The Dissertation Committee for Sifeng Lin Certifies that this is the approved version of the following dissertation:

Optimization Models and Methods for Transportation Services

Committee:

Anantaram Balakrishnan, Supervisor

Jonathan F. Bard, Co-Supervisor

Prakash Mirchandani

John J. Hasenbein

Nedialko Dimitrov

Optimization Models and Methods for Transportation Services

by

Sifeng Lin, B.E.; M.S.E.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

The University of Texas at Austin

August 2015

Acknowledgements

I am deeply grateful to my advisors, Professor Anant Balakrishnan and Professor Jonathan Bard for their patience, help, and support. I would also like to offer my sincere thanks to Professor Prakash Mirchandani for his guidance in the "Network Design Problem with Service Requirement", as well as in writing this dissertation. Besides, I would like to thank Professor John Hasenbein and Professor Ned Dimitrov to serve on my committee. I am very thankful to Dr. Alper Uygur for his support on the train dispatching problem. Finally, I deeply appreciate the endless support, love, caring and understanding from Lyn Chen.

Optimization Models and Methods for Transportation Services

Sifeng Lin, Ph.D.

The University of Texas at Austin, 2015

Supervisor: Anantaram Balakrishnan

Co-Supervisor: Jonathan F. Bard

Managing transportation services efficiently is essential to both public and private sectors. This dissertation addresses three scheduling problems in modern transportation systems: the network design problem, the train dispatching problem, and the service route design problem. The transportation network design problem with service requirements designs arcs on a directed network and route commodities on the designed arcs so that i) commodities satisfy service requirements and ii) the total cost is minimized. We develop three mathematical programming models: a compact but weak arc-flow formulation, a large but strong path-flow formulation, and a hybrid formulation that uses both the arc-flow and the path-flow representations. We show that the hybrid formulation can significantly strengthen the LP formulation without introducing many variables. To find a good hybrid formulation, we develop columnization and decolumnization algorithms that uses the LP relaxation information to identify commodities that should use the path-flow representation. We also develop valid inequalities for commodities using the path-flow representation. The train dispatching problem schedules the movements of trains on scarce railroad tracks so as to improve the average velocity of trains. We develop a mathematical programming model and

strengthen the model using valid inequalities. Besides, we present a heuristic to find a feasible solution quickly, which can serve as the warm-start solution to the MIP solver. For the third problem, we seek to design vehicle routes to deliver and pickup orders for a major grocery chain. We design a GRASP that can incorporate various operational requirements, including warehouse loading capacity, loading sequence, time window requirements, truck volume and weight capacities, and driver time limits. Our GRASP procedure consists of two phases: the solution construction (Phase I) and the Tabu search (Phase II). We show that the neighborhood structure of solutions is highly degenerate, which limits the solution space explored by the Tabu search. We apply the Tabu search with random variable neighborhood to increase the solution space explored.

Table of Contents

List of Tables

List of Figures

Chapter 1. Introduction

The effective and efficient management of transportation services is essential to the distribution of goods and services. With increased complexities of modern business operations, a firm with decision support systems for transportation planning can obtain significant competitive advantage. The objective of this research is to examine three important problems in transportation planning. Specifically, we develop mathematical models and solution methods for (i) the train dispatching problem in the railroad industry, (ii) the transportation network design problem with end-to-end service constraints, and (iii) the route design for delivery vehicles with backhauling.

We first study the train dispatching problem, which is one of the top priorities for railway companies. The train dispatching problem aims to optimize the movement of freight trains on railway tracks to reduce train waiting times and increase the average train velocity. To model the problem, we discretize time into periods and model movements of trains using arcs in the time space network. Our model incorporates various service requirements, including time window constraints, headways between consecutive trains, maintenance of way, and train priorities. To solve the resulting integer program, we explore several model enhancement strategies and propose a sequential routing heuristic to generate an initial feasible solution. We test our solution strategies using real-life data from a Class I railroad company.

Our second problem is the transportation network design problem with service constraints (NDSR). Since infrastructure networks are usually capital-intensive, minimizing the cost has been a major concern in transportation network design. However, increased global competition has forced firms to take the responsiveness and reliability into account. To address this problem, Balakrishnan et al. (2014) propose the

network design problem with service requirement. On top of the traditional fixed-charge multi-commodity network design representation, the NDSR problem incorporates additional constraints to ensure that each commodity's route satisfies various service requirements. We present three formulations for this problem: the weak but compact arc-flow formulation, the strong but large path-flow formulation, and a hybrid formulation that applies the arc-flow representation to some commodities and the path-flow representation to others. To identify a hybrid formulation that achieves the strength of path-flow formulation and the compactness of the arc-flow formulation, we propose two strategies, both of which decide path-flow commodities by iteratively solving the LP relaxation. To further improve the performance of the hybrid formulation, we develop valid inequalities based on the path-flow representation.

The third problem, the route design for delivery vehicles with backhauling, extends the traditional capacitated vehicle routing problem with time window (CVRPTW) to capture additional sequencing and warehouse capacity considerations. The problem entails routing vehicles to deliver replenishment orders from the warehouse to stores and pick up salvage orders from stores back to the warehouse, while considering the loading capacity at the warehouse, loading and unloading time at each store, sequencing of different types of items on the truck trailer, and volume and weight capacities of the trailer. Since size of the problem is too large to apply exact solution method, we use the greedy randomized adaptive search procedure to solve the problem.

The rest of this proposal is organized as follows. In Chapter 2, we show the modeling and solution strategies for train dispatching. Chapter 3 discusses the weight-constraint network design problem. Chapter 4 presents the store servicing routes design problem and our solution strategy. Chapter 5 concludes this dissertation.

Chapter 2. Network Design with Service Constraints

2.1. INTRODUCTION

Transportation networks play a critical role in both private and public sectors. Since building such infrastructure networks is usually capital-intensive, network designers have traditionally adopted strategies that emphasize cost minimization. A consequence of an undue focus on cost minimization is that the optimal design ends up being sparse—which, in turn, entails long routes for some commodities which may result in transportation delays and unresponsive supply chains. A singular focus on cost can thus have adverse consequences along other criteria.

With increasing global competition, shorter product life-cycles and more demanding customers, firms are being forced to take into account additional factors such as responsiveness and reliability while making network design decisions. To address this problem, Balakrishnan et al. (2014) proposed an extension of the traditional fixed-charge, multi-commodity network design problem. Their model incorporates additional constraints to ensure that each commodity's chosen route satisfies various service requirements. The service requirements are modeled by assigning weights to the arcs and ensuring that the sum (or product) of the arc weights for each commodity does not exceed the specified limit. The goal of this paper is to develop a new solution method for this important problem, which we refer to as the Network Design with Service Requirements (NDSR) problem.

We can use two opposite approaches for modeling the NDSR problem. The first one defines arc flow variables to represent the commodity flow on arcs and enforces flow conservation equations to ensure that flows constitute a path, as desired, from the commodity's origin node to its destination node. The advantages of such formulation are obvious: it is compact (in the sense that the model size is polynomial) and it can

easily incorporate additional constraints like the service, or weight—requirements that exist in the NDSR problem. In the NDSR context, such a formulation has a weak linear programming relaxation since its solution permits the weight constraint to be satisfied on average; in other words, the decomposition of the LP optimal solution may incur origin-destination flow along paths that do not satisfy one or more weight constraints.

Alternatively, we can model the NDSR problem using path flows. In this case, we enumerate all feasible paths for each commodity, i.e., all paths from the commodity's origin node to its destination node whose weights for each service metric are within the specified limits, and explicitly model the commodity flows on these feasible paths. By excluding the infeasible paths that violate any of the service requirements, this path-flow formulation provides a tighter linear programming relaxation; however, since the number of feasible paths may be exponential, solving the resulting formulation can be computationally difficult.

To synergistically exploit the compactness of the arc-based formulation and tightness of the path-based formulation, we propose a hybrid formulation that uses the arc-flow representation for some commodities and the path-flow representation for others. We show that the commodities with stringent service requirements favor the path-flow representation, while arc-flow representation is preferred for commodities with loose service requirements. However, instead of deciding a priori as to whether a commodity should have the arc-based or the path-based representation, we use the information contained in the optimal solutions of linear programming relaxations of continuously evolving formulations in the solution process. Thus, rather than using a static codification, we categorize the commodities dynamically, adapting not only to the original network structure but also to the information from progressively stronger linear programming relaxations.

To decide the representation of each commodity based on the information in the linear programming relaxation solutions, we use two methods that are respectively grounded in estimated bound improvement and in the flow decomposition. These procedures help find a hybrid formulation that has only slightly larger number of variables than the pure arc-flow representation but only slightly worse bound than the pure path-flow representation.

For the commodities using the path-flow representation, we develop new valid inequalities that help us tighten the hybrid formulation. The linear programming solution of the path-flow representation contains a decomposition of commodity flow into paths; this information facilitates taking unions and intersections of the paths to develop these new valid inequalities.

This study advances current state of knowledge in several aspects. First, it provides a novel approach to effectively solve the NSDR problem. Second, it introduces the idea of hybrid formulation, which may be extendable to other problem contexts that have a block-diagonal structure. Third, this paper uses the information contained in the path-flow representation of commodity flows to help define new valid inequalities that strengthen the hybrid model. Finally, the computations demonstrate that our solution approach based on the new hybrid formulation is effective, and that the Pareto principle applies in this context as well: using the path-based representation for just a small proportion of the commodities results in successfully closing a large proportion of the integrality gap.

The rest of this chapter is organized as follows. Section 2.2 introduces the NDSR problem, reviews the arc-flow formulation discussed in Balakrishnan et al. (2014), presents the path-flow formulation, and introduces the hybrid formulation and our dynamic reformulation strategies. In Section 2.3, we develop the new valid inequalities

based on path flow representation, as well as reviewing some of the valid inequalities discussed in Balakrishnan et al. (2014). Section 2.4 presents the computational results and Section 2.5 provides the concluding remarks.

2.2. MODEL FORMULATION

In the Network Design with Service Requirements (*NDSR*) problem, we are given a directed network with a set of available point-to-point links, and a set of commodities with their corresponding origins and destinations and service requirements. The service requirements can model the allowable end-to-end delay, reliability, and number of arcs or nodes traversed by the commodity. We seek to select a subset of arcs of the given network and route commodities on origin-destination paths along the selected arcs to minimiz the sum of the fixed costs for selecting the arcs and the variable costs for routing the commodities while meeting the service requirements. Specifically, consider a directed network G: (N, A) with the node set N = {1, 2, …, n} representing the origin nodes, destination nodes or transshipments points for the commodities, and arc set A = $\{(i, j): i, j \in N\}$, with $|A| = m$, representing facilities or possible interconnections for routing flows. Let K represent the set of commodities, and for each commodity $k \in K$, let s_k denote its origin node and t_k its destination node. We wish to route the commodities in K on simple paths from their respective source nodes to their destination nodes using the selected arcs. Since the arcs are uncapacitated, we can normalize the demand of each commodity to one. Let $f_{ij} \geq 0$ denote the fixed cost of selecting arc (i, j), and $c_{ij}^k \geq 0$ denote the cost to route each unit of commodity k on that arc. Each commodity has up to L service requirements, indexed by l, that any feasible path for the commodity must satisfy. For each service requirement l and arc (i, j), we associate a nonnegative weight $w_{ij}^{k,l}$ corresponding to commodity k. The sum of the weights of the

arcs on the selected path for commodity k and service requirement l must not exceed the specified upper limit on the weight $W^{k,l}$. This model also applies in situations where the reliabilities of the chosen paths are required not to exceed pre-specified levels. The reliability of a path is the product of the reliabilities of the arcs comprising the path. Taking the logarithm of both sides of the reliability requirement results in an additive rather than a multiplicative constraint.

The NDSR problem aims to select the set of arcs and route each commodity from its origin to its destination to minimize the total fixed and variable costs, while satisfying the weight constraint of each commodity. The NDSR problem generalizes several well-known *NP-Complete* problems, including the fixed-charge network design, weight-constrained shortest path, and hop-constrained network design problems. Without the weight constraint, the problem reduces to the traditional uncapacitated fixed-charge network design problem, which has been extensively studied (e.g., Magnanti and Wong 1984, Balakrishnan et al. 1997, Randazzo and Luna 2001, Agarwal and Aneja 2012). If the design costs are zero and there is only one weight metric, or if there is only one commodity and one weight metric, the problem reduces to the weight-constrained shortest path problem (Righini and Salani 2008, Carlyle et al. 2008, Dumitrescu and Boland 2003); Pugliese and Guerriero (2013) provide an recent survey of exact solution approaches for the problem. When the weights are the same for all arcs, i.e., $w_{ij}^{k,l} = w^{k,l}$, the problem reduces to the hop-constrained network design problem; Balakrishnan and Altinkemer (1992) propose a Lagrangian-based algorithm to solve the problem, and Pirkul and Soni (2003) present an alternative formulation that models the number of hops explicitly. When the problem has $n - 1$ commodities with a common origin node and distinct destination nodes, a single service level requirement and same weights for all

arcs, the problem reduces to hop constrained minimum spanning tree problem on a directed network (Dahl and Gouveia, 2004).

Recent research has studied the NDSR problem and its variants. Balakrishnan et al. (2014) model the NDSR problem using arc-flow variables and propose various valid inequalities to strengthen the model. Holmberg and Yuan (2003) consider a variant of the NDSR problem in which the commodities are not restricted to be routed on one a single path; they propose a formulation that models the flow of commodities on paths and apply column generation to solve the problem. Both these approaches have some disadvantages, as we discuss below, and prepare us to develop a hybrid approach.

2.2.1. Arc-flow formulation

Balakrishnan et al. (2014) propose an arc-flow formulation by defining two sets of variables: arc design variables and arc routing variables. Design variable z_{ij}, equals one if the solution selects arc (i, j) and is zero otherwise; arc routing variable x_{ij}^k equals one if commodity k is routed on arc (i, j) and zero otherwise. Using these variables, the NDSR problem has the following integer programming formulation, denoted as Model [AF]:

Model [AF] $$\text{Min} \sum_{(i,j)\in A} f_{ij} z_{ij} + \sum_{k\in K} \sum_{(i,j)\in A} c_{ij}^k x_{ij}^k \qquad (2.1)$$

subject to:

$$\sum_{j:(i,j)\in A} x_{ij}^k - \sum_{j:(j,i)\in A} x_{ji}^k = \begin{cases} 1 \text{ if } i = s_k \\ -1 \text{ if } i = t_k \\ 0 \text{ otherwise} \end{cases} \quad \forall k \in K, i \in N \qquad (2.2)$$

$$\sum_{(i,j)\in A} w_{ij}^{k,l} x_{ij}^k \le W^{k,l} \quad \forall k \in K, l = 1, \ldots, L, \qquad (2.3)$$

$$x_{ij}^k \le z_{ij} \quad \forall k \in K, (i, j) \in A, \text{ and} \qquad (2.4)$$

$$x_{ij}^k, z_{ij} \in \{0,1\} \quad \forall k \in K, (i, j) \in A. \qquad (2.5)$$

The objective function (2.1) minimizes the sum of total fixed costs of installing arcs and the routing costs of all commodities. Constraints (2.2) are the flow conservation equations to ensure that each commodity is routed from its origin to its destination. Constraints (2.3), which we refer to as the weight constraints, enforce the requirement that the total weight of arcs used by each commodity does not exceed the limit $W^{k,l}$ for each metric l. The forcing constraints (2.4) impose the condition that a commodity can only be routed on a path when the underlying arc is selected. Constraints (2.5) impose the binary requirement on the flow and design variables.

2.2.2. Path-flow formulation

An alternative way to formulate the problem is to model the routing path of each commodity explicitly, instead of using arc flow variables and imposing flow conservation equations (2.2) on arc flow variables; we call this new formulation path-flow formulation. Let Π^k denote the set of feasible paths for commodity k, i.e. paths originating at s_k and ending at t_k whose total weight is within $W^{k,l}$ for every metric l; let $\Pi_{ij}^k = \{p \in \Pi^k : (i, j) \in p\}$ denote the set of feasible paths for commodity k that contain arc (i, j). Let $C_p^k = \sum_{(i,j)\in p} c_{ij}^k$ represent the total variable cost of routing commodity k on path p. As earlier, we let z_{ij} equal one if the design selects arc (i, j) and zero otherwise. In addition, we define the path routing variable y_p^k, where a value of one for y_p^k indicates that commodity k flows on path p, and a value of zero indicates that it does not. Using the design and path routing variables, the path flow formulation, denoted by Model [PF], is as follows:

Model [PF] Min $$\sum_{(i,j)\in A} f_{ij} z_{ij} + \sum_{k\in K} \sum_{p\in\Pi^k} C_p^k y_p^k \qquad (2.6)$$

subject to:

$$\sum_{p\in\Pi^k} y_p^k = 1 \qquad \forall k \in K, \qquad (2.7)$$

$$\sum_{p \in \Pi_{ij}^k} y_p^k \le z_{ij} \qquad \forall k \in K_P, (i,j) \in A, \tag{2.8}$$

$$y_p^k, z_{ij} \in \{0,1\} \qquad \forall (i,j) \in A, k \in K, p \in \Pi^k. \tag{2.9}$$

Constraints (2.7) require that each commodity should be routed on one feasible path. Constraints (2.8) specify that a commodity k is routed on a path p only when every arc on p has been selected by the design. Constraints (2.9) impose the binary requirement on the flow and design variables.

2.2.3. Comparing Model [AF] and Model [PF] Models

We discuss the advantages and disadvantages of formulations Model [AF] and Model [PF] in this subsection, based on which we propose a new model to overcome the disadvantages of both models in Section 2.2.4. Denote the optimal linear programming relaxation value of Model [AF] and Model [PF] as L_{AF} and L_{PF} respectively.

Proposition 2.1. For any given problem instance, $L_{AF} \le L_{PF}$.

Proof. Let (**y**,**z**) denote an optimal solution to the linear programming relaxation of Model [PF]. Define $x_{ij}^k = \sum_{p \in \Pi^k : (i,j) \in p} y_p^k$, then (**x**,**z**) satisfies constraints (2.2) and (2.4). In addition,

$$\sum_{(i,j) \in A} w_{ij}^{k,l} x_{ij}^k = \sum_{(i,j) \in A} w_{ij}^{k,l} \sum_{p \in \Pi_{ij}^k} y_p^k = \sum_{p \in \Pi_k} \Big(\sum_{(i,j) \in p} w_{ij}^{k,l} \Big) y_p^k \le \sum_{p \in \Pi_k} W^{k,l} y_p^k = W^{k,l}.$$

Hence, (**x**,**z**) is feasible to the linear programming relaxation of formulation [AF]. Thus, $L_{AF} \le L_{PF}$.■

Proposition 2.1 shows that Model [AF] has a weaker linear programming relaxation than Model [PF]. Intuitively, the linear programming relaxation of Model [AF] is weaker since commodities can meet the weight limit constraint on average, i.e., by combining flows on paths with large and small weight values. On the other hand, Model [PF] excludes all infeasible paths, and so has a smaller feasible space.

