algorithms is able to handle larger and more complex problems, but it requires the time horizon to be discretized into a set of time periods. This treatment is unfortunately impractical for TISP. In TISP, the duration of a task varies from less than half an hour to a few weeks. Therefore, the unit of time intervals should be no longer than one hour in order to ensure accuracy, as discussed in Chapter 2. Suppose that one-hour intervals are used. Then a typical scheduling horizon of 8 weeks contains about 320 time periods (each week has 40 work hours), which means that the decision variables should be duplicated into 320 copies. Because the number of decisions variables has already been large without duplication (e.g., thousands of tasks and tens of teams), the TSN formulation will be very large-scale and hence unlikely to be solvable by MIP algorithm within a reasonable amount of computation time (e.g., less than one hour, considering the fact that the schedule may need update every day). Furthermore, for long-term planning (e.g., with a time horizon of one year), the number of variables will be extremely large and TSN model will be even less practical.

Heuristic algorithms are well-known to be suitable for complex and large-scale optimization problems, and they often yield good near-optimum solution very quickly. Therefore, heuristics are chosen to solve TISP. Although an explicit mathematical formulation is not necessary when heuristic algorithms are used, we still present it to help describe the problem. The formulation will be based on the VRP model in Chapter 2, because a continuous time dimension is more precise for TISP.

3.3 Model Formulation

In this section, we will first present a VRP-based core model for TISP, and then different types of side constraints and costs. As discussed later in this section, most side constraints are very difficult to handle with MIP algorithms.

3.3.1 Core Model

Let S be the set of track segments to be inspected periodically. Because a segment may be inspected multiple times in the scheduling horizon, we define a task as a single inspection activity on a segment. So each segment $s \in S$ has a set of tasks, denoted as $I_{\text{Seg},s}$. Let $I = \bigcup_{s \in S} I_{\text{Seg},s}$ be the set of tasks on all segments. Let $t_{\text{task},i}$ be the duration of task $i \in I$, i.e., the time needed to inspect the segment.

A track segment has two endpoints, one called the low milepost and the other called the high milepost. The inspection team starts to perform a task from one endpoint, and completes the task at the other endpoint. Because a segment may be longer than 100 miles, the two endpoints generally cannot be represented by the same spatial node, and we need to decide from which endpoint the team starts a task (i.e., the direction for the team to perform a task). In the model, the two possible directions of a task are represented by two directed arcs in opposite directions between the low milepost and the high milepost. Let E be the set of all directed arcs, and E_i be the set of arcs representing the direction of task $i \in I$ (a task only has two possible directions so $|E_i| = 2$). Let $t_{\text{travel},e_1 e_2}$ be the travel time from the head of arc $e_1 \in E$ (i.e., the finish point of the corresponding task) to the tail of arc $e_2 \in E$ (i.e., the start point of the corresponding task).

Let K be the set of inspection teams. For each team $k \in K$, an extra start arc $e_{\text{start},k}$ is added to E to represent the initial location of a team at the beginning of the scheduling horizon. Both of its endpoints are at the initial location of the team. In this way, the travel time from the initial location of team k to any other arcs can be defined.

Let $\mathbf{x} = \{x_{e_1 e_2 k} : e_1, e_2 \in E, k \in K\}$ be the set of binary decision variables for the movements of teams; i.e., $x_{e_1 e_2 k} = 1$ if team $k \in K$ travels directly from the head of arc $e_1 \in E$ to the tail of arc

$e_2 \in E$; and 0 otherwise. Let $\mathbf{u} = \{u_i : i \in I\}$ be the set of real-valued start times of tasks; i.e., u_i represents the start time of task $i \in I$. Because the railroad requires a team immediately move to the next task once it finishes the current task, $\mathbf{u}$ is determined once $\mathbf{x}$ is determined, i.e., $\mathbf{u}$ can be written as $\mathbf{u}(\mathbf{x})$. Without losing generality, let the scheduling horizon start at time 0. Let U be the horizon length as well as the end time of the scheduling horizon.

Let $C_{\text{cost item}}(\mathbf{x})$ be the cost function for a certain cost item (e.g., travel costs or side constraint penalty costs). Because $\mathbf{u}$ can be determined by $\mathbf{x}$, $C_{\text{cost item}}(\mathbf{x})$ can be written as a function of $\mathbf{x}$ only.

The core model (without side constraints) can be formulated as follows:

$$\text{(TISP)} \qquad \min \sum_{\text{all cost items}} C_{\text{cost item}}(\mathbf{x}), \tag{3.1}$$

s.t.

$$\sum_{e \in E} x_{e_{\text{start},k} e k} = 1, \qquad \forall k \in K, \tag{3.2}$$

$$\sum_{e_2 \in E} x_{e_1 e_2 k} - \sum_{e_2 \in E} x_{e_2 e_1 k} \le 0, \qquad \forall e_1 \in E \setminus \{e_{\text{start},k}\}, k \in K, \tag{3.3}$$

$$u_i + t_{\text{task},i} + t_{\text{travel},e_1 e_2} + U(x_{e_1 e_2 k} - 1) \le u_j, \qquad \forall e_1 \in E_i, e_2 \in E_j, i, j \in I, k \in K, \tag{3.4}$$

$$u_j + U(x_{e_1 e_2 k} - 1) \le u_i + t_{\text{task},i} + t_{\text{travel},e_1 e_2}, \qquad \forall e_1 \in E_i, e_2 \in E_j, i, j \in I, k \in K, \tag{3.5}$$

$$x_{e_1 e_2 k} \in \{0,1\}, \qquad \forall e_1, e_2 \in E, k \in K. \tag{3.6}$$

Objective (3.1) minimizes the summation of all costs. Constraints (3.2) and (3.3) enforce flow conservation. Constraints (3.4) establish the relationships between task start times and team routes, as well as eliminate subtours. Constraints (3.5) require a team to immediately move to the next task once the current task is finished. Constraints (3.6) define binary variables.

There are a few differences between TISP (3.1)-(3.6) and VRP (2.1)-(2.6). In TISP each task has two directions, while in the VRP model an activity is at a single spatial point. VRP constraints (2.4) require that every activity be performed exactly once, while TISP does not enforce that. Instead, TISP has periodicity constraints which will add a penalty cost to the objective value if a task is not performed within the scheduling horizon, as introduced in the next subsection.

The following subsections will introduce side constraints in more detail. These side constraints could be implemented either as hard constraints (i.e., they must not be violated) or as soft ones (i.e., they can be violated at a penalty cost).

3.3.2 Periodicity Constraints

Periodicity constraints require that all segments are inspected periodically at certain frequencies. We use $u_{\text{finish},i}$ to denote the finish time of task $i \in I$. Let $i_{\text{Seg},s0}$ represent the last task on segment $s \in S$ before the start of the scheduling horizon, and $I_{\text{Seg},s} = \left\{ i_{\text{Seg},s1}, \ldots, i_{\text{Seg},s\left|I_{\text{Seg},s}\right|} \right\}$ be the tasks to be scheduled within the scheduling horizon. For $o = 1, \ldots, \left|I_{\text{Seg},s}\right|$, always let task $i_{\text{Seg},s(o-1)}$ be performed before $i_{\text{Seg},so}$, and let $\Delta u_{i_{\text{Seg},so}}$ be the difference between the finish times of tasks $i_{\text{Seg},so}$ and $i_{\text{Seg},s(o-1)}$, i.e., let $\Delta u_{i_{\text{Seg},so}} = u_{\text{finish},i_{\text{Seg},so}} - u_{\text{finish},i_{\text{Seg},s(o-1)}}$. The railroad has a preferred inspection interval $t_{\text{Int},i}$ for every task $i \in I$, which means task i shall preferably be performed within time $t_{\text{Int},i}$ from the last inspection on the segment of i . In other words, for $s \in S$ and $o = 1, \ldots, \left|I_{\text{Seg},s}\right|$,

$\Delta u_{i_{\mathrm{Seg},so}}$ should not be significantly larger than $t_{\mathrm{Int},i_{\mathrm{Seg},so}}$. It shall be noted that the preferred interval of a task is usually defined based on the segment of this task. For example, if segment $s \in S$ has a preferred inspection interval $t_{\mathrm{SegInt},s}$, then $t_{\mathrm{Int},i} = t_{\mathrm{SegInt},s}$ for every task $i \in I_{\mathrm{Seg},s}$.

For $s \in S$ and $o = 1,\ldots,\left|I_{\mathrm{Seg},s}\right|$, because the start time of task $i_{\mathrm{Seg},so}$, $u_{i_{\mathrm{Seg},so}}$, is a decision variable in $\mathbf{u}$, the finish time of task $i_{\mathrm{Seg},so}$, $u_{\mathrm{finish},i_{\mathrm{Seg},so}} = u_{i_{\mathrm{Seg},so}} + t_{\mathrm{task},i_{\mathrm{Seg},so}}$, shall also be determined. For $o = 0$, however, $u_{\mathrm{finish},i_{\mathrm{Seg},s0}}$, the finish time of task $i_{\mathrm{Seg},s0}$, should be given as an input.

The periodicity constraints and corresponding penalty costs are formulated as follows:

$$\mathrm{C}_{\mathrm{Per}}(\mathbf{x}) = \sum_{i \in I} c_{\mathrm{Per},i}(\Delta u_i), \tag{3.7}$$

$$\Delta u_{i_{\mathrm{Seg},s1}} = u_{i_{\mathrm{Seg},s1}} + t_{\mathrm{task},i_{\mathrm{Seg},s1}} - u_{\mathrm{finish},i_{\mathrm{Seg},s0}}, \qquad \forall s \in S, \tag{3.8}$$

$$\Delta u_{i_{\mathrm{Seg},so}} = u_{i_{\mathrm{Seg},so}} + t_{\mathrm{task},i_{\mathrm{Seg},so}} - u_{i_{\mathrm{Seg},s(o-1)}} - t_{\mathrm{task},i_{\mathrm{Seg},s(o-1)}}, \quad \forall o \in \left\{2,\ldots,\left|I_{\mathrm{Seg},s}\right|\right\}, s \in S, \tag{3.9}$$

$$u_i \geq U\left(1 - \sum_{e_1 \in E_i} \sum_{e_2 \in E} \sum_{k \in K} x_{e_1 e_2 k}\right), \qquad \forall i \in I, \tag{3.10}$$

where $\mathrm{C}_{\mathrm{Per}}(\cdot)$ is the total costs incurred by periodicity constraints, and $c_{\mathrm{Per},i}(\cdot)$ is the penalty cost function of task $i \in I$ which is related to $t_{\mathrm{Int},i}$. Generally, $c_{\mathrm{Per},i}(\Delta u_i)$ should be increasing when $\Delta u_i > t_{\mathrm{Int},i}$, i.e. when the task is not performed within its preferred interval. Equation (3.7) defines $\mathrm{C}_{\mathrm{Per}}(\cdot)$ as the summation of $c_{\mathrm{Per},i}(\cdot)$ of all tasks. Equations (3.8) and (3.9) define $\Delta u_{i_{\mathrm{Seg},so}}$ as the interval between the finish times of $i_{\mathrm{Seg},s(o-1)}$ and $i_{\mathrm{Seg},so}$. Constraints (3.10) ensure the start time of a task is at least U (i.e., the end of scheduling horizon) if the task is not assigned to any team.

If a periodicity constraint is hard, i.e., Δu_i should never exceed $t_{\mathrm{Int},i}$ for task $i \in I$, then the following constraints shall be added to the model:

$$\Delta u_i \le t_{\mathrm{Int},i}, \qquad \forall i \in I \text{ with hard constraints.} \tag{3.11}$$

There is an expression $\sum_{e_1 \in E_i} \sum_{e_2 \in E} \sum_{k \in K} x_{e_1 e_2 k}$ in (3.10). Let $y_i = \sum_{e_1 \in E_i} \sum_{e_2 \in E} \sum_{k \in K} x_{e_1 e_2 k}$. Then $y_i = 1$ if task $i \in I$ is assigned to some team and performed within the scheduling horizon, or 0 otherwise. In most VRP models, there are constraints associated with y_i which require every activity to be performed exactly once. If such constraints are soft, a cost item $c_{\mathrm{task},i} y_i$ will be added to the objective function; here, $c_{\mathrm{task},i}$ is the penalty cost if task $i \in I$ is not assigned to any team (i.e., not performed within the scheduling horizon). If such constraints are hard, they can be written as:

$$y_i = 1, \qquad \forall i \in I,$$

which are equivalent to (2.4) in VRP.

In TISP, however, constraints (3.7)-(3.10) are used to ensure that the model tends to assign tasks to teams. A task not assigned to any team has a start time of at least U, and a penalty cost $c_{\mathrm{PER},i}(\cdot)$ will be incurred if this task is due before U.

Figure 6 illustrates the form of $c_{\mathrm{Per},i}(\Delta u_i)$ in RISP used by a railroad company. In this particular RISP, $t_{\mathrm{Int},i_{so}} = t_{\mathrm{SegInt},s}$, and $t_{\mathrm{SegInt},s}$ varies from one month to one year, depending on various characteristics of the segment $s \in S$. For example, segments with heavy traffic should be inspected more frequently because the rail tracks deteriorate faster and the impact of potential defects is also larger (i.e., higher accident risks, train delay costs and rolling stock maintenance costs). Other factors include the volumes of hazard material traffic and passenger traffic. Besides the preferred interval $t_{\text{SegInt},s}$, the company also has an "allowed interval" $t_{\text{SegInt},s}^{\text{A}}$ and a "required interval" $t_{\text{SegInt},s}^{\text{R}}$ for every segment $s \in S$. As the names suggest, (i) an actual inspection interval $\Delta u_{i_{\text{Seg},so}} \in \left(t_{\text{SegInt},s}, t_{\text{SegInt},s}^{\text{A}} \right]$ is allowed but not preferred, (ii) the case $\Delta u_{i_{\text{Seg},so}} \in \left(t_{\text{SegInt},s}^{\text{A}}, t_{\text{SegInt},s}^{\text{R}} \right]$ should be avoided, and (iii) the case $\Delta u_{i_{\text{Seg},so}} > t_{\text{SegInt},s}^{\text{R}}$ may require overtime work if it happens, and incurs very high penalty costs. The values of $t_{\text{SegInt},s}^{\text{A}}$ and $t_{\text{SegInt},s}^{\text{R}}$ also vary across segments. Usually they equal $t_{\text{SegInt},s}$ multiplied by some constants between 1.1 and 1.3. For very important segments, it is possible to have $t_{\text{SegInt},s} = t_{\text{SegInt},s}^{\text{A}} = t_{\text{SegInt},s}^{\text{R}}$, meaning that $\Delta u_{i_{so}}$ is prohibited from exceeding the preferred interval. Based on the description from the company, a convex penalty cost function, $c_{\text{Per},i}(\Delta u_i)$, is developed as shown in Figure 6. $c_{\text{Per},i}(\Delta u_i)$ is the same for all tasks $i \in I_{\text{Seg},s}$ in segment $s \in S$. The penalty cost is relatively low when Δu_i is within $t_{\text{SegInt},s}$, but becomes extremely high once Δu_i exceeds $t_{\text{SegInt},s}^{\text{R}}$. In this way, the model is able to prioritize the tasks correctly according to the company's requirement.

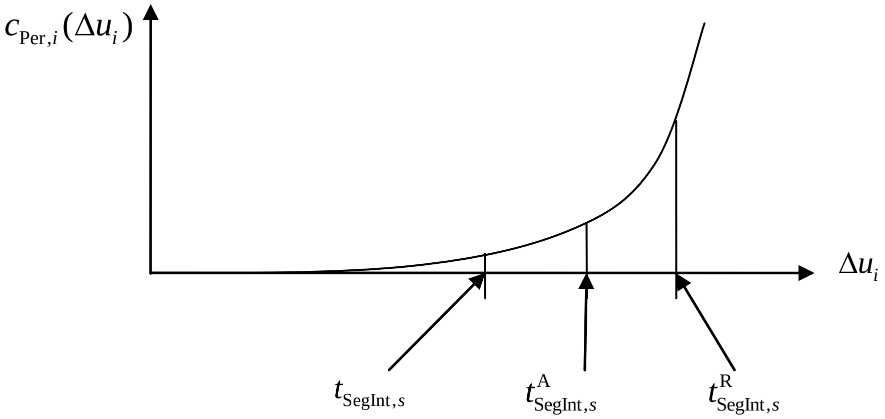


Figure 6. Periodicity constraint penalty cost in rail inspection scheduling.

Because $c_{\text{Per},i}(\Delta u_i)$ is a non-linear function, it cannot be directly solved with MIP algorithms. Although a piecewise linear function can be used to approximate $c_{\text{Per},i}(\Delta u_i)$, such approximation will introduce additional binary variables and side constraints, and will make the model very difficult to solve. This is one of the reasons that MIP algorithms are not used for TISP.

3.3.3 Non-Simultaneity Constraints

Non-simultaneity constraints require certain pairs of tasks not to be performed simultaneously. For example, two tasks in the same subdivision should not be performed simultaneously; otherwise they may disturb each other and obstruct train traffic. Such constraints are called subdivision non-simultaneity constraints. Inspection tasks must be helped by a certain railroad employee such as division engineer or roadmaster. This railroad employee is usually in charge of a set of segments. If the railroad employee is helping with some task, he or she cannot help with other tasks in his or her territory. Hence, two tasks involving the same railroad employee should not be performed simultaneously. Such constraints are called roadmaster non-simultaneity constraints. In the case that more than two tasks shall not be performed simultaneously, a non-simultaneity constraint is defined for each pair of them.

Let $<i,j>_{\mathrm{NS}}$ be a pair of tasks which should not be performed simultaneously, where $i,j\in I$. Also we use $<i,j>_{\mathrm{NS}}$ to represent a single non-simultaneity constraint. The non-simultaneity constraints and corresponding penalty costs are formulated as follows:

$$\mathrm{C}_{\mathrm{NS}}(\mathbf{x})=\sum_{\forall \text{soft} <i,j>_{\mathrm{NS}}} c_{\mathrm{NS},ij}t_{\mathrm{NS},ij}\ , \tag{3.12}$$

$$\begin{aligned} t_{\mathrm{NS},ij} &=\left(\min\left\{u_i+t_{\mathrm{task},i},u_j+t_{\mathrm{task},j}\right\}-\max\left\{u_i,u_j\right\}\right)^+ \\ &=\min\left\{t_{\mathrm{task},i},t_{\mathrm{task},j},\left(t_{\mathrm{task},i}+u_i-u_j\right)^+,\left(t_{\mathrm{task},j}+u_j-u_i\right)^+\right\}, \quad \text{for all soft } <i,j>_{\mathrm{NS}}\ , \end{aligned} \tag{3.13}$$

$$\min\left\{t_{\mathrm{task},i}+u_i-u_j,t_{\mathrm{task},j}+u_j-u_i\right\}\leq 0, \quad \text{for all hard } <i,j>_{\mathrm{NS}}\ , \tag{3.14}$$

where $\mathrm{C}_{\mathrm{NS}}(\cdot)$ is the total costs incurred by all soft non-simultaneity constraints, $c_{\mathrm{NS},ij}$ is the penalty cost per unit time for soft constraint $<i,j>_{\mathrm{NS}}$, and $t_{\mathrm{NS},ij}$ is the overlapping time of task i and j. Expression $(\cdot)^+$ represents $\max\left\{0,\cdot\right\}$. Clearly, tasks performed by the same team can never violate any non-simultaneity constraints, and hence a non-simultaneity constraint always involve two teams.

$\mathrm{C}_{\mathrm{NS}}(\cdot)$ is part of the objective function (3.1). Soft non-simultaneity constraints (3.13) involves another level of minimization when calculating $t_{\mathrm{NS},ij}$. Additional binary variables will be needed if such two-level minimization is formulated into an MIP form. In that case, constraints (3.13) can be replaced by:

$$\upsilon_{\mathrm{NS},ij1} + \upsilon_{\mathrm{NS},ij2} + \upsilon_{\mathrm{NS},ij3} = 2\,, \qquad \text{for all soft } <i, j>_{\mathrm{NS}}, \tag{3.15}$$

$$t_{\mathrm{NS},ij} \geq \min\left\{t_{\mathrm{task},i}, t_{\mathrm{task},j}\right\}(1-\upsilon_{\mathrm{NS},ij1})\,, \qquad \text{for all soft } <i, j>_{\mathrm{NS}}, \tag{3.16}$$

$$t_{\mathrm{NS},ij} \geq t_{\mathrm{task},i} + u_i - u_j - (U + t_{\mathrm{task},i})\upsilon_{\mathrm{NS},ij2}\,, \qquad \text{for all soft } <i, j>_{\mathrm{NS}}, \tag{3.17}$$

$$t_{\mathrm{NS},ij} \geq t_{\mathrm{task},j} + u_j - u_i - (U + t_{\mathrm{task},j})\upsilon_{\mathrm{NS},ij3}\,, \qquad \text{for all soft } <i, j>_{\mathrm{NS}}, \tag{3.18}$$

$$t_{\mathrm{NS},ij} \geq 0\,, \qquad \text{for all soft } <i, j>_{\mathrm{NS}}, \tag{3.19}$$

$$\upsilon_{\mathrm{NS},ij1}, \upsilon_{\mathrm{NS},ij2}, \upsilon_{\mathrm{NS},ij3} \in \{0,1\}\,, \qquad \text{for all soft } <i, j>_{\mathrm{NS}}, \tag{3.20}$$

where $\upsilon_{\mathrm{NS},ij1}$, $\upsilon_{\mathrm{NS},ij2}$ and $\upsilon_{\mathrm{NS},ij3}$ are auxiliary binary variables for soft non-simultaneity constraint $< i, j >_{\mathrm{NS}}$. Constraints (3.15) require that exactly 2 of the 3 auxiliary variables have value 1, and thus 1 of them has value 0. $\upsilon_{\mathrm{NS},ij1}$, $\upsilon_{\mathrm{NS},ij2}$ and $\upsilon_{\mathrm{NS},ij3}$ correspond to constraints (3.16), (3.17) and (3.18) respectively. If an auxiliary variable equals 1, the corresponding constraint is always satisfied. Otherwise, from the corresponding constraints $t_{\mathrm{NS},ij}$ must be at least as large as the smallest among $\min\left\{t_{\mathrm{task},i}, t_{\mathrm{task},j}\right\}$, $(t_{\mathrm{task},i} + u_i - u_j)$ and $(t_{\mathrm{task},j} + u_j - u_i)$. Because $c_{\mathrm{NS},ij}$ is always positive and $\mathrm{C}_{\mathrm{NS}}(\cdot)$ is minimized in the model, $t_{\mathrm{NS},ij}$ should be as small as possible. Therefore, in the optimum solution, $t_{\mathrm{NS},ij}$ exactly equals the smallest among the three as long as the smallest one is positive, or 0 otherwise.

In a similar way, hard non-simultaneity constraints (3.14) can also be written in an MIP form, as follows:

$$(U + t_{\mathrm{task},i})\upsilon_{\mathrm{NS},ij} \geq t_{\mathrm{task},i} + u_i - u_j\,, \qquad \text{for all hard } <i, j>_{\mathrm{NS}}, \tag{3.21}$$

$$(U + t_{\text{task},j})(1 - \upsilon_{\text{NS},ij}) \geq t_{\text{task},j} + u_j - u_i, \qquad \text{for all hard } <i, j>_{\text{NS}}, \qquad (3.22)$$

$$\upsilon_{\text{NS},ij} \in \{0,1\}, \qquad \text{for all hard } <i, j>_{\text{NS}}, \qquad (3.23)$$

where $\upsilon_{\text{NS},ij}$ is an auxiliary binary variable for hard non-simultaneity constraint $<i, j>_{\text{NS}}$.

It shall be noted that the equivalent MIP forms of (3.13) and (3.14) introduce additional binary variables and side constraints, which make the model difficult to solve using MIP algorithms.

Non-simultaneity constraints are more relevant in short-term scheduling than in long-term planning. In short-term scheduling, the railroad would like to know the schedule of every task in order to arrange the teams accordingly. In long-term planning, however, the railroad does not directly adopt the obtained schedule because of the existence of many future uncertainties (e.g., the actual task duration). Rather, the results from long-term planning are used to forecast performance statistics (such as the number of overdue tasks) and conduct what-if analyses. Hence, in long-term planning, it is not necessary to go into the detail of non-simultaneity constraints. Furthermore, because the penalty costs of non-simultaneity constraints are calculated based on the schedules of two teams, the accuracy are more susceptible to the uncertainties in track inspection. Therefore, in our study, non-simultaneity constraints are only considered in short-term scheduling but ignored in long-term planning.

3.3.4 Time Window Constraints

Time window constraints require a task to be performed at certain times. One possible reason is to avoid certain exogenous conflicts, which may involve inspection activities by Federal

Railroad Administration (FRA) geometry cars or maintenance projects by production teams. Time window constraints share some similarities with non-simultaneity constraints, and they can be formulated as follows:

$$\mathrm{C}_{\mathrm{TW}}(\mathbf{x}) = \sum_{\forall \text{soft } <i,j>_{\mathrm{TW}}} c_{\mathrm{TW},ij} t_{\mathrm{TWO},ij} \ , \tag{3.24}$$

$$\begin{aligned} t_{\mathrm{TWO},ij} &= \left(\min\left\{ u_i + t_{\mathrm{task},i}, u_{\mathrm{TW},j} + t_{\mathrm{TW},j} \right\} - \max\left\{ u_i, u_{\mathrm{TW},j} \right\} \right)^+ \\ &= \min\left\{ t_{\mathrm{task},i}, t_{\mathrm{TW},j}, \left(t_{\mathrm{task},i} + u_i - u_{\mathrm{TW},j} \right)^+, \left(t_{\mathrm{TW},j} + u_{\mathrm{TW},j} - u_i \right)^+ \right\}, \end{aligned}$$

$$\text{for all soft } <i,j>_{\mathrm{TW}}\ , \tag{3.25}$$

$$\min\left\{ t_{\mathrm{task},i} + u_i - u_{\mathrm{TW},j}, t_{\mathrm{TW},j} + u_{\mathrm{TW},j} - u_i \right\} \le 0, \qquad \text{for all hard } <i,j>_{\mathrm{TW}}\ , \tag{3.26}$$

where $\mathrm{C}_{\mathrm{TW}}(\mathbf{x})$ is the total costs incurred by all soft time window constraints, $<i, j>_{\mathrm{TW}}$ is a pair of task $i \in I$ and time window j from a time window constraint, $c_{\mathrm{TW},ij}$ is the penalty cost per unit time for soft time window constraint $<i, j>_{\mathrm{TW}}$, $u_{\mathrm{TW},j}$ is the start time of time window j, $t_{\mathrm{TW},j}$ is the duration of time window j, and $t_{\mathrm{TWO},ij}$ is the overlapping time of task $i \in I$ and time window j.

Because $u_{\mathrm{TW},j}$ and $t_{\mathrm{TW},j}$ are given as input, $t_{\mathrm{TWO},ij}$ is a piecewise linear function of u_i, as shown in Figure 7. Similar to non-simultaneity constraints, additional binary variables and constraints would be needed if time window constraints were formulated in the MIP form.

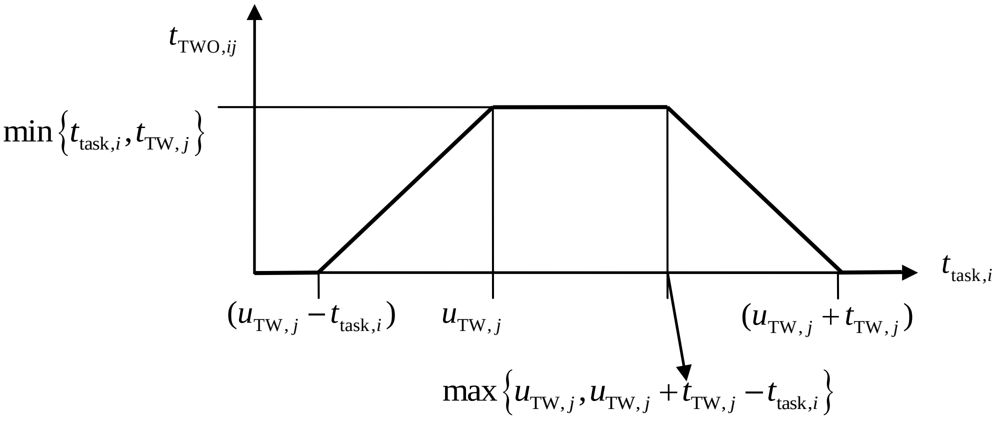


Figure 7. Time window constraint penalty costs in track inspection scheduling.

3.3.5 Preference Constraints

Preference constraints require that some tasks should be performed only by certain teams. There are a few reasons for that. The railroad does not want to send an inspection team too far from their homes, so that the crew members are able to go home during the weekend. The railroad also wants to keep an inspection team working in its familiar territory so as to improve efficiency.

We let a task-team pair $< i,k >_{\text{Pref}}$ represent a preference constraint, which requires task i not to be performed by team k, where $i \in I$, $k \in K$. Preference constraints are formulated as follows:

$$\mathrm{C}_{\text{Pref}}(\mathbf{x}) = \sum_{\forall \text{soft } <i,k>_{\text{Pref}}} \sum_{e_1 \in E_i} \sum_{e_2 \in E} c_{\text{Pref},ik} x_{e_1 e_2 k} , \tag{3.27}$$

$$x_{e_1 e_2 k} = 0 , \quad \forall e_1 \in E_i, e_2 \in E, \text{ for all hard } < i,k >_{\text{Pref}} , \tag{3.28}$$

where $\mathrm{C}_{\mathrm{Pref}}(\cdot)$ is the total penalty costs incurred by all soft preference constraints, and $c_{\mathrm{Pref},ik}$ is the penalty cost if task $i \in I$ is performed by team $k \in K$.[4] Equation (3.27) sums up the penalty costs of all soft preference constraints. Constraints (3.28) define hard constraints, enforcing that task $i \in I$ must not be performed by team $k \in K$.

Preference constraints introduce neither new variables nor new constraints. The cost term $\mathrm{C}_{\mathrm{Pref}}(\cdot)$ is directly added to objective function (3.1). Constraints (3.28) can be implicitly enforced by removing all corresponding variables $x_{e_1e_2k}$ from the model. Therefore, hard preference constraints actually help reduce the number of variables.

However, it should be noted that preference constraints are the only constraints which introduce the distinction among teams. If there were no preference constraints, all teams would be identical, and there would be no difference for a task to be performed by one team or another (as long as the start time does not change). Only because of the preference constraints, subscript k has to be introduced in $\left\{x_{e_1e_2k}\right\}$, making the number of variables k times of the case when teams are not distinguished.

Therefore, the impact of the preference constraints on the solution speed could be either positive or negative, depending on the number of hard preference constraints and the specific algorithm used to solve the problem. As we will discuss in Section 3.4, preference constraints generally do not cause extra difficulty if the problem is solved by heuristic algorithms, because different teams are always processed separately. Therefore, as long as there are hard constraints, the impact of preference constraints is always positive because they strictly eliminate variables and reduce the solution space. For MIP algorithms, the impact of preference constraints is

[4] Note that $c_{\mathrm{Pref},ik}$ can be negative if team k is preferred to perform task i.