A natural question is: How poorly can the linear programming relaxation of Model [AF] perform relative to the linear programming relaxation of Model [PF]? We can construct examples to show that the relative gap between the optimal linear programming relaxation values, i.e., $(L_{AF} - L_{PF}) / L_{PF}$, can be infinity.

Table 1 compares the size of the two formulations: the size of the arc-flow formulation is polynomial in the size of the network, whereas the number of variables in the path flow model can be exponential. As the number of service metrics L increases, the difficulty of solving Model [AF] may increase since the number of constraints increases, while it is easier to solve Model [PF] due to fewer number of variables. Likewise, when the weight limit requirement becomes more stringent, i.e. $W^{k,l}$ is decreased, number of variables in Model [PF] decreases and so does the effort to solve Model [PF], whereas the size of Model [AF] remains unchanged.

Table 1. Comparison of Model [AF] and Model [PF] models

Criteria	**Model [AF]**	**Model [PF]**
# of variables	$\|A\|(\|K\|+1)$	$\|A\|+\sum_{k\in K}\|\Pi^k\|$
# of constraints	$\|K\|(n+L+\|A\|)$	$\|K\|(1+\|A\|)$
# of service metrics L ↑	# of constraints ↑ # of variables unchanged	# of constraints unchanged # of variables ↓
Weight limits $W^{k,l}$ ↓	# of variables unchanged	# of variables ↓
Strength of linear programming relaxation	Lower	Greater

The pros and cons of Model [AF] and Model [PF] exhibit the classic tradeoff in integer programming formulations: compactness versus strength of the model. The stringency of the weight limit can affect the relative effectiveness of the two models.

When the weight limit is loose (i.e. when $W^{k,l}$ is relatively large), Model [AF] becomes more advantageous; in this case, the weight limit constraint is not likely to be binding in the linear programming relaxation of Model [AF], which implies the difference between L_{AF} and L_{PF} may not be large. However, $|\Pi^k|$ may be high for every commodity k and the higher number of variables may add to the difficulty of solving Model [PF]. If the weight limit is stringent, Model [PF] contains fewer path variables making it easier to solve. On the other hand, the weight limit constraint is very likely to be violated by the linear programming relaxation of Model [AF], resulting in the solution using infeasible paths (and satisfying the weight constraint on average), and hence a larger difference in L_{AF} and L_{PF} values.

2.2.4. Hybrid formulation

The comparison of Model [AF] and Model [PF] suggests that an approach based on path-flow variables is likely to be better for tightly-constrained commodities, and using arc-flow variables is likely to perform better otherwise. Since the weight limits of commodities are unlikely to be all stringent or all loose in the same problem instance, we are motivated to use the path-flow representation for commodities with tight weight constraints, and the arc-flow representation for others.

We use $K_A \subseteq K$ to denote the set of commodities for which the model uses the arc-flow representation and $K_P \subseteq K$ to denote the set of commodities for which the model uses the path-flow representation. Note that K_A and K_P form a partition of K, i.e. $K_A \cap K_P = \varnothing$ and $K_A \cup K_P = K$. Given the partitions (K_A, K_P), the hybrid formulation Model [HF(K_A, K_P)] can be represented as follows:

Model [HF(K_A, K_P)] Min $\sum_{(i,j)\in A} f_{ij} z_{ij} + \sum_{k\in K_A} \sum_{(i,j)\in A} c_{ij}^k x_{ij}^k + \sum_{k\in K_P} \sum_{p\in\Pi^k} C_p^k y_p^k$ (2.10)

subject to:

$$\sum_{j:(i,j)\in A} x_{ij}^k - \sum_{j:(j,i)\in A} x_{ji}^k = \begin{cases} 1 \text{ if } i = O(k) \\ -1 \text{ if } i = D(k) \\ 0 \text{ otherwise} \end{cases} \qquad \forall k \in K_A, i \in N \qquad (2.11)$$

$$\sum_{(i,j)\in A} w_{ij}^k x_{ij}^k \le W_{ij}^k \qquad \forall k \in K_A, l, = 1, \ldots, L, \qquad (2.12)$$

$$x_{ij}^k \le z_{ij} \qquad \forall k \in K_A, (i,j) \in A, \qquad (2.13)$$

$$\sum_{p\in\Pi_{ij}^k} y_p^k = 1 \qquad \forall k \in K_P, \qquad (2.14)$$

$$\sum_{p\in\Pi^k:(i,j)\in p} y_p^k \le z_{ij} \qquad \forall k \in K_P, (i,j) \in A, \qquad (2.15)$$

$$y_p^k \in \{0,1\} \qquad \forall k \in K_P, p \in \Pi^k, \qquad (2.16)$$

$$z_{ij}, x_{ij}^k \in \{0,1\} \qquad \forall k \in K_A, (i,j) \in A, \qquad (2.17)$$

If K_A = K, the model reduces to the weaker but more compact arc-flow formulation; if K_P = K, the model reduces to the larger but stronger path flow formulation.

The number of variables in Model [HF(K_A, K_P)] is $|A||K_A|+|A|+\sum_{k\in K_P}|\Pi^k|$, larger than the number of variables in Model [AF] and smaller than that in Model [PF]. We denote the optimal linear programming relaxation value of Model [HF(K_A, K_P)] as $L_{HF(K_A,K_P)}$.

Proposition 2.3. For any problem instance and any commodity partition (K_A, K_P), $L_{AF} \le L_{HF(K_A,K_P)} \le L_{PF}$.

Proof. The proof of this proposition is similar to the proof of Proposition 2.1. ■

Proposition 2.3 indicates that the strength of the model is also between Model [AF] and Model [PF]. Thus, we can view Model [HF(K_A, K_P)] as the compromise between the arc-flow and the path-flow representation: (i) compared to Model [AF], Model [HF(K_A, K_P)] has a stronger linear programming relaxation but without an undue

increase in the model size, but (ii) compared to Model [PF], the hybrid model has fewer number of variables but a weaker linear programming relaxation bound.

While Model [HF(K_A, K_P)] achieves an advantageous middle ground, we need to determine K_A and K_P before we can use the model. The partition of K into K_A and K_P is essential to the performance of Model [HF(K_A, K_P)]; we would like to choose a partition that brings about the appropriate mix of compactness of Model [AF] and strength of Model [PF]. As discussed earlier, the tightness of the weight limit values can greatly affect the effectiveness of the path-flow and the arc-flow representations: if commodity k has stringent weight limits, it is more advantageous to include it in the set K_P; but hand, if commodity k has loose weight limits, it is better to include it in the set K_A.

2.2.5. Columnization methods

Since determining the tightness of the weight limits *a priori* is difficult, we adopt an iterative method that exploits linear programming relaxation solutions to guide the partitioning of the set of commodities into those for which the arc-based formulation is more appropriate and those for which the path-based formulation is more appropriate. Starting with K_A=K, we refer to the process of moving a commodity k from K_A to K_P as columnizing commodity k. Given the linear programming solution to the current hybrid model, we propose two possible methods to evaluate the choice of commodities to columnize: the flow-decomposition method and the estimate-bound method.

The flow-decomposition method examines the flow decomposition of the arc flow commodities to decide which commodities to be columnized. Given any linear programming relaxation solution with arc-flow values for commodity k, we decompose the arc-flow into a set of path flows (Ahuja et al. 1993), denoted as P^k. If $\sum_{(i,j)\in p} w_{ij}^{k,l} > W^{k,l}$ for some path $p \in P^k$ and metric l, i.e., the solution attempts to use

some infeasible path, the weight metric *l* may be stringent for commodity *k*. In the flow decomposition method, we columnize a commodity *k* if $\sum_{(i,j)\in p} w_{ij}^{k,l} x_{ij}^{k} > W^{k,l}$ for any path $p \in P^k$. We tested several strategies for flow decomposition in a network, and based on this testing, used the shortest-weight path first strategy when decomposing the arc-flow into path flows. With such a strategy, the path with shortest-weight are considered first during the flow decomposition.

The estimate-improvement method tires to estimate the potential improvement of the linear programming relaxation solution and use this estimated improvement to decide which commodities to columnize. For each commodity *k* and arc (*i*, *j*) with $x_{ij}^{k} > 0$, we must have $x_{ij}^{k} \le z_{ij}$. If $x_{ij}^{k} < z_{ij}$, then arc (*i*, *j*) is not specially designed to route commodity *k*. If $x_{ij}^{k} = z_{ij}$, we can divide the design value z_{ij} into two parts: the portion of value specially designed for commodity *k* and the portion of value used by other commodities. Given the LP solution $(\bar{x}, \bar{y}, \bar{z})$ to hybrid Model [HF(K_A, K_P)], the portion of value used by commodities other than *k* is the maximum flow value of those commodities, represented by

$$\zeta_{ij}^{k}(K_A, K_P) = \max\left\{ \max_{k' \in K_A \setminus \{k\}} \bar{x}_{ij}^{k'}, \max_{k' \in K_P} \sum_{p \in \Pi^{k}:(i,j)\in p} \bar{y}_{p}^{k'} \right\} \quad \forall k \in K_A, (i,j) \in A.$$

Accordingly, the portion of the arc designed specifically for commodity *k* is $\bar{z}_{ij} - \zeta_{ij}^{k}(K_A, K_P)$. Thus, we can estimate the contribution of commodity *k* to the total cost as

$$E^{k}(K_A, K_P) = \sum_{(i,j)\in A} \left(c_{ij}^{k} \bar{x}_{ij}^{k} + f_{ij} \max\{0, \bar{z}_{ij} - \zeta_{ij}^{k}(K_A, K_P)\} \right) \quad \forall k \in K_A.$$

We can solve the following LP problem to estimate the contribution of commodity *k* to the total cost after columnizing commodity *k*.

$$D^{k}(K_A, K_P) = \min \sum_{p \in \Pi^{k}} C_{p}^{k} y_{p}^{k} + \sum_{(i,j)\in A} f_{ij} u_{ij} \qquad (2.18)$$

subject to

$$\sum_{p\in\Pi^k} y_p^k = 1 \tag{2.19}$$

$$\sum_{p\in\Pi_{ij}^k} y_p^k - u_{ij} \le \zeta_{ij}^k(K_A, K_P) \qquad \forall a \in A \tag{2.20}$$

$$y_p^k \ge 0 \qquad \forall p \in \Pi^k \tag{2.21}$$

$$u_{ij} \ge 0 \qquad \forall (i, j) \in A \tag{2.22}$$

In the above model, u_{ij} is the portion of the arc specially designed for commodity k. The following proposition shows that $D^k(K_A, K_P) - E^k(K_A, K_P)$ gives an upper bound on the improvement in the LP relaxation value after columnizing commodity k. Thus, we can use $D^k(K_A, K_P) - E^k(K_A, K_P)$ as a criterion to decide if commodity k should be columnized.

Proposition 2.4. $L_{\mathrm{H}(K_A\setminus\{k\},K_P\cup\{k\})} - L_{\mathrm{H}(K_A,K_P)} \le D^k(K_A,K_P) - E^k(K_A,K_P)$

Proof: Denote $l_{\mathrm{H}(K_A,K_P)} = L_{\mathrm{H}(K_A,K_P)} - E^k(K_A,K_P)$. Assume the optimal solution to the LP relaxation of $\mathrm{H}(K_A, K_P)$ is (x^1, y^1, z^1) and the optimal solution to formulation (2.18)-(2.22) is (y^2, u^2). We can define a feasible solution (x^3, y^3, z^3) to formulation $\mathrm{H}(K_A\setminus\{k\}, K_P\cup\{k\})$ as follows:

$x_{ij}^{k'3} = x_{ij}^{k'1}$ for all $k' \in K_A\setminus\{k\}$ and $(i, j) \in A$

$y_p^{k'3} = x_p^{k'1}$ for all $k' \in K_P$ and $p \in \Pi^k$

$y_p^{k'3} = y_p^{k'2}$ for $k' = k$ and $p \in \Pi^k$

$z_{ij}^3 = u_{ij}^2 + \zeta_{ij}^k(K_A, K_P)$ for all $(i, j) \in A$.

The objective function value of solution (x^3, y^3, z^3) is $l_{\mathrm{H}(K_A,K_P)} + D^k(K_A,K_P)$. Thus, we have $l_{\mathrm{H}(K_A,K_P)} + D^k(K_A,K_P) - L_{\mathrm{H}(K_A\setminus\{k\},K_P\cup\{k\})} \ge 0$, which is equivalent to

$$L_{\mathrm{H}(K_A\setminus\{k\},K_P\cup\{k\})} - L_{\mathrm{H}(K_A,K_P)} < D^k(K_A,K_P) - E^k(K_A,K_P) \text{ .■}$$

2.2.6. Decolumnization method

As we columnize more commodities, some of the commodities columnized earlier may no longer need the path-flow representation. So, we may need a decolumnization

algorithm to transform the commodity from the path-flow representation to the arc-flow version. Given the LP relaxation solution of a hybrid formulation, we propose an estimate-bound method to decolumnize commodities.

Given the LP solution $(\bar{x}, \bar{y}, \bar{z})$ to Model [HF(K_A, K_P)] the portion of value used by commodities other than k is the maximum flow value of those commodities, represented by

$$\xi_{ij}^k(K_A, K_P) = \max\left\{\max_{k' \in K_A} x_{ij}^{k'}, \max_{k' \in K_P \setminus \{k\}} \sum_{p \in \Pi_{ij}^k} y_p^{k'}\right\} \text{ for } k \in K_P \text{ and } (i,j) \in A.$$

Accordingly, the portion of the arc designed specifically for commodity k is $\bar{z}_{ij} - \zeta_{ij}^k(K_A, K_P)$. Thus, we can estimate the contribution of commodity k to the total cost as

$$F^k(K_A, K_P) = \sum_{p \in \Pi^k} C_p^k y_p^k + \sum_{(i,j) \in A} f_{ij} \max\{0, \bar{z}_{ij} - \zeta_{ij}^k\}.$$

We can solve the following problem to estimate the contribution of commodity k to the total cost after columnizing commodity k.

$$G^k(K_A, K_P) = \min \sum_{(i,j) \in A} c_{ij}^k x_{ij}^k + \sum_{(i,j) \in A} f_{ij} u_{ij}$$

subject to

$$\sum_{(i,j):(i,j) \in A} x_{ij}^k - \sum_{(j,i):(j,i) \in A} x_{ji}^k = \begin{cases} 1 & \text{if } i = O(k) \\ -1 & \text{if } i = D(k) \\ 0 & \text{otherwise} \end{cases} \forall i \in N \tag{2.23}$$

$$\sum_{(i,j) \in A} w_{ij}^k x_{ij}^k \le W^k \tag{2.24}$$

$$x_{ij}^k - u_{ij} \le \xi_{ij}(K_A, K_P) \quad \forall (i,j) \in A \tag{2.25}$$

$$x_{ij}^k \ge 0 \qquad \forall (i,j) \in A \tag{2.26}$$

$$u_{ij} \ge 0 \qquad \forall (i,j) \in A \tag{2.27}$$

In the formulation, u_{ij} denotes the portion of the arc (i, j) designed to incorporate the flow of commodity k. Proposition 2.5 shows that $F^k(K_A, K_P) - G^k(K_A, K_P)$ gives a lower bound on the decrease in the LP relaxation value, after decolumnizing commodity

k. Therefore, the value of $F^k(K_A, K_P) - G^k(K_A, K_P)$ gives a criterion on whether commodity k should be decolumnized: if the value of $D^k(K_A, K_P) - E^k(K_A, K_P)$ is high, we should not decolumnize k.

Proposition 2.5. $L_{\mathrm{H}(K_A,K_P)} - L_{\mathrm{H}(K_A\cup\{k\},K_P\setminus\{k\})} \geq F^k(K_A,K_P) - G^k(K_A,K_P)$

Proof: Similar to the proof of Proposition 2.4.■

2.2.7. Dynamic columnization and decolumnization

In this section, we integrate the columnization and decolumnization methods to find a good hybrid formulation. To prevent the size of hybrid formulation from becoming too large, we require that commodities that are considered to be candidates for columnization have a limited number of feasible paths. Specifically, commodity k is a candidate for assignment to K_P only when $|\Pi^k| < U$, where U is a parameter.

Starting with $K_A = K$, we iteratively solve the linear programming relaxation problem and use the resulting solution to decide the set of additional commodities to columnize. When no commodities can be columnized (because no commodity satisfied our columnization check), we start to decolumnize commodities, and when no commodities can be decolumnized, the resulting partition of commodity is returned Figure 1 illustrates the procedure.

`Procedure dynamic_columnization_and_decolumnization`

Input: NDSR problem instance with network G and set of commodities K
U, the upper bound for number of feasible paths of any commodity $k \in K_P$
Method to evaluate the weight tightness

Output: (K_A, K_P), a partition of commodity set K

Step 1: $K_A = K$ and $K_P = \varnothing$

Step 2: Solve the linear programming relaxation of Model [HF (K_A, K_P)]
$K_a = \varnothing$
For each commodity $k \in K_A$ with $|\Pi^k| < U$
If we use the *estimate-improvement method* and
$D^k(K_A, K_P) - E^k(K_A, K_P) > 0$ for some k
Add k to K_a
Break
Else if we use the *flow-decomposition method*
Decompose the solution into path flows, denoted as P^k
If $\sum_{(i,j)\in p} w_{ij}^{k,l} > W^{k,l}$ for some $p \in P^k$ and some l
Add k to K_a
If $K_a == \varnothing$
Go to step 4
Else
Go to Step 3

Step 3. $K_A := K_A \setminus K_a$ and $K_P := K_P \cup K_a$
Go to Step 2.

Step 4. Solve the linear programming relaxation of Model [HF (K_A, K_P)]
$K_a = \varnothing$
For each commodity $k \in K_P$
If $F^k(K_A, K_P) - G^k(K_A, K_P) \leq 0$ for some k
Add k to K_a
If $K_a == \varnothing$
Stop the procedure and return (K_A, K_P)
Else
Go to Step 5

Step 5. $K_A := K_A \cup K_a$ and $K_P := K_P \setminus K_a$
Go to Step 4.