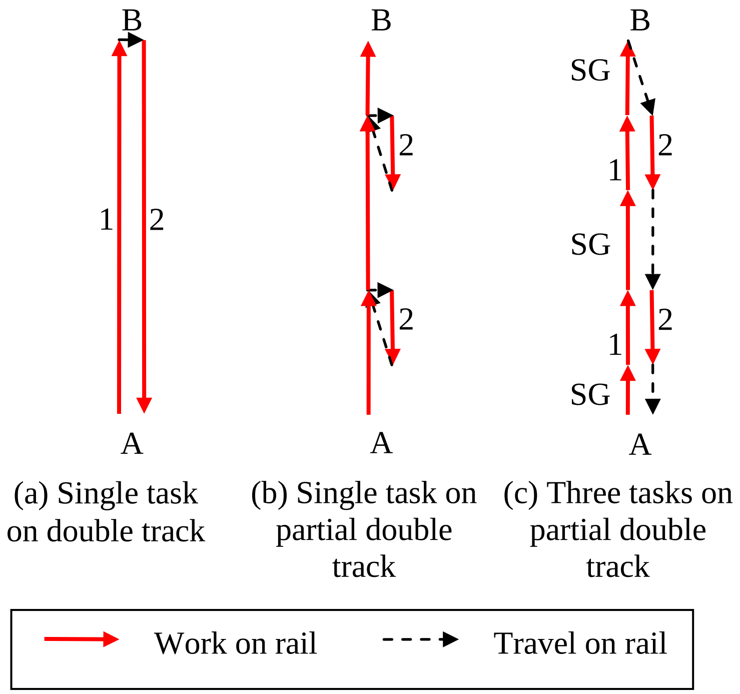
positive only if the number of hard preference constraints is large (i.e., most $x_{e_1 e_2 k}$ variables are eliminated).

3.3.6 Network Topology Constraints

Different from most routing and scheduling problems, where tasks are spatially distributed points and teams are able to travel between tasks freely, TISP are constrained by railroad network topology. One example is that every task is represented by two arcs instead of a single point, as explained in Subsection 3.3.1. In addition, both RISP and GISP have topology constraints that must be addressed.

Rail inspection teams use rail inspection vehicles to perform tasks. Rail inspection vehicles are hi-rail (highway-rail) trucks that can travel both on highway and rail tracks. When performing a task, the team drives on the rail track from one endpoint of the segment to the other. When traveling between two consecutive tasks, the team may travel either on rail or on highway. Generally, if the two tasks are connected by rail and the distance between them is small, the team will travel on rail; otherwise, the team will first find a grade crossing to get onto the highway, and then travel on the highway to the next task.

These rules usually apply to single track lines. When the tasks are on double track lines, the movement of a team becomes more complex. Figure 8 illustrates a few possible travel patterns of rail inspection teams on double track lines. If the task involves both tracks, as shown in Figure 8(a), the team may first work on track 1 from point A to B, then turn around, and work on track 2 from B to A. In this case, both endpoints of this task are at A. Similarly, the team could also start from point B and go back to point B. Sometimes the task is on a partial double track line (i.e., parts of the line are double track and parts are single track). The team may use the same movement as in Figure 8(a) if the proportion of double track is large. Otherwise, the team may only make one movement from A to B while inspecting the small pieces of track 2 along the way, as shown in Figure 8(b). Note that two endpoints of the task are different in Figure 8(b). The variety of possibilities associated with endpoints should be addressed based on the given movement pattern and used as part of the model input.



(a) Single task on double track

(b) Single task on partial double track

(c) Three tasks on partial double track

Figure 8. Travel patterns of rail inspection teams.

Figure 8(c) gives an example when a partial double track line is split into three segments respectively covering track SG (single), track 1 and track 2. Such split is very common because the three tracks usually have different traffic volumes and thus require different inspection intervals. Generally, track SG has more train traffic than track 1 or track 2, because traffic on tracks 1 and 2 (double track) merges onto track SG (single track). Suppose a team makes a movement from point A to B. It can either (i) perform one single task on one of the three segments, or (ii) perform one task on track SG and another on track 1 or 2, or (iii) inspect all three segments. In Figure 8(c), the team uses alternative (ii) for tracks SG and 1, and then goes back to inspect track 2 with alternative (i). In alternatives (ii) and (iii), two or more tasks are combined into one. Such combinations are endogenously determined by the optimization model. If they are formulated into mathematical constraints, the form will be very complex. Luckily, such combinations can be addressed relatively easily if heuristic algorithms are used to solve the problem. In our proposed model, alternatives (i) and (ii) are allowed while (iii) is relatively rare and therefore not considered.

A significant difference between rail inspection and geometric inspection is that geometry inspection teams use a different type of inspection vehicles, which are rail-bound and can only travel on rail tracks. Therefore, GISP involves a different set of network topology constraints from those for RISP. For example, geometry inspection vehicles can only turn around at certain locations in the railroad network (e.g., wyes and sufficiently large turn tables). These vehicles can occasionally make reverse movements along the track, but such movements are slow, disruptive to the inspection operations, and therefore undesirable. In additional, geometry inspection teams must return to certain locations (e.g., yards and terminals) at the end of the day in order to store the inspection vehicles and to change the crews.

3.3.7 Discrete Working Time Constraints

In our problem there are some practical issues with regard to the time dimension. On one hand, inspection teams do not work during weekends and holidays unless paid for overtime. So the continuous time horizon used in the VRP model is not completely realistic. On the other hand, tracks do not stop deteriorating during weekends and holidays[5], and a continuous time horizon is thus needed.

One way to address this problem is to use the approximate formulation for TISP. It can be assumed that a team works 7 days a week, but with only 5/7 efficiency. In this way, the continuous time horizon can still be used. Because the efficiency of every team is reduced, this method tends to bring the start times of tasks forward rather than postpone them. If there were only 2 days during one weekend, the error in the calculation of inspection intervals would be at most 2 days. For a task with preferred interval of 1 month or 31 days, the error would be 6.5%. However, for longer holidays (e.g., as long as 1 week), such approximation will cause larger errors. Furthermore, given the tight schedules of some inspection teams, even two days' error may not be acceptable. For a task with preferred inspection interval of 1 month, the periodicity constraint penalty cost increases very fast when inspection interval is close to the required interval. The cost may change dramatically if the start time of the task changes only one day, and the prioritization of tasks will not be correct if the costs are calculated with such large errors.

An exact way to formulate weekends and holidays in the mathematical model is to use two sets of time variables. One is continuous variables to calculate task durations and travel times and to formulate side constraints such as periodicity constraints. The other one is discrete variables to represent the workdays of teams and to ensure no teams work during weekends and holidays. Additional constraints are also needed to create the relationship between these two sets of variables. Such variables and constraints will greatly increase the complexity of the model. For example, it is difficult to formulate the case when part of the task is done on Friday afternoon and part is done on Monday morning. Such a model, even if formulated, is unlikely to

[5] Sometimes train traffic decreases during holidays, and the inspection frequencies can be reduced accordingly, but that does not happen frequently.

be tractable with MIP algorithms. Luckily, heuristic algorithms are able to handle these complex relationships.

3.3.8 Travel Costs

We assume that travel costs are proportional to the travel time t_{travel,e_1e_2} , as follows:

$$\mathrm{C}_{\text{travel}}(\mathbf{x}) = c_{\text{travel}} \sum_{k\in K} \sum_{e_1\in E_i} \sum_{e_2\in E} t_{\text{travel},e_1e_2} x_{e_1e_2k} \ , \tag{3.29}$$

where $\mathrm{C}_{\text{travel}}(\mathbf{x})$ is the total travel costs and c_{travel} is the travel cost per unit time.

In most routing and scheduling problems, travel cost is one of the major cost items. For example, in railroad transportation problems and highway transportation problems, fuel costs and driver wages are normally dominating. However, in TISP, travel costs are not a major concern. In practice, rail inspection teams are often contracted and paid at a fixed hourly rate. Longer travels do not impose higher costs to the railroad company. In GISP, there are only a few geometry inspection teams, and their fuel costs are relatively negligible compared to high importance of side constraints (such as periodicity constraints). Furthermore, even if travel costs are not explicitly considered, the model will still tend to minimize them indirectly. The reason is that when a team spends more time on travel, it will spend less time performing tasks. Periodicity constraints tend to schedule more tasks within the horizon, and thus tend to reduce the travel time.

In the proposed model, we still consider travel costs. However, c_{travel} is set to be a very low value. So it affects the decision only when no other side constraints are dominating.

3.4 Algorithm

As discussed earlier, TISP is very difficult if solved with MIP algorithms. We therefore propose customized heuristics to handle this problem based on an "incremental horizon" approach. The basic idea is as follows. Suppose the original problem has a total scheduling horizon $[0,\hat{U}]$. Instead of solving the whole problem at once, the algorithm solves a smaller version with a short horizon $[0,U]$, where $U<\hat{U}$. Once a solution is obtained for this smaller problem, we increase U incrementally and solve the extended problems until $U=\hat{U}$. It turns out that this solution approach is able to improve the solution quality, as discussed in Subsection 3.4.4.

Figure 9 illustrates the proposed algorithm framework. The algorithm typically starts with a short scheduling horizon (one to three weeks) and an empty set of tasks. It first generates a task for every segment which does not have an unscheduled task (i.e., a task not assigned to any team). A greedy algorithm and a task-interchange algorithm are then applied to add the newly generated tasks to the schedule and optimize the schedule. The algorithm terminates if all stopping criteria are met, i.e., (i) $U=\hat{U}$, (ii) no new tasks are generated in the current iteration, and (iii) the current solution is not improved in the current iteration; otherwise the algorithm increases the scheduling horizon U by a certain value (one to three weeks) if $U<\hat{U}$, and repeats the process of generating tasks and applying heuristics.

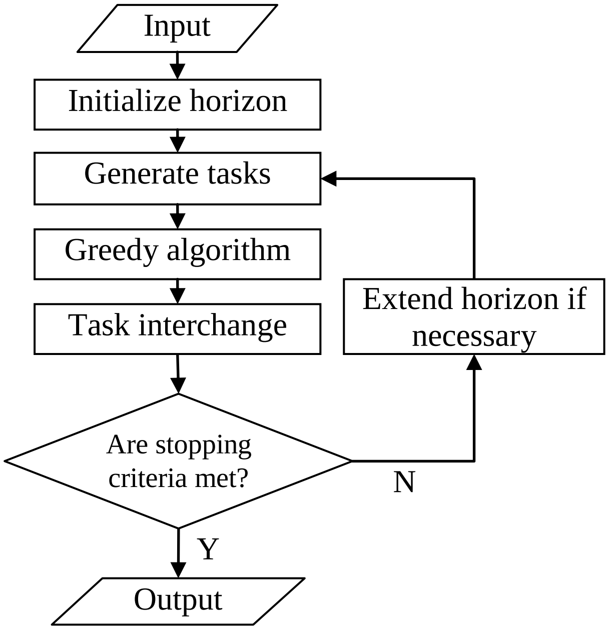


Figure 9. Algorithm framework of track inspection scheduling.

The following subsections discuss the key algorithm modules in more detail.

3.4.1 Task Generation

The optimum value of $\left|I_{\text{Seg},s}\right|$, i.e., the number of tasks to be scheduled on each segment, is unknown at the beginning of the algorithm. For example, suppose the preferred inspection interval of a segment is 31 days and the scheduling horizon is 80 days. It is possible to schedule 2 tasks with an average interval of 35 days within the horizon, or 3 tasks with an average interval of 25 days, or even more tasks with a shorter interval. One way to address this problem is to generate many tasks at the beginning of the algorithm to ensure that they are sufficient for scheduling throughout the solution process. For example, we may generate 5 tasks for the segment mentioned above if we are confident that no more than 5 tasks should be considered in the optimal solution. However, that may incur many more variables than necessary and slow down the solution speed. Determining the sufficient number of tasks also requires ad hoc trials.

In the proposed algorithm, tasks are generated for a segment only when necessary, i.e., when the segment does not have any unscheduled tasks. Clearly, if a segment still has an unscheduled task, there is no need to generate a new task. At the beginning of the algorithm, no segment has any tasks, and a task is generated for each of them. Then greedy and task-interchange algorithms are applied to schedule some of the tasks. After one iteration, the algorithm goes back to the task generation step, and new tasks are generated for those segments without unscheduled tasks. In this way, every segment has at least one unscheduled task, and greedy algorithm and task-interchange algorithm can always find a task to schedule from each segment.

3.4.2 Greedy Algorithm

The proposed greedy algorithm constructs an initial solution which can then be improved by local search.

Let $I_{\mathrm{U}} = \{i : y_i = 0, i \in I\}$, i.e., the set of tasks which are not scheduled within the model scheduling horizon. Let $c_{\mathrm{add},i}(<k,e>)$ and $c_{\mathrm{addPer},i}(<k,e>)$ be respectively the total costs and the periodicity constraint penalty cost incurred by task $i \in I_{\mathrm{U}}$ if it is append to the end of the route of team $k \in K$ and performed in direction $e \in E_i$. The algorithm is described as follows:

Step 1: Terminate the algorithm if all teams have enough tasks to work until U.

Step 2: Find the unscheduled task with the highest periodicity constraint penalty cost,

$$i^* = \arg\max_{i \in I_{\mathrm{U}}} c_{\mathrm{addPer},i} \left(\underset{<k,e> \in <K,E_i>}{\arg\min} c_{\mathrm{add},i}(<k,e>) \right),$$

and the team and arc which minimize the total costs incurred by i^*,

$$<k^*, e^* >= \underset{<k,e> \in <K,E_{i^*}>}{\arg\min} c_{\mathrm{add},i^*}(<k,e>) .$$

Step 3: Append task i^* to the end of the route of team k^*, and let i^* follow direction e^* (i.e., set $x_{e_{\mathrm{last},k} e^* k^*} = 1$ where $e_{\mathrm{last},k}$ is the last arc which team $k \in K$ currently travels to, and update all related variables, such as u_{i^*}, accordingly). Go to Step 1.

In Step 2, i^* is appended to the end of the route of team k^* in direction e^* so that the incremental total costs incurred by i^* are minimized. The proposed greedy algorithm iteratively selects such task i^* and adds it to the schedule. This strategy is intuitive. Since i^* incurs the highest periodicity constraint penalty cost after being added to schedule, it should not be further delayed otherwise even higher cost will be incurred[6].

However, as most greedy algorithm, the proposed greedy algorithm is not able to find an optimum solution in most cases. For example, suppose there are two tasks and one team. Task 1 has a preferred interval of 1 year and a duration of 2 weeks. Task 2 has a preferred interval of 1 month and a duration of 1 day. In the beginning of the horizon, task 1 is overdue for 1 day in terms of the preferred interval if it is immediately performed, and task 2 is 1 day before overdue. Therefore, the greedy algorithm will schedule task 1 first, and schedule task 2 after task 1. However, since task 1 is 2 weeks long, after it is finished task 2 has been about 2 weeks overdue,

[6] Note that the current algorithm always adds the new task to the end of a route. A possible improvement is to insert the new task to the best location along a route. More details will be discussed in Chapter 6.

which is about 50% of its preferred interval and is generally prohibited. The optimum solution should be scheduling task 2 before task 1. In that way task 2 is only delayed by 1 more day (i.e., the duration of task 2), and is overdue for a total of 2 days, which is less than 1% of its preferred interval and is generally allowed. Because of such limitation of the greedy algorithm, we further propose a task-interchange algorithm to improve the initial solution obtained from the greedy algorithm.

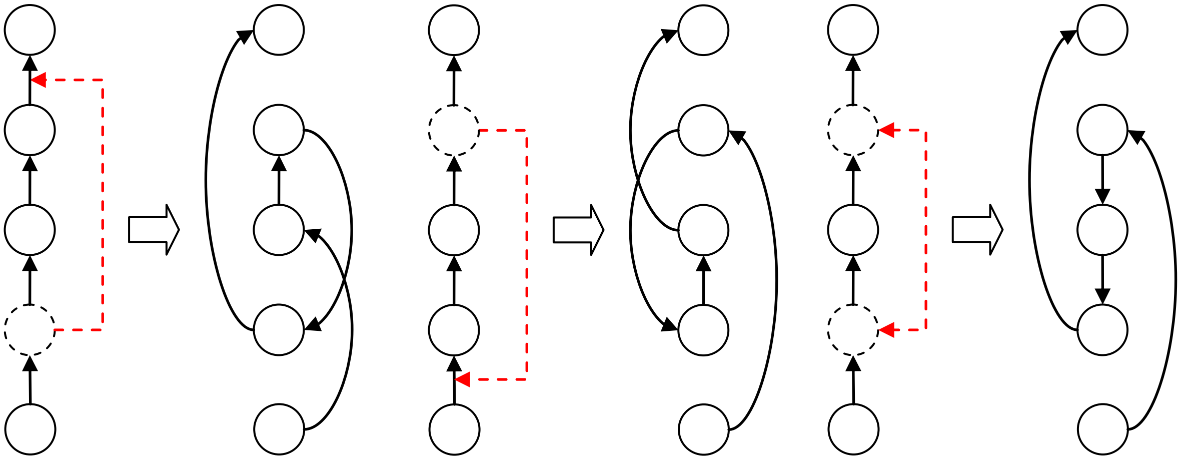
3.4.3 Task Interchange

Task interchange is a local search heuristic that can be used to improve an existing solution. As introduced in Section 2.2.3, local search algorithm checks the solutions within the neighborhood of the current solution (i.e., “check the moves”). If the checked solution has lower costs than the current one, the algorithm updates the current solution to the checked solution (i.e., “accept the move”). We consider seven types of neighborhood structure in task interchange, as follows:

1) One-team-insertion: This move changes the order of a scheduled task within its team. Figures 10(a) and (b) illustrate two possible one-team-insertions. In Figure 10(a), the start time of a task is sent forward, and in Figure 10(b), the start time of a task is sent backward. Let I_{team} be the average number of tasks assigned to a team[7]. Then the size of the one-team-insertion neighborhood (i.e., the number of all possible moves) is $O\left(|K|\, I_{\text{team}}^2\right)$.

2) One-team-swap: This move swaps the orders of two scheduled tasks within a team. Figure 10(c) illustrates a one-team-swap. The size of one-team-swap neighborhood is also $O\left(|K|\, I_{\text{team}}^2\right)$.

[7] Because all teams have the same scheduling horizon length, the numbers of tasks assigned to them should be similar as well.



(a) One-team-insertion (forward)

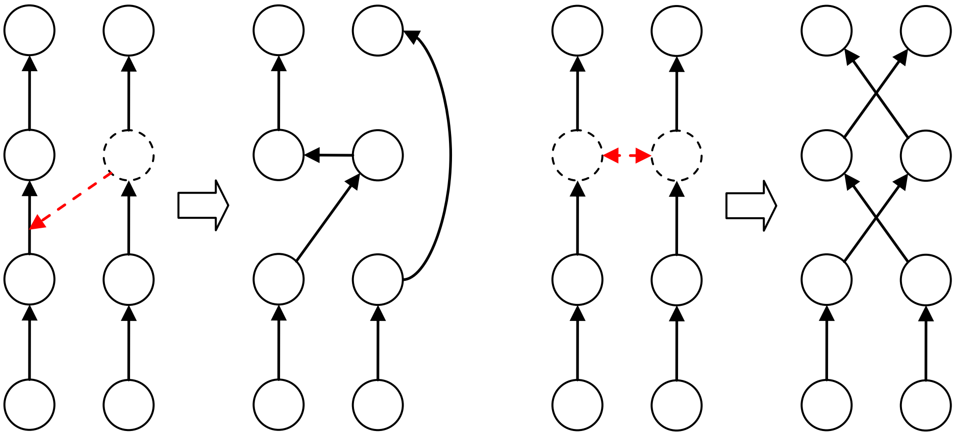
(b) One-team-insertion (backward)

(c) One-team-swap

Figure 10. Examples of task interchange within one team.

3) Two-team-insertion: This move removes a task from one team, and inserts it into the route of another team, as illustrated in Figure 11(a). The size of two-team-insertion neighborhood is $O\left(|K|^2 I_{\text{team}}^2\right)$. However, if there are many hard preference constraints, only the pairs of teams with overlapping territories (i.e., a set of segments which both teams can work on) should be checked for this move. Suppose, on average, a team has overlapping territories with K_{N} other teams. The size of two-team-insertion neighborhood can then be tightened to $O\left(|K| K_{\text{N}} I_{\text{team}}^2\right)$. In our RISP, K_{N} is generally much smaller than $|K|$.

4) Two-team-swap: This move swaps the positions (i.e., both the assigned team and the performed order) of two tasks between two teams, as illustrated in Figure 11(b). The size of two-team-swap neighborhood is also $O\left(|K| K_{\text{N}} I_{\text{team}}^2\right)$.



(a) Two-team-insertion (b) Two-team-swap

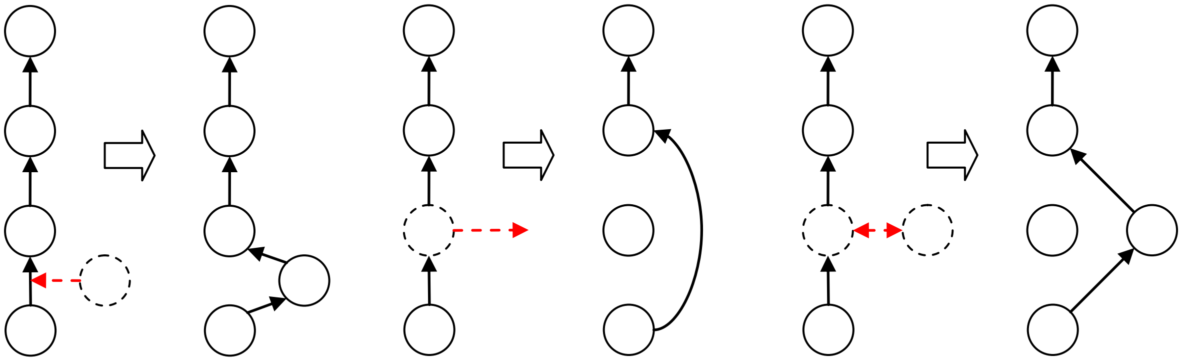
Figure 11. Examples of task interchange between two teams.

5) Add: This move inserts an unscheduled task into the route of a team, as illustrated in Figure 12(a). If all unscheduled tasks are assigned to a dummy team, this move can be viewed as a two-team-insertion between a real team and the dummy team (i.e., moving a task from the dummy team to the real team). The size of add neighborhood is $O\left(|K||I_{\mathrm{U}}|\right)$. Because during the interchange, we always keep (or generate) at least one unscheduled task for each segment, $|I_{\mathrm{U}}| \approx |S|$. So the size of add neighborhood can be written as $O\left(|K||S|\right)$.

6) Drop: This move removes a scheduled task from a team, as illustrated in Figure 12(b), and set it as unscheduled (i.e., by making its start time no earlier than U). This move can also be considered as a two-team-insertion between a real team and the dummy team, where a task is moved from the real team to the dummy team. The size of drop neighborhood is $O\left(|K| I_{\text{team}}\right)$.

7) Add-drop: This move is the combination of an add move and a drop move. A scheduled task is removed from its team and an unscheduled task takes over its position, as illustrated in Figure 12(c). This move can be viewed as a two-team-swap between a real team and the dummy

team. The size of add-drop neighborhood is $O\left(|K||I_{\mathrm{U}}|I_{\text{team}}\right)=O\left(|K||S|I_{\text{team}}\right)$. This move is usually the most time-consuming in short-term scheduling. However, as scheduling horizon length increases, I_{TEAM} also increases, and the sizes of two-team-insertion and two-team-swap, $O\left(|K|K_{\mathrm{N}}I_{\text{team}}^{2}\right)$, become comparable with that of add-drop.


(a) Add

(b) Drop

(c) Add-drop

Figure 12. Examples of task add/drop.

Exhaustive search is used in the proposed algorithm, i.e., all possible moves in the current solution should be enumerated before the algorithm terminates. The total size of task interchange is $O\left(|K|I_{\text{team}}^{2}\right)$ + $O\left(|K|K_{\mathrm{N}}I_{\text{team}}^{2}\right)$ + $O\left(|K||S|\right)$ + $O\left(|K|I_{\text{team}}\right)$ + $O\left(|K||S|I_{\text{team}}\right)$ = $O\left(|K|I_{\text{team}}\left(|S|+K_{\mathrm{N}}I_{\text{team}}\right)\right)$, which is acceptable in practice.

A few approaches are used to improve the solution speed. First, excessive calculations of costs are avoided. It can be seen that whenever a task is moved from one position to another, the start times of all its original successors (i.e., tasks performed after it in the same route) and new successors are affected. The schedules of other tasks, however, are not affected. Therefore, we trace the costs related to every task. Whenever a move is checked, the algorithm only visits the tasks whose start times are affected, calculates the cost change for each of them, and then calculates the total cost change to decide if the move should be accepted.

Second, duplicated moves are avoided. For example, swapping two consecutively performed tasks is equivalent to inserting the successor before the predecessor or the predecessor after the successor. Therefore, a task is not allowed to be inserted right before its predecessor or right after its successor in one-team-insertion,

Finally, preliminary checks are used to quickly reject those moves which make the solution infeasible or are unlikely to improve the solution. For example, the algorithm never allows a task to be inserted somewhere which will lead the inspection interval to significantly exceed the requirement. Also, for task $i_{\text{Seg},so}$, $s \in S$, $o = 1, \ldots, \left| I_{\text{Seg},s} \right|$, the algorithm never tries to make it start before $i_{\text{Seg},s(o-1)}$, or after $i_{\text{Seg},s(o+1)}$.

The sequence of checking moves is important. The algorithm always starts moving tasks from the end of the route. There are two reasons for this. First, such a strategy tends to introduce smaller disturbance to the solution. Second, such a strategy tends to bring forward the start time of an overdue task instead of postponing the previous task in the same segment. This can be illustrated by the following example. Suppose in segment $s \in S$ there are two tasks, $i_{\text{Seg},s1}$ and $i_{\text{Seg},s2}$, both performed by the same team, and only $i_{\text{Seg},s2}$ is overdue, i.e., the interval between the finish times of $i_{\text{Seg},s1}$ and $i_{\text{Seg},s2}$ exceeds the preferred inspection interval. For simplicity, assume no costs from other tasks are involved. If $i_{\text{Seg},s2}$ is moved first, the algorithm will insert it forward so that it will be performed within its preferred interval. However, if we move tasks from the start of the route, $i_{\text{Seg},s1}$ will be moved first. Because the periodicity constraint penalty cost function is convex, the algorithm will insert $i_{\text{Seg},s1}$ backward, in a way that the periodicity constraint penalty costs incurred by $i_{\text{Seg},s1}$ and $i_{\text{Seg},s2}$ are about the same. In that case, both tasks may be overdue, although in the optimum solution neither of them should be overdue.

Because a move may change the start times of many tasks and dramatically affect the current solution, the proposed task-interchange algorithm tends to be stuck at local optimal solutions. Hence, the quality of the final solution depends heavily on the initial solution. That is also why we propose the incremental horizon framework. In the hope to improve solution quality, simulated annealing (Osman, 1993) has also been tested. However, it failed to yield a good performance. The reason may probably be due to the fact that simulated annealing tends to disrupt the possibly desirable structure of the initial solutions from the greedy heuristic.

3.4.4 Incremental Horizon

The task-interchange algorithm has difficulty moving tasks in the early part of a long scheduling horizon, because such a move affects the start times of all tasks in the late part of the horizon and tends to dramatically change the objective value. Therefore, an incremental horizon approach is used. The algorithm starts with a decomposed problem with a short scheduling horizon, and increases the horizon gradually until it reaches the total scheduling horizon $\hat{U}$. In this way, the schedules of early tasks can be optimized when the model scheduling horizon is still short. Although this approach would not work well if the schedules of early tasks were highly dependent on the horizon length, fortunately, that is not the case in TISP. Because of the periodicity constraints, tasks due earlier should generally be scheduled in the early part of the horizon.

The initial horizon length and the increment of horizon length should not be longer than the minimum preferred inspection interval across all tasks. Otherwise a segment might require more than one inspection within the initial or incremental horizon length. Because at most one task is generated in each iteration, there may not be enough tasks to be scheduled in that iteration. We shall note that the initial and increased horizon lengths do affect the solution speed and therefore should be chosen carefully. Shorter initial or incremental horizon lengths require more algorithm iterations. In our RISP applications, the length is usually set to be around 1 to 3 weeks, because the minimum preferred inspection interval is normally one month and it is very unlikely that a segment requires two inspections within 3 weeks.

3.5 Case Studies

The proposed model and algorithms are applied to a full-scale RISP in a Class I railroad company. The algorithms are implemented in the Microsoft Visual C# environment on a personal computer with 2.66GHz dual core CPU and 3 GB RAM.

The first case study is conducted for a weekly scheduling problem instance in some week of a recent year (referred to as Year A from now on). This problem instance contains about 20 teams and more than 700 segments. The scheduling horizon is 8 weeks, and every segment has from 1 to 4 tasks to be scheduled. Every task involves one periodicity constraint, at least one preference constraint and at least two non-simultaneity constraints (one subdivision non-simultaneity constraints and at least one roadmaster non-simultaneity constraints). Some periodicity constraints and preference constraints are hard. Travel costs, discrete working time constraints and some network topology constraints are also considered. The initial and incremental horizon lengths are set to 3 weeks.

The solution time of the algorithm is less than 1 minute. The model solution is compared with the manual solution provided by experts from the company, as shown in Table 1. To protect data confidentiality, the statistics of both manual solution and model solution are scaled by a constant factor between 0.5 and 2. Such scaling does not affect the comparison of the results.

Table 1. Solution comparison for track inspection scheduling.

Statistics	*Manual solution*	*Model solution*	*Reduction (%)*
Total overdue percentage outside the required interval (%)	15.8	4.2	73.7
Total travel distance between tasks (miles per team per week)	63.2	47.4	25.0
Total non-simultaneity constraint overlapping duration (days per week)	0.66	0.42	37.5

The total overdue percentage outside the required interval is calculated as $\sum_{i \in I} \left((\Delta u_i - t^{\mathrm{R}}_{\mathrm{SegInt},s_i})^+ / t_{\mathrm{SegInt},s_i} \right) \times 100$, where s_i is the segment of task $i \in I$. It is generally the major proportion of periodicity constraint penalty costs if the interval of any task exceeds the required interval. Total travel distance is calculated as the average team travel speed multiplied by the total travel time, i.e., $\sum_{k \in K} \sum_{e_1 \in E} \sum_{e_2 \in E} t_{\mathrm{travel},e_1 e_2} x_{e_1 e_2 k}$. It is proportional to the travel costs. Although travel costs are low compared to periodicity constraint penalty costs, total travel distance is still an intuitive indicator of the solution quality, because less travel distance indicates more time to perform tasks. Total non-simultaneity constraint overlapping duration is calculated as $\sum_{\text{all } <i,j>_{\mathrm{NS}}} t_{\mathrm{NS},ij}$ (the overlapping duration of two tasks may be counted more than once if they are involved in multiple non-simultaneity constraints, e.g., a subdivision non-simultaneity constraint and a roadmaster non-simultaneity constraint). Because all non-simultaneity constraints are assumed to have the same penalty cost per unit time, this total overlapping duration is proportional to the non-simultaneity constraint penalty costs. It can be seen from Table 1 that the model solution has a better performance in terms of all three major statistics.

Figure 13 shows the results of the sensitivity analysis on the incremental horizon length (which is also used as the initial horizon length). The scenario with an incremental horizon length of 3 weeks (i.e., 21 days) is used as the benchmark. The statistics of other scenarios are compared to those of the benchmark scenario, and the percentage differences are shown in the Figure. It can be seen that as the incremental horizon length increases, the objective value generally increases while the solution time decreases (despite some fluctuations). When the incremental horizon length exceeds 34 days, no statistics are shown because the solution becomes infeasible (due to violations to hard constraints). Our observations imply the existence of a general tradeoff between the objective value and the solution time. A shorter incremental horizon length requires more algorithm iterations, but may better improve the solution. Also, the incremental horizon should not exceed the minimum preferred inspection interval, which is 31 days in our case. Otherwise there may not be enough tasks generated in an algorithm iteration, and the solution may incur high costs or even become infeasible.

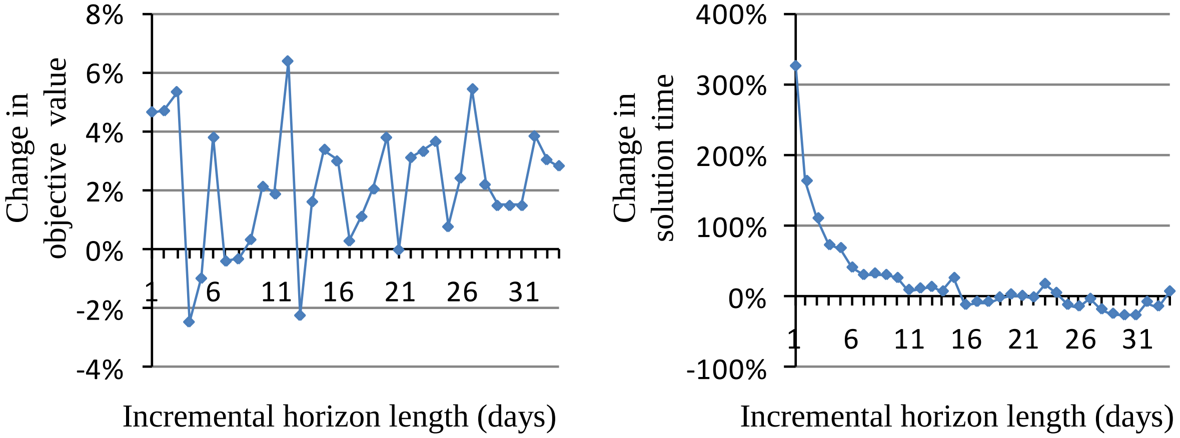
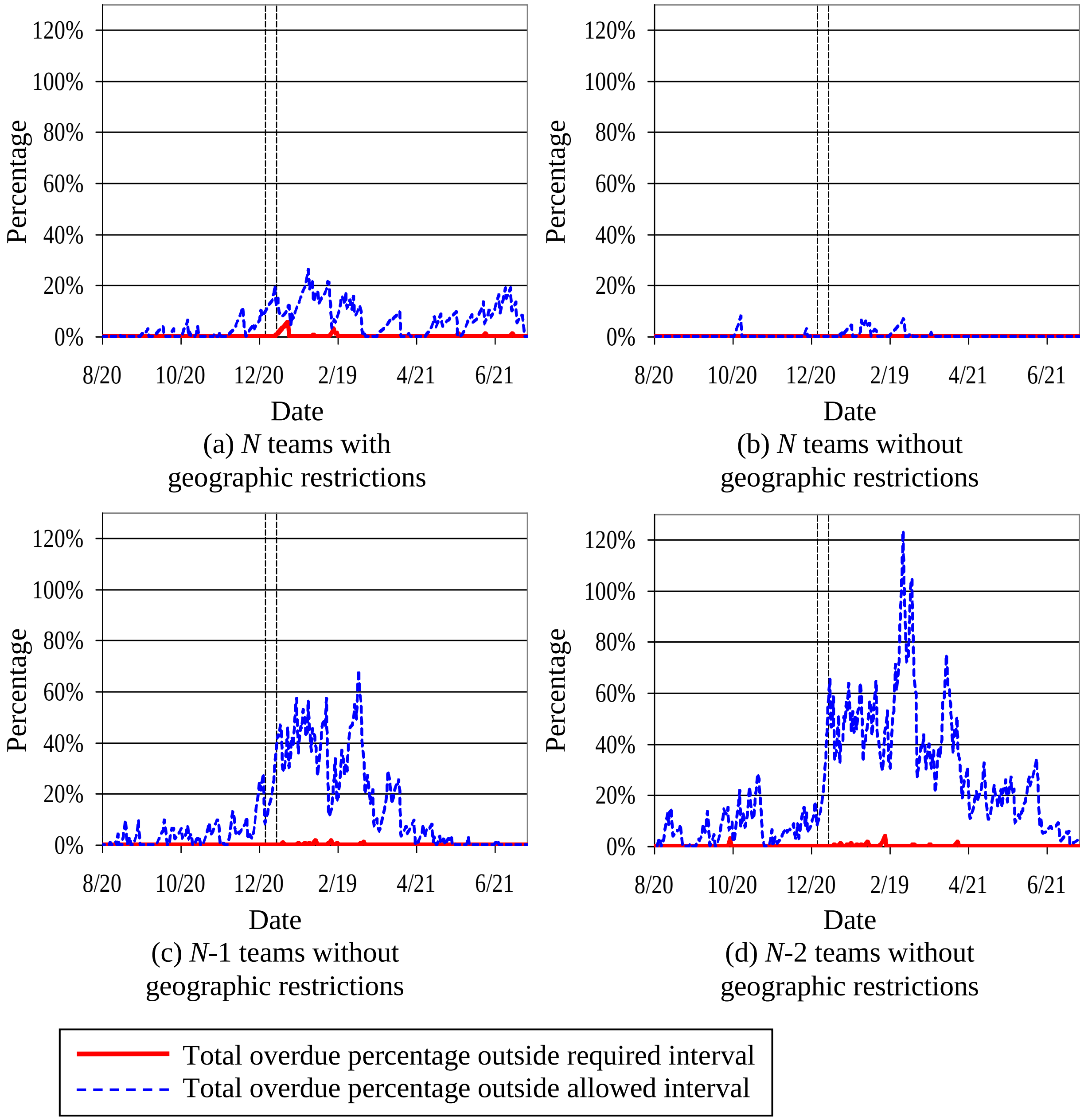


Figure 13. Sensitivity analysis on incremental horizon length
(benchmark scenario: incremental horizon length = 21 days).

The second case study is conducted for a long-term planning problem instance. The scheduling horizon is one year, and it starts from the summer of Year A. The input data and constraints are the same as those of the weekly scheduling, except that the horizon is extended and non-simultaneity constraints are ignored to improve solution speed (as discussed in Subsection 3.3.3). For such a long scheduling horizon, some segments may be inspected more than 12 times within the horizon. Thus the number of variables is much larger. The model is run for four different scenarios to perform what-if analyses: (a) using *N* (i.e., the current number of teams used by the railroad) teams with geographic restrictions (i.e., preference constraints which require a team to only work on certain segments), which is the current industry practice, and using (b) *N*, (c) *N*-1 and (d) *N*-2 teams without geographic restrictions (except some segments which have special policies and whose geographic restrictions are unlikely to be changed in the future). The model is able to obtain the solution within one hour for each of the four scenarios. The results are shown in Figure 14. The first month of the solutions has been removed in order to eliminate the influence of the manual schedule implemented before the horizon.



(a) *N* teams with geographic restrictions

(b) *N* teams without geographic restrictions

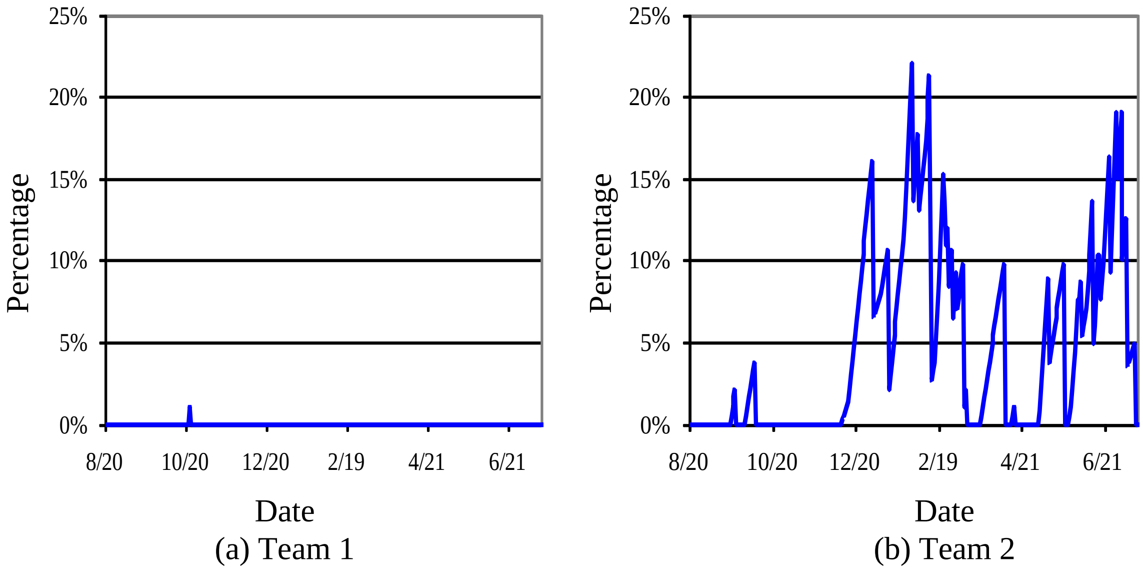
(c) *N*-1 teams without geographic restrictions

(d) *N*-2 teams without geographic restrictions

Figure 14. What-if analysis for one-year scheduling horizon.

The trends of the total overdue percentages outside the required and allowed inspection intervals are plotted for each scenario. Total overdue percentage outside the allowed inspection interval is defined in a similar way as that outside required inspection interval, i.e., $\sum_{i\in I}\left((\Delta u_i - t^{\mathrm{A}}_{\mathrm{SegInt},s_i})^{+} / t_{\mathrm{SegInt},s_i}\right)\times 100$. It can be seen that the model is able to eliminate almost all overdue percentage outside the required inspection intervals within the year. If the railroad keeps *N* inspection teams but remove the geographic restriction of them, most overdue percentage outside the allowed inspection interval can also be eliminated. If the railroad remove one or two inspection teams from the workforce, the percent of overdue outside the allowed inspection interval will increase, but there is still little overdue outside the required interval. So there is a tradeoff between the number of inspection teams and the overdue percentage. If the costs of extra teams in scenarios (a) and (b) can justify the reduced overdue percentage compared to (c) and (d), the railroad should probably keep all teams; otherwise it can reduce the number of teams.

It can be seen that in all four scenarios the overdue percentages begin to increase dramatically right after the long Christmas/New Year holiday which is indicated by vertical dashed lines. When there are *N* teams, the backlogs can dissipate rather quickly. However, when there are *N*-1 or *N*-2 teams, it will take a few months to clear the backlogs. This suggests a possibility of improving the inspection performance. If the railroad is able to have a short contract of a team only in the few weeks after Christmas/New Year vacation to catch up with the backlogs, maybe it is able to use only *N*-1 or *N*-2 teams during most time of the year while still be able to conduct most inspection tasks within the allowed inspection intervals.



(a) Team 1

(b) Team 2

Figure 15. Total overdue percentages outside allowed interval of individual teams.

It can also be seen that overdue percentages decrease when the geographic restrictions are removed. Without geographic restrictions, a team can help other teams perform tasks if no tasks in its own territory are due. Figure 15 compares the workloads of two teams in scenario (a) which are obviously not balanced. Team 1 has much less overdue percentage outside the allowed inspection interval than team 2, indicating that team 1 has less workload than team 2, and it may be beneficial to assign more segments to team 1 and fewer to team 2. This suggests that the railroad may be able to improve the inspection performance by revising the preference constraints.

Glossary of Symbols

$c_{\mathrm{add},i}(\cdot)$:	Total costs incurred by adding task $i \in I$ to the end of the route of a team,
$c_{\mathrm{addPer},i}(\cdot)$:	Periodicity constraint penalty costs incurred by adding task $i \in I$ to the end of the route of a team,
$C_{\mathrm{cost\ item}}(\cdot)$:	Total costs of a certain cost item,
$c_{\mathrm{NS},ij}$:	Non-simultaneity constraint penalty cost per unit time for $< i, j >_{\mathrm{NS}}$,
$\mathrm{C}_{\mathrm{NS}}(\cdot)$:	Total non-simultaneity constraint penalty costs,
$c_{\mathrm{Per},i}(\cdot)$:	Periodicity constraint penalty cost function for task $i \in I$,
$\mathrm{C}_{\mathrm{Per}}(\cdot)$:	Total periodicity constraint penalty costs,
$c_{\mathrm{Pref}\ ,ik}$:	Preference constraint penalty cost for team $k \in K$ to perform task $i \in I$
$\mathrm{C}_{\mathrm{Pref}}\ (\cdot)$:	Total preference constraint penalty costs,
$c_{\mathrm{task},i}$:	Penalty cost when task $i \in I$ is not scheduled within scheduling horizon,
c_{travel} :	Travel cost per unit time,
$\mathrm{C}_{\mathrm{travel}}(\cdot)$:	Total travel costs,
$c_{\mathrm{TW},ij}$:	Time window constraint penalty cost per unit time for $< i, j >_{\mathrm{TW}}$,
$\mathrm{C}_{\mathrm{TW}}(\cdot)$:	Total time window constraint costs,
E :	Set of arcs representing the direction to perform a task,
E_i :	Set of arcs representing the direction to perform task $i \in I$,
$e_{\mathrm{last},k}$:	The last arc which team $k \in K$ travels to,
$e_{\mathrm{start},k}$:	Initial location of team $k \in K$,
I :	Set of inspection tasks,
$i_{\mathrm{first},s}$:	First task in segment $s \in S$,
$< i, j >_{\mathrm{NS}}$:	Pair of tasks in a non-simultaneity constraint,
$< i, j >_{\mathrm{TW}}$:	Pair of a task and a time window in a time window constraint,
$< i, k >_{\mathrm{Pref}}$:	Pair of a task and a team in a preference constraint,
$I_{\mathrm{Seg},s}$:	Set of tasks on segment $s \in S$,
$i_{\mathrm{Seg},so}$:	The o th performed task on segment $s \in S$,
I_{team} :	Average number of tasks assigned to a team,
I_{U} :	Set of unscheduled tasks,
K :	Set of inspection teams,

K_{N}: Average number of teams which a team has overlapping territory with (excluding itself),

N: Number of teams in the railroad's current practice in the case study,

S: Set of track segments,

s_i: Segment of task $i \in I$,

$t_{\mathrm{Int},i}$ Preferred interval of task $i \in I$,

$t_{\mathrm{NS},ij}$: Overlapping duration of tasks i and j,

$t_{\mathrm{SegInt},s}$: Preferred interval of segment $s \in S$,

$t_{\mathrm{SegInt},s}^{\mathrm{A}}$: Allowed interval of segment $s \in S$,

$t_{\mathrm{SegInt},s}^{\mathrm{R}}$: Required interval of segment $s \in S$,

$t_{\mathrm{task},i}$: Duration of task $i \in I$,

$t_{\mathrm{travel},e_1e_2}$: Travel time from the head of arc $e_1 \in E$ to the tail of arc $e_2 \in E$,

$t_{\mathrm{TW},j}$: Duration of time window j,

$t_{\mathrm{TWO},ij}$: Overlapping duration of task i and time window j,

$\mathbf{u} = \{u_i\}$: Real variables for the start time of tasks,

U: End time of the scheduling horizon in each algorithm iteration,

$\hat{U}$: End time of the overall scheduling horizon,

$u_{\mathrm{finish},i}$ Finish time of task i,

Δu_i: Inspection interval of task $i \in I$,

$u_{\mathrm{TW},j}$: Start time of time window j,

$\mathbf{x} = \{x_{e_1e_2k}\}$: Binary variables for the movements of teams,

y_i: Binary variables indicating if task $i \in I$ is scheduled within scheduling horizon.

CHAPTER 4 PRODUCTION TEAM SCHEDULING

4.1 Introduction

Railroads usually identify capital track maintenance projects across the railroad network before the beginning of each year, and send out production teams to perform the projects during the year. The production team scheduling problem (PTSP) assigns every project to a team and determines its start time.

Solving PTSP often becomes a significant challenge to railroads. Many business constraints are imposed on PTSP due to various reasons such as railroad traffic operations, climate, and interactions among different projects. Some of these constraints are soft (i.e. violations can be tolerated at some costs if no better choice exists), and some are hard (i.e. violations should be prohibited). Poorly designed maintenance schedules may incur high maintenance and operating costs, or may not be even implementable due to violations to hard constraints. Furthermore, upon completing a project, a team has to travel immediately to the next project. There are very high costs associated with transporting the heavy machinery between consecutive project locations. This mandates that the travel distance be minimized while all projects are performed by suitable teams at suitable times.

Sometimes a project can be split into multiple parts, and each part can be scheduled separately to yield more flexibility. From now on, such a project will be called a “split project”. Unless specified, each part of a split project will be considered a separate project, and “project” may represent either an unsplit project or one part of a split project. Multiple parts of a split project may be performed by multiple teams simultaneously in order to reduce the impact on railroad operation. Suppose a project would have a duration of 10 weeks for one team. If it is split into two equal parts and performed by two teams simultaneously, it will be finished in 5 weeks and consequently its impact on railroad operation will be reduced by half. We call such a split project “folded”. Such practice makes the problem even more difficult.

Despite the high complexity of PTSP, current practice in the railroad industry mostly relies on the experience and knowledge of experts. The scheduling process is largely manual. It usually takes a group of experts more than one week, yet rarely provides a satisfying solution. This chapter proposes mathematical models and corresponding algorithms to solve PTSP effectively and efficiently.

4.2 Model and Algorithm Selection

Similar to TISP, PTSP is also a routing and scheduling problem with many side constraints. However, they have a few differences which significantly affect the selection of the model formulation and algorithm. First, TISP is solved very frequently, and its solution time should be as short as possible, preferably within a few minutes. On the other hand, PTSP is solved only once a year, and a solution time of a few hours (e.g., overnight) is acceptable. Second, the tasks in TISP may have durations varying from less than an hour to a few weeks, and the travel time between projects, which is variable in length, must be incorporated into the schedule. Hence it is difficult to discretize the scheduling horizon. In PTSP, however, all project durations are integer numbers of weeks. The travel time between projects normally occur during weekends and can be neglected in the scheduling process. Hence it is natural to discretize the scheduling time horizon into weeks. Finally, practical PTSP have many more hard side constraints than TISP. It is difficult to obtain a satisfying or even feasible solution without applying MIP algorithms. Because of all these reasons, a time-space network (TSN) model with MIP algorithm will be used to solve TISP in a discrete time horizon. Heuristic algorithms are also incorporated into the solution framework to address the large number of variables and constraints.

4.3 Model Formulation

4.3.1 Core Model

Let P be the set of projects, and K be the set of teams. Both teams and projects can be divided into multiple categories based on technical specialization, such as rail maintenance and T&S (timber and surfacing) maintenance. A project of a certain category can only be performed by teams from the same category. A category can be further divided into multiple types. For example, the rail team category includes large rail and small rail teams, and the rail project category includes curve patch and out-of-face projects. Each team can work on one or more types of projects based on their specializations. For convenience, we let $K_p \subseteq K$ be the subset of teams that can perform project $p \in P$, and $P_k \subseteq P$ be the subset of projects that can be performed by team $k \in K$.

Let $W = \{1, 2 \cdots, |W|\}$ denote the set of all weeks in the scheduling horizon (e.g., a year). Noting that productivities may vary across teams, we define t_{pk} as the duration (in terms of number of weeks) of project $p \in P$ if it is performed by team $k \in K_p$. Define the set of start weeks $W_{\text{start},k} = \{1, 2 \cdots, |W_{\text{start},k}|\}$ and the set of end weeks $W_{\text{end},k} = \{|W| - |W_{\text{end},k}| + 1, \cdots, |W|\}$, such that in the scheduling horizon, team $k \in K$ is allowed to start its work in any week $w \in W_{\text{start},k}$, and complete its work in any week $w \in W_{\text{end},k}$. The scheduling flexibility increases with the sizes of $W_{\text{start},k}$ and $W_{\text{end},k}$. If they are both singletons (i.e. a set with exactly one element), team $k \in K$

has to perform its work exactly in the designated weeks; on the other hand, if $W_{\text{start},k}$ and $W_{\text{end},k}$ are both equal to W, team $k \in K$ can choose its start or end week freely if no other constraints are present.

The spatial railroad network can be represented by an undirected graph with a set of vertices (i.e., track milestones) $V = \left\{1, 2, \cdots, |V|\right\}$ and a set of edges (i.e., railroad tracks) E. As discussed in Section 2.2.2, the TSN model duplicates the spatial network into multiple layers, one for each discrete time point. There are two types of flow (i.e., the routes of teams) in the time-space network: (i) flow within a spatial network layer, which corresponds to the travel between different projects; and (ii) flow across the layers, which corresponds to the movement of the team while performing a project. Define $\mathbf{x} = \{x_{v_1 v_2 kw}, \forall v_1, v_2 \in V, k \in K, w \in W\}$ as the set of type-(i) flow, where binary variable $x_{v_1 v_2 kw} = 1$ if team $k \in K$ travels from vertex $v_1 \in V$ to vertex $v_2 \in V$ in week $w \in W$; or 0 otherwise. Define $\mathbf{y} = \{y_{pkw}, \forall p \in P_k, k \in K, w \in W\}$ as the set of type-(ii) flow, where binary variable $y_{pkw} = 1$ if team $k \in K$ starts to work on project $p \in P$ in week $w \in W$; or 0 otherwise. Because a project has two endpoints and a team may start to work at either of them, we further define variables y^+_{pkw} and y^-_{pkw}, s.t. $y^+_{pkw} + y^-_{pkw} = y_{pkw}$, to indicate the direction the team follows. Suppose $v_1 \in V$ and $v_2 \in V$ are two endpoints of project $p \in P$ and $v_1 \le v_2$. Then $y^+_{pkw} = 1$ and $y^-_{pkw} = 0$ if team $k \in K$ starts project p from v_1 and ends at v_2; or $y^+_{pkw} = 0$ and $y^-_{pkw} = 1$ otherwise. Define $P^+_{vk} \subseteq P_k$, $\forall k \in K$, $v \in V$ as the subset of projects such that project $p \in P_k$ belongs to P^+_{vk} if (i) one of its endpoint is v and (ii) index v is equal to or smaller than the other endpoint. Similarly, the counterpart $P^-_{vk} \subseteq P_k$ contains those projects in P_k with endpoint index v larger than the other endpoint.

The core model (without side constraints) can be formulated as follows:

(PTSP)
$$\min \sum_{\text{all cost items}} C_{\text{cost item}}(\mathbf{x},\mathbf{y}), \tag{4.1}$$

s.t.

$$\sum_{w\in W_{\text{start},k}} \sum_{v\in V} x_{v_{\text{start},k}vkw} = 1, \qquad \forall k \in K, \tag{4.2}$$

$$\sum_{v'\in V} x_{vv'kw} - \sum_{v'\in V} x_{v'vkw} + \sum_{p\in P_{vk}^{+}} \left(y_{pkw}^{+} - y_{pk(w-t_{pk})}^{-} \right) + \sum_{p\in P_{vk}^{-}} \left(y_{pkw}^{-} - y_{pk(w-t_{pk})}^{+} \right) = 0,$$
$$\forall v \in V \setminus \{v_{\text{start},k}, v_{\text{end},k}\}, k \in K, w \in W, \tag{4.3}$$

$$\sum_{w\in W_{\text{end},k}} \sum_{v\in V} x_{vv_{\text{end},k}kw} = 1, \qquad \forall k \in K, \tag{4.4}$$

$$y_{pkw}^{+} + y_{pkw}^{-} = y_{pkw}, \qquad \forall p \in P_k, k \in K, w \in W, \tag{4.5}$$

$$\sum_{w\in W} \sum_{k\in K_p} y_{pkw} = 1, \qquad \forall p \in P, \tag{4.6}$$

$$y_{pkw}^{+}, y_{pkw}^{-} \in \{0,1\}, \qquad \forall p \in P_k, k \in K, w \in W. \tag{4.7}$$

The objective function (4.1) minimizes the summation of all costs. Constraints (4.2)~(4.4) ensure flow conservation: team $k \in K$ starts to work at the beginning of the scheduling horizon, and after finishing a project it moves to the next project immediately. Constraints (4.5) indicate the direction the team follows when performing a project. Constraints (4.6) ensure every project is performed exactly once. Constraints (4.7) define binary variables. Note that variables $\mathbf{x}$ do not have to be binary. The reason is that when $\mathbf{y}$ are binary there is always an optimum solution where $\mathbf{x}$ are integers, and such a solution can be easily found once $\mathbf{y}$ is determined.

4.3.2 Travel Costs

In PTSP, the travel of maintenance large machinery constitutes an important cost component. However, travel time is negligible because travel always occurs during weekends.[8] Similar to (3.29) in TISP, the travel cost can be calculated as follows:

$$\mathrm{C}_{\mathrm{travel}}(\mathbf{x}) = \sum_{w \in W} \sum_{k \in K} \sum_{v_1 \in V} \sum_{v_2 \in V} c_{\mathrm{travel},v_1 v_2 k} x_{v_1 v_2 k w} \ , \tag{4.8}$$

where $\mathrm{C}_{\mathrm{travel}}(\cdot)$ is the total travel cost, and $c_{\mathrm{travel},v_1 v_2 k}$ is the cost for team $k \in K$ to travel from vertex $v_1 \in V$ to $v_2 \in V$.

In this formula, $c_{\mathrm{travel},v_1 v_2 k}$ is assumed to be a simple linear function of the travel distance between v_1 and v_2. It is also assumed that the travel cost is incurred only when a team travels between different projects, but not while they are performing a project from one endpoint to the other. The reason is that the total distance of the latter type of movements is constant and inevitable.

4.3.3 Time Window and Preference Constraints

Time window constraints require a project to be performed in certain weeks. Weather is the most important factor behind such requirements. For example, it is generally not preferable to schedule projects in the northern U.S. regions during the winter time due to low temperature and heavy snowfall. There are also other reasons for time window constraints. For example, certain

[8] This is quite different from TISP, where travel time is important but travel cost is only a minor part of the total cost.

projects may be preferred to be performed within (or outside of) some specific weeks due to seasonal low (or high) railroad traffic volume (e.g., holiday season).

Define y'_{pkw} as follows:

$$y'_{pkw} = \sum_{w'=w-t_{pk}+1}^{w} y_{ptw'} , \qquad \forall p \in P_k , k \in K, w \in W . \tag{4.9}$$

So $y'_{pkw} = 1$ if team $k \in K$ is working on project $p \in P_k$ in week $w \in W$, or 0 otherwise. Let $< p, w >_{\mathrm{TW}}$ denote a time window constraint where project $p \in P$ should not be performed in week $w \in W$. Time window constraints can be formulated as follows:

$$\mathrm{C}_{\mathrm{TW}}(\mathbf{y}) = \sum_{\forall \text{soft } <p,w>_{\mathrm{TW}}} c_{\mathrm{TW},pw} \sum_{k \in K_p} y'_{pkw} , \tag{4.10}$$

$$y'_{pkw} = 0 , \quad \forall k \in K_p , \text{ for all hard } < p, w >_{\mathrm{TW}} , \tag{4.11}$$

where $\mathrm{C}_{\mathrm{TW}}(\cdot)$ is the total time window constraint penalty costs, and $c_{\mathrm{TW},pw}$ is the penalty cost for soft time window constraint $< p, w >_{\mathrm{TW}}$.

The formulation of time window constraints in PTSP is quite different from that in TISP, mostly because of the treatment on discrete versus continuous time horizons. In TISP, the overlapping duration of a task and a time window can be any real numbers, and the penalty cost is a non-linear function of the task start time. In PTSP, however, the overlapping duration is always an integer number of weeks, so a time window can be discretized into multiple weeks and the total penalty cost from that time window can be explicitly specified as the summation of individual penalties in each week, as in (4.10). It shall be noted that time window constraints introduce neither new variables nor new constraints to PTSP. Hard time window constraints (4.11) can be implemented by simply eliminating some y_{pkw} variables, which helps to reduce the scale of the problem (while increasing the difficulty of finding a feasible solution).

Preference constraints require that a project be performed by certain teams; e.g., the crew in a team prefers to work near their homes or in familiar territories. Let $< p,k >_{\mathrm{Pref}}$ denote a preference constraint where project $p \in P$ should not be performed by team $k \in K$. Preference constraints can be formulated as follows:

$$\mathrm{C}_{\mathrm{Pref}}(\mathbf{y}) = \sum_{\forall \mathrm{soft} <p,k>_{\mathrm{Pref}}} c_{\mathrm{Pref},pk} y_{pkw} \text{ ,} \tag{4.12}$$

$$y_{pkw} = 0 \text{,} \quad \text{for all hard } < p,k >_{\mathrm{Pref}} \text{ ,} \tag{4.13}$$

where $\mathrm{C}_{\mathrm{Pref}}(\cdot)$ is the total preference constraint penalty costs, and $c_{\mathrm{Pref},pk}$ is the cost for constraint $< p,k >_{\mathrm{Pref}}$ if it is soft.

Preference constraints in PTSP are similar to those in TISP. They introduce neither new variables nor new constraints, but they introduce distinctions among teams and thus increase the complexity of the problem.

4.3.4 Mutual Exclusion Constraints

Mutual exclusion (MX) constraints require certain projects not to be performed simultaneously in order to avoid severe traffic blockage. Generally, such constraints can be divided into three types: yard-related, junction-related and corridor-related. In yard-related MX constraints, a yard project and a non-yard project in the same subdivision should not be performed simultaneously, so that passing trains can stay in the yard when the non-yard project is being performed, or go through when the yard project is being performed. In junction-related MX constraints, no more than a certain number of subdivisions near the same junction should have simultaneous ongoing mainline projects, so that the traffic can use alternative routes when one subdivision is blocked. In corridor MX constraints, no more than a certain number of subdivisions in one corridor should have simultaneous ongoing single track projects, so that passing trains can wait at the endpoints of the blocked subdivision and pass whenever the team is not working. If MX constraints are violated, traffic will be significantly delayed and this is generally unacceptable to the railroads.