Figure 1. Procedure for dynamic columnization

2.3. STRENGTHENING THE HYBRID FORMULATION

In this section, we discuss how to further strengthen the hybrid formulation by applying valid inequalities. For the arc-flow proportion of the formulation, we apply various inequalities proposed in Balakrishnan et al. (2014); we also adapt their ρ OR-IF inequalities for commodities that have a path-flow representation. In addition, we discuss how the path-flow representation enables us to develop valid inequalities which we cannot replicate using just the arc flow variables.

2.3.1. Arc-flow representation

Balakrishnan et al. (2014) propose various inequalities to help strengthen Model [AF]. By exploiting the weight constraints, they derive several valid inequalities that tighten the linear programing relaxation of the model. For example, the incompatible *r*-Arc inequality

$$\sum_{(i,j)\in A'} x_{ij}^k \le r-1 \tag{2.28}$$

states that commodity can flow over at most $r-1$ of the arcs $A' \subseteq A$ for the selected route to avoid violating weight limits. The contingent routing inequality

$$x_{gh}^k \le \sum_{(i,j)\in A'} x_{ij}^k \tag{2.29}$$

stipulates that if commodity *k* flows on arc (*g*, *h*), then it must also be routed on at least one of the arcs in the set $A' \subseteq A$ to satisfy the weight constraints. While both inequalities are effective in closing the integrality gap for Model [AF], they are automatically satisfied by any linear programming solution for Model [PF], since all infeasible paths are already excluded from the formulation.

Balakrishnan et al. (2014) also combine the *r*-arc inequalities and the contingent routing inequalities to derive the generalized OR-IF inequality. Let us consider one specialization of this generalized inequality when *r* = 2 for the *r*-arc inequalities (*OR*

inequalities) and $|A'|=1$ for the contingent routing inequalities (*IF* inequalities). Let $I = \{1, 2, \ldots, Q\}$ denote a set of indices, with $I_{OR} \subseteq I$ denote the set of OR indices and $I_{IF} \subseteq I \setminus I_{OR}$ denote the set of IF indices. Each OR index $q \in I_{OR}$ has a corresponding 2-arc inequality $x^{k_q}_{i_q j_q} + x^{k_q}_{i_{q+1} j_{q+1}} \le 1$, and each IF index $q \in I_{IF}$ has a corresponding contingent routing constraint $x^{k_q}_{g_q h_q} \le x^{k_q}_{i_q j_q}$. Starting from these base inequalities, Balakrishnan [2014] derive the following ρ OR-IF inequality,

$$\sum_{q \in I_{OR}} (x^{k_q}_{i_q j_q} + x^{k_q}_{i_{q+1} j_{q+1}}) + \sum_{q \in I_{IF}} x^{k_q}_{g_q h_q} \le \sum_{q=1}^{Q} z_{a_q} + (\rho - 1)/2, \tag{2.30}$$

where $\rho = |I_{OR}|$. We can use $x^k_{ij} = \sum_{p \in \Pi^k_{ij}} y^k_p$ for $k \in K_P$ so that inequality (2.30) is applicable in the hybrid model.

2.3.2. Union-intersection inequality

The path-based representation for commodities is more "informative" than the arc-based representation: the arc-flow variables provide only local information about the flow on each arc, and a flow decomposition method is needed to determine the origin-destination path flow values (which may not be unique) from the arc-flow values. On the other hand, the path-flow representation explicitly models the flow on feasible paths; thus, not only do we know the decomposition of the path flows from s_k to t_k, but the arc flow on each arc (i, j) can be easily determined by using $x^k_{ij} = \sum_{p \in \Pi^k_{ij}} y^k_p$. Knowing the path flows permits us to take the unions and intersections of paths and thus derive new valid inequalities.

Proposition 2.6. For any Q arcs $(i_1, j_1), (i_2, j_2), \ldots, (i_Q, j_Q)$ and Q commodities $k_1, k_2, \ldots, k_Q$, the Union-Intersection inequality

$$\sum_{p \in \Pi^{k_Q}_{i_1 j_1} \cup \Pi^{k_Q}_{i_Q j_Q}} y^{k_Q}_p + \sum_{q=1}^{Q-1} \sum_{p \in \Pi^{k_q}_{i_q j_q} \cap \Pi^{k_q}_{i_{q+1} j_{q+1}}} y^{k_q}_p \le \sum_{q=1}^{Q} z_{i_q j_q} \tag{2.31}$$

is valid.

Proof. For any $q = 1, \ldots, Q-1$, the forcing constraints require that $\sum_{p \in \Pi^{k_q}_{i_q j_q} \cap \Pi^{k_q}_{i_{q+1} j_{q+1}}} y^{k_q}_p \le z_{i_q j_q}$ and $\sum_{p \in \Pi^{k_q}_{i_q j_q} \cap \Pi^{k_q}_{i_{q+1} j_{q+1}}} y^{k_q}_p \le z_{i_{q+1} j_{q+1}}$. In addition, we have $\sum_{p \in \Pi^{k_Q}_{i_1 j_1} \cup \Pi^{k_Q}_{i_Q j_Q}} y^{k_Q}_p \le 1$ (since the left hand side of this inequality is less than or equal to $\sum_{p \in \Pi^{k_q}} y^{k_q}_p$, which itself does not exceed one), and $\sum_{p \in \Pi^{k_Q}_{i_1 j_1} \cup \Pi^{k_Q}_{i_Q j_Q}} y^{k_Q}_p \le z_{i_1 j_1} + z_{i_Q j_Q}$ (since the left hand side of this inequality is less than or equal to $\sum_{p \in \Pi^{k_Q}_{i_1 j_1}} y^{k_Q}_p + \sum_{p \in \Pi^{k_Q}_{i_Q j_Q}} y^{k_Q}_p$, which is less than or equal to the right hand side of the inequality). Summing these four sets of inequalities, dividing both sides by two, and rounding down the right hand side give inequality (3.4). ■

Note that the Q commodities in inequality (2.31) need not be all distinct. To intuitively understand the inequality, let us look the inequality with $Q = 2$ for arcs (i_1, j_1), (i_2, j_2); enforcing inequalities for all pairs of k_1 and k_2 is equivalent to enforcing the inequality

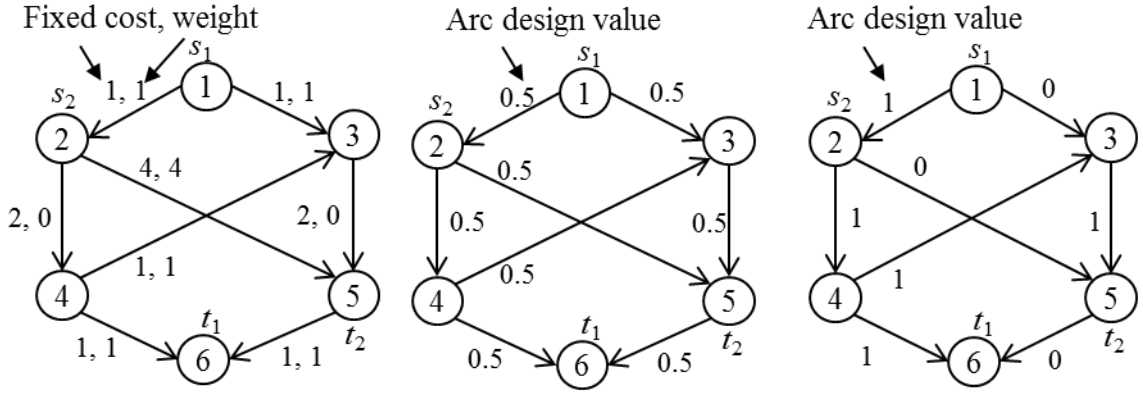
$$\max_k \sum_{p \in \Pi^k_{i_1 j_1} \cap \Pi^k_{i_2 j_2}} y^k_p + \max_k \sum_{p \in \Pi^k_{i_1 j_1} \cup \Pi^k_{i_2 j_2}} y^k_p \le z_{i_1 j_1} + z_{i_2 j_2} \tag{2.32}$$

Inequality (2.32) strengthens inequality $\sum_{p \in \Pi^k_{i_1 j_1} \cap \Pi^k_{i_2 j_2}} y^k_p + \sum_{p \in \Pi^k_{i_1 j_1} \cup \Pi^k_{i_2 j_2}} y^k_p \le z_{i_1 j_1} + z_{i_2 j_2}$ (it is valid since the left hand side is equivalent to $\sum_{p \in \Pi^k_{i_1 j_1}} y^k_p + \sum_{p \in \Pi^k_{i_2 j_2}} y^k_p$) for each commodity k by taking maximization over all commodities. Given any pair of arcs, we cannot use arc-flow variables to represent the flow of a commodity passing through both arcs, nor can we represent the flow passing through at least one of them; thus, it is hard, if not impossible, for us to enforce inequality (2.32) using the arc-flow representation.

To appreciate the effectiveness of the inequality, consider the example in Figure 2. In this example, $Q = 2$, we want to route commodity k_1 from node 1 to node 6 and route commodity k_2 from node 2 to node 5; the weight limit for both commodities is 5. The fixed costs and weights for the arcs are shown in Figure 2a; the costs of routing both

commodities on all the arcs are zero; note that both commodities have the same weights for all arcs. Figure 2b illustrates the design values in linear programming solution for formulation Model [PF]; in the solution, the design value for each arc is ½, the required flow of commodity k_1 splits equally on paths 1-2-4-6 and 1-3-5-6, and the flow of commodity k_2 splits equally on paths 2-4-3-5- and 2-5; the objective function value is 13/2.

To see the effectiveness of inequality (3.4), denote path 1-2-4-6 as p_1, path 1-3-5-6 as p_2, path 2-4-3-5 as p_3, and path 2-5 as p_4; the linear programming solution has $y_{p_1}^{k_1} = y_{p_2}^{k_1} = y_{p_3}^{k_2} = y_{p_4}^{k_2} = 1/2$. Since $\sum_{p\in\Pi_{24}^{k_1}\cup\Pi_{35}^{k_1}} y_p^{k_1} + \sum_{p\in\Pi_{24}^{k_2}\cap\Pi_{35}^{k_2}} y_p^{k_2} = y_{p_1}^{k_1} + y_{p_2}^{k_1} + y_{p_3}^{k_2}$, inequality (2.31) is simplified into $y_{p_1}^{k_1} + y_{p_2}^{k_1} + y_{p_3}^{k_2} \le z_{24} + z_{35}$; enforcing this constraint would cut off the solution and yield an integer (optimal) solution whose design value is shown in Figure 2c. In the solution, commodity k_1 uses path 1-2-4-6 and commodity k_2 uses path 2-4-3-5.



a. Network with arc fixed cost and weight

b. Solution of linear programming without inequality (2.31)

c. Solution of linear programming with inequality (2.31)

Figure 2. Example for Union-intersection inequality

To facilitate the study of the separation procedure, we first discuss the necessary conditions that must be met before an linear programming solution (y, z) can violate inequality (2.31).

Corollary 2.7. A linear programming solution (y,z) can violate inequality (2.31) only if

(i) $\sum_{p\in\Pi^{k_Q}_{i_Q j_Q}\cup\Pi^{k_Q}_{i_1 j_1}} y_p^{k_Q} > z_{i_1 j_1}/2 + z_{i_Q j_Q}/2,$ and

(ii) $\sum_{p\in\Pi^{k_q}_{i_q j_q}\cap\Pi^{k_q}_{i_{q+1} j_{q+1}}} y_p^{k_q} > 0$ for all $q = 1, 2, ..., Q-1.$

Proof: We need to show that if either condition is violated, inequality (2.31) is satisfied. First note that any linear programming solution satisfies $\sum_{p\in\Pi^{k_q}_{i_q j_q}\cap\Pi^{k_q}_{i_{q+1} j_{q+1}}} y_p^{k_q} \le z_{i_q j_q}/2 + z_{i_{q+1} j_{q+1}}/2$ for all $i = 1, \ldots, Q-1$. Now suppose $\sum_{p\in\Pi^{k_Q}_{i_Q j_Q}\cup\Pi^{k_Q}_{i_1 j_1}} y_p^{k_Q} \le z_{i_1 j_1}/2 + z_{i_Q j_Q}/2$. Adding these inequalities gives inequality (2.31). Hence, condition (i) is necessary. Next, suppose $\sum_{p\in\Pi^{k_h}_{i_h j_h}\cap\Pi^{k_h}_{i_{h+1} j_{h+1}}} y_p^{k_h} = 0$ for some h; adding inequalities $\sum_{p\in\Pi^{k_q}_{i_q j_q}\cap\Pi^{k_q}_{i_{q+1} j_{q+1}}} y_p^{k_q} \le z_{i_q j_q}$ for $q > h$, $\sum_{p\in\Pi^{k_q}_{i_q j_q}\cap\Pi^{k_q}_{i_{q+1} j_{q+1}}} y_p^{k_q} \le z_{i_{q+1} j_{q+1}}$ for $q < h$, and $\sum_{p\in\Pi^{k_Q}_{i_1 j_1}\cup\Pi^{k_Q}_{i_Q j_Q}} y_p^{k_Q} \le z_{i_1 j_1} + z_{i_Q j_Q}$ (implied by forcing constraints) gives inequality (2.31). Therefore, condition (ii) is necessary.■

Condition (i) implies that the total flow of k_Q on the feasible paths that contain either (i_1, j_1) or (i_Q, j_Q) must be high enough, and condition (ii) implies $\Pi^{k_q}_{i_q j_q} \bigcap \Pi^{k_q}_{i_{q+1} j_{q+1}}$ is non empty for all q = 1, 2, …, Q. As we will see later, these conditions help to speed up the separation procedure for inequality (2.31).

We create an undirected network $\bar{G} = (\bar{N}, \bar{A})$ based on the linear programming solution (**y**,**z**) as follows: for each arc $(i, j) \in A$ with $z_{ij} > 0$, we create a node, denoted as [i, j], corresponding to it; for each pair of nodes $[i_1, j_1], [i_2, j_2] \in \bar{N}$, if $Y_{[i_1, j_1][i_2, j_2]} = \max_k \sum_{p\in\Pi^k_{i_1 j_1}\cap\Pi^k_{i_2 j_2}} y_p^k > 0$, we create an undirected arc, with cost $d_{[i_1, j_1][i_2, j_2]} = z_{i_1 j_1}/2 + z_{i_2 j_2}/2 - Y_{[i_1, j_1][i_2, j_2]}$, between nodes [$i_1$, j_1] and [i_2, j_2]. Note that (a) Condition (ii) permits us to include only those node pairs [i_1, j_1] and [i_2, j_2] for which

$Y_{[i_1,j_1][i_2,j_2]} > 0$, and (b) $d_{[i_1,j_1][i_2,j_2]} > 0$ since $Y_{[i_1,j_1][i_2,j_2]} \le z_{[i_1,j_1]}$ and $Y_{[i_1,j_1][i_2,j_2]} \le z_{[i_2,j_2]}$. Besides, any simple path in network $\bar{G}$ corresponds to an inequality (2.31). For example, path $[i_1, j_1]$-$[i_2, j_2]$-…-$[i_Q, j_Q]$ corresponds to arcs inequality (2.31) defined by arcs (i_1, j_1) , …, (i_Q, j_Q) in network G and commodities $k_q = \arg\max_k \sum_{p \in \Pi^k_{i_q j_q} \cap \Pi^k_{i_q j_q}} y^k_p$ for q = 1, …, Q – 1 and $k_Q = \arg\max_k \sum_{p \in \Pi^k_{i_1 j_1} \cup \Pi^k_{i_Q j_Q}} y^k_p$; in addition, this path has cost $z_{i_1 j_1}/2 + z_{i_Q j_Q}/2 + \sum_{q=2}^{Q-1} z_{i_q} - \sum_{q=1}^{Q-1} \sum_{p \in \Pi^{k_q}_{i_q j_q} \cap \Pi^{k_q}_{i_{q+1} j_{q+1}}} y^{k_q}_p$,and the sum of this cost and $z_{i_1 j_1}/2 + z_{i_Q j_Q}/2 - \sum_{p \in \Pi^{k_Q}_{i_1 j_1} \cup \Pi^{k_Q}_{i_Q j_Q}} y^{k_Q}_p$ gives the difference between the right hand side and left hand side of inequality (2.31).

In the separation procedure, for a given pair of arcs (i_1, j_1) and (i_0, j_0), we seek the most violated inequality (2.31), if any, with $(i_Q, j_Q) = (i_0, j_0)$ for some number Q. We denote the cost of shortest path from $[i_1, j_1]$ to $[i_0, j_0]$ as $D_{[i_1,j_1][i_0,j_0]}$. To identify violated inequalities, condition (i) of Corollary 2.7 allows us to only study origin node [i_1, j_1] and destination node [i_0, j_0] with $E_{[i_1,j_1][i_0,j_0]} = z_{i_1 j_1}/2 + z_{i_0 j_0}/2 - \max_k \sum_{p \in \Pi^k_{i_1 j_1} \cup \Pi^{k_0}_{i_0 j_0}} y^k_p < 0$. If $E_{[i_1,j_1][i_0,j_0]} + D_{[i_1,j_1][i_0,j_0]} < 0$, we have identified an violated inequality ; otherwise no such inequality is violated. Figure 3 describes the detailed procedure.

`Procedure separate_union-intersection_inequalities`

Input: (**y**, **z**), linear programming solution of Model [PF]
arcs (i_1, j_1) and (i_0, j_0)

Output: the most violated Union-Intersection inequality (2.31)

Step 1. Calculate $D = z_{i_1 j_1}/2 + z_{i_0 j_0}/2 - \max_k \sum_{p \in \Pi^k_{i_0 j_0} \cup \Pi^k_{i_1 j_1}} y^k_p$

If $D \geq 0$

Stop the procedure and there is no violated inequality.

Step 2. Create the intersection network $\bar{G} = (\bar{N}, \bar{A})$

Find the shortest path between nodes $[i_1, j_1]$ and $[i_0, j_0]$; denote the path as $[i_1, j_1]$-$[i_2, j_2]$-…-$[i_{Q-1}, j_{Q-1}]$-$[i_0, j_0]$ and its cost as E

If $D + E < 0$

Return the inequality (2.31) with arcs $(i_1, j_1), \ldots, (i_{Q-1}, j_{Q-1}), (i_0, j_0)$ and commodities $k_q = \arg\max_k \sum_{p \in \Pi^k_{i_q j_q} \cap \Pi^k_{i_{q+1} j_{q+1}}} y^k_p$ for $q = 1, \ldots, Q-1$, and commodity $k_0 = \arg\max_k \sum_{p \in \Pi^k_{i_0 j_0} \cup \Pi^k_{i_1 j_1}} y^k_p$.