Although the MX constraints may be caused by various reasons, they can be formulated in a similar way. We first define $P_{\mathrm{MX}} \subseteq P$ as a subset of projects that belong to the same network element (e.g., subdivisions) over which the MX constraints are defined. We say that "project set" P_{MX} is "curfewed" in week $w \in W$ if at least one project in P_{MX} is being performed in that week. We further define F_{MX} as a set of P_{MX} that are mutually exclusive to each other (i.e., the maximum allowed number of simultaneous curfewed P_{MX} in F_{MX} should not exceed a certain value, $m_{\mathrm{MX},F_{\mathrm{MX}}}$). Then every MX constraint can be represented by an F_{MX}. Usually $m_{\mathrm{MX},F_{\mathrm{MX}}} = 1$, but in some cases $m_{\mathrm{MX},F_{\mathrm{MX}}}$ can be great than one.

In practice, P_{MX} may be in very different forms for different MX constraints. For a yard MX constraint, P_{MX} may contain a single unsplit project (in this case, P_{MX} is a singleton) or all parts of a split project. For a junction MX constraint, P_{MX} is defined for a subdivision and contains all mainline projects in that subdivision. For a corridor MX constraint, P_{MX} is defined for a subdivision and contains all single track projects in that subdivision.

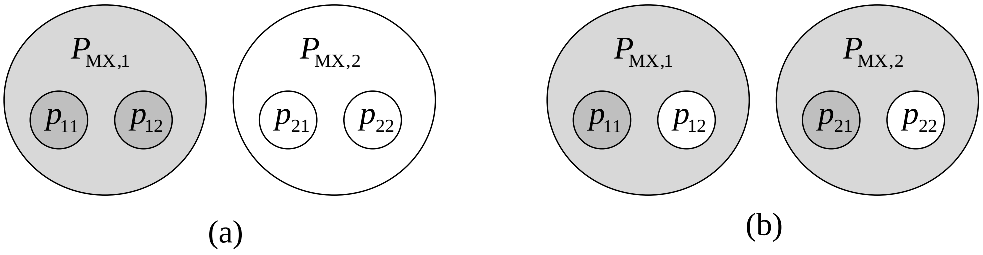


Figure 16. An example of a mutual exclusion constraint.

Figure 16 further illustrates an example of an MX constraint in a certain week. This MX constraint requires that the two projects sets from set $F_{\mathrm{MX}} = \left\{ P_{\mathrm{MX},1}, P_{\mathrm{MX},2} \right\}$ cannot be curfewed at the same time (i.e., $m_{\mathrm{MX},F_{\mathrm{MX}}} = 1$), where each P_{MX} contains two projects (i.e., $P_{\mathrm{MX},1} = \left\{ p_{11}, p_{12} \right\}$, $P_{\mathrm{MX},2} = \left\{ p_{21}, p_{22} \right\}$). Figure 16(a) shows a snapshot of a schedule that does not violate the MX constraint. In this schedule, both p_{11} and p_{12} are being performed simultaneously in some week (highlighted with gray color), and hence project set $P_{\mathrm{MX},1}$ is curfewed (also highlighted with gray color). However, because project set $P_{\mathrm{MX},2}$ is not curfewed, the total number of curfewed project sets is 1, and the MX constraint is not violated. Figure 16(b), on the other hand, shows a schedule which violates the MX constraint. Both p_{11} and p_{21} are performed in that week, and they respectively make project sets $P_{\mathrm{MX},1}$ and $P_{\mathrm{MX},2}$ curfewed. So the total number of curfewed project sets is 2, and the MX constraint is violated.

An MX constraint may be violated multiple times. Let $\upsilon_{\mathrm{MX},wF_{\mathrm{MX}}}$ be the exceedance of the total number of curfewed project sets over $m_{\mathrm{MX},F_{\mathrm{MX}}}$ in week $w \in W$. In Figure 16(b), $\upsilon_{\mathrm{MX},wF_{\mathrm{MX}}} = 2 - m_{\mathrm{MX},F_{\mathrm{MX}}} = 1$. If the MX constraint is violated in more than one week, the number of violations to that constraint will be the summation of $\upsilon_{\mathrm{MX},wF_{\mathrm{MX}}}$ over all weeks. We shall note too that even in one week an MX constraint may be violated multiple times. For example, if there were one more curfewed project set in Figure 16(b), $\upsilon_{\mathrm{MX},wF_{\mathrm{MX}}}$ would be 2.

The MX constraints can be formulated as follows:

$$\mathrm{C}_{\mathrm{MX}}(\mathbf{y}) = \sum_{\forall \text{soft } F_{\mathrm{MX}}} c_{\mathrm{MX},F_{\mathrm{MX}}} \sum_{\forall w \in W} \upsilon_{\mathrm{MX},wF_{\mathrm{MX}}} \,, \tag{4.14}$$

$$\upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}} \begin{cases} \geq y'_{pkw} & \text{if } |P_{\mathrm{MX}}| \geq 2 \\ = y'_{pkw} & \text{if } |P_{\mathrm{MX}}| = 1 \end{cases} , \qquad \forall k \in K_p, w \in W, p \in P_{\mathrm{MX}} \text{, for all } P_{\mathrm{MX}} \,, \tag{4.15}$$

$$\sum_{P_{\mathrm{MX}} \in F_{\mathrm{MX}}} \upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}} \leq m_{\mathrm{MX},F_{\mathrm{MX}}} + \upsilon_{\mathrm{MX},wF_{\mathrm{MX}}} \,, \quad \forall w \in W \text{, for all soft } F_{\mathrm{MX}} \,, \tag{4.16}$$

$$\sum_{P_{\mathrm{MX}} \in F_{\mathrm{MX}}} \upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}} \leq m_{\mathrm{MX},F_{\mathrm{MX}}} \,, \qquad \forall w \in W \text{, for all hard } F_{\mathrm{MX}} \,, \tag{4.17}$$

$$\upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}} \geq 0 \,, \qquad \text{for all } \upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}} \,, \tag{4.18}$$

$$\upsilon_{\mathrm{MX},wF_{\mathrm{MX}}} \geq 0 \,, \qquad \text{for all } \upsilon_{\mathrm{MX},wF_{\mathrm{MX}}} \,, \tag{4.19}$$

where $\mathrm{C}_{\mathrm{MX}}(\cdot)$ is the total MX constraint penalty costs, $c_{\mathrm{MX},F_{\mathrm{MX}}}$ is the unit penalty cost (i.e., penalty cost per violation) of soft constraint F_{MX}, and $\upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}}$ is the auxiliary variable which equals 1 if project set P_{MX} is curfewed in week w, or 0 otherwise.

Equation (4.14) calculates $\mathrm{C}_{\mathrm{MX}}(\cdot)$ as the summation of the penalty costs caused by all soft MX constraints. Constraints (4.15) create the relationship between projects and project sets, i.e., a project set is curfewed in a certain week if and only if at least one of its projects is performed in that week. Constraints (4.16) calculate $\upsilon_{\mathrm{MX},wF_{\mathrm{MX}}}$ for a soft MX constraint. Constraints (4.17) require the number of curfewed project sets in a hard MX constraint must be no larger than $m_{\mathrm{MX},F_{\mathrm{MX}}}$ in any week. Constraints (4.18) and (4.19) respectively define non-negative variables $\upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}}$ and $\upsilon_{\mathrm{MX},wF_{\mathrm{MX}}}$.

It can be seen that P_{MX} does not have subscript F_{MX}, because a project set may be involved in multiple MX constraints. In constraints (4.15), "$=$" instead of "$\leq$" is used if a project set is a singleton. In that way, auxiliary variable $\upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}}$ is added to the model only if needed (i.e., a project set contains more than one project), otherwise it is replaced by y'_{pkw}. Because the number of singleton P_{MX} in a problem instance is usually large, this treatment can eliminate many unnecessary variables and constraints.

4.3.5 Precedence Constraints

The technical nature of certain projects requires them to be performed in a certain sequence. For example, in a rail project, the rail team has to pull all spikes out of the ties, align or replace rails, and then nail the spikes back. If a T&S team has already laid new ties under the rails, the rail project will damage the new ties and shorten their lifetime. Hence, if a rail project and a T&S project are performed on a same track section, the rail project should be performed first.

Precedence constraints are used to ensure that all projects are performed in the right order. Let $< p,q >_{\text{Prec}}$ represent a precedence constraint which requires project $p \in P$ to start at least $t_{\text{Prec},pq}$ weeks before project $q \in P$ starts. Then the precedence constraints can be formulated as follows:

$$\mathrm{C}_{\text{Prec}}(\mathbf{y}) = \sum_{\forall \text{soft } <p,q>_{\text{Prec}}} c_{\text{Prec},pq} \upsilon_{\text{Prec},pq} \ , \tag{4.20}$$

$$\sum_{k \in K_p} y_{pkw} \le \sum_{w'=w+t_{\text{Prec},pq}}^{|W|} \sum_{k \in K_q} y_{qkw'} + \upsilon_{\text{Prec},pq} \,, \qquad \forall w \in W \text{, for all soft } < p,q >_{\text{Prec}} , \tag{4.21}$$

$$\sum_{k \in K_p} y_{pkw} \le \sum_{w'=w+t_{\text{Prec},pq}}^{|W|} \sum_{k \in K_q} y_{qkw'} \ , \qquad \forall w \in W \text{, for all hard } < p,q >_{\text{Prec}} , \tag{4.22}$$

where $\mathrm{C}_{\text{Prec}}(\cdot)$ is the total precedence constraint penalty cost, $c_{\text{Prec},pq}$ is the penalty cost for soft constraint $< p,q >_{\text{Prec}}$, and $\upsilon_{\text{Prec},pq}$ indicates whether a soft precedence constraint $< p,q >_{\text{Prec}}$ is violated. Equation (4.20) calculates $\mathrm{C}_{\text{Prec}}(\cdot)$ as the summation of the penalty costs caused by all soft precedence constraints. Constraints (4.21) and (4.22) respectively define soft and hard precedence constraints.

4.3.6 Simultaneity and Non-Simultaneity Constraints

Simultaneity constraints require that two projects be performed simultaneously by two teams in order to reduce technical difficulty or the impact on traffic operations. We define “overlapping ratio” as the overlapping duration of two projects divided by their maximum possible overlapping duration, for some given team assignments. In a hard simultaneity constraint, the overlapping ratio must be 1. In a soft simultaneity constraint, the penalty cost is proportional to the difference between 1 and the overlapping ratio, i.e., the penalty cost increases as the overlapping duration of two projects decreases. Suppose two projects in a simultaneity constraint are assigned to two teams and have durations of t and t' respectively. If their overlapping duration is $\min\{t,t'\}$, which is the maximal value possible, then the overlapping ratio is 1 and the penalty cost is zero.

Let T_p be the set of all possible durations that $p \in P$ could have, and $K_{\mathrm{Prod},pt} \subseteq K_p$ be the set of teams which can perform project $p \in P$ within exactly t weeks. We use a pair of projects $< p,q >_{\mathrm{S}}$ to denote a simultaneity constraint. Suppose project $p \in P$ is performed by some team with duration $t \in T_p$, and project $q \in P$ is performed by some team with duration $t' \in T_q$. Define auxiliary variable $\upsilon'_{\mathrm{S},pqtt'w}$ so that $\upsilon'_{\mathrm{S},pqtt'w} = 1$ if either of the two projects is performed in week $w \in W$, or 0 otherwise. Then $\sum_{w\in W} \upsilon'_{\mathrm{S},pqtt'w}$ is the number of weeks when at least one of the projects is performed. The value of $\sum_{w\in W} \upsilon'_{\mathrm{S},pqtt'w}$ can be at most $\left(\max\{t,t'\} - \min\{t,t'\}\right)$, when the overlapping ratio is 0, and at least $\max\{t,t'\}$, when the overlapping ratio is 1. Hence the overlapping ratio can be calculated as $\sum_{t\in T_p}\sum_{t'\in T_q} \frac{\sum_{w\in W} \upsilon'_{\mathrm{S},pqtt'w} - \max\{t,t'\}}{\min\{t,t'\}}$. Define auxiliary variables $\upsilon_{\mathrm{S},pq}$ = (1 – overlapping ratio) to indicate the number of violations to soft constraint $< p,q >_{\mathrm{S}}$.

Simultaneity constraints can be formulated as follows:

$$\mathrm{C_S}(\mathbf{y}) = \sum_{\forall \mathrm{soft} <p,q>_{\mathrm{S}}} c_{\mathrm{S},pq} \upsilon_{\mathrm{S},pq}\,, \tag{4.23}$$

$$\upsilon_{\mathrm{S},pq} \ge \sum_{t\in T_p}\sum_{t'\in T_q} \frac{\sum_{w\in W}\upsilon'_{\mathrm{S},pqtt'w} - \max\left\{t,t'\right\}}{\min\left\{t,t'\right\}}, \quad \text{for all soft } <p,q>_{\mathrm{S}}, \tag{4.24}$$

$$\sum_{w\in W}\upsilon'_{\mathrm{S},pqtt'w} \le \max\left\{t,t'\right\}, \quad \forall t\in T_p, t'\in T_q, \text{ for all hard } <p,q>_{\mathrm{S}}, \tag{4.25}$$

$$\upsilon'_{\mathrm{S},pqtt'w} \ge \sum_{k\in K_{\mathrm{Prod},pt}} y'_{pkw}, \quad \forall t\in T_p, t'\in T_q, w\in W, \text{ for all } <p,q>_{\mathrm{S}}, \tag{4.26}$$

$$\upsilon'_{\mathrm{S},pqtt'w} \ge \sum_{k\in K_{\mathrm{Prod},qt'}} y'_{pkw}, \quad \forall t\in T_p, t'\in T_q, w\in W, \text{ for all } <p,q>_{\mathrm{S}}, \tag{4.27}$$

$$\upsilon_{\mathrm{S},pq} \ge \sum_{w\in W}(y_{pkw} + y_{qkw}) - 1, \quad \forall k\in K_p \cap K_q, \text{ for all soft } <p,q>_{\mathrm{S}}, \tag{4.28}$$

$$\sum_{w\in W}(y_{pkw} + y_{qkw}) \le 1, \quad \forall k\in K_p \cap K_q, \text{ for all hard } <p,q>_{\mathrm{S}}, \tag{4.29}$$

where $\mathrm{C}_{\mathrm{S}}(\cdot)$ is the total simultaneity constraint penalty costs and $c_{\mathrm{S},pq}$ is the unit penalty cost of soft constraint $<p,q>_{\mathrm{S}}$. Equation (4.23) calculates $\mathrm{C}_{\mathrm{S}}(\cdot)$ as the summation of all soft simultaneity constraint penalties. Constraints (4.24) and (4.25) respectively define soft and hard constraints. Constraints (4.26) and (4.27) define auxiliary variables. Constraints (4.28) and (4.29) are cuts, which are redundant but can help tighten the model formulation. Note that $\upsilon_{\mathrm{S},pq}$ can be a fraction between 0 and 1, which incurs a partial penalty cost.

Non-simultaneity constraints, on the contrary, require two projects not to be performed simultaneously. One possible reason is that these two projects are in the same subdivision and that subdivision cannot accommodate both of them due to limited space or tight train schedules. Opposite to simultaneity constraints, the overlapping ratio must be 0 in a hard non-simultaneity constraint, and the penalty cost is proportional to the overlapping ratio in soft non-simultaneity constraints.

We use a pair of projects $< p, q >_{\text{NS}}$ to denote a non-simultaneity constraint. Suppose project $p \in P$ is performed by some team with duration $t \in T_p$, and project $q \in P$ is performed by some time with duration $t' \in T_q$. Define auxiliary variable $\upsilon'_{\text{NS},pqtt'w}$ so that $\upsilon'_{\text{NS},pqtt'w} = 1$ if both projects are performed in week $w \in W$, or 0 otherwise. Then $\sum_{w \in W} \upsilon'_{\text{NS},pqtt'w}$ is the number of weeks when both projects are performed, and the overlapping ratio can be calculated as $\sum_{t \in T_p} \sum_{t' \in T_q} \frac{\sum_{w \in W} \upsilon'_{\text{NS},pqtt'w}}{\min\{t, t'\}}$.

We define auxiliary variable $\upsilon_{\text{NS},pq}$ to denote the overlapping ratio, i.e., the number of violations to soft constraint $< p, q >_{\text{NS}}$. The non-simultaneity constraints can be formulated as:

$$\mathrm{C}_{\text{NS}}(\mathbf{y}) = \sum_{\forall \text{soft} < p,q >_{\text{NS}}} c_{\text{NS},pq} \upsilon_{\text{NS},pq} , \tag{4.30}$$

$$\upsilon_{\text{NS},pq} \ge \sum_{t \in T_p} \sum_{t' \in T_q} \frac{\sum_{w \in W} \upsilon'_{\text{NS},pqtt'w}}{\min\{t, t'\}}, \quad \text{for all soft} < p, q >_{\text{NS}}, \tag{4.31}$$

$$\upsilon'_{\text{NS},pqtt'w} \ge \sum_{k \in K_{\text{Prod},pt}} y'_{pkw} + \sum_{k \in K_{\text{Prod},qt'}} y'_{qkw} - 1,$$

$$\forall t \in T_p, t' \in T_q, w \in W, \text{ for all soft} < p, q >_{\text{NS}}, \tag{4.32}$$

$$\sum_{k \in K_{\text{Prod},pt}} y'_{pkw} + \sum_{k \in K_{\text{Prod},qt'}} y'_{qkw} \le 1,$$

$$\forall t \in T_p, t' \in T_q, w \in W, \text{ for all hard} < p, q >_{\text{NS}}. \tag{4.33}$$

where $\mathrm{C}_{\text{NS}}(\cdot)$ is the total non-simultaneity constraint penalty costs and $c_{\text{NS},pq}$ is the unit penalty cost of soft constraint $< p, q >_{\text{NS}}$. Equation (4.30) calculates $\mathrm{C}_{\text{NS}}(\cdot)$ as the summation of all soft non-simultaneity constraint penalty costs. Constraints (4.31) and (4.33) respectively define soft and hard non-simultaneity constraints. Constraints (4.32) define auxiliary variables for soft non-simultaneity constraints. The number of violations $\upsilon_{\mathrm{NS},pq}$ to a non-simultaneity constraint can also be a fraction between 0 and 1.

Although both simultaneity and non-simultaneity constraints use the overlapping ratio to calculate the penalty costs, their overlapping ratios and the auxiliary variables are formulated in quite different ways. The reason is that simultaneity constraints maximize the overlapping ratios while non-simultaneity constraints minimize them. Therefore, different treatments are needed in the MIP model.

We note that a non-simultaneity constraint is similar to a special case of an MX constraint where $\left|F_{P_{\mathrm{MX}}}\right| = 2$, $\left|P_{\mathrm{MX}}\right| = 1$, $\forall P_{\mathrm{MX}} \in F_{P_{\mathrm{MX}}}$ and $m_{\mathrm{MX},F_{P_{\mathrm{MX}}}} = 1$. Indeed, their hard versions are the same. The only difference is that soft non-simultaneity constraints calculate the penalty costs based on the overlapping ratio and those soft MX constraints calculate the costs based on the overlapping duration. For a certain overlapping duration, the overlapping ratio may be different due to varying project durations across teams.

4.3.7 Consecution Constraints

Consecution constraints require multiple projects to be performed consecutively by the same team in order to improve efficiency. Although the duration of a project is always an integer number of weeks, the actual time needed to perform the tasks is often more or less than the integer number of weeks. Suppose there are two projects A and B with actual durations of 0.5 week and 1.5 weeks respectively. After rounding up, their integer durations are 1 and 2 weeks.

Hence it may appear that 1 + 2 = 3 weeks are needed to finish both projects. However, if they are spatially close to each other and are performed consecutively by the same team, the team may be able to move to the next project in the middle of the week and finishes both of them in 1.5 + 0.5 = 2 weeks. This saves one week. Hence, we use consecution constraints to specify such possible savings. It shall be noted that consecution constraints do not depend on the order of the projects. In the above example, the team can perform project A either before or after B.

Let P_{Con} denote a set of projects that are specified in a consecution constraint. We also use P_{Con} to represent a consecution constraint. Let $K_{P_{\text{Con}}} = \bigcup_{p\in P_{\text{Con}}} K_p$, i.e., the set of teams which can perform at least one project in P_{Con}. Let $t_{\text{Con},P_{\text{Con}}pqk} = \sum_{p'\in P_{\text{Con}}\cap P_k\setminus\{p\}\setminus\{q\}} t_{p'k}$, i.e., the total duration of all projects except p and q in P_{Con} for team $k \in K_{P_{\text{Con}}}$. The consecution constraints can be formulated as follows:

$$\mathrm{C}_{\text{Con}}(\mathbf{y}) = \sum_{\forall \text{soft } P_{\text{Con}}} c_{\text{Con},P_{\text{Con}}} \upsilon_{\text{Con},P_{\text{Con}}} , \tag{4.34}$$

$$\upsilon_{\text{Con},P_{\text{Con}}} \ge y_{pkw} - \sum_{w'=w-t_{qk}-t_{\text{Con},P_{\text{Con}}pqk}}^{w+t_{pk}+t_{\text{Con},P_{\text{Con}}pqk}} y_{qkw'} ,$$

$$\forall w \in W, k \in K_{P_{\text{Con}}} ,\ \forall p, q \in P_{\text{Con}}, p \ne q, \text{ for all soft } P_{\text{Con}} , \tag{4.35}$$

$$y_{pkw} \le \sum_{w'=w-t_{qk}-t_{\text{Con},P_{\text{Con}}pqk}}^{w+t_{pk}+t_{\text{Con},P_{\text{Con}}pqk}} y_{qkw'} ,$$

$$\forall w \in W, k \in K_{P_{\text{Con}}} ,\ \forall p, q \in P_{\text{Con}}, p \ne q, \text{ for all hard } P_{\text{Con}} , \tag{4.36}$$

$$\upsilon_{\text{Con},P_{\text{Con}}} \ge \sum_{w\in W} y_{pkw} - \sum_{w\in W} y_{qkw} , \qquad \forall k \in K_{P_{\text{Con}}} ,\ \forall p, q \in P_{\text{Con}}, p \ne q, \text{ for all soft } P_{\text{Con}} , \tag{4.37}$$

$$\sum_{w \in W} y_{pkw} = \sum_{w \in W} y_{qkw} \ , \qquad \forall k \in K_{P_{\text{Con}}} \ , \ \forall p, q \in P_{\text{Con}}, p \neq q \text{, for all hard } P_{\text{Con}} \ , \qquad (4.38)$$

where $\mathrm{C}_{\text{Con}}(\cdot)$ is the total consecution constraint penalty costs, $c_{\text{Con},P_{\text{Con}}}$ is the penalty cost of soft constraint P_{Con}, and $\upsilon_{\text{Con},P_{\text{Con}}}$ is the number of violations to soft constraint P_{Con}. Equation (4.34) calculates $\mathrm{C}_{\text{Con}}(\cdot)$ as the summation of all soft consecution constraint penalty costs. Constraints (4.35) and (4.36) require all projects in P_{Con} to be performed consecutively. Constraints (4.37) and (4.38) require all projects in P_{Con} to be performed by the same team.

In constraints (4.35) and (4.36), if a project $p \in P_{\text{Con}}$ is performed in week $w \in W$, any other project $q \in P_{\text{Con}}$ should be performed no earlier than week $w - t_{qk} - t_{\text{Con},P_{\text{Con}}pqk}$ and no later than week $w + t_{pk} + t_{\text{Con},P_{\text{Con}}pqk}$. This condition holds for all projects $p \in P_{\text{Con}}$ if and only if the projects in P_{Con} are performed consecutively. In consecution constraints, the number of violations to a constraint can only be 1 or 0. Note that we define $y_{pkw} = 0$ when $p \notin P_k$. That is, if a project is performed by a team which cannot perform all the projects in P_{Con}, the consecution constraint must be violated.

4.3.8 Split Project Constraints

In order to reduce the impact of track maintenance on train traffic operations, some projects are split into multiple parts and performed simultaneously by multiple teams (i.e., the project is "folded"). Such practice is helpful to reduce violations to MX constraints. However, sometimes it may incur higher travel cost or cause violations to other side constraints. For example, a project may be required by a preference constraint to be only performed by one team. If it is split and assigned to two teams, the preference constraint will be violated. Therefore, only a subset of projects should be selected to fold.

Sometimes a project must be folded. In that case, it is split and simultaneity constraints are applied to each pair of its parts. More often, the model needs to decide whether a split project (which is split into two parts) should be folded or not. If not, the parts of the split project should be performed consecutively by one team, which is equivalent to the case that the project is not split and is still performed as a whole.

Split project constraints require two parts of a split project to be either performed simultaneously by two teams, or consecutively by one team. Let $< p, q >_{\mathrm{SP}}$ represent a split project constraint, where $p \in P$ and $q \in P$ are the two parts of a split project. Because p and q are originally from one project, it can be seen that $K_p = K_q$. Also, p and q should not be part of any existing simultaneity constraint or consecution constraint. The split project constraints can be considered as a special case of a disjunction (i.e., logical “or”) of a new simultaneity constraint and a new consecution constraint, and they can be formulated as follows:

$$\mathrm{C}_{\mathrm{SP}}(\mathbf{y}) = \sum_{\forall \mathrm{soft}\, <p,q>_{\mathrm{SP}}} c_{\mathrm{SP},pq} \upsilon_{\mathrm{SP},pq} , \tag{4.39}$$

$$\upsilon_{\mathrm{Con},pq} \geq y_{pkw} - y_{qk(w+t_{pk})} , \qquad \forall w \in W, k \in K_p \text{, for all } < p, q >_{\mathrm{SP}} , \tag{4.40}$$

$$\upsilon_{\mathrm{Con},pq} \geq \sum_{w \in W} y_{pkw} - \sum_{w \in W} y_{qkw} , \qquad \forall k \in K_p \text{, for all } < p, q >_{\mathrm{SP}} , \tag{4.41}$$

$$\upsilon_{\mathrm{S},pq} \geq \sum_{t \in T_p} \sum_{t' \in T_q} \frac{\sum_{w \in W} \upsilon'_{\mathrm{S},pqtt'w} - \max\{t, t'\}}{\min\{t, t'\}} , \qquad \text{for all } < p, q >_{\mathrm{SP}} , \tag{4.42}$$

$$\upsilon'_{\mathrm{S},pqtt'w} \geq \sum_{k \in K_{\mathrm{Prod},pt}} y'_{pkw} , \qquad \forall t \in T_p, t' \in T_q, w \in W \text{, for all } < p, q >_{\mathrm{SP}} , \tag{4.43}$$

$$\upsilon'_{\mathrm{S},pqtt'w} \geq \sum_{k \in K_{\mathrm{Prod},qt'}} y'_{pkw}\,, \qquad \forall t \in T_p, t' \in T_q, w \in W\text{, for all } < p,q >_{\mathrm{SP}}\,, \tag{4.44}$$

$$\upsilon_{\mathrm{SP},pq} = \upsilon_{\mathrm{Con},pq} + \upsilon_{\mathrm{S},pq} - 1, \qquad \text{for all soft } < p,q >_{\mathrm{SP}}\,, \tag{4.45}$$

$$\upsilon_{\mathrm{Con},pq} + \upsilon_{\mathrm{S},pq} = 1\,, \qquad \text{for all hard } < p,q >_{\mathrm{SP}}\,, \tag{4.46}$$

$$\upsilon_{\mathrm{SP},pq} \geq 0\,, \qquad \text{for all soft } < p,q >_{\mathrm{SP}}\,, \tag{4.47}$$

where $\mathrm{C}_{\mathrm{SP}}(\cdot)$ is the total split project constraint penalty costs, $c_{\mathrm{SP},pq}$ is the penalty cost of soft constraint $< p,q >_{\mathrm{SP}}$, $\upsilon_{\mathrm{SP},pq}$ is the number of violations to soft constraint $< p,q >_{\mathrm{SP}}$, and $\upsilon_{\mathrm{Con},pq}$, $\upsilon_{\mathrm{S},pq}$, and $\upsilon'_{\mathrm{S},pqtt'w}$ are auxiliary variables. Equation (4.39) calculates $\mathrm{C}_{\mathrm{SP}}(\cdot)$ as the summation of all soft split project constraint penalty costs. Constraints (4.40) and (4.41) are similar to soft consecution constraints, requiring that $p \in P$ and $q \in P$ are performed consecutively by the same team, otherwise $\upsilon_{\mathrm{Con},pq}$ will equals 1. Constraints (4.42)-(4.44) are equivalent to soft simultaneity constraints, requiring that $p \in P$ and $q \in P$ are performed simultaneously by two teams, otherwise $\upsilon_{\mathrm{Con},pq}$ will be (1 - their overlapping ratio). Constraints (4.45) and (4.46) require one of $\upsilon_{\mathrm{Con},pq}$ and $\upsilon_{\mathrm{S},pq}$ to be 0, which means either the consecution or simultaneity constraint should be satisfied, respectively with the soft formulation and hard formulation. Constraints (4.47) are cuts, which are redundant but can help tighten the model.

Constraints (4.40) are a simplified version of consecution constraints (4.35). In split project constraints, the consecution constraint is a special case of general consecution constraints where $|P_{\mathrm{Con}}| = 2$. Furthermore, if the consecution constraint is satisfied, p and q can be considered as one unsplit project, and the sequence that they are performed is not important. So we can always require p to be performed before q in the consecution constraint.

4.3.9 Limit Constraints

Limit constraints limit the total duration of certain projects performed by each team. For example, most projects are 4-day projects (i.e., requiring a team to work 4 days a week) but some are 5-day projects (i.e., requiring a team to work 5 days a week) because of different train schedules at project sites. When performing a 5-day project, the crew only has 2 days of off-work time in a week. Such projects are unfavorable to the crew, and a team should not perform too many such projects in the scheduling horizon.

We use pair $< P_{\text{Lim}}, k >$ to represent a limit constraint, where team $k \in K$ should not perform more than $m_{\text{Lim},P_{\text{Lim}}k}$ weeks of projects from set $P_{\text{Lim}} \subseteq P$. Limit constraints can be formulated as follows:

$$\mathrm{C}_{\text{Lim}}(\mathbf{y}) = \sum_{\forall \text{soft } <P_{\text{Lim}},k>} c_{\text{Lim},P_{\text{Lim}}k} \upsilon_{\text{Lim},P_{\text{Lim}}k} \,, \tag{4.48}$$

$$\upsilon_{\text{Lim},P_{\text{Lim}}k} \geq \sum_{p \in P_{\text{Lim}}} \left(t_{pk} \sum_{w \in W} y_{pkw} \right) - m_{\text{Lim},P_{\text{Lim}}k} \,, \quad \text{for all soft } < P_{\text{Lim}}, k >, \tag{4.49}$$

$$\upsilon_{\text{Lim},P_{\text{Lim}}k} \geq 0 \,, \quad \text{for all soft } < P_{\text{Lim}}, k >, \tag{4.50}$$

$$\sum_{p \in P_{\text{Lim}}} \left(t_{pk} \sum_{w \in W} y_{pkw} \right) \leq m_{\text{Lim},P_{\text{Lim}}k} \,, \quad \text{for all hard } < P_{\text{Lim}}, k >, \tag{4.51}$$

where $\mathrm{C}_{\text{Lim}}(\cdot)$ is the total limit constraint penalty costs, $c_{\text{Lim},P_{\text{Lim}}k}$ is the penalty cost per week of soft constraint $< P_{\text{Lim}}, k >$, and $\upsilon_{\text{Lim},P_{\text{Lim}}k}$ is the number of violations to soft constraint $< P_{\text{Lim}}, k >$. Equation (4.48) calculates $\mathrm{C}_{\text{Lim}}(\cdot)$ as the summation of all soft limit constraint penalty costs.

Constraints (4.49) define $\upsilon_{\text{Lim},P_{\text{Lim}}k}$ as the exceedance of the total duration of projects in P_{Lim} performed by team $k \in K$ over $m_{\text{Lim},P_{\text{Lim}}k}$. Constraints (4.50) define non-negative variables $\upsilon_{\text{Lim},P_{\text{Lim}}k}$. Constraints (4.51) define hard limit constraints.

4.3.10 Relay Rail Constraints

In practice, "relay rail" is the recycled rail that is still in good condition for reuse. Using relay rail helps reduce the purchase of costly new rails. Relay rail is usually supplied from track lines with relatively heavy traffic and reused for lines with relatively light traffic (such as industrial and spur tracks). A rail project may either supply or demand relay rail. Relay rail constraints require the projects supplying relay rail to be scheduled before projects demanding relay rail for a certain amount of processing and transporting time, so that the demand can be met. Our current model does not address which supplying project serves which demanding project; this matching problem is a part of the relay rail sourcing problem and will be studied in the future.

There are different types of relay rails in terms of weight and size. Let P_{relay} be the set of projects that either supply or demand a certain type of relay rail. We also use P_{relay} to represent a relay rail constraint. Let r_p be the net output of relay rail from project $p \in P_{\text{relay}}$, which is positive for projects supplying relay rail, and negative for those demanding relay rail. Let $t_{\text{relay},p}$ be the number of weeks after the start of project $p \in P_{\text{relay}}$ when its supplied relay rail becomes available (usually a few months in practice) or its demanded relay rail needs to be transported to site (usually 0).

Sometimes relay rail is also supplied from other sources than projects, such as relay rail factories. Let $r_{\text{other},w}$ be the net output of relay rail from other sources in week $w \in W$. The net quantity of available relay rail in week $w \in W$ can be calculated as $\sum_{w'=1}^{w}\left(\sum_{p \in P_{\text{relay}}}\left(r_p \sum_{k \in K_p} y_{pk(w'-t_{\text{relay},p})}\right)+r_{\text{other},w'}\right)$. Define $\upsilon_{\text{relay},P_{\text{relay}}}$ as the total shortage of relay rail during the scheduling horizon, which equals the negative of the minimum net quantity of available relay rail among all weeks.

Relay rail constraints can be formulated as:

$$\text{C}_{\text{relay}}(\mathbf{y}) = \sum_{\forall \text{soft } P_{\text{relay}}} c_{\text{relay},P_{\text{relay}}} \upsilon_{\text{relay},P_{\text{relay}}} , \tag{4.52}$$

$$\upsilon_{\text{relay},P_{\text{relay}}} \geq -\sum_{w'=1}^{w}\left(\sum_{p \in P_{\text{relay}}}\left(r_p \sum_{k \in K_p} y_{pk(w'-t_{\text{relay},p})}\right)+r_{\text{other},w'}\right), \qquad \forall w \in W \text{, for all soft } P_{\text{relay}} , \tag{4.53}$$

$$\upsilon_{\text{relay},P_{\text{relay}}} \geq 0, \qquad \text{for all soft } P_{\text{relay}} , \tag{4.54}$$

$$\sum_{w'=1}^{w}\left(\sum_{p \in P_{\text{relay}}}\left(r_p \sum_{k \in K_p} y_{pk(w'-t_{\text{relay},p})}\right)+r_{\text{other},w'}\right) \geq 0, \qquad \forall w \in W \text{, for all hard } P_{\text{relay}} , \tag{4.55}$$

where $\text{C}_{\text{relay}}(\cdot)$ is the total relay rail constraint penalty costs and $c_{\text{relay},P_{\text{relay}}}$ is the penalty cost per unit of relay rail shortage of soft constraint P_{relay}. Equation (4.52) calculates $\text{C}_{\text{relay}}(\cdot)$ as the summation of all soft relay rail constraint penalty costs. Constraints (4.53)-(4.55) defines soft and hard relay rail constraints.

4.3.11 “Blitz” Constraints

The “blitz” constraints involve a set of special projects (blitz projects) which can only be performed during certain consecutive weeks (blitz weeks) in a year. During blitz weeks, many teams (blitz teams) need to travel to blitz projects and work intensively on them (Burns and Franke, 2005). Sometimes a team has to interrupt its current project to work on blitz, and continue that project after coming back from blitz. In the studied PTSP, blitz projects are in subdivisions close to coal mines and are performed during coal mine summer vacation. In railroad business, coal traffic cannot be interrupted, and hence no maintenance projects shall be performed in coal mine subdivisions when coal mines are in operation. However, coal mines typically have a two-week vacation in the summer, providing the railroads with the only possible time window to conduct maintenance on those subdivisions.

In order to implement such practice, the model forces all blitz teams to move to the vertex representing blitz location during blitz weeks. For those projects interrupted by blitz, the model extends their durations t_{pk} by two weeks during blitz weeks. All constraints related to project durations should be modified during blitz weeks accordingly. For example, formula (4.36) should be replaced by the following formulas in hard consecution constraints:

$$y_{pkw} \le \begin{cases} \displaystyle\sum_{w'=w-t_{qk}-t_{\mathrm{Con},P_{\mathrm{Con}}pqk}}^{w+t_{pk}+t_{\mathrm{Con},P_{\mathrm{Con}}pqk}} y_{qkw'}, & w < \min W_{\mathrm{B}} - t_{pk} - t_{\mathrm{Con},P_{\mathrm{Con}}pqk} \\ & \text{or } w > \max W_{\mathrm{B}} + t_{qk} + t_{\mathrm{Con},P_{\mathrm{Con}}pqk}, \\ \displaystyle\sum_{w'=w-t_{qk}-t_{\mathrm{Con},P_{\mathrm{Con}}pqk}}^{w+t_{pk}+t_{\mathrm{Con},P_{\mathrm{Con}}pqk}+|W_{\mathrm{B}}|} y_{qkw'}, & \min W_{\mathrm{B}} - t_{pk} - t_{\mathrm{Con},P_{\mathrm{Con}}pqk} \le w < \min W_{\mathrm{B}}, \\ \displaystyle\sum_{w'=w-t_{qk}-t_{\mathrm{Con},P_{\mathrm{Con}}pqk}-|W_{\mathrm{B}}|}^{w+t_{pk}+t_{\mathrm{Con},P_{\mathrm{Con}}pqk}} y_{qkw'}, & \max W_{\mathrm{B}} < w \le \max W_{\mathrm{B}} + t_{qk} + t_{\mathrm{Con},P_{\mathrm{Con}}pqk}, \end{cases}$$

$$\forall w \in W \setminus W_{\mathrm{B}}, k \in K_{P_{\mathrm{Con}}} \cap K_{\mathrm{B}},\ \forall p,q \in P_{\mathrm{Con}}, p \ne q, \text{ for all hard } P_{\mathrm{Con}}, \quad (4.56)$$

$$y_{pkw} \le \sum_{w'=w-t_{qk}-t_{\mathrm{Con},P_{\mathrm{Con}}pqk}}^{w+t_{pk}+t_{\mathrm{Con},P_{\mathrm{Con}}pqk}} y_{qkw'} ,$$

$$\forall w \in W, k \in K_{P_{\mathrm{Con}}} \setminus K_{\mathrm{B}},\ \forall p, q \in P_{\mathrm{Con}}, p \ne q, \text{ for all hard } P_{\mathrm{Con}}, \quad (4.57)$$

where W_{B} is the set of blitz weeks and K_{B} is the set of blitz teams. All other duration-related constraints must be treated for the blitz weeks in a similar fashion. Note that blitz projects have fixed schedules and are not included in P .

4.4 Algorithm

Again, large-scale routing and scheduling problems with many side constraints is well known to be computationally intractable. Basic MIP algorithms such as linear relaxation and branch and bound generally incur significant difficulty solving such problems. Therefore, we propose a heuristic algorithm (combined with MIP algorithms) to solve PTSP.

The flowchart in Figure 17 illustrates the proposed algorithm framework. First, projects with high probabilities of violating MX constraints are identified and split. Then projects (after splitting) are tentatively scheduled by solving a scheduling model. Finally PTSP is solved iteratively with two types of local search algorithms: (i) decomposition and restriction with MIP algorithm, and (ii) block interchange. The solution process terminates when some stopping criteria are met (e.g., a predetermined maximum solution time is reached).

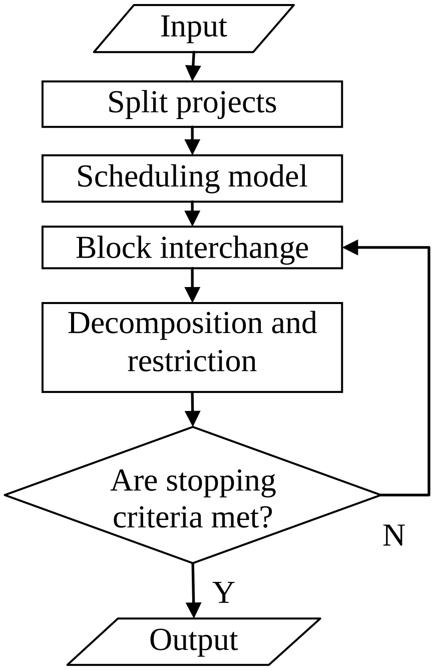


Figure 17. Algorithm framework of production team scheduling.

The key feature of the proposed algorithm is the iterative application of two types of local search algorithms. While each of the two local search algorithms may get stuck in local optima, the local optimum of one neighborhood structure may not be that of the other one. By exploring different neighborhood structures, the two local search algorithms can effectively improve the solution by helping each other leave local optima.

This section also introduces some other auxiliary algorithms that are developed to reduce the input data of the model. The reduced data keeps the problem instance unchanged but reduces the number of variables and constraints.

4.4.1 Project Splitting

As explained before, some projects are split or folded in order to reduce the violations to side constraints. Some projects are required to be split or folded by the railroad. In addition, the algorithm needs to determine by itself which other projects should be folded in order to satisfy MX constraints. Generally, the algorithm is only allowed to split a project into two parts. If the project split by the algorithm is not folded, its two parts must be performed consecutively by one team.

The number of projects to split is determined in an ad hoc way. Generally, there is a tradeoff between travel costs and penalty costs for the number of split projects: splitting more projects usually increases travel costs and decreases penalty costs, and vice versa. This is intuitive: splitting projects can reduce the violations to side constraints, but teams have to travel more distance because the total number of projects is increased after splitting. Also, splitting more projects would reduce the solution speed, because more variables and constraints are required.

The following strategy is used to prioritize the projects for splitting. For a project $p \in P$, compute its expected duration $\bar{t}_p = \frac{1}{|K_p|} \sum_{k \in K_p} t_{pk}$ assuming it has equal probability to be assigned to any team $k \in K_p$. For each project set P_{MX} for MX constraints, compute its minimum curfewed duration $t_{\text{O},P_{\text{MX}}}$ as $\max_{p \in P_{\text{MX}}} \bar{t}_p$ (this duration is achieved when all projects in P_{MX} are performed simultaneously). Because $t_{\text{O},P_{\text{MX}}}$ is usually far smaller than $|W|$, assume that the curfewed duration of P_{MX} may start from any week of the scheduling horizon with equal probabilities. Then for any two project set P_{MX} and P'_{MX} in a MX constraint F_{MX} with

$m_{\mathrm{MX},F_{\mathrm{MX}}}=1$ (most MX constraints have $m_{\mathrm{MX},F_{\mathrm{MX}}}=1$), the expected penalty cost (a very high cost can be used for hard MX constraints) incurred by P_{MX} and P'_{MX} can be estimated as

$$\bar{c}_{\mathrm{MX},F_{\mathrm{MX}}P_{\mathrm{MX}}P'_{\mathrm{MX}}}=c_{\mathrm{MX},F_{\mathrm{MX}}}t_{\mathrm{O},P_{\mathrm{MX}}}t_{\mathrm{O},P'_{\mathrm{MX}}}/|W|.$$

The total expected penalty cost incurred by P_{MX} from MX constraint F_{MX} can be estimated as

$$\bar{c}_{\mathrm{MX},F_{\mathrm{MX}}P_{\mathrm{MX}}}=\sum_{P'_{\mathrm{MX}}\in F_{\mathrm{MX}}\setminus\{P_{\mathrm{MX}}\}}\bar{c}_{\mathrm{MX},F_{\mathrm{MX}}P_{\mathrm{MX}}P'_{\mathrm{MX}}}=c_{\mathrm{MX},F_{\mathrm{MX}}}t_{\mathrm{O},P_{\mathrm{MX}}}\sum_{P'_{\mathrm{MX}}\in F_{\mathrm{MX}}\setminus\{P_{\mathrm{MX}}\}}t_{\mathrm{O},P'_{\mathrm{MX}}}/|W|.$$

For a project $p\in P_{\mathrm{MX}}$, the expected penalty costs incurred by it from F_{MX} can be estimated as

$$\bar{c}_{\mathrm{MX},F_{\mathrm{MX}}P_{\mathrm{MX}}p}=\frac{\bar{t}_p}{t_{\mathrm{O},P_{\mathrm{MX}}}}\bar{c}_{\mathrm{MX},F_{\mathrm{MX}}P_{\mathrm{MX}}}=c_{\mathrm{MX},F_{\mathrm{MX}}}\bar{t}_p\sum_{P'_{\mathrm{MX}}\in F_{\mathrm{MX}}\setminus\{P_{\mathrm{MX}}\}}t_{\mathrm{O},P'_{\mathrm{MX}}}/|W|.$$

If p is involved in multiple MX constraints, $\bar{c}_{\mathrm{MX},p}$, the total expected MX constraint penalty costs incurred by it, can estimated as the summation of all $\bar{c}_{\mathrm{MX},F_{\mathrm{MX}}P_{\mathrm{MX}}p}$:

$$\begin{aligned}\bar{c}_{\mathrm{MX},p}&=\sum_{P_{\mathrm{MX}}:p\in P_{\mathrm{MX}}}\ \sum_{F_{\mathrm{MX}}:P_{\mathrm{MX}}\in F_{\mathrm{MX}},m_{\mathrm{MX},F_{\mathrm{MX}}}=1}\bar{c}_{\mathrm{MX},F_{\mathrm{MX}}P_{\mathrm{MX}}p}\\&=\frac{\bar{t}_p}{|W|}\sum_{P_{\mathrm{MX}}:p\in P_{\mathrm{MX}}}\ \sum_{F_{\mathrm{MX}}:P_{\mathrm{MX}}\in F_{\mathrm{MX}},m_{\mathrm{MX},F_{\mathrm{MX}}}=1}\left(c_{\mathrm{MX},F_{\mathrm{MX}}}\sum_{P'_{\mathrm{MX}}\in F_{\mathrm{MX}}\setminus\{P_{\mathrm{MX}}\}}t_{\mathrm{O},P'_{\mathrm{MX}}}\right).\end{aligned}$$

Excluding the unsplittable projects (e.g., projects with the duration of one week and those required by the company not to be split) and the split projects (i.e., projects which have already been split), the project with the highest $\bar{c}_{\mathrm{MX},p}$ is selected to split. If it can be performed by teams of different productivities, it will be temporarily assigned to the team with the highest productivity in order to reduce its duration; otherwise it will be split into two parts. Either case, $\bar{c}_{\mathrm{MX},p}$ of all projects should be recalculated before the next project is selected to split.

4.4.2 Scheduling Model

Our scheduling model is a relaxed variant of PTSP that we use to obtain an initial solution for local search algorithms. In PTSP, it can be seen that $\mathbf{x}$ only appears in travel costs but not in side constraints. If travel costs are ignored, $\mathbf{x}$ can be eliminated and PTSP becomes a pure scheduling problem without the spatial dimension. Define sets $W'_{\text{start},k} = \left\{1, 2\cdots, \left|W_{\text{start},k}\right| - 1\right\}$ and $W'_{\text{end},k} = \left\{|W| - \left|W_{\text{end},k}\right| + 2, \cdots, |W|\right\}$. The simplified core model (without side constraints) can be formulated as follows:

$$\text{(PTSP2)} \quad \min \sum_{\text{all cost items}} C_{\text{cost item}}(\mathbf{y}), \tag{4.58}$$

s.t.

$$\sum_{p \in P_k} y_{pkw} \le \sum_{p \in P_k} y_{pk(w+1)}, \qquad \forall k \in K, w \in W'_{\text{start}}, \tag{4.59}$$

$$\sum_{p \in P_k} y_{pkw} = 1, \qquad \forall k \in K, w \in W \setminus W'_{\text{start},k} \setminus W'_{\text{end},k}, \tag{4.60}$$

$$\sum_{p \in P_k} y_{pkw} \ge \sum_{p \in P_k} y_{pk(w-1)}, \qquad \forall k \in K, w \in W'_{\text{end}}, \tag{4.61}$$

(4.5)-(4.7).

PTSP2 is equivalent to PTSP with $c_{\text{travel},v_1v_2k} = 0, \forall v_1, v_2 \in V, k \in K$. Because the elimination of $\mathbf{x}$, PTSP2 has much fewer variables than PTSP. All side constraints and costs except travel costs (4.8) can be included in PTSP2, because they are only related to $\mathbf{y}$. By solving PTSP2 with some of the side constraints, an initial solution can be obtained.

In practice, PTSP2 is solved using commercial MIP solver. We reduce the time needed to find an initial solution, by considering only time window and preference constraints in PTSP2, since these constraints do not add any new variables or constraints to the model. If other side constraints (such as MX constraints) were added, PTSP2 generally cannot be solved within a reasonable time., In our experiments, the exact optimum solution for PTSP2 can usually be obtained within 5 minutes for real-world problem instances (as discussed in more detail in Section 4.5).

4.4.3 Decomposition and Restriction

In practice, PTSP is often very large-scale and difficult to solve. However, the original problem may be decomposed into restricted subproblems, which can be solved separately using MIP algorithms. In our work, a team-wise decomposition scheme is used. Each subproblem only contains the decision variables related to a small subset of the teams and the projects assigned to these teams (in the current solution). Other decision variables are temporarily fixed. In this sense, every subproblem is a restriction version of the original problem. As such, a subproblem has much fewer variables and constraints and can be solved relatively quickly with MIP algorithms. By iteratively formulating and solving such subproblems, the overall solution is continuously improved. To ensure a fast solution speed, we restrict the maximum number of teams in a subproblem to two.

This decomposition and restriction (D&R) method is essentially a local search algorithm where a move is defined as the update of decision variables related to a subproblem. For a given problem, we can define $|K|$ subproblems each containing one team, and $|K|(|K|-1)/2$ subproblems each containing two teams. Hence, the size of the neighborhood is

$|K|+|K|(|K|-1)/2=|K|(|K|+1)/2$, which is not a very large number. However, it is still impractical to exhaustively enumerate all possible moves, because testing a move (i.e., solving a subproblem) is relatively time-consuming compared to ordinary local search algorithms (to ensure solution speed and quality, the solution time limit for one subproblem is usually set to be a few minutes). Therefore, we use non-exhaustive search which allows the algorithm to terminate before all possible moves are enumerated.

Local search algorithm requires the objective value of a given solution to be always quantifiable. Therefore, all hard side constraints are relaxed into soft constraints (but with very high penalty costs) at the beginning of the algorithm, while they are set back to hard constraints (which often reduces the number of variables) once they become satisfied during the algorithm iterations.

A few approaches are applied to help improve the efficiency of the D&R search. They include (i) adding cuts to the MIP model, (ii) adding augmented cost function to the MIP model, (iii) tightening the MIP model for subproblems, and (iv) using a heuristic strategy to prioritize subproblems in local search.

When solving a subproblem with the MIP algorithm, linear programming (LP) relaxation, is applied to the MIP model. In LP relaxation, integer constraints such as (4.7) are relaxed. Cuts are the constraints redundant in the MIP model but not in the LP relaxation. For example, cuts (4.47) are added to the split project constraints, which require $\upsilon_{\mathrm{SP},pq}$ to be always non-negative. Clearly, it always holds in a feasible solution: projects $p \in P$ and $q \in P$ can never satisfy both simultaneity constraint and consecution constraint, hence $\upsilon_{\mathrm{Con},pq}$ and $\upsilon_{\mathrm{S},pq}$ cannot be both zero. However, $\upsilon_{\mathrm{SP},pq}$ may be negative in an LP relaxation solution. Such cuts can help reduce the integrality gap of LP relaxation (i.e., the minimum known upper bound of the difference between the optimum objective values of the MIP model and its LP relaxation), and hence improve the efficiency of MIP algorithms. Constraints (4.28) and (4.29) are also cuts, which are added to the simultaneity constraints. They enforce a simultaneity constraint to be fully violated (i.e. overlapping ratio equals 0) when two projects are performed by the same team.

Augmented cost function is used to penalize some features of the local optimal solution and help the algorithm leave a local optimum. For example, in a hard MX constraint $F_{P_{\text{MX}}}$, if there are more than one project set P_{MX} containing more than one project, $F_{P_{\text{MX}}}$ is called a difficult MX constraint, in the sense that it is difficult to be satisfied in D&R search. Still use the example in Figure 16. Suppose in the current solution the four projects are assigned to four different teams and have some common overlapping weeks. In order to satisfy this MX constraint, the only possible way in D&R search is to move both p_{11} and p_{12} out of the overlapping weeks, or to move both p_{21} and p_{22} out. Such move can only be achieved by solving the subproblem containing both p_{11} and p_{12}, or that containing both p_{21} and p_{22}. Because there are totally $|K|(|K|+1)/2$ subproblems, those two subproblems may not be solved frequently, or even solved during the algorithm. Even if they are solved, the algorithm may not be able to reschedule both p_{11} and p_{12} or both p_{21} and p_{22} at one time because of other side constraints. In order to address this problem, a penalty is added to each of the four projects when they are overlapping. As such, the algorithm is able to move them out one by one. Another example is consecution constraints, where an additional penalty is added if two projects within P_{Con} is not performed by the same team. In this way, the algorithm tends to first assign them to the same team, and then schedule them to be consecutively performed.

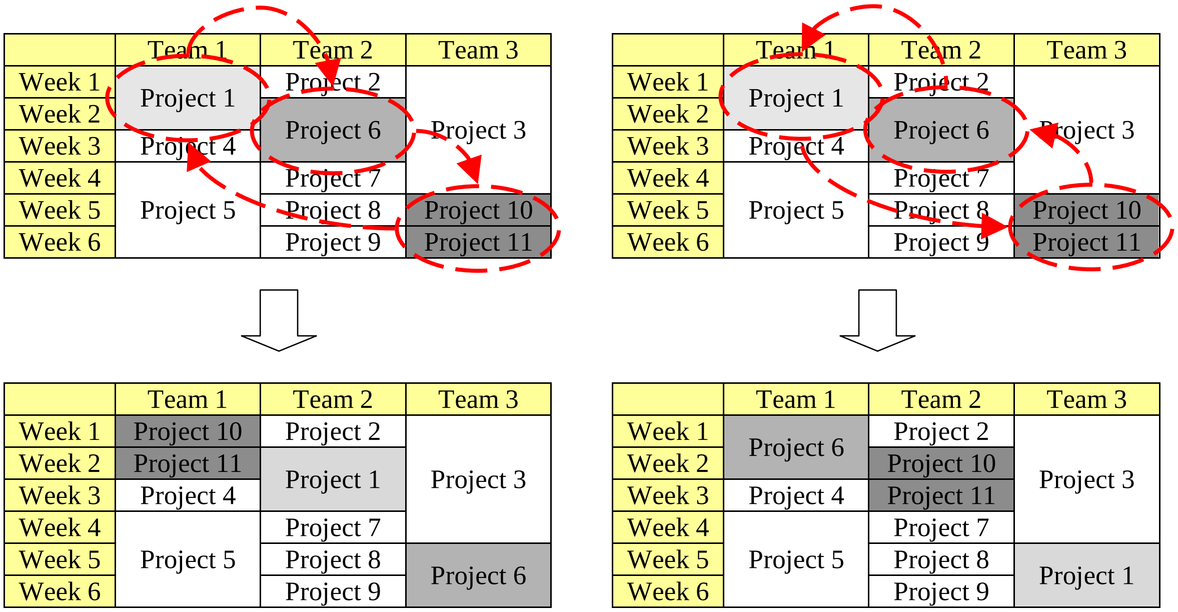
The proposed MIP model can be directly applied to the subproblems. However, because many variables are fixed in the restricted subproblems, the model for subproblems can be further tightened in order to improve the solution speed. For example, suppose there are two projects in a consecution constraint. If only one project is in the subproblem, the consecution constraint must be violated, because it is impossible to assign both projects to one team by solving this subproblem. Hence a fixed penalty cost can be added to the objective function and this consecution constraint can be eliminated from the subproblem. In another example, suppose p_1 and p_2 are two projects in a project set P_{MX} and only p_1 is in the subproblem. Then we can define $\upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}} = 1$ for the weeks when p_2 is performed, and $\upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}} = y'_{p_1kw}$ for the weeks when p_2 is not performed. In this way, $\upsilon'_{\mathrm{MX},wP_{\mathrm{MX}}}$ can be represented by existing variables and no additional variable is needed.

In D&R search, a heuristic strategy is used to quickly prioritize subproblems to determine the order of solving them. Generally, it is desirable to first solve the subproblems that are more promising for improvements. A subproblem is considered promising if it has a large initial integrality gap. However, the integrality gap is unknown until its LP relaxation is solved (which can usually be done quickly). In the proposed search strategy, if the gap of a subproblem is unknown, the algorithm estimates it with the costs incurred by that subproblem multiplied by an ad hoc coefficient. Such estimation is most effective when a few constraint violations are dominating (i.e., a large proportion of the total cost is caused by them). In that case, the strategy is able to identify the subproblems involved with those violations relatively easily. However, when all hard constraints and high cost soft constraints become satisfied, it becomes difficult to estimate the gap accurately because of the complex tradeoffs among side constraints. Besides using the integrality gap as a criterion for prioritization, the algorithm also keeps a tabu list; i.e., giving low priority to a subproblem which has been solved recently, because it is unlikely to improve the solution by solving the same subproblem a second time. However, the algorithm will again consider that subproblem as new after a certain number of changes have been made on the schedules of its projects (e.g., after solving other subproblems or performing the block-interchange search).

Although the D&R search is enhanced in a few ways, it may still get stuck in local optima. Therefore, another type of local search, i.e., block interchange, is implemented to help the D&R search leave the local optima.

4.4.4 Block Interchange

In the block-interchange search, a “block” is a sequence of consecutively performed projects by a team. The duration of a block is the summation of the project durations, and the category of a block is just the category of the projects in this block. In a block-interchange move, multiple blocks with the same duration and category exchange their positions (i.e., start times and assigned teams). Figure 18(a) illustrates a block interchange, where three blocks, one containing project 1, one containing project 6, and one containing projects 10 and 11, are interchanged.



	Team 1	Team 2	Team 3
Week 1	Project 1	Project 2	Project 3
Week 2		Project 6	
Week 3	Project 4		
Week 4	Project 5	Project 7	
Week 5		Project 8	Project 10
Week 6		Project 9	Project 11

	Team 1	Team 2	Team 3
Week 1	Project 10	Project 2	Project 3
Week 2	Project 11	Project 1	
Week 3	Project 4		
Week 4	Project 5	Project 7	
Week 5		Project 8	Project 6
Week 6		Project 9	

(a)

	Team 1	Team 2	Team 3
Week 1	Project 1	Project 2	Project 3
Week 2		Project 6	
Week 3	Project 4		
Week 4	Project 5	Project 7	
Week 5		Project 8	Project 10
Week 6		Project 9	Project 11

	Team 1	Team 2	Team 3
Week 1	Project 6	Project 2	Project 3
Week 2		Project 10	
Week 3	Project 4	Project 11	
Week 4	Project 5	Project 7	
Week 5		Project 8	Project 1
Week 6		Project 9	

(b)

Figure 18. Two schemes of a 3-block interchange.

We define an n-block interchange as the move associated with n blocks all changing positions. Suppose two blocks are initially in positions A and B (represented by A-B) in the current solution, after a 2-block interchange their positions will become B-A. When $n > 2$, an n-block interchange includes multiple interchange schemes. In a 3-block interchange, A-B-C may become B-C-A or C-A-B. Figure 18(a) and (b) illustrate two different 3-block interchange schemes. Note that A-C-B is not a valid 3-block interchange, because the first block is still in its initial position A after the move. Such a move can be performed with a 2-block interchange, and is not considered as a 3-block interchange in order to avoid duplication. In a 4-block interchange, there are totally 9 interchange schemes, i.e., A-B-C-D may become B-A-D-C, C-A-D-B, D-A-B-C, B-D-A-C, C-D-A-B, D-C-A-B, B-C-D-A, C-D-B-A or D-C-B-A.

In our implementation, we use $n \leq 4$. The reason is that the schedule is restricted by many side constraints, and a block interchange with $n > 4$ is very unlikely to improve the solution.

This is verified by the observation from numerical studies on the real-world problem instances. For the same reason, the duration of a block is also restricted to be no more than 10 weeks.

Teams may have different total durations of their assigned projects. If only the blocks with the same duration are interchanged, the total project duration of a team will never change. To make the search more powerful, the interchange of “tail blocks” is allowed. A tail block is a block at the end of the schedule plus some empty weeks. Figure 19 illustrates a tail block interchange. In this example, one tail block contains a single project 3, and the other contains projects 6 and 7, and an empty week highlighted by the cell filled with diagonal lines. After the move, project 3 is assigned to team 2, and projects 6 and 7 are assigned to team 1. The total project durations of team 1 and team 2 are also interchanged.

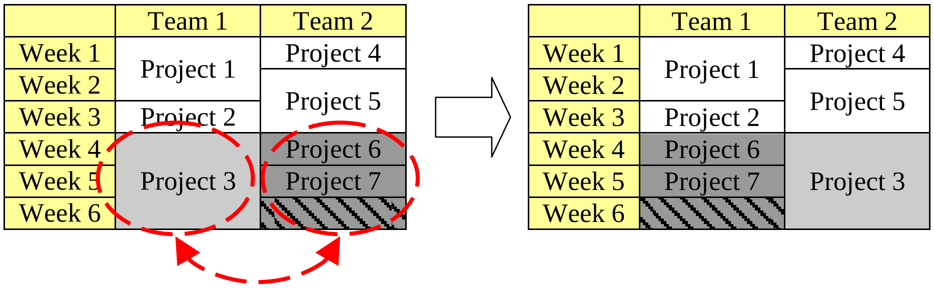


Figure 19. An example of tail block interchange.