Else

Stop the procedure and there is no violated inequality

Figure 3. Procedure to separate union-intersection inequality

2.3.3. Q-union inequality

Proposition 2.8 Let Q be an odd integer number. Given Q arcs $(i_1, j_1), (i_2, j_2), \ldots, (i_Q, j_Q)$, and Q commodities $k_1, k_2, \ldots, k_Q$, the Q-union inequality

$$\sum_{p \in \Pi^{k_Q}_{i_Q j_Q} \cup \Pi^{k_Q}_{i_1 j_1}} y^{k_Q}_p + \sum_{q=1}^{Q-1} \sum_{p \in \Pi^{k_q}_{i_q j_q} \cup \Pi^{k_q}_{i_{q+1} j_{q+1}}} y^{k_q}_p \leq \sum_{q=1}^{Q} z_{i_q j_q} + (Q-1)/2 \qquad (2.33)$$

is valid.

Proof: Consider the valid inequalities $\sum_{p \in \Pi^{k_q}_{i_q j_q} \cup \Pi^{k_q}_{i_{q+1} j_{q+1}}} y^{k_q}_p \leq z_{i_q j_q} + z_{i_{q+1} j_{q+1}}$ and $\sum_{p \in \Pi^{k_q}_{i_q j_q} \cup \Pi^{k_q}_{i_{q+1} j_{q+1}}} y^{k_q}_p \leq 1$ for $q = 1, \ldots, Q-1$, $\sum_{p \in \Pi^{k_Q}_{i_Q j_Q} \cup \Pi^{k_Q}_{i_1 j_1}} y^{k_Q}_p \leq z_{i_1 j_1} + z_{i_Q j_Q}$, and $\sum_{p \in \Pi^{k_Q}_{i_Q j_Q} \cup \Pi^{k_Q}_{i_1 j_1}} y^{k_Q}_p \leq 1$; summing up these inequalities, dividing both sides by two, and rounding down the right hand side, we obtain inequality (2.33).■

Balakrishnan et al. (2014) considers a specialization of the ρ OR-IF inequality for the arc-flow formulation, called ρ OR inequality, in Model [AF] model; for an odd

number Q, arcs (i_1, j_1), (i_2, j_2), …, (i_Q, j_Q), and commodities $k_1, k_2, ..., k_Q$, the ρ OR inequality

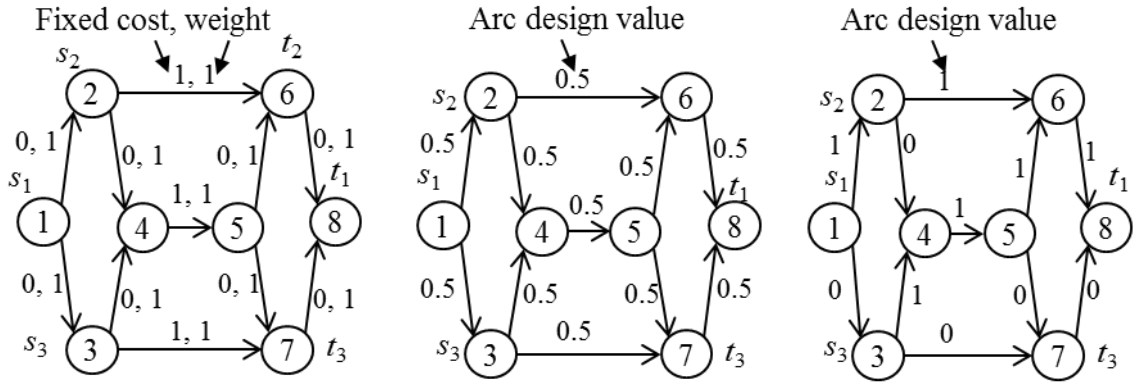
$$x_{i_Q j_Q}^{k_Q} + x_{i_1 j_1}^{k_Q} + \sum_{q=1}^{Q-1} (x_{i_q j_q}^{k_q} + x_{i_{q+1} j_{q+1}}^{k_q}) \le \sum_{q=1}^{Q} z_{i_q j_q} + (Q-1)/2 \qquad (2.34)$$

is valid if $x_{i_q j_q}^{k_q} + x_{i_{q+1} j_{q+1}}^{k_q} \le 1$ for q = 1, …, Q – 1 and $x_{i_Q j_Q}^{k_Q} + x_{i_1 j_1}^{k_Q} \le 1$. We can view inequality (2.34) as a specialization of inequality (2.33) when $x_{i_q j_q}^{k_q} + x_{i_{q+1} j_{q+1}}^{k_q} \le 1$ (or equivalently $\Pi_{i_q j_q}^{k_q} \bigcap \Pi_{i_{q+1} j_{q+1}}^{k_q} = \varnothing$) for q = 1, …, Q – 1 and $x_{i_Q j_Q}^{k_Q} + x_{i_1 j_1}^{k_Q} \le 1$(or equivalently $\Pi_{i_1 j_1}^{k_1} \bigcap \Pi_{i_Q j_Q}^{k_Q} = \varnothing$). In the arc-flow representation, without knowing decomposition of the path flows from s_k to t_k, we can only enforce the special case, i.e. inequality (2.34), but not the general inequality (2.33).

We demonstrate the effectiveness of the inequality using the example in Figure 4. This problem instance has three commodities k_1, k_2, and k_3 with origins s_1, s_2, and s_3 and destinations t_1, t_2, and t_3 respectively. The weight limit for each commodity is 3 and the cost to route each commodity on each arc is 0. The numbers in Figure 4a present the fixed cost and weight for each arc. Figure 4b illustrates the linear programming solution to Model [PF]. In the solution, the design value for each arc is ½ and the objective function value is 3/2; the flow of commodity k_1 splits equally on paths 1-2-6-8, and 1-3-7-8, the flow of commodity k_2 splits equally on paths 2-6 and 2-4-5-6, and the flow of commodity k_3 splits equally on paths 3-7 and 3-4-5-7. Denoting paths 1-2-6-8, 1-3-7-8, 2-6, 2-4-5-6, 3-7, and 3-4-5-7 as p_1, p_2, p_3, p_4, p_5, and p_6, we have

$$\sum_{p \in \Pi_{26}^{k_1} \cup \Pi_{37}^{k_1}} y_p^{k_1} + \sum_{p \in \Pi_{26}^{k_2} \cup \Pi_{45}^{k_2}} y_p^{k_2} + \sum_{p \in \Pi_{37}^{k_3} \cup \Pi_{45}^{k_3}} y_p^{k_3} = y_{p_1}^{k_1} + y_{p_2}^{k_1} + y_{p_3}^{k_2} + y_{p_4}^{k_2} + y_{p_5}^{k_3} + y_{p_6}^{k_3},$$

and inequality (2.33) is simplified to $y_{p_1}^{k_1} + y_{p_2}^{k_1} + y_{p_3}^{k_2} + y_{p_4}^{k_2} + y_{p_5}^{k_3} + y_{p_6}^{k_3} \le z_{26} + z_{45} + z_{37} + 1$; Adding this constraint cuts off the solution and yields an integer (optimal) solution shown in Figure 4c.



a. Network with arc fixed cost and weight value

b. Solution of linear programming without inequality (2.33)

c. Solution of linear programming with valid inequality (2.33)

Figure 4. Example for Q-union inequality

To separate violated inequality (2.33), we create an undirected arc-union network $G' = (N', A')$ based on the linear programming solution (**y**, **z**) as follows: for each node arc $(i, j) \in A$ with $z_{ij} > 0$, we create a node corresponding to it, denoted as $[i, j]$. For each pair of nodes $[i_1, j_1], [i_2, j_2] \in N'$, if $Y_{[i_1,j_1][i_2,j_2]} = \max_k \sum_{p \in \Pi^k_{i_1 j_1} \cup \Pi^k_{i_2 j_2}} y^k_p > 0$, we create an undirected arc, with cost $d_{[i_1,j_1][i_2,j_2]} = Y_{[i_1,j_1][i_2,j_2]} - z_{i_1 j_1}/2 - z_{i_2 j_2}/2$, between them. Any simple cycle $[i_1, j_1]$-$[i_2, j_2]$-…-$[i_Q, j_Q]$-$[i_1, j_1]$ in G', with Q being an odd number, corresponds to an inequality (2.33) defined by arcs (i_1, j_1), …, (i_Q, j_Q) and commodities $k_q = \arg\max_k \sum_{p \in \Pi^k_{i_q j_q} \cup \Pi^k_{i_q j_q}} y^k_p$ for q = 1, …, Q – 1 and $k_Q = \arg\max_k \sum_{p \in \Pi^k_{i_1 j_1} \cup \Pi^k_{i_Q j_Q}} y^k_p$; the cost of the cycle is $\max_k \sum_{p \in \Pi^{k_Q}_{i_Q j_Q} \cup \Pi^{k_Q}_{i_1 j_1}} y^{k_Q}_p + \sum_{q=1}^{Q-1} \sum_{p \in \Pi^{k_q}_{i_q j_q} \cup \Pi^{k_q}_{i_{q+1} j_{q+1}}} y^{k_q}_p - \sum_{q=1}^{Q} z_{i_q j_q}$, and if this cost is bigger than (Q – 1)/2, we find a violated inequality (2.33). Our cutting plane method implements the inequality with Q = 3 and applies conditions in Corollary 3.8 to speed up the separation by reducing the number of possible arc triplets. Conditions in Corollary 3.8 indicates

that the design values or the total flow on two arcs should be large enough to violate inequality (2.33).

Corollary 3.8. Inequality (2.33) with $Q = 3$, i.e. inequality

$$\sum_{p\in\Pi^{k_1}_{i_1j_1}\cup\Pi^{k_1}_{i_2j_2}} y_p^{k_1} + \sum_{p\in\Pi^{k_2}_{i_2j_2}\cup\Pi^{k_2}_{i_3j_3}} y_p^{k_2} + \sum_{p\in\Pi^{k_3}_{i_3j_3}\cup\Pi^{k_3}_{i_1j_1}} y_p^{k_3} \le z_{i_1j_1} + z_{i_2j_2} + z_{i_3j_3} + 1$$

can be violated by the linear programming solution (**y, z**) only if the following conditions are met:

(i) $z_{i_1j_1} + z_{i_2j_2} + z_{i_3j_3} > 1$

(ii) $\sum_{p\in\Pi^k_a\cup\Pi^k_b} y_p^k > z_a$ and $\sum_{p\in\Pi^k_a\cup\Pi^k_b} y_p^k > z_b$ for all triplets $[a, b, k] = [(i_1, j_1), (i_2, j_2), k_1], [(i_2, j_2), (i_3, j_3), k_2], [(i_3, j_3), (i_1, j_1), k_3]$.

(iii) $\sum_{p\in\Pi^k_a\cup\Pi^k_b} y_p^k > 1 - z_c$ for all quadruplets $[a, b, c, k] = [(i_1, j_1), (i_2, j_2), (i_3, j_3), k_1], [(i_2, j_2), (i_3, j_3), (i_1, j_1), k_2], [(i_3, j_3), (i_1, j_1), (i_2, j_2), k_3]$.

Proof: (i) We have $\sum_{p\in\Pi^{k_3}_{i_3j_3}\cup\Pi^{k_3}_{i_1j_1}} y_p^{k_3} + \sum_{p\in\Pi^{k_1}_{i_1j_1}\cup\Pi^{k_1}_{i_2j_2}} y_p^{k_1} + \sum_{p\in\Pi^{k_2}_{i_2j_2}\cup\Pi^{k_2}_{i_3j_3}} y_p^{k_2} \le 2z_{i_1j_1} + 2z_{i_2j_2} + 2z_{i_3j_3}$, since $\sum_{p\in\Pi^{k_3}_{i_1j_1}\cup\Pi^{k_3}_{i_3j_3}} y_p^{k_3} \le z_{i_1j_1} + z_{i_3j_3}$, $\sum_{p\in\Pi^{k_1}_{i_1j_1}\cup\Pi^{k_1}_{i_2j_2}} y_p^{k_1} \le z_{i_1j_1} + z_{i_2j_2}$, and $\sum_{p\in\Pi^{k_2}_{i_2j_2}\cup\Pi^{k_2}_{i_3j_3}} y_p^{k_2} \le z_{i_2j_2} + z_{i_3j_3}$. If inequality (2.33) is violated, $2z_{i_1j_1} + 2z_{i_2j_2} + 2z_{i_3j_3} > z_{i_1j_1} + z_{i_2j_2} + z_{i_3j_3} + 1$, which implies inequality $z_{i_1j_1} + z_{i_2j_2} + z_{i_3j_3} > 1$.

(ii) Since $\sum_{p\in\Pi^{k_2}_{i_2j_2}\cup\Pi^{k_2}_{i_3j_3}} y_p^{k_2} \le z_{i_2j_2} + z_{i_3j_3}$ and $\sum_{p\in\Pi^{k_2}_{i_2j_2}\cup\Pi^{k_2}_{i_3j_3}} y_p^{k_2} \le 1$, we have inequality (2.33) violated only if $\sum_{p\in\Pi^{k_3}_{i_3j_3}\cup\Pi^{k_3}_{i_1j_1}} y_p^{k_3} > z_{i_1j_1}$. The proof for the other cases of Condition (ii) is similar.

(iii) Since $\sum_{p\in\Pi^{k_1}_{i_1j_1}\cup\Pi^{k_1}_{i_2j_2}} y_p^{k_1} + \sum_{p\in\Pi^{k_2}_{i_2j_2}\cup\Pi^{k_2}_{i_3j_3}} y_p^{k_2} \le z_{i_1j_1} + 2z_{i_2j_2} + z_{i_3j_3}$, inequality (2.33) can only be violated if $\sum_{p\in\Pi^{k_3}_{i_1j_1}\cup\Pi^{k_3}_{i_3j_3}} y_p^{k_3} + z_{i_1j_1} + 2z_{i_2j_2} + z_{i_3j_3} > z_{i_1j_1} + z_{i_2j_2} + z_{i_3j_3} + 1$, which implies inequality $\sum_{p\in\Pi^{k_3}_{i_1j_1}\cup\Pi^{k_3}_{i_3j_3}} y_p^{k_3} > 1 - z_{i_2j_2}$. We can similarly prove the other cases.■

2.4. COMPUTATIONAL RESULTS

Our computational testing has several goals. First, we would like to demonstrate that the hybrid formulation can help solve the NDSR problem to optimality and that it is robust across a wide range of problems. Second, we would like to show that the hybrid formulation can help increase the lower bound without unduly increasing the model size. Third, we would like to evaluate the impact of using dynamic adaptive strategy for columnization of some of the commodities.

To conduct the study, we used the following approach to generate test problems with varying sizes, costs, and service requirements. The method randomly locates each node on a rectangular grid, connects all nodes with a spanning tree, and adds arcs to ensure that the network is strongly connected. The fixed cost for each arc is a combination of the Euclidean distance between its endpoints and a random component. Balakrishnan et al. (1989) suggests that the ratio of variable to fixed costs can influence computational performance; we use a parameter γ to represent this ratio: higher values of this parameter correspond to higher relative variable costs. The number of metrics L is one. The weight of each arc is randomly generated between $1-\delta$ and $1+\delta$, where δ is a given parameter. By varying the value of δ, we can generate problems with different service requirements. The weight limit of commodity k is $\alpha w^k + \beta$, where (i) w^k is the length of the shortest weight path for commodity k from its origin to its destination, (ii) α and β is are random numbers. For conciseness, we use notation $\alpha \in [\alpha_1, \alpha_2]$ to denote that the values of α is randomly generated in the closed interval specificed by α_1 and α_2.

We compare three solution methods: the arc-based method, the path-based method, and the hybrid method. The arc-based method uses CPLEX to solve Model [AF]. The path-based method uses CPLEX to solve Model [PF] with the path-based

valid inequalities applied. The hybrid method first uses dynamic columnization and decolumnization to identify commodities that are suited for path flow representations and then uses CPLEX to solve the resulting hybrid formulation, applying all the model strengthening techniques in Section 2.3. In the subsequent tables we use "EB" to denote the estimate-bound columnization method and "FD" to denote the flow-decomposition columnization method. We set the limit on number of paths U is set to 15000 in the dynamic modeling process to keep the problem size manageable.

We implement all methods in JAVA using CPLEX 12.4 to solve the optimization problem and run under Ubuntu Linux on a Dell Poweredge 2950 workstation with two hex-core, hyperthreading 3.33 GHz Xeon processors and 24 GB of shared memory. For each problem, the bound-and-bound terminates after 20 minutes or when the integrality gap become less than 0.1%.

We first consider different problem sizes with $\gamma = 0.1$, $\delta = 0.2$, $\alpha \in [1, 1.2]$, and $\beta \in [1, 7]$. The size of the problems are represented by $n/m/|K|$, where n and m are the number of nodes and arcs in the network and K is the set of commodities. Table 2 compares the performance of the arc-based, the hybrid, and the path-based methods for three scenarios with different number of arcs and number of commodities. The results, which are the averages of five instances, show that the hybrid method outperforms the arc-based and the path-based methods: it always finds a lower gap. The final gaps for the arc-based and the path-based methods are much larger and increase dramatically as the problem size increases. The hybrid formulation results in the lowest solution time. For 40/240/160 and 40/200/200 problems, due to the large number of feasible paths, we cannot solve the path-based formulation. We can also see that two dynamic columnization methods perform similarly.

Table 3 further establishes the superiority of our hybrid formulation by comparing the model size and the linear programming relaxation bounds of three alternative formulations. It shows that our approach usually columnized a small portion of the commodities and thus has only slightly more variables than the arc-based formulation. On the other hand, the number of feasible paths are so large that it is either infeasible to formulate the path-based model (in the allotted memory) or it takes an enormous computational effort to solve the resulting path-based formulation. Although the hybrid formulation has much fewer variables, its linear programming relaxation bound is almost the same as that of the path-based formulation. In other words, a little columnization can go a long way. By harnessing the respective strengths of the arc- and path-based formulations, our approach of dynamically columnizing commodities results in an approach that is methodologically sound and computationally successful.

A comparison of the two columnization methods shows that their performance are quite similar. Both alternatives columnize nearly the same number of commodities and taking almost the same amount of time to perform the columnization.

For the robustness of our solution method, we fix the problem size to 40/200/160 and compare the result of problems with different values of α, β, γ and δ in Table 4. As we can see, the hybrid formulations are better than the arc flow and path flow formulations in all possible instances.