On the scale of a practical problem, the size of the neighborhood is very large. Exhaustive local search would be impractical. Hence we use randomized local search strategy, where moves are randomly selected from the neighborhood to test. The algorithm first stores all blocks of the current schedule in multiple pools, which are defined based on team categories, team productivities, block durations and block types (i.e., whether a block is a tail block), so that only the blocks from the same pool can be interchanged with one another. Suppose a pool contains m

blocks. Then there are $\binom{m}{n}$ different n-block combinations within that pool. By calculating the total number of block combinations across all pools and all possible values of n, the algorithm is able to randomly select block combinations with equal probabilities. After a set of blocks is selected, all possible interchange schemes are enumerated. Randomized local search terminates when the solution is not improved after a certain number of trials.

Similar to the local search implementation in TISP, the costs related to every project are tracked. In order to improve the solution speed, the algorithm only calculates the cost changes associated with the interchanged blocks when a move is made.

Different from node interchange in VRP and task interchange in TISP, block interchange does not interchange blocks with different durations. For example, it does not allow insertion of a block into a schedule. The reason is that such a move will change the start times of all following projects, and with many side constraints in the problem instance, it tends to cause many unacceptable violations. Furthermore, such a move requires the recalculation of costs associated with many following projects, and will greatly reduce the solution speed.

Compared with D&R search, block-interchange search is much faster and is able to interchange projects among more than two teams. On the other hand, D&R search is able to optimize the schedules of all projects in one or two teams which block-interchange search may not be able to achieve. Therefore, the two types of searches can complement each other. The algorithm applies the block-interchange search whenever a subproblem is solved in the D&R search. Such alternations between the two searches are found to be able to improve the solution much more efficiently and effectively than any one of them alone.

In practice, we also implemented a parallel version of block interchange. Multiple threads are used to search the neighborhood of the current solution parallelly. Once a thread finds a better solution, the current solution is updated for all threads. Such parallel search is able to utilize the multiple cores of modern computers.

4.4.5 Data Reduction

Before solving the model, we first consolidate the input data in order to reduce the problem instance scale. There are two reasons that the data shall be reduced: (i) the raw data are provided by the railroad and may not be the most efficient way to formulate the problem; (ii) during local search we only care about the variables and constraints related to the subproblems or blocks, and the entire data set for the whole problem may not be efficient to formulate the subproblems or to calculate the costs related to blocks. Data reduction can reduce the number of variables and constraints in the model and hence improve the solution speed. Note that the data reduction mentioned here should not cause any loss of accuracy. Some data reductions can be automatically performed by MIP software when the subproblems are solved. Others need to be implemented using customized algorithm.

The spatial railroad network data can generally be reduced by removing unnecessary vertices and edges. A vertex is called “important” if it is an endpoint of a project. The goal of network reduction is to keep all important vertices in the network, preserve the shortest path distances among them, and reduce the number of “unimportant” vertices and edges. Assuming a vertex and an edge could cause the same computational complexity, the objective is to minimize the sum of the numbers of vertices and edges.

First, unimportant vertices and their incidental edges can be removed by adding new edges between each pair of its neighbors. If an unimportant vertex has degree one, the only edge incident to it can be simply removed. If it has degree two, its two incident edges can be removed while a new edge should be added to connect its two neighboring vertices. If it has degree three, the three incident edges can be removed but three new edges must be added to connect the three neighboring vertices. For vertices with four degrees, four edges can be removed but six has to be added. The examples for degree 1, 2, 3 and 4 are shown in Figure 20 (Δn and Δm are respectively the change of vertex number and the change of edge number). Therefore, unless Δn is given more than twice weight as that of Δm, only unimportant vertices with degree no larger than three are removed. The algorithm repeatedly removes such vertices until there is none left.

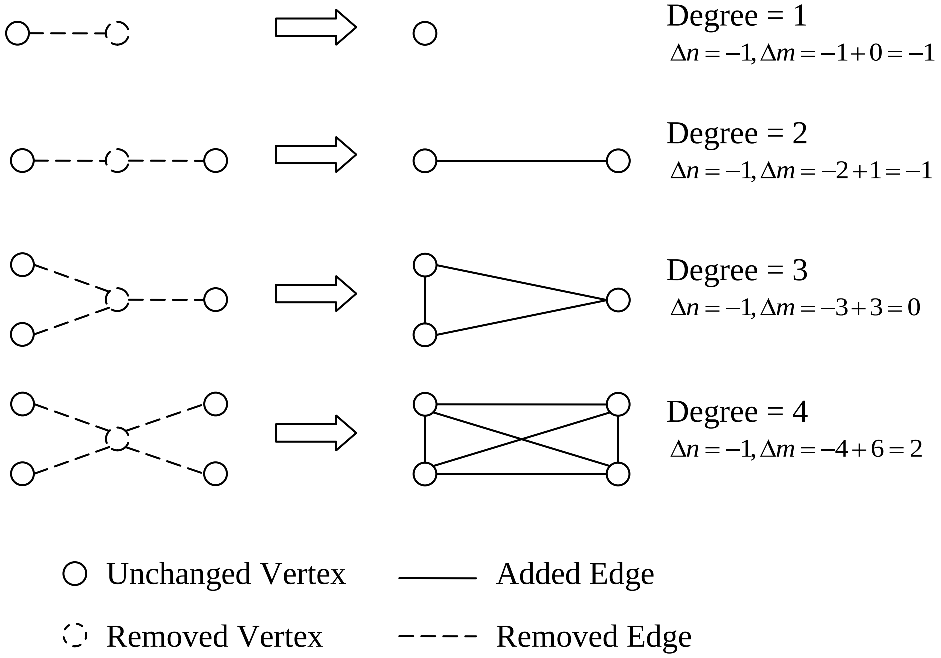

Degree = 1
$\Delta n = -1, \Delta m = -1 + 0 = -1$
Degree = 2
$\Delta n = -1, \Delta m = -2 + 1 = -1$
Degree = 3
$\Delta n = -1, \Delta m = -3 + 3 = 0$
Degree = 4
$\Delta n = -1, \Delta m = -4 + 6 = 2$
Unchanged Vertex
Added Edge
Removed Vertex
Removed Edge

Figure 20. Examples of removing unimportant vertices.

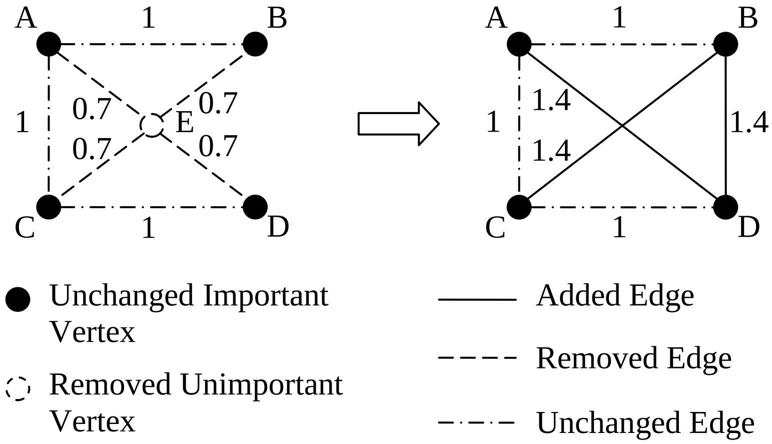


Figure 21. An example of a removable vertex with degree 4.

After the above process, removable vertices may still exist. An example is shown in Figure 21. Suppose A, B, C, D are important vertices and E is an unimportant one. The lengths of edges are shown in the figure. Though E’s degree is four, it can still be removed by adding only three new edges AD, BC and BD with length 1.4. These three new edges are necessary, because without them the shortest distances between AD, BC, and BD will become 2, 2 and 3, instead of 1.4 as in the original network. However, it is not necessary to add a new edge between AB. The original shortest path between A and B is edge AB, which does not pass through E, so the removal of E does not change the shortest distance between A and B. For the same reason, it is also not necessary to add any new edge between AC or CD.

Generally, let $N(v)$ be the set of neighbors of an unimportant vertex v. Let $S(v)$ be the set of unordered pairs (u_1, u_2) such that $u_1, u_2 \in N(v)$ and path u_1-v-u_2 is a subgraph (i.e. part) of some shortest path between two important vertices. It is easy to see that if degree(v) $\geq \left|S(v)\right|$, vertex v and all its incident edges can be replaced by new edges between every vertex pair in $S(v)$, and the number of edges in the network is reduced or remains unchanged. The algorithm repeatedly removes such vertices until there is none left.

Let n be the number of vertices in the original network. In each iteration, Floyd-Warshall algorithm (Floyd, 1962) is used to find the shortest path between every two vertices, and has time complexity $O(n^3)$. Two $n \times n$ matrices, *Dist*, which is used to store the shortest distances, and *Prev*, which is used to store the predecessor vertices of the shortest paths, are generated by Floyd-Warshall algorithm. To find a removable vertex, the algorithm needs to go through *Prev* to calculate the number of times that a vertex is used in a shortest path, so the time complexity is $O(n^2)$. To remove a vertex, the algorithm also needs to go through *Prev* to both remove and add edges, so the time complexity is also $O(n^2)$. Only one vertex is removed per iteration, so the total time complexity of the algorithm is $O(n(n^3 + n^2 + n^2)) = O(n^4)$. Currently it has not been proved that if this algorithm can guarantee an optimum solution, i.e., a network with fewest vertices and edges. However, it is easy to implement and has good performance for the practical problem instance scale.

MX set reduction is another way to reduce the scale of the problem instance. In the raw data provided by the railroad, most MX sets only contains two project sets, i.e., $\left|F_{P_{\text{MX}}}\right| = 2$. The reason is that the experts from the railroad review pairs of project sets (e.g., subdivisions) and identify those where two project sets are mutually exclusive. Multiple MX pairs can often be consolidated into a single MX set. For example, for project sets 1, 2 and 3, suppose {1, 2}, {2, 3} and {1, 3} are three MX pairs with the same penalty cost. If the raw data is used as the input, there will be three MX constraints. However, if they are consolidated into a single MX set {1, 2, 3}, only one constraint is needed.

The goal of MX set reduction is to obtain a minimum number of MX sets from the provided MX pairs. Because MX pairs with different penalty costs cannot be consolidated, we can first divide MX pairs into multiple groups based on their penalty costs, and perform data reduction on each group separately. For a certain group, we can construct an undirected graph with project sets as vertices and MX pairs as edges. The MX set reduction problem is equivalent to the minimum clique cover problem on the graph, where a clique (a subset of vertices where every two vertices are connected by an edge) represents a MX set. Minimum clique cover problem is NP-complete (Karp, 1972). In PTSP, we use a simple constructive heuristic algorithm to solve the minimum clique cover problem. The algorithm is as follows:

Step 1: Terminate the algorithm if there is no edge in the graph.

Step 2: Select an arbitrary edge, set its two endpoints as the current clique, and remove that edge from the graph.

Step 3: Try to find an arbitrary vertex which is adjoining to all vertices in the current clique. If such vertex exists, remove all edges between it and the current clique, add it to the current clique, and repeat Step 3; otherwise go to Step 4.

Step 4: Record the current clique. Go to Step 1.

The above algorithm may not always yield the optimal solution. However, because the railroad network is a planar network and most MX sets are composed by spatially adjoining project sets, it can generally obtain a good solution.

4.5 Case Studies

The proposed model and algorithm are implemented in the Microsoft Visual C++ environment on a personal computer with 2.66GHz dual core CPU and 3 GB RAM. Commercial MIP solver

CPLEX (version 1.12, multi-thread) is used to solve subproblems. The time limit of solving a restricted subproblem is set to 3 minutes.

In the first study, the proposed approach is applied to the PTSP in a Class I railroad company for a recent year, Year A[9]. About 300 projects need to be assigned to about 20 teams and scheduled in about 50 weeks of the year (after reserving some weeks for vacation). The detailed statistics of the input data regarding teams, projects, etc., are summarized in Table 2. The numbers of MX constraints are before data reduction. Note that not every type of side constraints proposed in the model is used in this problem instance due to limited data availability. The penalty costs of all soft constraints are provided by the company based on the impacts of the violations on its business operations. The number of projects to split is determined in an ad hoc way.

[9] This year may be different from the one mentioned in previous case studies.

Table 2. Input data statistics for production team scheduling (Year A data).

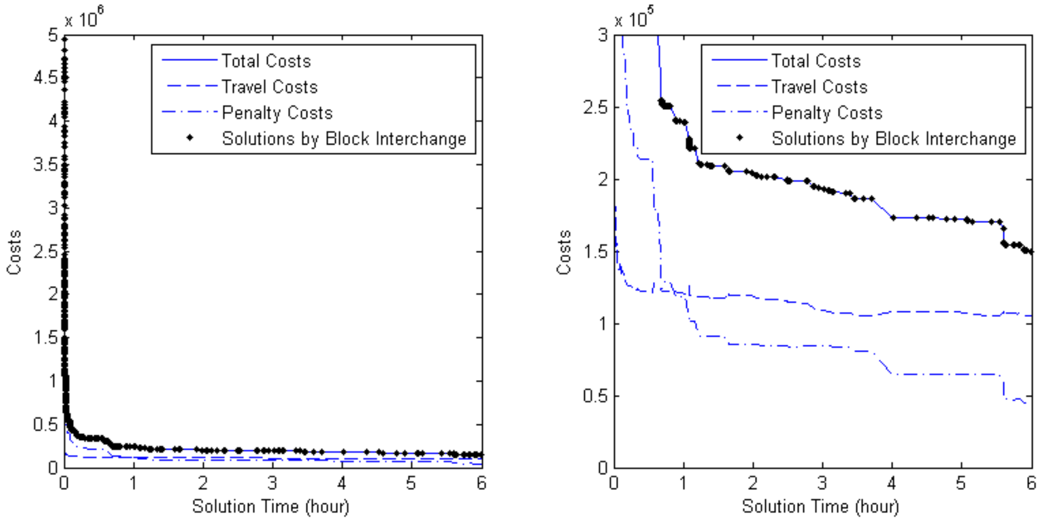
Team	# of teams	About 20
	# of team categories	2
	# of team productivities	2
	# of team types	5
Project	# of projects	About 300
	# of project types	7
# of weeks		About 50
# of side constraints (hard/soft)	Yard MX	32/0
	Junction MX	275/189
	Corridor MX	0/10
	Time window	2,423/628
	Preference	667/608
	Precedence	25/0
	Simultaneity	6/6
	Consecution	2/0
	Split Project	0/32

Table 3 shows the statistics of the model solution (after 6 hours of computation time). The unit of all cost entries in the table is the average cost needed for moving one team by one mile between projects. To protect data confidentiality, the cost entries are scaled by a constant factor between 0.5 and 2. In the statistics of "number of violations", one side constraint may be violated multiple times as introduced in Section 4.3. It can be seen that the model solution is able to satisfy all hard constraints and most soft constraints. Such a solution would be very difficult to obtain if manual procedure were used.

Table 3. Solution statistics for production team scheduling (Year A data).

Travel costs		104,458
Soft side constraints (penalty costs / # of violations)	Junction MX	4,086 / 61
	Corridor MX	13,599 / 58
	Time window	11,589 / 34
	Preference	14,068 / 130
	Simultaneity	0 / 0
	Split Project	0 / 0
Total costs (travel + penalty)		147,801
Total hard constraint violations		0

Figure 22 plots how the total costs (the sum of the travel costs and penalty costs, including the penalty costs of hard constraints internally set in the algorithm) converge over time. The running time of the scheduling model (less than 5 minutes) is not included. After an initial solution was obtained by the scheduling model, block-interchange search was applied and it reduced total costs rapidly within about 10 minutes. Then the model iteratively used D&R search and block-interchange search to continuously reduce the costs.



(a) (b)

Figure 22. Cost reduction over running time (Year A data).

In this solution, the number of split projects is 32, which is predetermined based on the experience from previous years. We have also tried to split 20 or 40 projects, and there is still one hard constraint violation after 6 hours of running time for both cases. The reason is as explained in Subsection 4.4.1: splitting too few projects will impose difficulty regarding the MX constraints, while splitting too many will increase the complexity of the problem and reduce the solution speed.

The algorithm for PTSP is non-deterministic, because block interchange uses a randomized search strategy, and the parallel computing used in both block interchange and CPLEX is non-deterministic. Therefore, different runs of algorithms may provide different solutions. In order to test the average performance of the algorithm, we ran the algorithm 9 more times with the same input. The average total costs of all 10 runs are 149,507, and the sample standard deviation is 6,140. In light of the Central Limit Theorem, we assume that the mean of the total costs of a solution approximately follows the normal distribution. Then the confidence interval for the mean of the total costs can be calculated as $\bar{X} \pm t_{\alpha/2,n-1}\frac{s}{\sqrt{n}}$, where $\bar{X}$ is the average costs, s is the sample standard deviation, n is the sample size (i.e., the number of runs), α is the desired significance level, and $t_{\alpha/2,n-1}$ is the upper critical value of the Student's *t*-distribution with n-1 degrees of freedom. We use $\alpha = 0.05$, and the confidence interval for the mean of the total costs is $149,507 \times (1 \pm 3.5\%)$ = (144,294, 154,721). We did not perform more runs because 3.5% has been small enough for the purpose of testing the performance of the algorithm.

In the second study, the proposed model and algorithm are applied to the PTSP in a Class I railroad company for another recent year, Year B, when a manual solution obtained by experts is also available for comparison. In this data set, about 300 projects need to be assigned to about 20 teams and scheduled in about 50 weeks of the year. The detailed statistics of the input data regarding teams, projects, etc., are summarized in Table 4.

Table 4. Input data statistics for production team scheduling (Year B data).

Team	# of teams	About 20
	# of team categories	2
	# of team productivities	2
	# of team types	5
Project	# of projects	About 300
	# of project types	7
# of weeks		About 50
# of side constraints (hard/soft)	Junction MX	0/273
	Corridor MX	0/12
	Time window	2,305/2,334
	Precedence	24/45

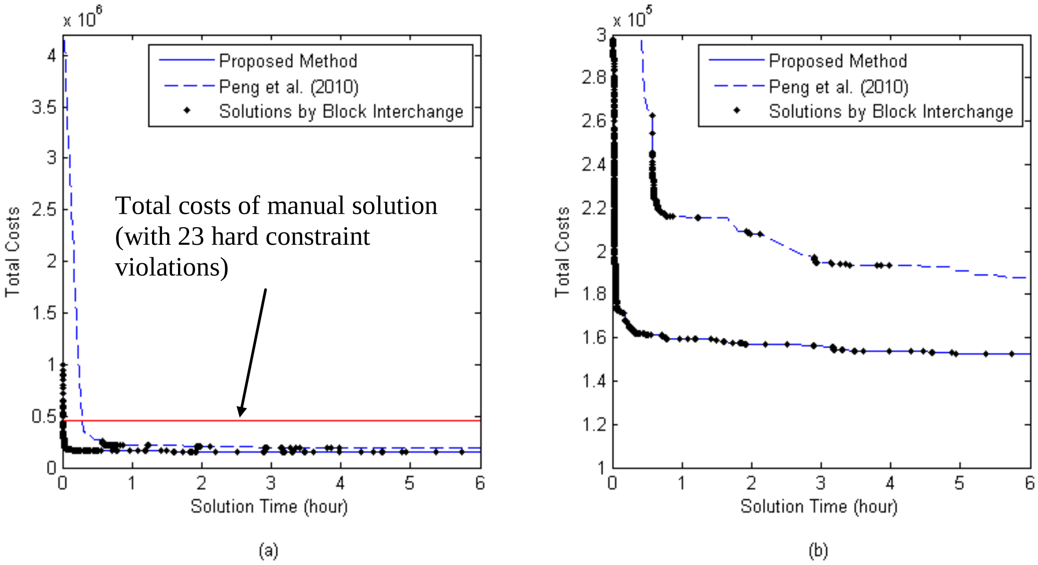
An approach proposed by Peng et al. (2010) was also implemented in the same hardware and software environment to solve the Year B data. There are two major differences between the approaches proposed in this dissertation study and Peng et al. (2010). First, in the block-interchange search used by Peng et al. (2010), a block can have only one project, and a set of blocks has only one interchange scheme. Hence their block interchange is much less powerful than the proposed one. Second, Peng et al. (2010) used a clustering model instead of the scheduling model to obtain an initial solution. The clustering model is not able to address as many side constraints as the scheduling model.

Table 5 shows the comparison of the model solutions by the approaches proposed by Peng et al. (2010) and this dissertation study (after 6 hours of computation time), and the benchmark solution manually developed by the experienced planning experts from the railroad (based on the same input data). To protect data confidentiality, the cost entries are scaled by a constant factor between 0.5 and 2. The scaling operation does not impact the comparison between different solutions. It can be seen the proposed approach reduces the total costs by 66.7% compared to the manual solution. It also satisfies all hard constraints while the manual solution has 23 violations. Compared to Peng et al. (2010) approach, the proposed approach reduced the total costs by 18.8%.

Table 5. Solution comparison for production team scheduling (Year B data).

Costs and violations		*Manual procedure*	*Peng et al.(2010) approach*	*Proposed approach*
Travel costs		161,944	158,598	139,921
Soft side constraints (penalty costs / # of violations)	Junction MX	24,117 / 72	8,709 / 26	2,345 / 7
	Corridor MX	12,661 / 54	5,158 / 22	3,751 / 16
	Time window	257,965 / 95	14,664 / 41	6,062 / 36
	Precedence	670 / 20	603 / 18	435 / 13
Total costs (travel + penalty)		457,357	187,733	152,647
Hard side constraints (# of violations)	Time window	15	0	0
	Precedence	8	0	0
Total hard constraint violations		23	0	0

Figure 23 plots how the total costs converge over time. The time to obtain an initial solution is less than 5 minutes for both approaches and is not included. By comparing the total costs at time 0, it can be seen the scheduling model in the proposed approach can obtain a much better initial solution than the clustering model in Peng et al. (2010) approach. The total costs from both approaches were reduced dramatically in the first hour and became lower than the manual solution. However, those from the proposed approach dropped much faster because of the improved block interchange. In fact, the proposed approach is able to obtain in 3 minutes a solution as good as that by Peng et al. (2010) approach in 6 hours.



Total costs of manual solution (with 23 hard constraint violations)

(a)

(b)

Figure 23. Cost reduction over running time (Year B data).

Glossary of Symbols

$\mathrm{C}_{\text{Con}}(\cdot)$: Total consecution constraint penalty costs,

$c_{\text{Con},P_{\text{Con}}}$: Consecution constraint penalty cost when projects in P_{Con} are not performed consecutively by the same team,

$C_{\text{cost item}}(\cdot)$: Total costs of a certain cost item,

$\mathrm{C}_{\text{Lim}}(\cdot)$: Total limit constraint penalty costs,

$c_{\text{Lim},<P_{\text{Lim}},k>}$: Limit constraint penalty cost per week when more than $m_{\text{Lim},P_{\text{Lim}}k}$ weeks of projects from P_{Lim} are performed by team $k \in K$,

$\mathrm{C}_{\text{MX}}(\cdot)$: Total mutual exclusion constraint penalty costs,

$c_{\text{MX},F_{\text{MX}}}$: Mutual exclusion constraint penalty cost per project per week for constraint F_{MX},

$\mathrm{C}_{\text{NS}}(\cdot)$: Total non-simultaneity constraint penalty costs,

$c_{\text{NS},pq}$: Non-simultaneity constraint penalty cost when projects $p,q \in P$ are performed simultaneously,

$\mathrm{C}_{\text{Prec}}(\cdot)$: Total precedence constraint penalty costs,

$c_{\text{Prec},pq}$: Precedence constraint penalty cost when project $q \in P$ is not started after at least $t_{\text{Prec},pq}$ weeks after the start of project $p \in P$,

$\mathrm{C}_{\text{Pref}}(\cdot)$: Total preference constraint penalty costs,

$c_{\text{Pref},pk}$: Preference constraint penalty cost when project $p \in P$ is performed by team $k \in K$,

$\mathrm{C}_{\text{relay}}(\cdot)$: Total relay rail constraint penalty costs,

$c_{\text{relay},P_{\text{relay}}}$: Relay rail constraint penalty cost per unit of relay rail shortage in constraint P_{relay},

$\mathrm{C}_{\text{S}}(\cdot)$: Total simultaneity constraint penalty costs,

$c_{\text{S},pq}$: Simultaneity constraint penalty cost when projects $p,q \in P$ are not performed simultaneously,

$\mathrm{C}_{\text{SP}}(\cdot)$: Total split project constraint penalty costs,

$c_{\text{SP},pq}$: Split project penalty cost when projects $p,q \in P$ are performed neither simultaneously by different teams nor consecutively by the same team,

$\mathrm{C}_{\text{travel}}(\cdot)$: Total travel costs,

$c_{\text{travel},v_1v_2k}$: Cost for team $k \in K$ to travel from vertex $v_1 \in V$ to $v_2 \in V$,

$\mathrm{C}_{\text{TW}}(\cdot)$: Total time window constraint penalty costs,

$c_{\text{TW},pw}$: Time window constraint penalty cost when project $p \in P$ is performed in week $w \in W$,

F_{MX}: Set of project sets in an mutual exclusion constraint,

K: Set of teams,

K_{B}: Set of blitz teams,

K_p: Set of teams which can perform project $p \in P$,

$K_{P_{\text{Con}}}$: Set of teams which can perform at least one project in P_{Con},

$K_{\text{Prod},pt}$: Set of teams which can perform project $p \in P$ with duration $t \in T_p$,

$m_{\text{Lim},P_{\text{Lim}}k}$: Maximum allowed weeks of projects from P_{Lim} performed by team $k \in K$ in a limit constraint,

$m_{\text{MX},F_{\text{MX}}}$: Maximum allowed simultaneous curfewed project sets from $F_{P_{\text{MX}}}$ in a mutual exclusion constraint,

P: Set of projects,

P_{Con}: Set of projects in a consecution constraint,

P_k: Set of projects which can be performed by team $k \in K$,

P_{Lim}: Set of projects used in limit constraints,

P_{MX}: Project set used in mutual exclusion constraints,

P_{relay}: Set of projects in a relay rail constraints,

P_{vk}^{+}: Set of projects which can be performed by team $k \in K$ and whose endpoints v, v' satisfy $v \le v'$,

P_{vk}^{-}: Set of projects which can be performed by team $k \in K$ and whose endpoints v, v' satisfy $v > v'$,

$< p,k >_{\text{Pref}}$: Pair of project $p \in P$ and team $k \in K$ in a preference constraint,

$< P_{\text{Lim}}, k >$: Pair of a set of projects P_{Lim} and team $k \in K$ in a limit constraint,

$< p,q >_{\text{NS}}$: Pair of projects $p,q \in P$ in a non-simultaneity constraint,

$< p,q >_{\text{Prec}}$: Pair of projects $p,q \in P$ in a precedence constraint,

$< p,q >_{\text{S}}$: Pair of projects $p,q \in P$ in a simultaneity constraint,

$< p,q >_{\text{SP}}$: Pair of projects $p,q \in P$ in a split project constraint,

$< p,w >_{\text{TW}}$: Pair of project $p \in P$ and week $w \in W$ in a time window constraint,

$r_{\text{other},w}$: Net relay rail output from other sources in week $w \in W$,

r_p: Net relay rail output from project $p \in P_{\text{relay}}$,

$t_{\text{Con},P_{\text{Con}}pqk}$: Total duration of all projects except p and q in P_{Con} for team $k \in K_{P_{\text{Con}}}$,

T_p: Set of possible durations of project $p \in P$,

t_{pk}: Duration of project $p \in P$ for team $k \in K$,

$t_{\text{Prec},\,pq}$: Number of weeks after the start of project $p \in P$ when project $q \in P$ can be started in a precedence constraint,

$t_{\text{relay},\,p}$: Number of weeks after the start of project $p \in P_{\text{relay}}$ when its supplied relay rail becomes available or its demanded relay rail should be available on site,

V: Set of vertices,

W: Set of weeks,

W_{B}: Set of blitz weeks,

$W_{\text{end},k}$: Set of end weeks for team $k \in K$,

$W_{\text{start},k}$: Set of start weeks for team $k \in K$,

$\mathbf{x} = \{x_{v_1 v_2 kw}\}$: Binary variables indicating if team $k \in K$ travels from vertex $v_1 \in V$ to vertex $v_2 \in V$ in week $w \in W$,

$\mathbf{y} = \{y_{pkw}\}$: Binary variables indicating if team $k \in K$ starts to perform project $p \in P$ in week $w \in W$,

y^{+}_{pkw}, y^{-}_{pkw}: Binary variables indicating the direction the team follows when performing a project,

y'_{pkw}: Binary variables indicating if team $k \in K$ is performing project $p \in P$ in week $w \in W$.

CHAPTER 5 JOB-TO-PROJECT CLUSTERING

5.1 Introduction

Maintenance jobs are the smallest unit of a capital track maintenance activity. Thousands of them are created every year based on the defects identified during track inspection. A typical rail job covers from a few hundred to over a hundred thousand feet of rail track, and a typical timber and surfacing (T&S) job involves the replacement of from a few hundred to tens of thousands of ties. The duration of a job varies from less than one day to a few weeks. It is difficult to directly schedule these maintenance jobs in production team scheduling problem (PTSP), because that would induce a prohibitive number of decision variables and side constraints.

Fortunately, a few nice characteristics of PTSP make it possible to cluster the jobs into fewer projects and address a portion of the side constraints during the clustering process. First, production teams can only make long-distance travels during weekends. Jobs performed between long-distance travels must be spatially close to each other, and their total duration should be close to an integer number of weeks. Second, jobs performed within one week need to have some common properties, such as being in the same subdivision or track line, so that train operations can be rescheduled easily. These requirements make it natural to first cluster the jobs into projects with durations of integer numbers of weeks, i.e., by solving a job-to-project clustering problem (JTPCP), and then use the projects as the basic scheduling units in PTSP. Such a two-step procedure is much more practical than directly scheduling jobs in PTSP.

Typically, the railroad would like to minimize the total duration of projects in order to reduce the costs. In the studied JTPCP, each production team is contracted at a flat payment rate of about tens of thousands of dollars per week. For a certain clustered project, for example, if it takes 0.6 week to finish all jobs in this project, the billable project duration will still be one week because the team generally cannot travel to the next project in the middle of the week. The railroad will have to pay for one full week, while 40% of the cost could otherwise be reduced or avoided. Hence, minimizing the total duration of projects is generally achieved by reducing the rounding errors of individual project durations.

While minimizing the total duration of all projects, the railroad has to consider many side constraints which make the problem very complex. For example, certain jobs from different subdivisions should not be clustered into one project; otherwise it will be difficult to reschedule train traffic through those subdivisions. In the current practice, JTPCP is solved manually. The procedure usually takes about one week, yet the solution quality may still not be satisfying. In this chapter, we will develop mathematical models and solution algorithms to effectively solve the clustering problem for large-scale real-world instances.

5.2 Model Formulation

5.2.1 Core Model

Let I be the set of jobs. Because every job has two endpoints, we use two directed arcs to represent the possible direction to carry out a job. Let E be the set of all arcs, and E_i be the set of arcs representing the directions of job $i \in I$, where $|E_i| = 2$. Let $t_{\text{travel},e_1 e_2}$ be the travel time from the head of arc $e_1 \in E$ (i.e., the finishing point of one job) to the tail of arc $e_2 \in E$ (i.e., the start point of another job), and T_{travel} be the maximum allowed travel time.