Table 2. IP comparison for arc-based, hybrid, and path-based methods

(Averages over five instances)

Problem Size $n/m/\|K\|$	Average Final Gap[a] (%)				# Solved to Optimality				Average CPLEX Time if Solved to Optimality (secs)			
	Arc	EB[b]	FD[c]	Path	Arc	EB[b]	FD[c]	Path	Arc	EB[b]	FD[c]	Path
40/200/160	11.1	2.5	2.1	6.1	0	1	1	0	NA	160	135	NA
40/240/160	13.7	4.4	4.0	NA	0	0	0	NA	NA	NA	NA	NA
40/200/120	10.1	1.6	1.3	5.9	0	3	3	0	NA	396	519	NA
40/200/200	11.6	2.6	2.7	NA	0	1	1	0	NA	283	346	NA

[a]Final Gap = (Final upper bound – Final lower bound) / Final upper bound × 100%

[b]EB: the hybrid formulation generated using estimate-bound columnization method

[c]FD: the hybrid formulation generated using flow-decomposition columnization method

Table 3. Model size and linear programming relaxation bounds comparison for arc-based, hybrid, and path-based methods

(Averages over five instances)

Problem Size $n/m/\|K\|$	Average of Ratio of #Variables[a]			Average of Hybrid Bound Quality[b]		Average # of Columnized Commodities		Average Time to Columnize and Decolumnize	
	EB[b]	FD[c]	Path	EB[b]	FD[c]	EB[b]	FD[c]	EB[b]	FD[c]
40/200/160	1.99	1.97	57.93	0.9743	0.9742	49.6	49.4	115	100
40/240/160	2.20	2.21	NA	NA	NA	50.0	49.8	135	149
40/200/120	2.53	2.57	64.48	0.9615	0.9608	37.8	37.0	387	358
40/200/200	1.92	1.91	NA	NA	NA	57.4	57.6	121	116

[a]Ratio of #Variables = # of variables in the hybrid or the path-based method / # of variables in the arc-based method

[b]EB: the hybrid formulation generated using estimate-bound columnization method

[c]FD: the hybrid formulation generated using flow-decomposition columnization method

[d]Hybrid Bound Quality = $(L_{HF} - L_{AF}) / (L_{PF} - L_{AF})$, where L_{HF}, L_{AF}, and L_{PF} are the linear programming relaxation bounds of the hybrid, the arc-based, and the path-based formulations.

Table 4. Comparison of arc-based and hybrid methods with different values of α, β, γ and δ

		Average Final Gap[a] (%)			
	Values	**Arc**	**EB[b]**	**FD[c]**	**Path**
α	[0.1, 1.1]	11.2	1.3	1.6	3.8
	[1.2, 1.4)	8.0	2.5	2.2	NA
β	[1, 6]	13.3	0.8	0.6	1.0
	[3, 8]	6.4	1.6	1.3	NA
γ	0.05	20.6	6.4	6.5	14.9
	0.15	4.6	0.0	0.2	1.1
δ	0.1	10.9	2.1	1.7	6.0
	0.3	12.7	2.2	1.8	5.8

[a]Final Gap = (Final upper bound – Final lower bound) / Final upper bound × 100%
[b]EB: the hybrid formulation generated using estimate-bound columnization method
[c]FD: the hybrid formulation generated using flow-decomposition columnization method

2.5. CONCLUDING REMARKS

In this chapter, we examined the arc-flow, path-flow, hybrid formulations for the NDSR problem. The hybrid formulation is powerful since it leverages the advantages of both the arc-flow and the path-flow formulation. Specifically, the arc-flow representation results in a compact formulation but a weak relaxation bound, while the path-flow representation leads to a tight formulation but with an excessive number of variables. By applying a dynamic columnization and decolumnization strategy, we can construct a hybrid formulation that has a strong linear programming bound but is also manageable in size. To further improve the algorithmic effectiveness, we developed and implemented valid inequalities that strengthen the path-flow representation.

To apply the path-flow representation for any commodity, our current approach requires enumerating all the possible feasible paths for the commodity. As an extension, it is appealing to apply the column generation technique (Barnhart et al. 1998): start with a promising subset of paths and generate other paths when necessary by solving a pricing

problem, i.e., the constrained shortest path problem. The resulting model may be called "hybrid column generation" since column generation is only applied to a subset of commodities and other commodities use the original representation, i.e., the arc-flow representation. This approach could lead to solving much larger problem instances. It would be also interesting to study how to develop and implement the "hybrid column generation" idea in other problem contexts. Likewise, the idea of the hybrid method can be extended to other areas of integer programming, including Benders decomposition and Lagrangian relaxation.

The path-flow variables available in the path-flow representation enable us to develop new valid inequalities; we cannot readily formulate equivalent inequalities in the arc-flow representation. Identifying inequalities of this type and using them in other situations could lead to solution approach improvements.

We could also investigate the NDSR model defined on undirected networks. In these networks, a selected arc permits flow of commodities in both directions. These and other promising directions can prove to be a feasible area of research on the NDSR and related problems.

Chapter 3. The Train Dispatching Problem

3.1. INTRODUCTION

The freight rail industry is an important segment of the transportation sector. According to a study by Association of American Railroads (2013), freight railroads move about 40% of the total freight in US (measured in ton-miles), more than any other transportation mode. As a cost-effective and environmentally-friendly transportation mode, rail transportation will continue to play important roles as transportation needs grow nationwide to interconnect geographically expanding supply chains and to transport new commodities such as shale oil. With rising fuel prices and growing concerns about greenhouse effects, rail transport is also gaining popularity due to its energy efficiency. To meet the increasing demand for rail freight services, railroad companies are focusing on first improving the utilization of their existing resources before investing in expensive capacity expansion projects. Among these resources, capital-intensive railway tracks are one of the main bottlenecks that limit the flow of freight traffic. The utilization of tracks depends on how well the train dispatchers orchestrate the movement of trains through each territory. The tracks are shared by trains, with varying speeds and priorities, traveling in both directions. To avoid collisions, dispatchers must decide whether and how long to hold a particular train at various sidings to permit other trains to meet (and cross) or pass (overtake) it. These sequencing and scheduling decisions govern the effective velocity of trains, defined as the distance traveled divided by the total travel and waiting time for each train; in turn, the velocity averaged over all trains determines track occupancy and utilization. According to a recent report (GE Report 2010), every mile-per-hour increase in average train velocity can yield annual savings of millions of dollars in capital and expenses. Single-track territories, used primarily to reduce construction cost, are the greatest capacity bottleneck on the train transit lines (Kittelson and

Associates, 2003). Two tracks usually have around four times more capacity than a single track (Abril et al., 2008); however, a four-track line only has 50% more capacity than a double line (Kittelson and Associates, 2003). Optimizing the train movements in single-track territories is critical to manage the flow of trains in the whole rail network. The goal of this paper is to develop a model and an effective solution method to optimize the movement plans for freight trains passing through each dispatching territory so as to maximize average velocity in that territory. We propose several modeling and methodological enhancements, and demonstrate using real data on actual train schedules and track characteristics for a U.S. freight railroad that these enhancements are very effective in reducing solution time.

Freight trains carry many different goods ranging from commodities such as coal, petroleum, and agricultural products to automobiles and intermodal freight. Unlike passenger trains whose itineraries are fixed well in advance, freight trains do not follow a fixed schedule. Rather, their routes and timing vary from week to week depending on the volume of traffic between various locations. Based on the type of freight they carry and their schedule requirements, trains have different priorities for movement and hence their relative priorities in terms of passing or crossing other trains. To manage the traffic on the freight rail network, the system is partitioned into "territories," each covering one or more parallel tracks between two "terminal" locations. Dispatchers, one for each territory, are responsible for the short-term decisions of planning the movement of trains and managing the traffic in their respective territories. Each dispatcher faces the following decision problem: given the set of trains that will traverse the dispatcher's territory over the planning horizon (e.g., next 12 hours), the attributes of each train, including its priority, speed, and the time and location at which it enters the territory, and the characteristics of the territory in terms of its physical configuration, capacities of

track segments, and locations of stations and sidings, the train dispatching problem seeks to sequence these trains and plan their meet and pass events so as to maximize average train velocity (or minimize the total waiting time) while satisfying various operational and safety requirements. Prior research has discussed mathematical programming models for train dispatching, but founds that solving real world problem to near-optimality is too time-consuming for the models to be useful in practice. Therefore, most papers largely focus on heuristic solution methods and do not emphasize modeling or methodological refinements to optimally solve actual problems within reasonable time.

We propose an integer programming model for the train dispatching problem that uses a discrete time representation, and explore techniques to solve real-life problem instances to optimality. Our model maximizes the weighted velocity of trains, taking into account various practical requirements for railway operations including operational rules regarding trailing of trains, headway requirements between trains, track unavailability, and train priorities. Solving the model using standard solvers (e.g., CPLEX) is too time-consuming to be used for real-time planning; the dispatchers expect that a useful tool should return a good plan within several minutes. When we applied CPLEX to a base model (without enhancements) for real problem instances, the solver cannot find solutions within 4% MIP gap after 5 minutes for most instances(see Section 3.6). To improve solution performance, we develop and incorporate several modeling enhancements. We first propose strong non-concurrency constraints that exploit the unidirectional movement property on each track segment; these inequalities are not only tighter than previous track capacity constraints (e.g., Şahin et al. 2008), but also reduce the model size by eliminating the need for separate constraints to avoid train-swap conflicts at block transitions (Harrod 2011). Further, based on the non-concurrency and headway requirements, we refine the partition of segments into sections so as to

strengthen the non-concurrency constraints while also reducing model size. We propose another set of non-concurrency inequalities based on a train's movement to further tighten the model. Moreover, we strengthen the pairwise unidirectional inequalities discussed in Cacchiani et al. (2010), extend these inequalities to more than one segment, and generalize the inequalities to incorporate more than one train. We also develop separation procedures for these inequalities so as to add them dynamically (as user cuts) during the branch and cut process. Finally, we develop a sequential dispatching heuristic, with randomization, to find good solutions quickly. These solutions can serve to warm-start and accelerate exact solution procedures. Computational tests, using real problem instances demonstrate that our modeling and methodological enhancements vastly improve the performance of exact solution methods.

Our work can contribute to this area in at least two aspects. First, the existing literature overwhelmingly rely on heuristics to solve the problem, and little effort has been devoted to improving the performance of the exact solution process. Our work may be the first to explore ways to solve the problem optimally. Furthermore, solution methods, especially those designed for train dispatching in US, are rarely validated through real-life instances. Usually, the test instances are randomly generated (Şahin et al. 2008) or simplified (Harrod, 2011). In contrast, we demonstrate the validity of our methods by testing more than 20 instances from a Class I railroad company in US.

The rest of this chapter is organized as follows. Section 3.2 reviews related literature. We describe our model formulation in Section 3.3, and develop techniques to tighten and reduce the model in Section 3.4. Section 3.5 discusses our heuristic solution procedure. We test our model and solution method based on data from a major U.S. railroad company, and report computational results in Section 3.6. Section 3.7 offers concluding remarks.

3.2. LITERATURE REVIEW

As one of the most important operational planning problems for railroads, the train dispatching problem or train timetabling problem (typically defined in the context of planning movements of trains, particularly passenger trains in Europe) has drawn considerable attention from both researchers and practitioners. Cordeau et al. (1998), Törnquist (2006), and Lusby et al. (2011) provide extensive surveys of related models and solution methods.

Caprara et al. (2002) prove that the train timetabling problem is NP-complete by polynomially transforming any instance of a maximum independent set problem into a simplified version of the problem. The problem can be viewed as a job-shop problem with blocking and no-wait constraints (Corman et al. 2010) or a multicommodity network flow problem with additional constraints (Caprara et al. 2002), which are both known to be NP-complete.

To formulate the train dispatching problem as a mathematical program, researchers have considered two broad approaches – continuous time models (e.g., Carey 1994a, Carey 1994b, and Carey and Lockwood 1995, Higgins et al. 1996, Zhou and Zhong 2007, and Mu and Dessouky 2011) and discrete time models (e.g. Brännlund et al. 1998, Caprara et al. 2002, and Cacchiani et al. 2010, Cacchiani and Caprara 2008, Şahin et al. 2008, and Harrod 2011) – to represent train travel and waiting times.

The key part of the discrete-time mathematical programming approach is how to model the headway requirement and track capacity requirement (i.e. prevent overtaking and meeting of trains in segments). Caprara et al. (2002) consider a passenger train timetabling problem on uni-directional single-track railroads. They develop three sets of clique constraints to enforce the headway between trains when entering and exiting stations and to prevent overtaking of trains inside any track. By using pre-specified

segment traversal time, Caprara et al. (2006) develop a stronger version of the non-overtaking constraints to reduce computation time. Cacchiani et al. (2010) generalize the model in Caprara et al. (2006) to incorporate the movement of freight trains on bi-directional tracks. Şahin et al. (2008) and Harrod (2011) enforce the headway and track capacity requirements by dividing each segment into smaller pieces and forbidding simultaneous occupancy of each piece by trains.

Since solving the problem using standard commercial solvers (e.g. CPLEX) is too time-consuming, various mathematical-programming-based heuristics are developed to solve the problem. For example, Şahin et al. (2008) propose a LP-greedy construction heuristic; Caprara et al. (2002) provide a heuristic based on Largrangian Relaxation; Mu and Dessouky (2011) develop various heuristics to reduce the model size. An alternative to optimization-based heuristics is the discrete-event heuristic (e.g. Dorfman and Medanic 2004, Cai et al. 1998, and Şahin 1999). One major advantage of discrete-event heuristic models over the optimization model is that they can provide a detailed description of the transient behavior of the railway system and incorporate many real-world concerns that are difficult to formulate mathematically (Cai et al. 1998).

3.3. MODEL FORMULATION

3.3.1. Problem description

The operations planning process in freight railroads begins with decisions on which trains to run (between which locations and at what times) over the next week or so, based on the actual and projected volume of shipments between various origin-destination pairs. This train operations plan specifies, for each train, the starting location, intermediate stopping locations (e.g., to pick up and drop off freight cars, change crews), final destination, and its planned starting time and desired arrival times at

the intermediate and final locations. To effectively manage the traffic and dynamically plan the movements of the scheduled trains, railroad companies partition their networks into territories, each assigned to a dispatcher. A dispatching territory typically covers 100 to 200 miles of single or parallel tracks flanked by "terminal" locations at the two ends of the territory, and having intermediate sidings and stations where trains can wait to let other trains cross or pass. Each train typically traverses multiple territories on its origin-to-destination route. Our model seeks to optimize the movements of trains inside a specific territory over a short-term planning horizon of, say, 12 hours.

We refer to any location within a territory that contains one or more sidings where trains can wait and/or change tracks as a *station*. Trains can meet or pass other trains only at the *stations*. A "meet" occurs when two trains traveling in opposite directions cross each other safely without colliding into each other. One of the trains uses the mainline while the other uses or waits on a siding. A "pass" event happens when a fast or high priority train overtakes another train traveling in the same direction; the latter train waits on a siding to permit the former train to pass. We refer to the mainline track(s) between two adjacent stations as *segment*. Without loss of generality, we assume that stations and segments alternate along the territory, i.e., each intermediate station is flanked by two segments on either side, and each station has two adjacent segments. Since segments do not contain intermediate locations where trains can wait, trains traveling in opposite directions cannot simultaneously use the same track on a segment. But, to increase the utilization of tracks, trains traveling in the same direction are permitted to trail each other, i.e., multiple trains can use a track at the same time as long as they are separated by a minimum required distance, called the *trailing headway*. Operationally, dispatchers usually specify this separation requirement in terms of *blocks*; each block typically corresponds to the portion of track between two adjacent signals on a

segment. At most one train can occupy each block at any time, and dispatchers usually separate two successive trains by at least one unoccupied block so that the trailing train can move safely at its nominal speed with sufficient stopping distance. To model this separation requirement, we treat every two adjacent blocks as a section and permit at most one train (traveling in either direction) to occupy a section at any time; specifically, if a segment has K blocks, indexed as blocks 1, 2,⋯, K, we define blocks k and $k + 1$ as section k for $k = 1, 2, \cdots, K - 1$. Another safety requirement pertains to the minimum required time separation between trains crossing a control point. A *control point*, located at any junction where a track splits into multiple tracks or vice versa, represents a railroad switch to guide trains from one track to another. In particular, every station has one control point at each of its two endpoints (that demarcate the station from its adjacent segments). To ensure safe operation, successive trains passing through a control point (in either direction) must be separated by a minimum required time (typically, five to seven minutes); we refer to this restriction as the *control point headway*.

3.3.2. Notation

To formulate the train dispatching problem, we adopt a discrete time modeling approach (as in Caprara et al. 2002 and Şahin et al. 2008). In this approach, we divide the planning horizon into fine-grained time intervals or *periods* (e.g., each period is one or two minutes). Let H denote the total number of time periods in the planning horizon, indexed as $t = 1, 2, \ldots, H$. Consider a territory with stations $s \in S$, segments $m \in M$, and control points $p \in P$. Both stations and segments are called edges and let $e \in S \cup M$ represent an edge in the territory. Each track segment m is partitioned into one or more *sections*, denoted as G_m; each section $s \in G_m$ can be occupied by no more than one train at any time. As discussed earlier, these sections are defined to ensure that trailing trains

(traveling in the same direction) maintain sufficient inter-train headway or distance. For each train $q \in Q$, we know its direction of travel, denoted as + or – (e.g., + is eastbound or northbound and – is westbound or southbound), the sequence of stations or segments that it must traverse, its priority (used to decide the importance or weight for maximizing average velocity), any arrival time windows or hard time window, and the train's traversal (travel) time on each segment or station that is passes through. Let o^q and d^q respectively denote train q's starting and ending stations in the territory. Assume, without loss of generality, that these starting and ending locations are stations where train q can wait (e.g., the terminals for future trains that pass through the territory, or intermediate stations for trains that are already in the territory). We define θ_p as the minimum required control point headway (in time periods) between two successive trains passing through control point p in either direction.

For every edge e and train $q \in Q_e$, we are given the time τ_e^q (in number of periods) for train q to traverse that edge. We assume for simplicity that, at every station, the train's traversal time (including any track crossover time) is the same on all the tracks at that station. With this assumption (which largely holds in practice, particularly when we take into account the control point headway requirements), we do not need to distinguish between assignments and movements of trains along different tracks within the station. So, if α_{st} denotes the number of available parallel tracks at station s at time t, we can simply impose an aggregate capacity of α_{st} on the number of trains that are within the station in period t. We can readily extend the model to permit varying traversal times on different tracks, but at the expense of adding more decision variables. Trains can only wait at stations (at most train one per track at any time), and not on segments. Based on a train's entry and traversal times, waiting time restrictions, track availability, and any explicit arrival time window requirements at intermediate stations or the end of

the territory, we can determine possible time periods at which the train can enter each edge e (control point p), permitting us to narrow the time periods in which the train can be at edge e (control point p). Let T_e^q (T_p^q) be the subset of periods in which train $q \in Q_m$ can enter edge e (control point p). At the origin station $s = o_q$, for trains that are already within the territory at the start of the planning horizon, this time window may include only the time period at which the train entered that station (with appropriate adjustments to traversal time so that entry time is at or after time zero). On the other hand, for future trains that will enter the territory later, the time window may include all periods until the end of the horizon (including a dummy period H+1) if the train is permitted to wait at its starting location. Since trains cannot wait on segments, for every segment m and each section $g \in G_m$, we can determine the time needed for a train to enter (and leave) section g after it enters segment m. Specifically, let σ_g^q (λ_g^q) be the time needed for train q to travel from the beginning of segment m to the beginning (end) of section $g \in G_m$. Our model largely focuses on single-track territories in which every segment has a single bi-directional track; with modest changes it also extends to multi-track segments. All the notation (indices, sets, and parameters) needed for the model formulation is listed as follows:

Indices and sets

s	index of a station
m	index of a segment
e	index of an edge
t	index of time
g	index of a section
p	index of a control point
$+,$	indices of train directions
S	set of stations in the territory
M	set of segments in the territory
Q	set of trains to be dispatched
P	set of control points in the territory

Parameters

H	number of time periods in the planning horizon
G_m	set of sections in segment m
θ_p	control point headway (number of periods) at control point p
a_{st}	number of available tracks in station s at time t
o^q	origin edge of train q
d^q	destination edge of train q
τ_e^q	traversal time of train q on edge e
Q_e^k	set of trains that will enter edge e in direction k, where $k \in \{+,\ \}$
Q_e	set of trains that will enter edge e
Q_p	set of trains that will use control point p
f_{st}^q	cost for train q to wait at station s at time t
c_{et}^q	cost for train q to enter edge e at time t

T_e^q set of time periods that train q can enter edge e

T_p^q set of time periods that train q can enter control point p

σ_g^q time (number of periods) for train q to travel from the beginning of segment m, where $g \in G_m$, to the beginning of section g

λ_g^q time (number of periods) for train q to travel from the beginning of segment m, where $g \in G_m$, to the end of section g

b_e^q edge that train q travels on before it enters edge e

b_p^q edge that train q travels on before it uses control point p

Main dispatching decisions are when each train should enter each segment or station, and whether it should wait at a station. To capture these decisions, we define the following decision variables: x_{et}^q =1 if train q enters edge e at time t, and 0 otherwise; y_{st}^q =1 if train q waits at station s (after entering sation s) at time t, and 0 otherwise. Note that we define variables based on segments, rather than on blocks, as in Şahin et al. (2008) and Harrod (2011), thereby dramatically reducing the number of variables.

To make good use of track resources, we can either maximize the weighted velocity of trains or minimize the weighted waiting time of trains. Assume ε^q is the weight for train q. If we want to minimize the weighted waiting time, we can set $f_{st}^q = \varepsilon^q$ for all $s \in S$ and $t \in T_s^q$, $c_{et}^q = (t - t^q)\ \varepsilon^q$ if $e = o^q$, and $c_{et}^q = 0$ otherwise. Next, we will show how to set the objective function coefficients so that the weighted velocity is maximized. Let τ^q represent the total runtime train q use to traverse all the stations and segments in the territory, and D^q represent the total length of such stations and segments. Let t_e^q be the earliest time train q can enter segment or station e, achieved when train q does not wait before entering a. If train q enters d^q at time t, the total time it spends in

the territory is $\tau^q + t - t^q_{d^q}$, and its velocity is $D^q/(\tau^q + t - t^q_{d^q})$. Accordingly, the objective function coefficients can be set in the following way

$$\overline{f}^q_{st} = 0;\ \overline{c}^q_{et} = -\varepsilon^q D^q/(\tau^q + t - t^q_{o^q}) \text{ if } e = d^q, \text{ and } \overline{c}^q_{et} = 0 \text{ if } e \neq d^q \qquad (3.1)$$

Coefficients (3.1) use $c^t_{d^q}$ to record the velocity of train q when q arrives at destination d^q. Alternatively, we can set the coefficients in the following way

$$\begin{aligned} \hat{f}^q_{st} &= \varepsilon^q D^q/(\tau^q + t - t^q_{o^q} + 1) - \varepsilon^q D^q/(\tau^q + t - t^q_{o^q}) \\ \hat{c}^q_{et} &= -\varepsilon^q D^q/(\tau^q + t - t^q_a) \text{ if } e = o^q \\ \hat{c}^q_{et} &= 0 \text{ if } e \neq o^q \end{aligned} \qquad (3.2)$$

Proposition 3.1. Objective function coefficients (3.1) and (3.2) are equivalent.

Proof: We will show that for each train q, both objective function coefficients (3.1) and (3.2) give the same value in any feasible solution. Given any solution (**x, y**), we assume without loss of generality that train q waits in station s_ω at time t_ω, i.e., $y^q_{s_\omega t_\omega} = 1$, for $\omega = 1, 2, ..., \Omega$ and that train q enters edge o^q at time t^*, i.e., $x^q_{o^q t^*} = 1$. Without loss of generality, we assume that $t_1 < t_2 < \cdots < t_\Omega$. Flow conservation constraint imply that $t_\omega = t^q_{s_\omega} + \omega + t^* - t^q - 1$ and that $t^* - t^q = t_1 - t^q_1$. Thus, the velocity of train q given by coefficients (3.2) is

$$\begin{aligned} &\sum_{e\in S\cup M}\sum_{t\in T^q_e} \hat{c}^q_{et} x^q_{et} + \sum_{s\in S}\sum_{t\in T^q_s} \hat{f}^q_{st} y^q_{st} \\ &= -\left(\hat{c}^q_{o^q t^*} + \sum_{\omega=1}^{\Omega} \hat{f}^q_{s_w t_w}\right) \\ &= \frac{D^q}{\tau^q + t^* - t^q} + \sum_{\omega=1}^{\Omega}\left(\frac{\varepsilon^q D^q}{\tau^q + t_\omega - t^q_{s_\omega} + 1} - \frac{\varepsilon^q D^q}{\tau^q + t_\omega - t^q_{s_\omega}}\right) \\ &= \frac{D^q}{\tau^q + t^* - t^q} + \sum_{\omega=1}^{\Omega}\left(\frac{\varepsilon^q D^q}{\tau^q + t^q_{s_\omega} + \omega + t^* - t^q - t^q_{s_\omega}} - \frac{\varepsilon^q D^q}{\tau^q + t^q_{s_\omega} + \omega + t^* - t^q - 1 - t^q_{s_\omega}}\right) \\ &= \frac{D^q}{\tau^q + t^* - t^q} + \sum_{\omega=1}^{\Omega}\left(\frac{\varepsilon^q D^q}{\tau^q + \omega + t^* - t^q} - \frac{\varepsilon^q D^q}{\tau^q + \omega + t^* - t^q - 1}\right) \\ &= \frac{D^q}{\tau^q + \Omega + t^* - t^q} \end{aligned}$$

Assume that train q arrives at d^q at time t', i.e., $x^q_{d^q t'} = 1$. The flow conservation constraints guarantee that $t' = \Omega + t^* - t^q$. Thus, the velocity of train q given by coefficients (3.1) is

$$\sum_{e \in S \cup M} \sum_{t \in T_e^q} \bar{c}^q_{et} x^q_{et} + \sum_{s \in S} \sum_{t \in T_s^q} \bar{f}^q_{st} y^q_{st} = \bar{c}^q_{d^q t'} x^q_{d^q t'} = \frac{D^q}{\tau^q + \Omega + t^* - t^q}$$

which is the same to the velocity given by coefficient profile coefficients (3.2). ♦

Setting objective function coefficients according to (3.1) requires each train to enter its destination, which may not be achieved within the planning horizon. On the other hand, it is not necessary to send the train to its destination to get a valid velocity under coefficients (3.2); if a train ends up in certain location before their destinations, they are assumed to run unimpeded from that location to its destination.

Our computational experience show that the commercial LP solver (e.g., CPLEX) can solve the model with coefficients (3.2) much faster than the model with coefficients (3.1). Thus, we will use coefficients (3.2) in subsequent discussions. To model the implicit time window requirement, we define $c^q_{et} = \hat{c}^q_{et} + \xi^q_{et}$ and use ξ^q_{et} to reflect the preference for train q to enter edge e at time t.

3.3.3. Mathematical formulation

Given the notation defined in the previous section, we can formulate the train dispatching problem as follows:

$$z = \min \sum_{e \in S \cup M} \sum_{q \in Q_e} \sum_{t \in T_e^q} c^q_{et} x^q_{et} + \sum_{s \in S} \sum_{q \in Q_s} \sum_{t \in T_s^q} f^q_{st} y^q_{st} \tag{3.3}$$

subject to

$$\sum_{t \in T_e^q} x^q_{et} = 1 \qquad \forall q \in Q, e = o^q \tag{3.4}$$

$$x^q_{mt_1} = x^q_{st_2} \qquad \forall s \in S \setminus \{o^q\}, q \in O_s, t_2 \in T_s^q, m = b_s^q, t_1 = t_2 - \tau_m^q \tag{3.5}$$

$$x^q_{st_1} + y^q_{s,t_2-1} = x^q_{mt_2} + y^q_{st_2} \qquad \forall m \in M \setminus \{o^q\}, q \in Q_m, t_2 \in T^q_s, s = b^q_m, t_1 = t_2 - \tau^q_s \tag{3.6}$$

$$\sum_{q \in Q_s} y^q_{st} + \sum_{q \in Q_s} \sum_{t_1 = t - \tau^q_s + 1}^{t} x^q_{st_1} \le \alpha_{st} \qquad \forall s \in S, t = 1, \cdots, H \tag{3.7}$$

$$\sum_{q \in Q_p : e = b^q_p} \sum_{t_1 = t - \tau^q_e + 1}^{t - \tau^q_e + \theta_p} x^q_{et_1} \le 1 \qquad \forall p \in P, t \in T^q_p \tag{3.8}$$

$$\sum_{q \in Q_m} \sum_{t' = t - \lambda^q_g + 1}^{t - \sigma^q_g} x^q_{mt'} \le 1 \qquad \forall m \in M, g \in G_m, t \in T^q_m \tag{3.9}$$

$$\sum_{\substack{q \in Q^+_m, \\ t' = t - \sigma^q_g}} x^q_{mt'} + \sum_{\substack{q \in Q^-_m, \\ t' = t - \lambda^q_g}} x^q_{mt'} \le 1 \qquad \forall m \in M, g \in G_m, t \in T^q_m \tag{3.10}$$

$$x^q_{et} \in \{0,1\} \qquad \forall (q, e, t) \in A \tag{3.11}$$

$$y^q_{st} \in \{0,1\} \qquad \forall (q, s, t, t+1) \in A \tag{3.12}$$

The objective function (3.3) minimizes the total cost so that the weighted velocity is maximized or the total waiting time is minimized. Constraints (3.4) make sure that each train enters the territory and is thus dispatched. Constraints (3.5) and (3.6) are flow conservation constraints. Constraints (3.5) specify that if train q enters station s at time t_2, it must have entered segment $m = b^q_s$ at time $t_2 - \tau^q_m$; constraint (3.6) ensures that if train q enters the segment m or waits at its previous station s at time t_2, it must have entered station s at time $t_2 - \tau^q_s$ or waited at station s at time $t_2 - 1$. Constraints (3.7) are the station capacity constraints to ensure that the number of trains moving and waiting in a station s at any time t should not exceed the number of tracks available at the station. Since we assume that the travel time through a station is the same for all tracks at the station, we do not define separate variables for the movement and waiting of trains on each parallel track within a station; using a single variable for a train's movement or waiting at a station not only reduces the problem size but also avoids symmetry in the feasible solution space. We can later apply a post-processing procedure

to assign the trains to specific tracks within a station. Constraints (3.8) enforce the control point headway requirement: at most one train can pass through control point p for every θ_p time units.

Constraints (3.9) permit at most one train to occupy each section in any time period. Since a section consists of two adjacent blocks, constraints (3.9) guarantee that two trailing trains are separated by at least one unoccupied block. Constraints (3.9) can ensure that no meeting or overtaking occurs inside segments. Observe that, if a train overtakes or meets another train inside a segment, they have to appear in some section of that segment at the same time; therefore, permitting no more than one train to use each track section at any time can prevent meets and passes inside segments. Constraints (3.10) prevent trains from crossing each other at the boundary point of two adjacent blocks. As shown in Harrod (2011), the formulation are not valid without constraints (3.10).

The above formulation can solve for the optimal schedule for a given set of trains. To model the *no-wait* trains, we apply a two-stage hierarchical procedure. In the first stage, we solve the problem with only no-wait trains; in the second stage, we fix the solution of no-wait trains and solve for the plans of the rest of the trains.

3.4. MODEL ENHANCEMENTS

Constraints (3.9) specify that at most one train can occupy a specific section at any time; so, trains traveling in opposite directions should never appear in a particular section at the same time. In fact, in any collision-free train schedule, trains traveling in the opposite directions on a segment should never occupy *any* section of the segment (not just a particular section) at the same time. We refer to this property as the *unidirectional movement property*. In this section, we will study how to exploit the unidirectional

movement property to strengthen the model. Specifically, we propose non-concurrency constraints based on sections as well as train movement. For notational convenience, we assume that the control point headway is θ for all control points.

3.4.1. Non-concurrency constraint

One implication of the unidirectional movement property is that, if a train is traveling on any section of a segment at time *t*, then no trains traveling in the opposite direction can travel on any section of this segment at the same time. The following constraints (3.13) enforce this requirement. Similar to constraint (3.9), constraint (3.13) also ensures that at most one train is allowed in each section. Further, as shown in Proposition 3.2, constraints (11) dominate constraints (3.9) and (3.10), i.e., the latter constraints are redundant when we include constraints (3.13) in the model.

$$\max_{g\in G_m}\sum_{q\in Q_m^+}\sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^q + \max_{g\in G_m}\sum_{q\in Q_m^-}\sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^q \le 1 \quad \forall m\in M, t\in \bigcup_{q\in Q_m} T_m^q \tag{3.13}$$

Proposition 3.2. Constraint (3.13) dominates constraints (3.9) and (3.10).

Proof: The proposition follows directly from the fact that any g ∈ G_m, we have

$$\max_{g\in G_m}\sum_{q\in Q_m^+}\sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^q \ge \sum_{q\in Q_m^+}\sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^q \quad \text{and} \quad \max_{g\in G_m}\sum_{q\in Q_m^-}\sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^q \ge \sum_{q\in Q_m^-}\sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^q . \blacklozenge$$

We call constraint (3.13) the *non-concurrency* constraint, since it simultaneously enforces unidirectional movement in segments, section capacity, and no-crossing of trains at section boundaries, thus tightening the model. By introducing indicator variables u_{mt}^+ and u_{mt}^-, we can linearize constraint (3.13) as follows: let $u_{mt}^-(u_{mt}^+)=1$ if any train is traveling in direction $-(+)$ on segment *m* at time *t*, making the segment unavailable to trains traveling in direction $+(-)$, and 0 otherwise. Constraints (3.14), (3.15) and (3.16) capture these definitions and enforce non-concurrency on segment *m*.

$$u_{mt}^{+} \geq \sum_{q \in Q_s^{+}} \sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^{q} \qquad \forall m \in M, g \in G_m, t \in T \tag{3.14}$$

$$u_{mt}^{-} \geq \sum_{q \in Q_s^{-}} \sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^{q} \qquad \forall m \in M, g \in G_m, t \in T \tag{3.15}$$

$$u_{mt}^{+} + u_{mt}^{-} \leq 1 \qquad \forall m \in M, t \in T \tag{3.16}$$

As we can see in Figure 5, if a train $q \in Q_m^{+}$ enters segment m at time $t - \tau_m^q - \theta + 1$, constraints (3.14), (3.15) and (3.16) require that $u_{mt'}^{+}$ =1 for $t' \in \{t' : t - \tau_m^q - \theta + 1 \leq t' \leq t - \theta + 1\}$. The headway at control point 4 require that no trains traveling in direction – can enter segment *m* between time $t - q + 1$ and *t*, which implies that $\sum_{q' \in Q_s^{-}} \sum_{t_1=t+1}^{t+\theta} x_{mt_1}^{q'} = 0$ for all $g \in G_m$. If we also have $\sum_{q' \in Q_s^{+} \setminus \{q\}} \sum_{g \in G_m} \sum_{t_1 \in T_g^q(t)} x_{mt_1}^{q} = 0$ for all $g \in G_m$, i.e., no other trains are traveling on *m* in direction + at time *t*, we could have three possible feasible solutions that give the same objective function value: $(u_{mt'}^{+}, u_{mt'}^{-})$ =(0, 1) or (1, 0) or (0, 0) for $t+1 \leq t' \leq t+\theta$. In constraints (3.14), (3.15), and (3.16), u_{mt}^{+} and u_{mt}^{-} are auxiliary variables, and their actual values are not our concern as long as we have correct *x* values. To break the symmetry, we can enforce $u_{mt'}^{+}$=1 and for $u_{mt'}^{-}$=0 $t+1 \leq t' \leq t+\theta$ if a train $q \in Q_m^{+}$ enters segment *m* at time $t - \tau_m^q + 1$. One of the possible implementations is applying constraints (3.17) and (3.18).

$$u_{mt}^{+} \geq \sum_{q \in Q_s^{+}} \sum_{t_1=t-\tau_m^q-\theta+1}^{t_1=t-\tau_m^q} x_{st_1}^{q} \qquad \forall m \in M, g \in G_m^{+}, t \in \bigcup_{q \in Q} T_s^q \tag{3.17}$$

$$u_{mt}^{-} \geq \sum_{q \in Q_s^{-}} \sum_{t_1=t-\tau_m^q-\theta+1}^{t_1=t-\tau_m^q} x_{st_1}^{q} \qquad \forall m \in M, g \in G_m^{-}, t \in \bigcup_{q \in Q} T_s^q \tag{3.18}$$

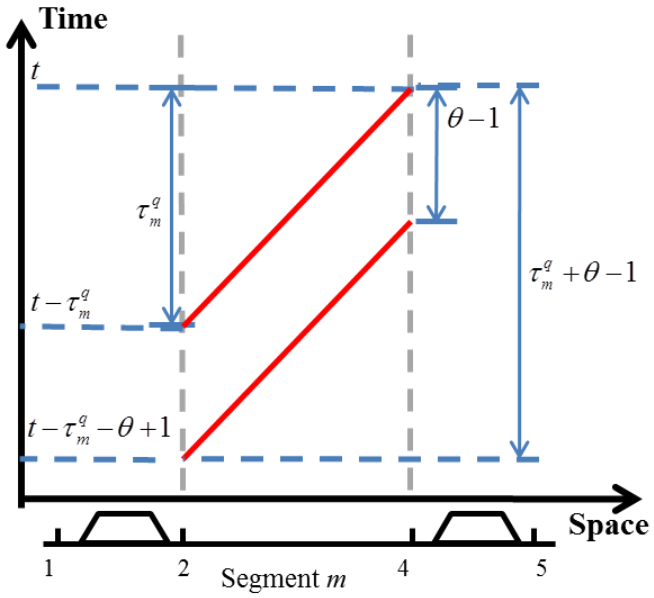


Figure 5. Unidirectional movements

3.4.2. Refining the track sections

For trains traveling in the same direction, we enforce two types of headway requirement, namely trailing headway and control point headway, to keep safe distances between trains. This section discusses how to refine the trailing headway by exploiting the control point headway.