Similar to the definitions in Subsection 4.3.1, we divide production teams into multiple types. Let K be the set of team types[10]. For $k \in K$, let T_k be the total number of available work weeks of all type-*k* teams in the planning horizon (i.e., one year in this study), and $c_{\text{team},k}$ be the cost for a type-*k* team to work for one week (referred to as "working cost" from now on). Let $t_{\text{job},ik}$ be the duration of job $i \in I$ when it is performed by a type-*k* team.

Let P be the set of projects, and P_k be the projects which are assigned to team type-*k*, for all $k \in K$. In JTPCP, a project is only assigned to one team type[11], and hence P_k is disjoint across *k*. Let t_p be the duration of project $p \in P$ in the unit of weeks. For each project $p \in P$, a virtual arc $e_{\text{start},p}$ is created and added to E. The travel time from arc $e_{\text{start},p}$ to any arcs is 0, i.e., $t_{\text{travel},e_{\text{start},p}e} = 0$, $\forall e \in E$. Every project $p \in P$ is composed of a sequence of arcs, starting with the virtual arc $e_{\text{start},p}$. When a maintenance team performs a project, it traverses these arcs and performs the corresponding jobs. Let $\mathbf{x} = \left\{ x_{e_1 e_2 p} : e_1, e_2 \in E, p \in P \right\}$ be the set of decision variables, such that $x_{e_1 e_2 p} = 1$ if arc e_1 is scheduled right after arc e_2 in project p, or 0 otherwise.

Let $C_{\text{cost item}}(\cdot)$ be the cost function of side constraints. The core model of JTPCP (without side constraints) can be formulated in the form of a vehicle routing problem (VRP) as follows:

$$\text{(JTPCP)} \quad \min \sum_{k \in K} \left(c_{\text{team},k} \sum_{p \in P_k} t_p \right) + \sum_{\text{other cost items}} C_{\text{cost item}}(\mathbf{x}), \qquad (5.1)$$

s.t.

[10] Note that in Chapter 4 *K* denotes the actual teams, while here *K* denotes the team types (each type may include one or more teams).

[11] This is different from PTSP, where a project may be reassigned to another team type.

$$\sum_{e\in E} x_{e_{\text{start},p}ep} = 1, \qquad \forall p \in P, \tag{5.2}$$

$$\sum_{e_2\in E} x_{e_1e_2p} - \sum_{e_2\in E} x_{e_2e_1p} \le 0, \qquad \forall e_1 \in E \setminus \{e_{\text{start},p}\},\, p \in P, \tag{5.3}$$

$$u_i + 1 + U(x_{e_1e_2p} - 1) \le u_j, \qquad \forall e_1 \in E_i, e_2 \in E_j, i, j \in I, p \in P, \tag{5.4}$$

$$\sum_{e_1\in E_i}\sum_{e_2\in E}\sum_{p\in P} x_{e_1e_2p} = 1, \qquad \forall i \in I, \tag{5.5}$$

$$t_p \ge \sum_{i\in I}\sum_{e_1\in E_i}\sum_{e_2\in E} (t_{\text{job},ik} + t_{\text{travel},e_1e_2}) x_{e_1e_2p}, \quad \forall p \in P_k, k \in K, \tag{5.6}$$

$$\sum_{p\in P_k} t_p \le T_k, \qquad \forall k \in K, \tag{5.7}$$

$$x_{e_1e_2p} = 0, \qquad \forall (e_1,e_2) \in \left\{(e_1,e_2) \in E^2 : t_{\text{travel},e_1e_2} > T_{\text{travel}}\right\},\, p \in P, \tag{5.8}$$

$$x_{e_1e_2p} \in \{0,1\}, \qquad \forall e_1, e_2 \in E, p \in P, \tag{5.9}$$

$$t_p \in \mathbb{N}_0, \qquad \forall p \in P. \tag{5.10}$$

Here parameter U is a large constant number that should not be smaller than the maximum number of jobs in a project.[12] Auxiliary variables $\mathbf{u} = \{u_i : i \in I\}$ help eliminate subtours, whereas u_i indicates the relative sequence in which job $i \in I$ is performed in a project. Specifically, let $i_{p1}, i_{p2}, \ldots, i_{p|I_p|}$ be the sequence of jobs in project $p \in P$, where I_p is the set of jobs in project p. Then we have $u_{i_{p2}} = u_{i_{p1}} + 1,\ u_{i_{p3}} = u_{i_{p1}} + 2, \ldots,\ u_{i_{p|I_p|}} = u_{i_{p1}} + |I_p| - 1$.

Objective (5.1) minimizes the total costs. Constraints (5.2) and (5.3) ensure flow conservations. Constraints (5.4) eliminate subtours using auxiliary variables $\mathbf{u}$. Constraints (5.5)

[12] Obviously, $|I|$ is a choice for the value of U, because no project can contain more than $|I|$ jobs.

require every job to be assigned to a project. Constraints (5.6) calculate the duration of a project as the ceiling of the total time needed for the team to perform all jobs in the project and to travel between the jobs. Constraints (5.7) require that the total project duration assigned to a team type should not exceed the total available weeks of that team type in the planning horizon. Constraints (5.8) prohibit any travel longer than the pre-determined limit T_{travel}. Constraints (5.9) and (5.10) define the binary and integer variables respectively.

Besides the core model, multiple types of side constraints should also be considered in JTPCP to ensure practicality and relevance. These side constraints could be either soft or hard. The following subsections introduce these side constraints.

5.2.2 Mutual Exclusion Constraint

Mutual exclusion (MX) constraints require certain jobs not to be assigned to the same project. Usually jobs from different subdivisions should not be assigned to one project, otherwise train operations in multiple subdivisions will be affected when the project is performed. Sometimes a job can only be performed by certain teams (not team types) because of the preference constraints in PTSP (i.e., the crews of those teams have their homes near that job or are familiar with the track section of that job; see Subsection 4.3.3). If two such jobs must be performed by two different teams, obviously they should not be assigned to one project. Note that the MX constraints in JTPCP are applied to the project dimension (i.e. the team dimension in traditional VRP models) of the schedule, while the MX constraints in PTSP are applied to the time dimension of the schedule.

Define y_{ip} as follows:

$$y_{ip} = \sum_{e_1 \in E_i} \sum_{e_2 \in E} x_{e_1 e_2 p} \,, \qquad \forall i \in I, p \in P \,. \tag{5.11}$$

So $y_{ip} = 1$ if job $i \in I$ is assigned to project $p \in P$, or 0 otherwise. Let $\{i, j\}_{\mathrm{MX}}$ be a pair of jobs which should not be assigned to the same project, so $\{i, j\}_{\mathrm{MX}}$ can also be used to denote an MX constraint. If a project violates multiple soft MX constraints, i.e., the project contains multiple pairs of jobs which are in a soft MX constraint, only the highest penalty cost among those constraints will be incurred. Let $\upsilon_{\mathrm{MX},p}$ be the penalty cost incurred by project $p \in P$. MX constraints can be formulated as follows:

$$\mathrm{C}_{\mathrm{MX}}(\mathbf{x}) = \sum_{p \in P} \upsilon_{\mathrm{MX},p} \,, \tag{5.12}$$

$$c_{\mathrm{MX},ij}(y_{ip} + y_{jp} - 1) \le \upsilon_{\mathrm{MX},p} \,, \qquad \forall p \in P \text{, for all soft } \{i, j\}_{\mathrm{MX}} \,, \tag{5.13}$$

$$y_{ip} + y_{jp} \le 1 \,, \qquad \forall p \in P \text{, for all hard } \{i, j\}_{\mathrm{MX}} \,, \tag{5.14}$$

$$\upsilon_{\mathrm{MX},p} \ge 0 \,, \qquad \forall p \in P \,, \tag{5.15}$$

where $\mathrm{C}_{\mathrm{MX}}(\cdot)$ is the total MX constraint penalty costs, and $c_{\mathrm{MX},ij}$ is the penalty cost of soft constraint $\{i, j\}_{\mathrm{MX}}$. Equation (5.12) calculates $\mathrm{C}_{\mathrm{MX}}(\cdot)$ as the summation of all soft MX constraint penalty costs. Constraints (5.13) and (5.14) respectively define soft and hard MX constraints. Constraints (5.15) define non-negative variables $\upsilon_{\mathrm{MX},p}$.

5.2.3 Preference Constraints

Jobs can be divided into multiple types. Preference constraints require a certain type of jobs to be assigned to certain types of teams, usually because either only those teams have the capability of performing those jobs, or they are more efficient at those jobs. The preference constraints in JTPCP are similar to those in TISP and PTSP. Let H denote the set of job types, and $I_h \subset I$ denote the subset of jobs with type $h \in H$. Let $< h, k >_{\text{Pref}}$ denote a preference constraint where type-h jobs should not be assigned to type-k teams. Preference constraints can be formulated as follows:

$$\mathrm{C}_{\text{Pref}}(\mathbf{x}) = \sum_{\forall \text{soft } <h,k>_{\text{Pref}}} \left(c_{\text{Pref},hk} \sum_{i \in I_h} \sum_{p \in P_k} y_{ip} \right), \tag{5.16}$$

$$y_{ip} = 0, \quad \forall i \in I_h, p \in P_k, \text{ for all hard } < h, k >_{\text{Pref}}, \tag{5.17}$$

where $\mathrm{C}_{\text{Pref}}(\cdot)$ is the total preference constraint penalty costs, and $c_{\text{Pref},hk}$ is the cost for soft preference constraint $< h, k >_{\text{Pref}}$.

5.2.4 Limit Constraints

Limit constraints restrict the total quantity of a certain job attribute assigned to one project. For example, the average number of railroad crossings per mile in all non-yard T&S jobs in a project performed by a small T&S team should be bounded from above, because small T&S teams often do not have the proper equipment for crossing maintenance.

Let $f_{\text{Lim}}(\mathbf{y}_p)$ be the total quantity of a certain attribute in project $p \in P$, where variables $\mathbf{y}_p = \{y_{ip} : i \in I\}$ indicate which jobs in I are assigned to project $p \in P$. For example, suppose $m_{\text{crossing},i}$ equals the number of crossings in job $i \in I$ if the job is a non-yard T&S job, or 0 otherwise. $m_{\text{miles},i}$ equals the length of job $i \in I$ (in terms of miles) if the job is a non-yard T&S job, or 0 otherwise. Then the average number of crossings per mile of non-yard T&S jobs in project $p \in P$ can be written as:

$$f_{\text{Lim}}(\mathbf{y}_p) = \begin{cases} \sum_{i \in I} m_{\text{crossing},i} y_{ip} / \sum_{i \in I} m_{\text{miles},i} y_{ip}, & \text{if } \sum_{i \in I} m_{\text{miles},i} y_{ip} > 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5.18}$$

Suppose the quantity of an attribute from $f_{\text{Lim}}(\mathbf{y}_p)$ should not exceed $m_{\text{Lim},f_{\text{Lim}}k}$ units for a project performed by team type $k \in K$. We use $< f_{\text{Lim}}, k >$ to represent a limit constraint, which can be formulated as follows:

$$\text{C}_{\text{Lim}}(\mathbf{x}) = \sum_{\forall \text{soft} < f_{\text{Lim}},k>} \left(c_{\text{Lim},f_{\text{Lim}}k} \sum_{p \in P_k} \upsilon_{\text{Lim},f_{\text{Lim}}p} \right), \tag{5.19}$$

$$\upsilon_{\text{Lim},f_{\text{Lim}}p} \geq f_{\text{Lim}}(\mathbf{y}_p) - m_{\text{Lim},f_{\text{Lim}}k}, \qquad \forall p \in P_k, \text{ for all soft } < f_{\text{Lim}}, k >, \tag{5.20}$$

$$\upsilon_{\text{Lim},f_{\text{Lim}}p} \geq 0, \qquad \forall p \in P_k, \text{ for all soft } < f_{\text{Lim}}, k >, \tag{5.21}$$

$$f_{\text{Lim}}(\mathbf{y}_p) \leq m_{\text{Lim},f_{\text{Lim}}k}, \qquad \forall p \in P_k, \text{ for all hard } < f_{\text{Lim}}, k >, \tag{5.22}$$

where $\mathrm{C}_{\mathrm{Lim}}(\cdot)$ is the total penalty costs associated with limit constraints, $c_{\mathrm{Lim},f_{\mathrm{Lim}}k}$ is the penalty cost of soft limit constraint $< f_{\mathrm{Lim}}, k >$, and $\upsilon_{\mathrm{Lim},f_{\mathrm{Lim}}p}$ is the number of violations in project $p \in P_k$. Equation (5.19) defines $\mathrm{C}_{\mathrm{Lim}}(\cdot)$ as the summation of all soft limit constraint penalty costs. Constraints (5.20) define $\upsilon_{\mathrm{Lim},f_{\mathrm{Lim}}p}$ as the exceedance of $f_{\mathrm{Lim}}(\mathbf{y}_p)$ in project $p \in P_k$ over $m_{\mathrm{Lim},f_{\mathrm{Lim}}k}$. Constraints (5.21) define non-negative variables $\upsilon_{\mathrm{Lim},f_{\mathrm{Lim}}p}$ for soft limit constraints and constraints (5.22) define hard limit constraints.

Note that the limit constraints here in fact generalize those in PTSP. For the limit constraints of PTSP, $f_{\mathrm{Lim}}(\cdot)$ just equals the total duration of 5-day projects performed by a team.

5.2.5 Rounding Constraints

Rounding constraints allow the total job duration in a project to be rounded down rather than rounded up in certain conditions. Teams are able to work overtime for a certain percentage in sections where train schedules permit that. Although overtime working incurs extra costs per unit time, it may still save the overall costs because we may be able to round down the total job duration. For example, suppose the total job duration is 5.1 weeks in a project. It will be preferable to have the team work 5 weeks (including 0.1 week of overtime) rather than 6 weeks, as long as the overtime cost for 0.1 week is lower than the regular working cost for 1 week. Such rounding may also help satisfy some constraints such as (5.7).

Suppose the overtime cost (including the overtime payment and the penalty cost due to crew's unwillingness towards overtime) is r_{overtime} times as large as the regular working cost. For each job, there is a maximum overtime percentage $r_{\mathrm{JOVT},i}$, which means the maximum allowed overtime duration at the track section of that job is the total job duration multiplied by $r_{\mathrm{JOVT},i}$. If a project $p \in P$ contains jobs with different $r_{\mathrm{JOVT},i}$, the maximum overtime percentage of that project, $r_{\mathrm{POVT},p}$, will be defined as the minimum $r_{\mathrm{JOVT},i}$ among all its jobs. Define $t_{\mathrm{TJ},p}$ as the total job duration (including travel time) in project $p \in P$:

$$t_{\mathrm{TJ},p} = \sum_{i \in I} \sum_{e_1 \in E_i} \sum_{e_2 \in E} (t_{\mathrm{job},ik} + t_{\mathrm{travel},e_1 e_2}) x_{e_1 e_2 p} \, , \qquad \forall p \in P_k, k \in K \, . \tag{5.23}$$

The rounding constraints can be formulated as:

$$\mathrm{C}_{\mathrm{overtime}}(\mathbf{x}) = r_{\mathrm{overtime}} \sum_{k \in K} \left(c_{\mathrm{team},k} \sum_{p \in P_k} t_{\mathrm{overtime},p} \right), \tag{5.24}$$

$$r_{\mathrm{POVT},p} \le r_{\mathrm{JOVT},i} y_{ip} + \max_{j \in I} \left\{ r_{\mathrm{JOVT},j} \right\} (1 - y_{ip}) \, , \qquad \forall i \in I, p \in P \, , \tag{5.25}$$

$$t_p \ge \frac{1}{1 + r_{\mathrm{POVT},p}} t_{\mathrm{TJ},p} \, , \qquad \forall p \in P \, , \tag{5.26}$$

$$t_{\mathrm{overtime},p} \ge t_{\mathrm{TJ},p} - t_p \, , \qquad \forall p \in P \, , \tag{5.27}$$

$$t_{\mathrm{overtime},p} \ge 0 \, , \qquad \forall p \in P \, , \tag{5.28}$$

where $\mathrm{C}_{\mathrm{overtime}}(\cdot)$ is the total overtime costs, and $t_{\mathrm{overtime},p}$ is the overtime working duration of project $p \in P$. Equation (5.24) defines $\mathrm{C}_{\mathrm{overtime}}(\cdot)$ as the summation of overtime costs of all projects. Constraints (5.25)-(5.27) respectively calculate $r_{\mathrm{POVT},p}$, t_p and $t_{\mathrm{overtime},p}$. Constraints (5.28) define non-negative variables $t_{\mathrm{overtime},p}$. Note that constraints (5.6) in the core model should be replaced by constraints (5.26). These rounding constraints, especially the non-linear ones, (5.26), are difficult to solve.

5.2.6 Minimum Duration Constraints

Minimum duration constraints require that the total duration of all jobs in a project must be at least T_{job}, where $T_{\text{job}} < 1$ week. If a project cannot meet such a requirement, its jobs will instead be performed by local maintenance teams (which are often less costly than regular production teams). However, because of the lower efficiency and limited availability of local maintenance teams, the railroad would rather like to avoid such transfers from happening. Hence, the minimum duration constraints can be formulated as:

$$\mathrm{C}_{\text{transferred}}(\mathbf{x}) = \sum_{i \in I} \left[c_{\text{transferred},i} \left(1 - \sum_{p \in P} y_{ip} \right) \right], \tag{5.29}$$

$$t_{\text{TJ},p} \geq T_{\text{job}} (1 - x_{e_{\text{start},p} e_{\text{start},p} p}), \qquad \forall p \in P, \tag{5.30}$$

$$\sum_{p \in P} y_{ip} \leq 1, \qquad \forall i \in I, \tag{5.31}$$

where $\mathrm{C}_{\text{transferred}}(\cdot)$ is the total cost incurred by transferring jobs to local teams, and $c_{\text{transferred},i}$ is the cost of transferring job $i \in I$. Hence, Equation (5.29) calculates $\mathrm{C}_{\text{transferred}}(\cdot)$ as the summation of the costs incurred by all transferred jobs. Constraints (5.30) require a project with at least one job to have at least T_{job} week of total job duration. By the definition of $x_{e_{\text{start},p} e_{\text{start},p} p}$ in the core model, $x_{e_{\text{start},p} e_{\text{start},p} p} = 1$ if a project does not contain any jobs, or 0 otherwise. Constraints (5.31) require a job to be performed at most once. During implementation, constraints (5.31) should replace constraints (5.5) in the core model, because now with some penalty we allow a job not to be assigned to any projects.

5.2.7 Project Duration Constraints

Project duration constraints require the duration of a project to be within a certain range; otherwise the jobs in that project should be assigned to other team types to improve efficiency. There are a few major differences between minimum duration constraints and project duration constraints. First, a minimum duration constraint is applied to the total duration of jobs in a project (including travel time), while a project duration constraint is applied to the project duration, which may either be the ceiling or the floor of the total job duration (which is endogenously decided by the model as described in Subsection 5.2.5). More importantly, the duration limits in minimum duration constraints are given values, while those in project duration constraints depend on the project composition. These differences make the project duration constraints much more complex than the minimum duration constraints.

Let H_p be the subset of job types involved in project $p \in P$. Let $T_{\min,H'k}$ be the minimum duration of a project which contains job types $H' \subseteq H$ and is performed by team type $k \in K$. Similarly, let $T_{\max,H'k}$ be the maximum duration of such a project. Project duration constraints are all hard and can be formulated as follows:

$$T_{\min,H_pk} \le t_p \le T_{\max,H_pk}, \qquad \forall p \in P_k, k \in K. \qquad (5.32)$$

The difficulty of solving these constraints is that H_p is unknown until the composition of project $p \in P$ is determined. If the MIP algorithms were used, the constraints would have to be formulated in a much more complex way. For example, a decision variable may be needed for every possible composition (i.e. the combination of job types) of a project, resulting in an exponential number of integer decision variables.

5.3 Algorithm

A practical problem instance of JTPCP involves thousands of jobs and hundreds of projects. The presence of side constraints significantly increases the complexity of the problem. Fortunately, real-world problem instances have some good characteristics that allow us to develop heuristic algorithms to obtain a near-optimal solution.

Similar to PTSP, the jobs and team types in JTPCP can be divided into categories, such as rail and T&S. Jobs in a category can only be performed by team types in the same category. Unlike PTSP, in JTPCP there are no constraints imposed on jobs or teams across different categories. Therefore, the problem can be decomposed into subproblems by categories, and each subproblem can be solved separately.

Figure 24 illustrates the proposed algorithm framework. A greedy algorithm is first applied to obtain an initial solution. Then a local search algorithm and a feasibility heuristic algorithm are iteratively applied to improve the solution. The algorithm terminates when some stopping criteria are met, e.g., the current solution remains unchanged since the last iteration, or a certain number of iterations have been finished. The details of the algorithms will be discussed in the following subsections.

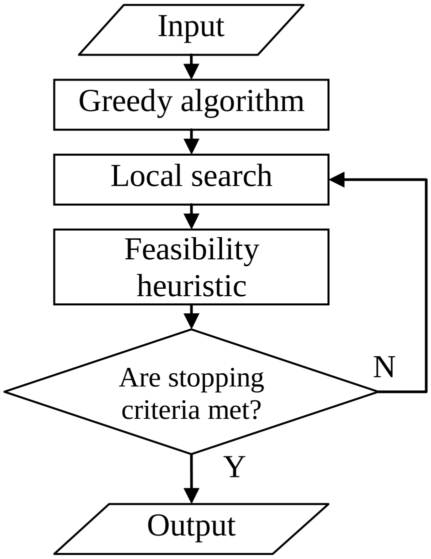


Figure 24. Algorithm framework of job-to-project clustering.

5.3.1 Greedy Algorithm

The proposed greedy algorithm obtains an initial solution. The optimum solution of JTPCP has some characteristics regarding the three most important constraints or costs: travel time, MX constraints and preference constraints. First, it is intuitive that the optimal solution tends to minimize the travel time between jobs, in order to satisfy constraints (5.8), and to minimize the total working costs which are closely related to the travel time. Second, the optimal solution should avoid assigning jobs from different subdivisions to a project, because most MX constraints are associated with jobs in different subdivisions. Third, the optimal solution tends to assign a job to its most preferred team type, i.e., the team type with the lowest preference constraint penalty cost. These characteristics allow us to obtain a good initial solution with constructive heuristic algorithm.

We divide the jobs into pools so that jobs in each pool have the same type and subdivision. For each pool, the cheapest-insertion algorithm (Rosenkrantz, et al., 1977) is used to assign jobs to projects, as follows:

Step 1: Let I' be the jobs in the current pool. Let P' be the set of projects for the current pool. Set $P' := \{p\}$, where p is a new project without any jobs.

Step 2: Find an unassigned job $i^* \in I'$ and a project $p^* \in P'$ with the lowest increased travel time after job i^* is assigned to p^*:

$$< i^*, p^* >= \underset{<i,p>\in<I',P'>}{\arg\min}\ c_{\text{add}}(< i, p >),$$

where $c_{\text{add}}(< i, p >)$ is the increased travel time after job i is assigned to project p. Define $c_{\text{add}}(< i, p >) = +\infty$ if constraints (5.8) are violated after the assignment.

Step 3: If $c_{\text{add}}(< i^*, p^* >) = +\infty$, create a new project, assign job i^* to it and add the new project to P'; otherwise assign job i^* to p^*.

Step 4: $I' = I' \setminus \{i^*\}$. Go to Step 2 if $I' \neq \varnothing$.

Step 5: $P = P \cup P'$, i.e., record the projects for the current pool.

The basic idea of the greedy algorithm is as follows: Every project $p \in P$ represents a sequence (or route) of jobs: $i_{p1}, i_{p2}, \ldots, i_{p|I_p|}$ for the maintenance team to follow. During the algorithm, whenever we assign a job to a project, we insert it into a place along the sequence so that the marginal increase of the travel time is minimized.

For jobs distributed over a general network or a continuous plane, the cheapest-insertion algorithm may not be able to obtain a solution with the minimum number of projects and minimum travel time. However, in JTPCP, the jobs in a subdivision are usually distributed along the one-dimensional track line. In those cases, the cheapest-insertion algorithm is able to obtain the optimal solution. This is intuitive. Jobs along a track can be considered as a path where each job is connected to its neighbors from both directions. In an optimal solution, the path is divided into multiple parts by breaking all links longer than T_{travel}, and each part represents a project. The cheapest-insertion algorithm starts from a job as a new project, and keeps adding its adjacent jobs to it, until both its endpoints reach a link longer than T_{travel} or the end of the track. Such procedure can obtain exactly the optimal solution described above.

From the greedy algorithm, we obtain a set of projects. The jobs in each project have the same job type and subdivision. Therefore, the initial solution has very few violations on MX constraints and preference constraints. However, the solution may violate other constraints and may have high working costs. So a local search algorithm and a feasibility algorithm are iteratively applied to improve the solution.

5.3.2 Local Search

The local search algorithm is primarily used to reduce (i) working costs and (ii) violations to limit and project duration constraints. It contains two types of neighborhood structures: (i) job interchange, which removes a job from one project and adds it to another; and (ii) project interchange, which removes a project from a team type and assigns it to another.

In job interchange, a prioritization strategy is used to prioritize the moves in order to improve the solution quality. The algorithm starts removing jobs from the projects whose total job duration contains a small fractional number (in the unit of weeks[13]). Intuitively, removing a job from such projects is more likely to improve the solution. For each project where jobs are to be removed, denoted as p, all other projects are sorted according to their similarities and closeness to p. That is, the algorithm first tries to add the removed jobs to those projects which have the same subdivisions and job types as p, and are spatially close to p. In this way, the algorithm can avoid violating MX and preference constraints or increasing the travel time after the move.

Given p and a project where jobs are to be added, denoted by q, the job interchange algorithm starts removing jobs from one endpoint of p (since a project represents a route of jobs it has two endpoints). If no further improvement can be made, it starts removing jobs from the other endpoint. It always starts from the endpoints, because the maintenance team generally does not want to skip any jobs on the way when it is moving along the track to perform the project. To the contrary, when the algorithm inserts a job into q, it always tries to find the insertion point to minimize the travel time. Figure 25 illustrates an example of job interchange, where a job at the finishing point of project 1 is inserted into project 2.

[13] For example, if the total job duration is 2.7 weeks, 0.7 is the fraction number.

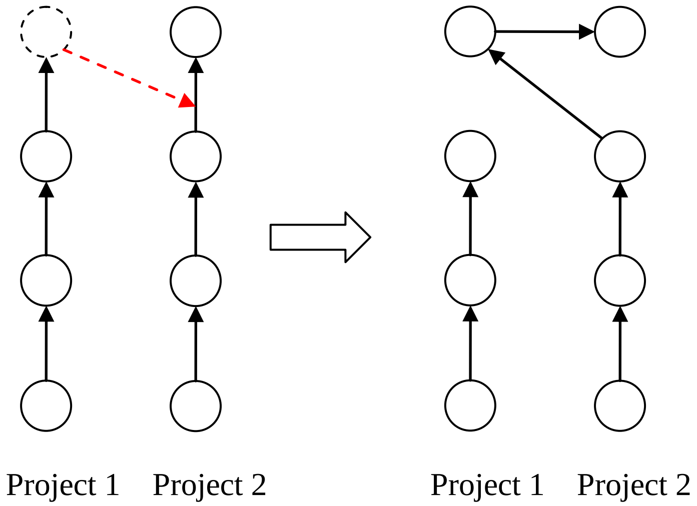


Figure 25. An example of job interchange.

In both job interchange and project interchange, constraints (5.7) are relaxed to explore the neighborhood more extensively. Otherwise, if the total project duration of a team type $k \in K$ reaches T_k, the algorithm will no longer assign more projects to k or insert jobs to projects performed by k --- this is not desirable in the search procedure. Minimum duration constraints are also relaxed for the same reason.

The local search algorithm determines whether the total job duration of a project is rounded up or down based on its value. Let $t_{\text{fraction},p} \in (0,1)$ be the fractional part of the total job duration in a project. If $t_{\text{fraction},p} \le r_{\text{POVT},p}(t_{\text{TJ},p} - t_{\text{fraction},p})$, rounding down the total job duration will be possible. The cost for working on the fractional part is $t_{\text{fraction},p} r_{\text{overtime}} c_{\text{team},k}$ if the total job duration is rounded down, or $1 \times c_{\text{team},k}$ otherwise. Therefore, the algorithm will round down the total job duration if $t_{\text{fraction},p} r_{\text{overtime}} \le 1$, i.e., $t_{\text{fraction},p} \le 1/r_{\text{overtime}}$. Such strategy is usually optimal because most of the benefits of overtime working come from the reduced total project durations.

5.3.3 Feasibility Heuristic

The feasibility heuristic algorithm is used to satisfy constraints (5.7) and minimum duration constraints (5.29)-(5.31), which are relaxed and may be violated during local search. There are two steps in the feasibility heuristic, each for one type of constraints.

The first step is to satisfy constraints (5.7). It iterates on the team types whose total project durations exceed T_k, and tries to assign their projects to other team types whose total project durations are below T_k. Suppose team type $k \in K$ violates constraints (5.7), and the algorithm tries to assign its projects to team type $k' \in K$. Generally, $T_{k'}$ is large enough to accept the excessive project duration from k in our problem instances. Let b_p be the increased costs caused by reassigning project $p \in P_k$ from k to k', or in other words, b_p can be considered as the benefit to keep p in team type k. Note that b_p is always non-negative because otherwise p should have already been reassigned to other team types in the local search.

Then we are facing a 0-1 knapsack problem. In the problem we have a set of projects P_k, each with a benefit b_p and a duration t_p. The objective is to select some projects with maximum total benefit, while the total duration of them do not exceed T_k. This 0-1 knapsack problem is known to be NP-complete (Karp, 1972). It cannot be solved in pseudo-polynomial time using the conventional dynamic programming algorithm which requires the benefits of projects to be integers (Martello and Toth, 1990). However, an exact optimal solution is not necessary in our context, since solving this problem is just a subroutine of the entire heuristic algorithm. So a heuristic method is used to obtain a near-optimal solution. We keep removing the projects from k starting from the ones with the lowest benefit-cost ratio b_p / t_p, until the total project duration in k becomes no larger than T_k.

The second step is to satisfy minimum duration constraints. For a project with a total job duration less than T_{job} , the algorithm first tries to insert all its jobs into another project as long as the cost is reduced. If such an insertion is impossible, the algorithm dismisses the project, and its jobs will be performed by local maintenance teams. Here we do not try to separately insert these jobs into multiple projects, because such an insertion will usually have already been done in the job-interchange algorithm should it be able to reduce costs.

5.4 Case Studies

The proposed model and algorithm are implemented in the Microsoft Visual C++ environment on a personal computer with 2.66GHz dual core CPU and 3 GB RAM. They are applied to solve JTPCP for a Class I railroad company for a recent year, Year A. In the problem instance, more than 2,300 jobs from 2 categories are to be assigned to 5 team types. The number of MX constraints exceeds 94,000.