Since trains must pass through a control point to enter or exit a segment, the control point headway can impact the trailing distances between trains. If a slow train trails a fast train, they must be at least θ time periods apart when entering a segment; the distance between them grows as they travel on the segment. Hence, the minimal distance between these two trains is at least the distance traversed by the fast train in θ time periods. The same result holds when a fast train trails a slow train. Assume l_m^+ and l_m^- are the distance traversed during θ time periods by the second slowest train that can travel on segment m in direction + and – respectively. Then in any feasible solution,

the distances between any two trailing trains traveling on segment m in direction + and – are no smaller than l_m^+ and l_m^- respectively.

Figure 6 provides a procedure to refine the track sections for segment m by making sections in G_m^+ and G_m^- no shorter than l_m^+ and l_m^- respectively. The procedure first finds the distance traversed during θ time period by the second slowest train in both directions in Step 1. In Step 2 and Step 3, the procedure defines the sections for trains traveling in both directions respectively by finding the starting and ending location of each section.

After the refinement, we use G_m^+ and G_m^- to refine constraints (3.14) and (3.15) as

$$u_{mt}^+ \geq \sum_{q \in Q_s^+} \sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^q \qquad \forall m \in M, g \in G_m^+, t \in T \tag{3.19}$$

$$u_{mt}^- \geq \sum_{q \in Q_s^-} \sum_{t'=t-\lambda_g^q+1}^{t-\sigma_g^q} x_{mt'}^q \qquad \forall m \in M, g \in G_m^-, t \in T \tag{3.20}$$

Since sections in G_m^+ and G_m^- are longer, constraints (3.19) and (3.20) have more variables movement variable at their right hand sides than constraints (3.14) and (3.15) respectively, which leads to stronger constraints. Besides, the number of constraints (3.19) and (3.20) is smaller than number of constraints (3.14) and (3.15), since G_m^+ and G_m^- may have fewer sections than G_m.

Procedure refine_section

Input: segment m; ending locations of blocks in m: $\mathbf{l}_m = (l_m(1), l_m(2), \cdots, l_m(K))$.

Output: G_m^+ and G_m^- (sections on segment m for directions + and −).

Step 1: set $l_m(0) = 0$; $G_m^+ = \phi$; $G_m^- = \phi$

l_m^+, l_m^- = distance traversed during θ time periods by the second slowest train traveling in direction + and − on segment m

Step 2: For $k = 1$ to $K - 1$

Section g_k^+ starts at $l_m(k-1)$ and ends at Max{ $l_m(k+1)$, $l_m(k-1)+l_m^+$ }

Add g_k^+ to G_m^+

If $l_m(k-1)+l_m^+ \geq l_m(K)$

Go to Step 3;

Step 3: For $k = 1$ to $K - 1$

Section g_k^- starts at $l_m(k-1)$ and ends at Max{ $l_m(k+1)$, $l_m(k-1)+l_m^-$ }.

add g_k^- to G_m^-.

If $l_m(k-1)+l_m^- \geq l_m(K)$

Terminate the procedure.

Figure 6. Procedure to refine section

In some situations, we may not have the exact block location (namely $l_m(k)$, for k =1, 2, ⋯, K), or we may not even have trailing headway requirement (e.g., Caprara et al. 2002). Instead of using pairwise unidirectional inequality (3.23) discussed later, we can apply the procedure described in Figure 7 to define sections, based on which we can enforce non-concurrency constraints (3.19) and (3.20). Since constraints (3.19), (3.20) and (3.16) prevent trains from occupying a segment in different directions at any time, we don't need to enforce constraints to prevent the simultaneous crossing of trains at the boundary point of two adjacent sections.

Procedure define_section

Input: segment m; length of m: L_m;

Output: G_m^+ and G_m^- (set of sections on segment m for directions + and −).

Step 1: l_m^+, l_m^- = distance traversed during θ time periods by the second slowest train traveling in direction + and − on segment m

Number of sections $K^+ = \lceil L_m / l_m^+ \rceil$ and $K^- = \lceil L_m / l_m^- \rceil$ in segment m;

$G_m^+ = \phi$; $G_m^- = \phi$

Step 2: For k =1 to $K^+ - 1$

Section g_k^+ starts at $(i-1)l_m^+$ and ends at il_m^+;

Add g_k^+ to G_m^+;

Section g^+ starts at $L_m - l_m^+$ and ends at L_m;

Add the section to G_m^+.

Step 3: For k =1 to $K^- - 1$

section g_k^- starts at $(i-1)l_m^-$ and ends at il_m^-;

add g_k^- to G_m^-;

Section g^- starts at $L_m - l_m^-$ and ends at L_m;

Add the section to G_m^-.

Figure 7. Procedure to define section

3.4.3. Train-based unidirectional inequality

The unidirectional movement property requires that at any time, only trains traveling in the same direction can appear in a segment, which is enforced by constraints (3.16). From this point of view, the right hand sides of constraints (3.14) and (3.15) (or the refined constraints (3.19) and (3.20)) are used to activate the binary variables u_{mt}^+ and u_{mt}^-. Since u_{mt}^+ (also u_{mt}^-) is a binary variable, the set of variables used to enforce its value should constitute a clique, which means that no more than one of them can be active at any feasible solution. Constraints (3.14) and (3.15) can be viewed as finding cliques based on the requirement that at most one train can occupy each section. Here,

we study ways to enforce unidirectional movement from the perspectives of each train. For simplicity, we assume that the minimum headway is θ for all control points from this point.

Since all trains move only in a single pre-specified direction, every train can enter every segment at most once, providing a way to identify cliques, which we call train-based cliques. As Figure 5 illustrates, if train *q* (without loss of generality, we assume it is traveling in direction +) enters segment *m* at a certain time *t*, it will exit the segment at time $t+\tau_m^q$, and so the train is traveling on the segment between time *t* and $t+\tau_m^q-1$. Accordingly, if any train *q* enters segment *m* at any time between $t-\tau_m^q+1$ and *t*, it is traveling on segment *m* at time *t*, thus $u_{mt'}^+ = 1$ and $u_{mt'}^- = 0$ for $t-\tau_m^q+1 \le t' \le t$; besides, the symmetry breaking constraints (3.17) and (3.18) enforce that $u_{mt'}^+ = 1$ and $u_{mt'}^- = 0$ for $t-\tau_m^q-\theta+1 \le t' \le t-\tau_m^q$. Hence, we can activate variables u_{mt}^+ and u_{mt}^- by using inequalities (3.21) and (3.22).

$$u_{mt}^+ \ge \sum_{t_1=t-\tau_m^q-\theta+1}^{t} x_{mt_1}^q \qquad \forall m \in M, q \in Q_m^+, t \in T \qquad (3.21)$$

$$u_{mt}^- \ge \sum_{t_1=t-\tau_m^q-\theta+1}^{t} x_{mt_1}^q \qquad \forall m \in M, q \in Q_m^-, t \in T \qquad (3.22)$$

Note that inequalities (3.21) and (3.22) do not dominate non-concurrency constraints (3.19) and (3.20), and vice versa: inequalities (3.21) and (3.22) are both enforced for each train but include more variables for each train; in contrast, inequalities (3.19) and (3.20) are enforced for all trains in a certain direction, but with fewer variables for each train.

3.4.4. Pairwise unidirectional inequalities

To prevent trains from meeting inside a segment in the bi-directional train timetabling problem, Cacchiani et al. (2010) propose a set of crossing constraints. Given

any pair of trains $q_1 \in Q_m^+$, $q_2 \in Q_m^-$ and a time window from time t_1 to $\bar{t}_1$ with $\bar{t}_1 - \tau_m^{q_2} - \theta + 1 \le t_1 + \tau_m^{q_1} + \theta - 1$, we can express their constraints as follows,

$$\sum_{t'=t_1}^{\bar{t}_1} x_{mt'}^{q_1} + \sum_{t'=t_2}^{\bar{t}_2} x_{mt'}^{q_2} \le 1 \qquad (3.23),$$

where $t_2 = \bar{t}_1 - \tau_m^{q_2} - \theta + 1$ and $\bar{t}_2 = t_1 + \tau_m^{q_1} + \theta - 1$. If we take $t_1 = t - \tau_m^{q_1} - \theta + 1$ and $\bar{t}_1 = t$, we have

$$\sum_{t=t-\tau_m^{q_1}-\theta+1}^{t} x_{mt}^{q_1} + \sum_{t=t-\tau_m^{q_2}-\theta+1}^{t} x_{mt}^{q_2} \le 1 \quad \forall q_1 \in Q_m^+, q_2 \in Q_m^-, m \in M^{q_1} \cap M^{q_2}, t \in T \qquad (3.24)$$

Proposition 3.3 shows that train-based non-concurrency inequalities (3.21), (3.22) and (3.16) are equivalent to inequality (3.24), a specifical case of general unidirectional inequality (3.23); by defining variables u_{mt}^+ and u_{mt}^-, the number of inequalities is significantly reduced.

Proposition 3.3. Constraint (3.24) is LP-equivalent to constraints (3.21), (3.22) and (3.16).

Proof: It is obvious that (3.24) is implied by (3.21), (3.22) and (3.16). We need to show that constraint (3.24) implies (3.21), (3.22) and (3.16). Since constraint (3.24) applies to all $q \in Q_m^+$, we have

$$\sum_{t=t-\tau_m^{q_1}-\theta+1}^{t} x_{mt}^{q_1} \le 1 - \sum_{t=t-\tau_m^{q_2}-\theta+1}^{t} x_{mt}^{q_2} \quad \forall q_1 \in Q_m^+, q_2 \in Q_m^-, m \in M^{q_1} \cap M^{q_2}, t \in T,$$

which is equivalent to

$$\max_{q \in Q_m^+} \sum_{t=t-\tau_m^{q_1}-\theta+1}^{t} x_{mt}^{q} \le 1 - \sum_{t=t-\tau_m^{q_2}-\theta+1}^{t} x_{mt}^{q_2} \qquad \forall q_2 \in Q_m^-, m \in M^{q_1} \cap M^{q_2}, t \in T.$$

for any train $q \in Q_m^+$. Applying the same logic to the above inequality for $q \in Q_m^-$, we can have,

$$\max_{q \in Q_m^+} \sum_{t=t-\tau_m^{q_1}-\theta+1}^{t} x_{mt}^{q} + \max_{q \in Q_m^-} \sum_{t=t-\tau_m^{q_1}-\theta+1}^{t} x_{mt}^{q} \le 1 \qquad \forall m \in M, t \in T_m^q \bigcup T_m^{q'} \qquad (3.25)$$

Linearizing (3.25), we can get constraints (3.21), (3.22) and (3.16). Constraints (3.21) , (3.22) and (3.16) are implied by constraints (3.24). Hence, these two set of constraints are LP-equivalent. ♦

The variables in the left hand side of constraint (3.23) form a clique. By expoliting the control point headway requirement, we can incoporate more variables in the clique and thus identify a inequality that is stronger than constraint, as demonstrated in proposition 3.4.

Proposition 3.4. For any t_1 , t_2 , $\bar{t}_1$, $\bar{t}_2$, $t(q')$ and $\bar{t}(q')$ with $t_2 = \bar{t}_1 - \tau_m^{q_2} - \theta + 1$, $\bar{t}_2 = t_1 + \tau_m^{q_1} + \theta - 1$, $t(q') = \text{Max}\{\bar{t}_1 - \theta + 1,\ \bar{t}_2 - \tau_m^{q'} - \theta + 1,\ t_1\}$, $\bar{t}(q') = \min\{\ t_1 + \theta - 1,\ t_2 + \tau_m^{q_2} + \theta - 1\}$ for $q' \in Q_m^+ \setminus \{q_1\}$, inequality (3.26) is valid.

$$\sum_{t'=t_1}^{\bar{t}_1} x_{mt'}^{q_1} + \sum_{q' \in Q_m^+ \setminus \{q_1\}} \sum_{t'=t(q')}^{\bar{t}(q')} x_{mt'}^{q'} + \sum_{t'=t_2}^{\bar{t}_2} x_{mt'}^{q_2} \le 1 \qquad (3.26),$$

Proof: Since $t(q') \ge t_1$ and $\bar{t}(q') \le t_1 + \theta - 1$, we have

$$\bar{t}(q') - t(q') \le t_1 + \theta - 1 - t_1 = \theta - 1;$$

therefore, the control point headway would imply that

$$\sum_{q' \in Q_m^+ \setminus \{q_1\}} \sum_{t'=t(q')}^{\bar{t}(q')} x_{mt'}^{q'} \le 1 \qquad (3.27)$$

If we know that a train $q \in Q^+$ enters segment m between time t_1 and $\bar{t}_1$, according to the control point headway, no trains traveling in direction + can enter segment m between time $\bar{t}_1 - \theta + 1$ and $t_1 + \theta - 1$, as required by the control point headway. Thus, we have

$$\sum_{t'=t_1}^{\bar{t}_1} x_{mt'}^{q_1} + \sum_{q' \in Q_m^+ \setminus \{q_1\}} \sum_{t'=t(q')}^{\bar{t}(q')} x_{mt'}^{q'} \le 1 \qquad (3.28).$$

For $q' \in Q_m^+ \setminus \{q_1\}$, we have $t(q') \ge \bar{t}_2 - \theta - \tau_m^{q'} + 1$ and $\bar{t}(q') \le t_2 + \tau_m^{q_2} + \theta - 1$, pairwise unidirectional inequalities would suggest that

$$\sum_{t'=t(q')}^{\bar{t}(q)} x_{mt'}^{q'} + \sum_{t'=t_2}^{\bar{t}_2} x_{mt'}^{q_2} \le 1 \qquad (3.29).$$

Combining with inequality (3.27), we have,

$$\sum_{q' \in Q_m^+ \setminus \{q_1\}} \sum_{t'=t(q')}^{\bar{t}(q')} x_{mt'}^{q'} + \sum_{t'=t_2}^{\bar{t}_2} x_{mt'}^{q_2} \le 1 \qquad (3.30).$$

As $t_2 \ge \bar{t}_1 - \tau_m^{q_2} - \theta + 1$ and $\bar{t}_2 \le t_1 + \tau_m^{q_1} + \theta - 1$, pairwise unidirectional inequality (3.23) is valid. Taking ½ [(3.28) + (3.30) + (3.23)] and rounding down the right hand side give (3.26).♦

Although we do not need inequalities (3.23) and (3.26) to define the feasible region in our model, adding them as cutting planes can help to strengthen our formulation. Since the number of inequalities (3.23) and (3.26) are too large to be fully identified *a priori*, we develop a separation procedure to identify violated inequalities during the branch and bound algorithm. Both inequalities are enforced for each pair of trains traveling on each specific segment in the opposite directions; for each pair of trains and a segment, any given values of t_1 and $\bar{t}$ can determine values of t_2 and $\bar{t}_2$ in inequality (3.23) and values of t_2, $\bar{t}_2$ $t(q')$, and $\bar{t}(q')$ in inequality (3.26). Thus, the complexity for separation is $O\left(\left|Q_m^+\right|\left|Q_m^-\right|\left|M\right|H^2\right)$. We can reduce the search procedure by applying the results in corollory 3.5: as long as some inequality is violated by a factional solution, we can always cut off the solution with inequalities with $x_{mt_1}^{q_1} > 0$ and $x_{m\bar{t}_1}^{q_1} > 0$. Accordinly, we can just search for inequalities with $x_{mt_1}^{q_1} > 0$ and $x_{m\bar{t}_1}^{q_1} > 0$. The detailed separation procedure is described in Appendix A.

Corollory 3.5. Given a fractional solution (**x**, **y, u**), if it does not satisfy all possible inequalities (3.26), there is a violated inequality (3.26) with $x_{mt_1}^{q_1} > 0$ and $x_{m\bar{t}_1}^{q_1} > 0$.

Proof: if the solution (**x**, **y, u**) violates inequality (3.23), without loss of generality, we assume there exists (t, $\bar{t}$, t_2, $\bar{t}_2$) such that

$$\sum_{t'=t}^{\bar{t}} x_{mt'}^{q_1} + \sum_{q' \in Q_m^+ \setminus \{q_1\}} \sum_{t'=t(q')}^{\bar{t}(q')} x_{mt'}^{q'} + \sum_{t'=t_2}^{\bar{t}_2} x_{mt'}^{q_2} > 1.$$

Taking $t_1 = \arg\min_{t'}\{t' : t' \geq t, x^{q_1}_{mt'} > 0\}$ and $\overline{t}_1 = \arg\max_{t'}\{t' : t' \leq \overline{t}, x^{q_1}_{mt'} > 0\}$, we have $\sum_{t'=t}^{\overline{t}} x^{q_1}_{mt'} = \sum_{t'=t_1}^{\overline{t}_1} x^{q_1}_{mt'}$ and $t \leq t_1 \leq \overline{t}_1 \leq \overline{t}$. Thus, we find a violated inequality (23), with $x^{q_1}_{mt_1} > 0$ and $x^{q_1}_{m\overline{t}_1} > 0$. ♦

3.4.5. Pairwise unidirectional inequalities across segments

Inequality (3.23) prevents the incompatible movements between two trains traveling on a specific segment in opposite directions. Lemma 3.6 extends the results to more than one segment.