Even for such a large-scale problem instance, the solution time is less than 10 seconds. All hard constraints are satisfied. Table 6 shows the statistics of the model solution. The first statistic is the percentage of the total duration of jobs assigned to a less preferred team type (i.e., not a team type with the minimum preference constraint penalty cost) in the total job duration. It is related to the total preference constraint penalty costs, $\mathrm{C}_{\mathrm{Pref}}(\cdot)$. The second statistic is proportional to the total costs incurred by transferred jobs, $\mathrm{C}_{\mathrm{overtime}}(\cdot)$. The third and fourth are related to the total working costs and overtime working costs. The last is related to the total MX constraint penalty costs, $\mathrm{C}_{\mathrm{MX}}(\cdot)$. Small values of these statistics indicate low costs and therefore a good solution.

Table 6. Solution statistics for job-to-project clustering (Year A data).

Percentage in the total job duration (%)	Total duration of jobs assigned to a less preferred team type	6.0
	Total duration of jobs transferred to local teams	0.8
	Total rounded-up fractions	11.2
	Total rounded-down fractions	0.4
Percentage in the number of projects (%)	Number of projects violating MX constraints	7.1

The proposed approach is also used to solve JTPCP for a Class I railroad company for a recent year, Year B, when a manual solution obtained by experts is available for comparison. In that problem instance, more than 2,200 jobs from 2 categories are to be assigned to 5 team types. The number of MX constraints exceeds 89,000. The proposed approach solves this instance in less than 10 seconds. The comparison between the model solution and the manual solution is shown in Table 7.

Table 7. Solution statistics for job-to-project clustering (Year B data).

Statistics		*Manual procedure*	*Proposed approach*	*Reduction (%)*
Total project duration (weeks)		-*	-*	11.0
Number of hard constraint violations		437	0	100.0
Percentage in the total job duration (%)	Total duration of jobs assigned to a less preferred team type	16.5	6.9	58.2
	Total duration of jobs transferred to local teams	0.0	1.4	-
	Total rounded-up fractions	11.9	9.7	18.5
	Total rounded-down fractions	0.6	0.4	33.3
Percentage in the number of projects (%)	Number of projects violating MX constraints	13.7	5.9	56.9

*The values are not shown to protect data confidentiality.

It can be seen that the model solution is significantly superior with regard to almost all statistics. In particular, all hard constraints are satisfied. The manual procedure does not have transferred jobs because the data of those jobs are not provided. The model solution involves some transferred jobs, but the penalty costs from them are compensated by the improvements in hard constraints and other penalty costs. The railroad estimates that our model is able to save tens of thousands of dollars in working costs per year. The model also greatly expedites the decision-making procedure (which used to take an expert at least one week).

Glossary of Symbols

$C_{\text{cost item}}(\cdot)$: Total costs of a certain cost item,

$\mathrm{C}_{\text{Lim}}(\cdot)$: Total limit constraint penalty costs,

$c_{\text{Lim},f_{\text{Lim}}k}$: Limit constraint penalty cost for $< f_{\text{Lim}}, k >$,

$\mathrm{C}_{\text{MX}}(\cdot)$: Total mutual exclusion constraint penalty costs,

$c_{\text{MX},ij}$: Mutual exclusion constraint penalty cost for $\{i, j\}_{\text{MX}}$,

$\mathrm{C}_{\text{overtime}}(\cdot)$: Total overtime penalty costs,

$\mathrm{C}_{\text{Pref}}(\cdot)$: Total preference constraint penalty cost,

$c_{\text{Pref},hk}$: Preference constraint penalty cost for $< h, k >_{\text{Pref}}$,

$c_{\text{team},k}$: Costs for a type $k \in K$ team to work for one week,

$\mathrm{C}_{\text{transferred}}(\cdot)$: Total transferred job costs,

$c_{\text{transferred},i}$: Cost incurred by job $i \in I$ if it is transferred,

E: Set of arcs,

E_i: Set of arcs representing the direction to perform job $i \in I$,

$e_{\text{start},p}$: Virtual start arc for project $p \in P$,

$f_{\text{Lim}}(\cdot)$: Function to calculate the quantity of a certain attribute of jobs in a limit constraint,

$< f_{\text{Lim}}, k >$: Pair of function $f_{\text{Lim}}(\cdot)$ and team type $k \in K$ in a limit constraint,

H: Set of job types,

$< h, k >_{\text{Pref}}$: Pair of job type $h \in H$ and team type $k \in K$ in a preference constraint,

H_p: Set of job types in project $p \in P$,

I: Set of jobs,

I_h: Set of type $h \in H$ jobs,

$\{i, j\}_{\text{MX}}$: Set of jobs in a mutual exclusion constraint,

I_p: Set of jobs in project $p \in P$,

i_{po}: The o th performed job in project $p \in P$,

K: Set of team types,

$m_{\text{Lim},f_{\text{Lim}}k}$: Maximum allowed quantity of a certain attribute of jobs in limit constraint $< f_{\text{Lim}}, k >$,

P: Set of projects,

P_k : Set of projects assigned to team type $k \in K$,

$r_{\mathrm{JOVT},i}$: Maximum overtime percentage for job $i \in I$,

r_{overtime} : Ratio between overtime penalty cost and regular working cost,

$r_{\mathrm{POVT},p}$: Maximum overtime percentage for project $p \in P$,

$t_{\mathrm{fraction},p}$: Fractional part of the total job duration in project $p \in P$,

T_{job} : Minimum total job duration in a project,

$t_{\mathrm{job},ik}$: Duration of job $i \in I$ when it is performed by a type $k \in K$ team,

T_k : Maximum total number of work weeks for type $k \in K$ teams,

$T_{\mathrm{min},H'k}$: Minimum project duration for team type $k \in K$ and job types $H' \subseteq H$,

$T_{\mathrm{max},H'k}$: Maximum project duration for team type $k \in K$ and job types $H' \subseteq H$,

$t_{\mathrm{overtime},p}$: Overtime working duration for project $p \in P$,

t_p : Duration of project $p \in P$,

$t_{\mathrm{TJ},p}$: Total job duration in project $p \in P$,

T_{travel} : Maximum allowed travel time,

$t_{\mathrm{travel},e_1e_2}$: Travel time from the head of arc $e_1 \in E$ to the tail of arc $e_2 \in E$,

$\mathbf{u} = \left\{u_i\right\}$: Variables for the relative sequence in which a job is performed in a project,

$\mathbf{x} = \left\{x_{e_1e_2p}\right\}$: Binary variables for the route of a project,

y_{ip} : Binary variables indicating if job $i \in I$ is assigned to project $p \in P$.

CHAPTER 6 CONCLUSION AND FUTURE RESEARCH

6.1 Conclusion

This Ph.D. research focuses on three interrelated optimization problems on railroad track maintenance: track inspection scheduling problem (TISP), production team scheduling problem (PTSP), and job-to-project clustering problem (JTPCP). All these problems have difficult side constraints and have very large-scale problem instances in practice. However, little research has been conducted on these problems, and the railroad companies mostly solve them manually solely based on the experience and knowledge of experts.

We first proposed a vehicle routing problem (VRP) model for TISP, and addressed many side constraints (some of which had never been studied before), such as the periodicity constraints, network topology constraints and discrete working time constraints. We developed a customized heuristic algorithm to solve the model by iteratively applying a constructive heuristic and a local search in an incremental scheduling horizon framework. The proposed modeling and solution approaches have been applied to real-world problem instances for both short-term scheduling and long-term planning. For the short-term scheduling problem, our numerical case study has shown that the proposed approach is able to obtain a much better solution than the manual procedure in a very short time. For the long-term planning problem (which cannot be solved manually), we conducted what-if analyses to draw managerial insights that could possibly help the railroad improve its operational efficiency.

Then we addressed the PTSP problem by formulating it as a time-space network (TSN) model with many types of side constraints (such as mutual exclusion constraints, consecution constraints and relay rail constraints). A simplified scheduling model was used to obtain an initial solution while two local search algorithms (decomposition and restriction (D&R), and block interchange) were used to iteratively improve the solution. Many additional solution techniques, such as data reduction, augmented cost function and subproblem prioritization, were developed to improve the computational performance of the proposed algorithm. The proposed approach has been applied to real-world problem instances and the solutions were able to satisfy all hard constraints and most soft constraints. The comparison with previous results has further indicated how our model significantly outperforms not only the manual solution practice in the industry, but also the earlier approach proposed by Peng et al. (2010).

We have also developed a mathematical model and solution algorithm for JTPCP. The formulated VRP-based model includes very difficult side constraints such as limit constraints and project duration constraints. A customized heuristic algorithm has been proposed to solve the model efficiently. The algorithm includes a constructive heuristic, a local search and a feasibility heuristic. Application of our model to real-world problem instances has shown that the proposed approach outperforms manual solution approach in terms of both solution quality and solution speed.

In the past few years, the proposed models and algorithms for all three problems have been adopted by a Class I railroad to help their practical operations. Our efforts have been shown to bring large cost savings and efficiency improvements. We are optimistic, therefore, that the ideas and methods developed in this dissertation shall also be useful to routing and scheduling problems in other contexts, such as railroad train scheduling, highway transportation scheduling and airline scheduling.

6.2 Future Research

The proposed approach for TISP has been implemented in practice to solve a real-world rail inspection scheduling problem (RISP). We plan to also apply it to a geometry inspection scheduling problem (GISP), which, despite some similar features, has very different input data and network topology constraints. We will continue working with the railroad industry to ensure that the proposed methods are suitable for GISP in practice.

We will work on improving the existing models and solution algorithms for TISP, PTSP and JTPCP. More side constraints will be incorporated into the models in order to better meet industry needs. There are many uncertainties associated with track inspection and maintenance activities (e.g., travel delay, equipment break-downs). Currently such uncertainties are addressed by re-solving the model with a new planning horizon whenever events deviate from the original plan. However, a more systematic approach (e.g., stochastic programming models with recourse) could be developed to better address the uncertainties.

There are also potential improvements on the algorithm side. Because of the large scale and high complexity of the studied problems, local search algorithms are applied to all of them. There are two possible ways to improve the local search algorithms. First, more neighborhood structures shall be explored. For example, the development of the PTSP algorithm has experienced four stages: (i) only D&R was used but no project could be exchanged between different teams in a subproblem; (ii) projects were allowed to be exchanged between different teams in D&R search; (iii) an additional search, block interchange, was added to the algorithm, but a block could contain only one project; (iv) a block was allowed to contain multiple projects in block interchange. In every stage we explored new neighborhood structure and observed significant improvements in solution quality. There are potentially more neighborhood structures which can improve the algorithm. For example, in the task-interchange search for TISP, an interchanged block contains at most one task. We may possibly improve the algorithm by allowing blocks with more than one task. In the D&R search for PTSP, subproblems are created based on team-wise decomposition. We could possibly improve the algorithm by allowing week-wise decomposed subproblems.

Furthermore, it might be worthwhile to develop more effective strategies to prioritize the moves in local search. In the D&R algorithm for PTSP, the prioritization of the subproblems relies on some ad-hoc parameters which may not be effective. One possible way to improve the algorithm is to develop a self-adaptive learning strategy. In such strategy, the values of parameters can be initially calculated based on the algorithm history logs from previous data sets, and then they are updated systematically and endogenously during the algorithm based on the log from the current data set. Such strategy could possibly identify more promising moves. The prioritization strategy can also be applied to improve task interchanges in TISP (where exhaustive local search is currently used) and block interchange in PTSP (where randomized local search is currently used).

In studied problems, the quality of the initial solution is important, because local search tends to be stuck at local optima for large problem instances with many side constraints. Hence the improvement of constructive heuristics may be beneficial. In TISP, the greedy algorithm can be improved by allowing inserting a new task anywhere in the current schedule, but not always at the end of a route.[14] It is also possible to develop better criteria to select new tasks to add. For example, currently the algorithm ranks the tasks based on periodicity constraint penalty cost, which is generally related to the overdue percentage. A possible improvement is to also take the

[14] In the VRP literature, similar ideas have led to the nearest-insertion, farthest-insertion, and cheapest-insertion heuristics (Rosenkrantz, et al., 1977).

preferred interval and task duration into consideration. For tasks with similar overdue percentages, those with shorter preferred intervals should be performed first, because their periodicity constraint penalty costs increase very fast. Those with shorter task durations should also have higher priorities, because they are less likely to delay other tasks. In PTSP, the initial solution is obtained by a scheduling model which only considers time window and preference constraints. A possible improvement is to decompose the problem so that more difficult constraints such as mutual exclusion and simultaneity constraints can also be considered.

The mixed integer programming (MIP) algorithm (used as a subroutine in PTSP) may be further improved by utilizing more advanced MIP solution techniques, such as row generation/Benders decomposition, column generation/Dantzig-Wolfe decomposition and Lagrange relaxation.

Besides the three optimization problems studied in this dissertation research, other problems on railroad track maintenance, such as relay rail sourcing problem and rail grinding scheduling problem, are all interesting and relevant topics for future research. It may be beneficial to further establish an integrated decision-making framework for both track maintenance and train operations. For example, with such a framework, the train schedules and the maintenance schedules would be better coordinated to enhance the overall system operational efficiency.

REFERENCES

1. Ahuja, R.K., Cunha, C.B. and Sahin, G. (2005). Network models in railroad planning and scheduling, *Tutorials in Operations Research*, 1, 54–101.

2. Alonso, F., Alvarez, M.J. and Beasley, J.E. (2008). A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions. *Journal of the Operational Research Society*, 59, 963–976.

3. Acharya, D., Sussman, J. and Martland, C. (1990). Application of expert systems and network optimization techniques in rail relay scheduling. *Journal of Transportation Research Forum*, 31(1), 1–16.

4. Association of American Railroads. (2010). *Railroad Statistics*. Available online at http://www.aar.org/~/media/aar/Industry%20Info/AAR%20Stats%202010%201123.ashx. Accessed on December 5, 2010.

5. Balakrishnan, N. and Wong, R.T. (1990). A network model for the rotating workforce scheduling problem. *Networks*, 20, 25–42.

6. Beck, J.C., Prosser, P. and Selensky, E. (2002). On the reformulation of vehicle routing probelms and scheduling problems. In *LNAI 2371, Proceedings of the Symposium on Abstraction, Reformulation and Approximation (SARA 2002)*, 282–289.

7. Beck, J.C., Prosser, P. and Selensky, E. (2003). Vehicle routing and job shop scheduling: What's the difference. Proceedings of *the 13th International Conference on Artificial Intelligence Planning and Scheduling*.

8. Beltrami, E.J. and Bodin, L.D. (1974). Networks and vehicle routing for municipal waste collection. *Networks*, 4(1), 65–94.

9. Benders, J.F. (1962). Partitioning procedures for solving mixed-variables programming problems.

Numerische Mathematik, 4, 238–252.

10. Bog, S., Nemani, A.K. and Ahuja, R.K. (2010). Iterative algorithms for the curfew planning problem. *Journal of the Operational Research Society*, online, 1–15.

11. Braysy, O. and Gendreau, M. (2005). Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms. *Transportation Science*, 39(1), 104-118.

12. Budai, G., Huisman, D. and Dekker, R. (2006). Scheduling preventive railway maintenance activities. *Journal of Operational Research Society*, 57, 1035–1044.

13. Burns, D. and Franke, M. (2005). Analyzing the blitz approach to m/w. *Railway Track and Structures*, 101(10), 33-38.

14. Cannon, D.F., Edel, K.-O., Grassie, S.L. and Sawley, K. (2003). Rail defects: An overview. *Fracture & Fatigue of Engineering Material & Structures*, 26(10), 865–886.

15. Chardaire, P., McKeown, G.P., Verity-Harrison, S.A. and Richardson, S.B. (2005). Solving a time-space network formulation for the convoy movement problem. *Operations Research*, 53(2), 219–230.

16. Cheu, R.L., Wang, Y. and Fwa, T.F. (2004). Genetic algorithm-simulation methodology for pavement maintenance scheduling. *Computer-Aided Civil and Infrastructure Engineering*. 19(6), 446–55.

17. Cheung, B.S.N., Chow, K.P., Hui, L.C.K. and Yong, A.M.K. (1999). Railway track possession assignment using constraint satisfaction. *Engineering Applications of Artificial Intelligence*, 12(5), 599–611.

18. Christofides, N. and Beasley, J.E. (1984). The period routing problem. *Networks*, 14(2), 237–256.

19. Coene, S., Arnout, A. and Spieksma, F.C.R. (2010). On a periodic vehicle routing problem. *Journal of the Operational Research Society*, 61, 1719–1728.

20. Dantzig, G.B. and Ramser, J.H. (1959). The truck dispatching problem. *Management Science*, 6(1), 80–91.

21. Dantzig, G.B. and Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research*, 8, 101-111.

22. De Jong, C., Kant, G. and Van Vliet, A. (1996). On finding minimal route duration in the vehicle routing problem with multiple time windows. Manuscript, Department of Computer Science, Utrecht University, The Netherlands. (Available online at http://www.cs.uu.nl/research/projects/alcom/wp4.3.html. Accessed on January 18, 2011.)

23. Derinkuyu, K., Balakrishnan, A. and Morales J. (2010). Scheduling rail grinding operations. Presented in *INFORMS Annual Meeting 2010*, Austin, TX.

24. Desrochers, M., Desrosiers, J. and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2), 342–354.

25. Dridi, L., Parizeau, M., Mailhot, A. and Villeneuve, J.P. (2008). Using Evolutionary Optimization Techniques for Scheduling Water Pipe Renewal Considering a Short Planning Horizon. *Computer-Aided Civil and Infrastructure Engineering*, 23(8), 625–635.

26. Durango-Cohen, P. and Madanat, S. (2008). Optimization of inspection and maintenance decisions for infrastructure facilities under performance model uncertainty: A Quasi-Bayes approach, *Transportation Research Part A*. 42, 1074–1085.

27. Federal Railroad Administration. (1999). Improving railroad safety and rail passenger technology through targeted research and demonstrations 1992-1997. *Technical report, DOT/FRA/ORD-99/02*, Washington, D.C.

28. Federal Railroad Administration. (2002). *Track safety standards compliance manual*. Available online at http://www.fra.dot.gov/Pages/460.shtml. Accessed on January 18, 2011.

29. Federal Railroad Administration Office of Safety Analysis. (2010). Available online at http://safetydata.fra.dot.gov/officeofsafety/. Accessed on December 5, 2010.

30. Fisher, M. L. (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27, 1–18.

31. Florian, M., Bushell, G. and Ferland, J. (1976). The engine scheduling problem in a rail network. *INFOR*, 14, 121–138.

32. Floyd, R.W. (1962). Algorithm 97: Shortest Path. *Communications of the ACM*, 5(6), 345.

33. Ford, L.R. and Fulkerson, D.R. (1958). A suggested computation for maximal multicommodity network flows. *Management Science*, 5, 97-101.

34. Francis, P.M., Smilowitz, K.R. and Tzur, M. (2006). The period vehicle routing problem with service choice. *Transportation Science*, 40(4), 439–454.

35. Francis, P., Zhang, G. and Smilowitz, K. (2007). Improved modeling and solution methods for the multi-resource routing problem. *European Journal of Operational Research*, 180, 1045–1059.

36. Francis, P.M., Smilowitz, K.R. and Tzur, M. (2008). The period vehicle routing problem and its extensions. In Golden, B. et al. (eds.), *The vehicle routing problem: Latest advances and new challenges*, Springer.

37. Funke, B., Grunert, T. and Irnich, S. (2005). Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics*, 11, 267-306.

38. Gendreau, M., Hertz, A. and Laporte, G. (1992). A new insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40, 1086−1093.

39. Gendreau, M., Potvin, J.-Y., Braysy, O., Hasle, G. and Lokketangen, A. (2008). Metaheuristics for the vehicle routing problem and its extensions: A categorized bibliography. In Golden, B. et al. (eds.), *The vehicle routing problem: Latest advances and new challenges*, Springer.

40. Glover, F. (1992). New ejection chain and alternating path methods for traveling salesman problems. In: Balci, O., Sharda, R., and Zenios, S. (eds), *Computer Science and Operations Research: New*

Developments in Their Interfaces, 449−509, Pergamon Press, Oxford.

41. Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer, Norwell, MA.

42. Golabi, K. and Pereira, P. (2003). An innovative pavement management and planning system for the road network of Portugal. *ASCE Journal of Infrastructure Systems*, 9(2), 75–80.

43. Gorman, M.F. and Kanet, J.J. (2010). Formulation and solution approaches to the rail maintenance production gang scheduling problem. *Journal of Transportation Engineering*, 136(8), 701–708.

44. Grimes G.A. and Barkan, C.P.L. (2006). Cost-effectiveness of railway infrastructure renewal maintenance. *Journal of Transportation Engineering*, 132(8), 601–608.

45. Halse, K. (1992). Modeling and solving complex vehicle routing problems. *Ph.D. Thesis*, Technical University of Denmark.

46. Hane, C., Barnhart, C., Johnson, E.L., Marsten, R.E., Nemhauser, G.L. and Sigismondi, G. (1995). The fleet assignment problem: Solving a large integer program. *Mathematical Programming*, 70(2), 211–232.

47. Hansen, P. and Mladenovic, N. (2005). Variable neighborhood search. In: *Search methodologies: Introductory tutorials in optimization and decision*. Springer.

48. Higgins, A. (1998). Scheduling of railway track maintenance activities and crews. *Journal of Operational Research Society*, 49, 1026–1033.

49. Higgins, A., Ferreira, L. and Lake, M. (1999). Scheduling rail track maintenance to minimise overall delays. In *Proceedings of the 14th International Symposium on Transportation and Traffic Theroy*, Jerusalem, Israel.

50. Hochbaum D.S. (1982). Heuristics for the fixed cost median problem. *Mathematical Programming*, 22, 148-162.

51. Ibaraki, T., Imahori, S., Kubo, M., Masuda, T., Uno, T. and Yagiura, M. (2005). Effective local search

algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science*, 39(2), 206-232.

52. ILOG. (2006). *ILOG CPLEX 10.0 User's Manual*.

53. Ioannou, G., Kritikos, M. and Prastacos, G. (2001). A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society*, 52, 523−537.

54. Jarrah, A.I.Z., Yu, G., Krishnamurthy, N. and Rakshit, A. (1993). A decision support framework for airline flight cancellations and delays. *Transportation Science*, 27(3), 266-280.

55. Jiang, X. and Adeli, H. (2004). Object-oriented model for freeway work zone capacity and queue delay estimation. *Computer-Aided Civil and Infrastructure Engineering*, 19(2), 144-56.

56. Kallehauge, B., Larsen, J., Madsen, O.B.G. and Solomon, M.M. (2005) Vehicle routing problem with time windows. In: Desaulniers, G., Desrosiers, J. and Solomon, M.M. (eds.), *Column Generation*, Springer, NY.

57. Kallehauge, B. (2008). Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers and Operations Research*, 35, 2307-2330.

58. Karp, R. (1972). Reducibility among combinatorial problems. In *Proceedings of a Symposium on the Complexity of Computer Computations*, Plenum Press.

59. Kliewer, N., Mellouli, T., and Suhl, L. (2006). A time-space network based exact optimization model for multi-depot bus scheduling. *European Journal of Operational Research*, 175, 1616-1627.

60. Kohl, N. and Madsen, O.B.G. (1997). An optimization algorithm for the vehicle routing problem with time windows based on Lagrangian relaxation. *Operations Research*, 45, 395-403.

61. Kwon, O.K., Martland, C.D. and Sussman, J.M. (1998). Routing and scheduling temporal and heterogeneous freightcar traffic on rail networks. *Transportation Research Part E*, 34(2) 101–115.

62. Lake, M., Ferreira, L. and Kozan, E. (2001). Heuristic techniques for scheduling railway track

maintenance. In: Kozan and Ohuchi (eds.), *Operations Research, Management Science at Work: Applying Theory in the Area Pacific Region, The International Series in Operations Research & Management Science*, Kluwer Academic Publishers.

63. Laporte, G. (1992). The Vehicle Routing Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3), 345-358.

64. Lee, C.Y., Lei, L. and Pinedo, M. (1997). Current trends in deterministic scheduling. *Annals of Operations Research*, 70, 1-41.

65. Leung, J.Y-T. (Ed.). (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Boca Raton, FL.

66. Li, G., Balakrishnan, A. and Roth, B. (2009). Effectively solving production gang scheduling problem for railway track maintenance projects. Presented in *INFORMS Annual Meeting 2009*, San Diego.

67. Li, Z., Madanu, S., Abbas, M. and Zhou, B. (2010). A Heuristic Approach for Selecting Highway Investment Alternatives. *Computer-Aided Civil and Infrastructure Engineering*, 25(6), 1-13.

68. Ma, W., Cheu, R.L. and Lee, D.H. (2004). Scheduling of lane closures using genetic algorithms with traffic assignments and distributed simulations. *Journal of Transportation Engineering*. 130(3), 322–329.

69. Martello, S. and Toth, P. (1990). *Knapsack Problems: Algorithms and Computer Implementations*. Wiley, Chichester, UK.

70. Morales, J., Balakrishnan, A. and Li, G. (2008). Optimal Routing of Geometry Cars. Presented in *INFORMS Annual Meeting 2008*, Washington, D.C.

71. Nemani, A.K., Bog, S. and Ahuja, R.K. (2010). Solving the Curfew Planning Problem. *Transportation Science*, 44(4), 506-523.

72. Ng, M., Lin, D.Y. and Waller, S.T. (2009). Optimal Long-Term Infrastructure Maintenance Planning Accounting for Traffic Dynamics. *Computer-Aided Civil and Infrastructure Engineering*, 24, 459–469.

73. Or, I. (1976). Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. *Ph.D. thesis*, Northwestern University, Evanston, Illinois.

74. Osman, I.H. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. *Annals of Operations Research*, 41, 421−452.

75. Ouyang, Y. (2007). Pavement resurfacing planning for highway networks: A parametric policy iteration approach. *Journal of Infrastructure Systems (ASCE)*, 13(1), 65-71.

76. Ouyang, Y., Li, X., Lai, Y.C., Barkan, C. and Kawprasert, A. (2009). Optimal locations of railroad wayside defect detection installations. *Computer-aided Civil and Infrastructure Engineering*, 24, 309-319.

77. Ouyang, Y. and Madanat, S. (2006). An analytical solution for the finite-horizon pavement resurfacing planning problem. *Transportation Research Part B*, 40(9), 767-778.

78. Ouyang, Y. and Madanat, S. (2004). Optimal scheduling of rehabilitation activities for multiple pavement facilities: exact and approximate solutions. *Transportation Research Part A*, 38(5), 347–65.

79. Peng, F., Kang, S., Li, X., Ouyang, Y., Somani, K. and Acharya, D. (2010). A heuristic approach to railroad track maintenance scheduling problem. *Computer-Aided Civil and Infrastructure Engineering*. In press.

80. Pinedo, M. (1995). *Scheduling: Theory, Algorithms and Systems*. PrenticeHall.

81. Potvin, J.-Y. and Rousseau, J.-M. (1993). A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66, 331−340.

82. Potvin, J.-Y. and Rousseau, J.-M. (1995). An exchange heuristic for routing problems with time windows, *Journal of the Operational Research Society*, 46, 1433−1446.

83. Retharekar, A. and Mobasher, A. (2010). Routing and scheduling preventive railway maintenance with predetermined desired reliability. Presented in *INFORMS Annual Meeting 2010*, Austin, TX.

84. Rosenkrantz, D., Sterns, R. and Lewis, P. (1977). An analysis of several heuristics for the travelling salesman problem, SIAM Journal on Computing, 6, 563–581.

85. Russell, R. (1977). An effective heuristic for the M-tour traveling salesman problem with some side conditions. *Operations Research*, 25, 517−524.

86. Savelsbergh, M.W.P. (1992). The vehicle routing problem with time windows: Minimizing route duration. *Journal on Computing*, 4, 146−154.

87. Schafer, D. and Barkan, C.P.L. (2008). A prediction model for broken rails and an analysis of their economic impact, In *Proceedings of the 2008 AREMA Conference*, Salt Lake City, Utah.

88. Schlake, B.W., Barkan, C.P.L. and Edwards, J.R. (2010). Impact of automated inspection technology on unit train performance. In *Proceedings of the 2010 Joint Rail Conference*, Urbana, Illinois.

89. Schlake, B.W., Edwards, J.R., Hart, J.M., Barkan, C.P.L, Todorovic, S., and Ahuja, N. (2009). Automated Inspection of Railcar Underbody Structural Components Using Machine Vision Technology. In *Proceedings of the TRB 88th Annual Meeting*, Washington, DC.

90. Schmid, V., Doerner, K.F., Hartl, R.F. and Salazar-Gonz ález, J.J. (2010). Hybridization of very large neighborhood search for ready-mixed concrete delivery problems. *Computers & Operations Research*, 37, 559-574.

91. Selensky, E. (2001). On mutual reformulation of shop scheduling and vehicle routing. In *Proceedings of the 20th UKPLANSIG*, 282–291.

92. Simson, S., Ferreira, L. and Murray, M. (1999). Modeling rail track maintenance planning using

simulation. In *Proceedings of the 15th National Conference of the Australian Society for Operations Research*, Australian Society for Operations Research, Gold Coast, Brisbane, 1159–1172.

93. Solomon, M.M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35, 254-265.

94. Steinzen, I., Gintner, V., Suhl, L. and Kliewer, N. (2010). A time-space network approach for the integrated vehicle- and crew-scheduling problem with multiple depots. *Transportation Science*, 44(3), 367-382.

95. Taillard, E., Badeau, P., Gendreau, M., Guertin, F. and Potvin, J.-Y. (1997). A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31, 170−186.

96. Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. Wiley.

97. Thengvall, B.G., Bard, J.F. and Yu, G. (2000). Balancing user preferences for aircraft schedule recovery during irregular operations. *IIE Transportations*, 32(3), 181-193.

98. Thompson, P.M. and Psaraftis, H.N. (1993). Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41, 935−946.

99. Tien, J.M. and Kamiyama, A. (1982). On manpower scheduling algorithms. *SIAM Review*, 24, 275-287.

100. Tolliver, D. and Benson, D. (2010). Freight railway track maintenance cost model. *Technical report, MPC-357*, Federal Highway Administration, Washington, D.C.

101. Toth, P. and Vigo, D. (eds.). (2001). *The Vehicle Routing Problem*. SIAM, Philadelphia, PA.

102. Unnikrishnan, A., Valsaraj, V., Damnjanovic, I. and Waller, S.T. (2009). Design and Management Strategies for Mixed Public Private Transportation Networks: A Meta-Heuristic approach. *Computer-Aided Civil and Infrastructure Engineering*, 24(4), 266-279.

103. Yan, S., Lai, W. and Chen, M. (2008). Production scheduling and truck dispatching of ready mixed

concrete. *Transportation Research Part E*, 44, 164-179.

104. Zante-de Fokkert, J.I. van, Hertog, D. den, Berg, F.J. van den, and Verhoeven, J.H.M. (2007). The Netherlands schedules track maintenance to improve track workers' safety. *Interfaces*, 37(2), 133-142.