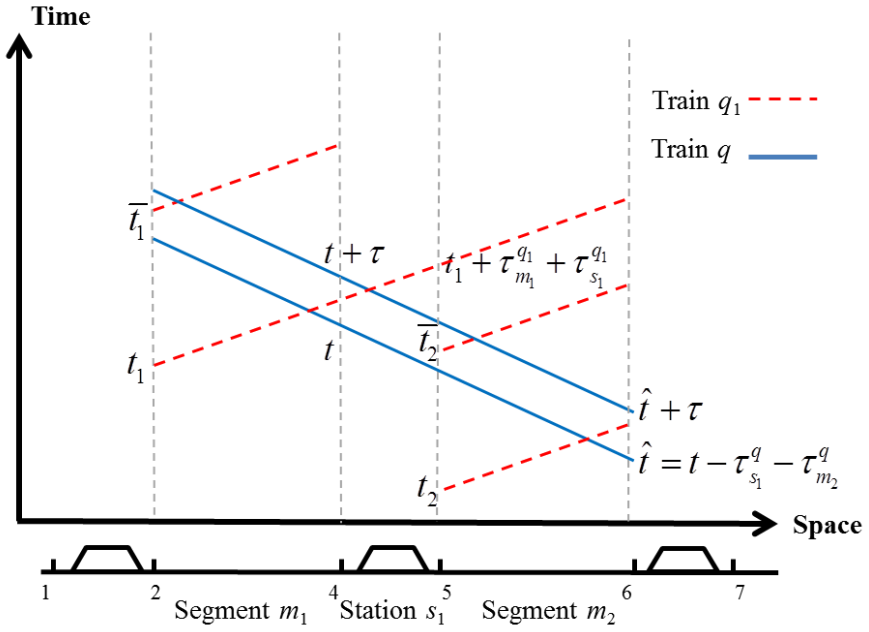


Figure 8. Illustration of unidirectional movements across segments

Lemma 3.6. Assume train q_1 (traveling in direction +) and train q (traveling in direction −) can both enter two adjacent segments m_1 and m_2, and the station s_1 between them. For $\hat{t}$, t, $\overline{t}_1$, $\overline{t}_2$, t_1, t_2 that satisfy

$$\hat{t} = t - \tau^{q}_{s_1} - \tau^{q}_{m_2},\ \overline{t}_1 = t + \tau^{q}_{m} + \theta - 1,\ t_2 = \hat{t} + \tau - \theta - \tau^{q_1}_{m_2} + 1$$

and either of the following condition holds:

(1) $t_1 = t + \tau - \tau_m^{q_1} - \theta + 1$ and $\overline{t_2} = \min\{t_1 + \tau_{m_1}^{q_1} + \tau_{s_1}^{q_1} - 1, \hat{t} + \tau_{m_1}^{q} + \theta - 1\}$,

(2) $\overline{t_2} = \hat{t} + \tau_{m_1}^{q} + \theta - 1$ and $t_1 = \max\{t + \tau - \tau_m^{q_1} - \theta + 1, \overline{t_2} - \tau_{m_1}^{q_1} - \tau_{s_1}^{q_1} + 1\}$.

Inequality

$$\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q} - y_{s_1(t-1)}^{q} + \sum_{t'=t_1}^{\overline{t_1}} x_{m_1 t'}^{q_1} + \sum_{t'=t_2}^{\overline{t_2}} x_{m_2 t'}^{q_1} \le 1 \tag{3.31}$$

is valid.

Proof: Here we prove that the proposition is true when condition (1) holds. If train q_1 enters segment m_1 between time t_1 and $\overline{t_1}$, it will enter segment segment m_2 no earlier than $t_1 + \tau_{m_1}^{q_1} + \tau_{s_1}^{q_1}$ (since $\overline{t_2} < t_1 + \tau_{m_1}^{q_1} + \tau_{s_1}^{q_1}$). Therefore, inequality

$$\sum_{t'=t_2}^{\overline{t_2}} x_{m_2 t'}^{q_1} + \sum_{t'=t_1}^{\overline{t_1}} x_{m_1 t'}^{q_1} \le 1 \tag{3.32}$$

is valid. If $t_1 = t + \tau - \tau_m^{q_1} - \theta + 1$ and $\overline{t_1} = t + \tau_m^{q} + \theta - 1$, the pairwise unidirectional inequality (3.23) in segment m_1 implies that inequality

$$\sum_{t'=t_1}^{\overline{t_1}} x_{m_1 t'}^{q_1} + \sum_{t'=t}^{\overline{t}} x_{m_1 t'}^{q} \le 1 \tag{3.33}$$

is valid. Since $\overline{t_2} \le t - \tau_{s_1}^{q} + \theta - 1$ and $t_2 = \hat{t} + \tau - \theta - \tau_{m_2}^{q_1} + 1$, inequality (3.23) in segment m_2 implies that inequality

$$\sum_{t'=t_2}^{\overline{t_2}} x_{m_2 t'}^{q_1} + \sum_{t'=\hat{t}}^{\hat{t}+\tau} x_{m_2 t'}^{q} \le 1 \tag{3.34}$$

is valid.

For notational convenience, we define $x^* = \left(\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q}\right)\left(\sum_{t'=\hat{t}}^{\hat{t}+\tau} x_{m_2 t'}^{q}\right)$. Since $\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q} \le 1$ and $\sum_{t'=\hat{t}}^{\hat{t}+\tau} x_{m_2 t'}^{q} \le 1$, we have $\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q} \ge x^*$ and $\sum_{t'=\hat{t}}^{\hat{t}+\tau} x_{m_2 t'}^{q} \ge x^*$. Thus, inequalities (3.33) and (3.34) would imply inequalities (3.35) and (3.36).

$$\sum_{t'=t_1}^{\overline{t_1}} x_{m_1 t'}^{q_1} + x^* \le 1 \tag{3.35}$$

$$\sum_{t'=t_2}^{\overline{t_2}} x_{m_2 t'}^{q_1} + x^* \le 1 \tag{3.36}$$

Taking ½[(3.32)+(3.35)+(3.36)] and rounding down the right hand side give inequality

$$\sum_{t'=t_2}^{\bar{t}_2} x_{m_2 t'}^{q_1} + \sum_{t'=t_1}^{\bar{t}_1} x_{m_1 t'}^{q_1} + x^* \le 1 \qquad (3.37).$$

Summing over flow conservation constraints for train q at station s_1 and segment m_1 between time t and $t+\tau$ gives $\sum_{t'=\hat{t}}^{\hat{t}+\tau} x_{m_2 t'}^{q} = \sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q} + y_{s_1 \bar{t}}^{q} - y_{s_1 (t-1)}^{q}$, which leads to $x^* = \left(\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q}\right)\left(\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q} + y_{s_1 \bar{t}}^{q} - y_{s_1 (t-1)}^{q}\right)$.

For any feasible solution **x**, $\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q}$=0 or 1 and ($\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q}$, $y_{s_1 \bar{t}}^{q}$)=(0 , 1) or (1, 0). Thus,

$$\left(\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q}\right)^2 = \sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q}, \; y_{s_1 \bar{t}}^{q} \times \sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q} = 0, \text{ and } \; y_{s_1 (t-1)}^{q} \times \sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q} \le y_{s_1 (t-1)}^{q}.$$

Accordingly, we know that inequality (3.38) is valid.

$$\sum_{t'=t}^{t+\tau} x_{m_1 t'}^{q} - y_{s_1 (t-1)}^{q} \le x^* \qquad (3.38)$$

Inequalities (3.37) and (3.38) imply that inequality (3.31) is valid.♦

We may view the development of inequality (3.31) as a lift-and-project cutting plane algorithm for 0-1 programs (Balas et al., 1993); we first lift the soluton space to a higher dimension by adding variable x^*, and then project back to the original solution space, i.e., eliminate variable x^*, by expoliting the our problem structure. Intuitively, we can interpret inequality (3.31) in the following way: if train q enters segment m_1 between time t and $t+\tau$, its movement is incompatible with the entry of train q_1 into segment m_1 between time t_1 and $\bar{t}_1$; if it has not waited at station s_1 at time $t-1$, it should have entered segment m_2 between time $\hat{t}$ and $\hat{t}+\tau$, and thus its movement should be in conflict with the entry of train q_1 into segment m_2 between time t_2 and $\bar{t}_2$; thus, if train q enters segment m_1 between time t and $t+\tau$, its movement is not compatible with the entry of q_1 into segment m_1 between time t_1 and $\bar{t}_1$ and the entry of train q_1 into segment m_2 between

time t_2 and $\overline{t}_2$. We can extend this logic to more than 2 segments, as stated in Proposition 3.7.

Proposition 3.7. Assume trains q_1 (traveling in direction +) and q (traveling in direction −) can enter K adjacent segments and the stations between these segments, denoted as m_1, s_1, m_2, s_2, $\cdots m_{K-1}$, s_{K-1}, m_K. For any t, $\hat{t}_1 = t$, $\hat{t}_{k'} = t - \sum_{k=1}^{k'-1} \tau^q_{m_k} - \sum_{k=1}^{k'-1} \tau^q_{s_k}$ for k' = 2, …, K, $\overline{t}_1 = t + \tau^q_{m_1} + \theta - 1$, $t_K = \hat{t}_K + \tau - \tau^q_{m_K} - \theta + 1$, $t_k \le \overline{t}_k$ for all k =1, 2, …, K, and either of the following conditions holds for each $k = 2, \cdots, K$,

$$t_{k-1} = \hat{t}_{k-1} + \tau - \tau^q_{m_{k-1}} - \theta + 1 \text{ and } \overline{t}_k = \min\{t_{k-1} + \tau^{q_1}_{m_{k-1}} + \tau^{q_1}_{s_{k-1}} - 1, \hat{t}_k + \tau^q_{m_k} + \theta - 1\}$$

$$\overline{t}_k = \hat{t}_k + \tau^q_{m_k} + \theta - 1 \text{ and } t_{k-1} = \max\{\overline{t}_k - \tau^{q_1}_{m_{k-1}} - \tau^{q_1}_{s_{k-1}} + 1, \hat{t}_{k-1} + \tau - \tau^q_{m_{k-1}} - \theta + 1\},$$

inequality

$$\sum_{t'=t}^{t+\tau} x^q_{m_1 t'} - \sum_{k=1}^{K-1} y^q_{s_k(\hat{t}_k - 1)} + \sum_{k=1}^{K} \sum_{t'=t_k}^{\overline{t}_k} x^{q_1}_{m_k t'} \le 1 \tag{3.39}$$

is valid.

Proof: Similar to Proof of Lemma 4.1. ♦

Since there are two possible situations corresponding to any $\overline{t}_k$ and t_{k-1} pair for k = 2, ⋯, K, number of possible situations grows exponentially with K. Corollory 3.7 shows that we can find the most violated inequality in $O(K)$ for any given pair of trains, sequence of adjacnet edges, t and τ, and thus we can identify the most violated inequality (3.39) in complexity $O\left(\left|Q^+_m\right|\left|Q^-_m\right|\left|M\right|H^2K\right)$.

Corollory 3.7. Given a fractional solution (**x**, **y, u)**, pair of trains q and q_1, seqence of adjacent edges m_1, s_1, m_2, s_2, $\cdots m_{K-1}$, s_{K-1}, m_K, and t and τ, there is a $O(K)$ algorithm to find the most violated inequality (3.39), if any.

Proof: To get the most violated ineqiaity (3.39), if any, we need to find the (t_{k-1}, $\overline{t}_k$) pairs for k = 2, ⋯, K that will maximize $\sum_{t'=t}^{t+\tau} x^q_{m_1 t'} - \sum_{k=1}^{K-1} y^q_{s_k(\hat{t}_k - 1)} + \sum_{k=1}^{K} \sum_{t'=t_k}^{\overline{t}_k} x^{q_1}_{m_k t'}$,

which is equivanet to maximizing $\sum_{k=1}^{K}\sum_{t'=t_k}^{\overline{t}_k} x_{m_k t'}^{q_1}$ $\sum_{k=1}^{K}\sum_{t'=t_k}^{\overline{t}_k} x_{m_k t'}^{q_1}$. We will demonstrate a dynamic programming algorithm to solve the problem. For the convenicence of notation, we define the following notation for any given *t* and τ :

$$\overline{t_1^1} = \overline{t_1^2} = t + \tau_{m_1}^q + \theta - 1,$$

$$t_K^1 = t_K^2 = \hat{t}_K + \tau - \tau_{m_K}^q - \theta + 1,$$

$$t_{k-1}^1 = \hat{t}_{k-1} + \tau - \tau_{m_{k-1}}^q - \theta + 1 \text{ for all } k = 2, \cdots, K,$$

$$\overline{t_k^1} = \min\{t_{k-1} + \tau_{m_{k-1}}^{q_1} + \tau_{s_{k-1}}^{q_1} - 1, \hat{t}_k + \tau_{m_k}^q + \theta - 1\} \text{ for all } k = 2, \cdots, K;$$

$$t_{k-1}^2 = \max\{\overline{t_k} - \tau_{m_{k-1}}^{q_1} - \tau_{s_{k-1}}^{q_1} + 1, \hat{t}_{k-1} + \tau - \tau_{m_{k-1}}^q - \theta + 1\} \text{ for all } k = 2, \cdots, K$$

$$\overline{t_k^2} = \hat{t}_k + \tau_{m_k}^q + \theta - 1 \text{ for all } k = 2, \cdots, K.$$

For k' = 1, ⋯, *K*−1 and *i* = 1, 2, we define

$$f_{k'}^i = \max\left\{\sum_{k=1}^{k'}\sum_{t'=t_k}^{\overline{t}_k} x_{m_k t'}^{q_1} : t_{k'} = t_{k'}^i; (t_{k-1}, \overline{t_k}) = (t_{k-1}^1, \overline{t_k^1}) \text{ or } (t_{k-1}^2, \overline{t_k^2})\} \forall k = 2, \cdots k'\right\}$$

and define $h_k^{ij} = f_{k-1}^i + \sum_{t'=t_k^i}^{\overline{t_k^j}} x_{m_k t'}^{q_1}$ if $t_k^i \le \overline{t_k^j}$, and 0 otherwise for *k* = 2, ⋯, *K* and *i, j* = 1, 2. Then we have $\max \sum_{k=1}^{K}\sum_{t'=t_k}^{\overline{t}_k} x_{m_k t'}^{q_1} = f_K^1 = f_K^2$. Accordingly, the dynamic programming recursion proceeds in the following fashion:

$$f_1^i = h_1^i = \sum_{t'=t_1^1}^{\overline{t}_1} x_{m_1 t'}^{q_1} \text{ for } i = 1, 2;$$

$$h_k^{ij} = f_{k-1}^i + \sum_{t'=t_k^i}^{\overline{t_k^j}} x_{m_k t'}^{q_1} \text{ if } t_k^i \le \overline{t_k^j}, \text{ and 0 otherwise}$$

$$f_k^j = \max\{h_k^{1j}, h_k^{2j}\}.$$

It is easy to check that the total number of updates is *O*(*K*). ♦

Our experience indicates that inequalities with more than two segments are rearely violated. Thus, we focus on a separation procedure for inequality (3.31). For each pair of trains traveling in opposite directions in any segmen, any possible value of *t* and $\overline{t}$ corresponds to two pairs of (t_1, $\overline{t}_1$, t_2, $\overline{t}_2$); thus, the complexity of separation for inequality (3.31) is $O\left(\left|Q_m^+\right|\left|Q_m^-\right|\left|M\right|H^2\right)$. Corollory 3.8 can help to accelerate the

separation procedure in practice by restricting the search to set of (t, $\overline{t}$, t_1, $\overline{t}_1$, t_2, $\overline{t}_2$) pairs with $x^q_{m\overline{t}} > 0$; $x^q_{m_1 t} > 0$ or $y^q_{s_1 t} > 0$, and $x^{q_1}_{m_1 t_1} > 0$ or $x^{q_1}_{m_2 \overline{t}_2} > 0$. The detailed procedure is discussed in Appendix A.

Corollory 3.8. Given a fractional solution (**x**, **y, u)**, if **x** do not satisfy all possible inequalities (3.39), we can find a violated inequality (3.31) with $x^q_{m\overline{t}} > 0$; $x^q_{m_1 t} > 0$ or $y^q_{s_1 t} > 0$, and $x^{q_1}_{m_1 t_1} > 0$ or $x^{q_1}_{m_2 \overline{t}_2} > 0$.

Proof: similar to the proof of Corollory 3.7. ♦

3.5. SEQUENTIAL DISPATCHING HEURISTIC

A heuristic solution procedure is important when the optimization model fails to find a good solution within reasonable amount of time. Since trains can only wait inside stations, our heuristic tries to move trains from one station to the next sequentially and iteratively. When the planning begins with some trains traveling at any segment, these trains move to the next stations during initialization. Besides, we treat terminals at the ends of the territory as stations with infinite capacities. Each iteration consists of three stages: the *pre-dispatching stage*, the *active train selection stage*, and the *dispatching stage*. In the pre-dispatching stage, system state is updated to avoid deadlocks; in the active train selection stage, an *active train* is chosen as the candidate to move forward in the dispatching stage; in the dispatching stage, the heuristic decides whether to delay the active train or move it forward.

3.5.1. Notation and definitions

At any point of the heuristic, we describe the state of the dispatching system by the station that each train is in, the movements and waiting of trains inside the territory that are already planned, and the track unavailability due to planned train movements and waiting and explicit time window requirements. We use Ψ to denote such a state.

Given any system state Ψ, we define the following notation to facilitate the discussion of the heuristic procedure:

$s(q, \Psi)$	the station train q is in
$m(q, \Psi)$	$= e^q_{s(q,\Psi)}$, i.e., the next segment that train q traverse after exiting $s(q, \Psi)$
$s_1(q, \Psi)$	$= e^q_{m(q,\Psi)}$, i.e., the next station that train q traverse after exiting $m(q, \Psi)$
$p_1(q, \Psi)$	train q's entry control points of $m(q, \Psi)$
$p_2(q, \Psi)$	train q's entry control points of $s_1(q, \Psi)$
$t(q, \Psi)$	time up until which the schedule of train q is determined at state Ψ
$Q(\Psi)$	set of trains that have not been sent to their destinations, ordered increasingly by $t(q, \Psi)$
$Q(\Psi, s)$	set of trains that are currently in station s
$w(s, q, \Psi)$	total waiting time of train q in station s in state Ψ
$c(\Psi)$	total cost of the movement plan if each train q leaves $s(q, \Psi)$ at $t(q, \Psi)$ and runs without waiting until its destination

Note that $c(\Psi)$ gives an upper bound on the best objective function value that can be achieved from state Ψ, and thus can be used as metric to compare different states. In any state Ψ, each train can either be moved forward to next station or held to wait at the current station for some time. In any given state Ψ, we define the following procedures:

- *move*(q, Ψ) makes train q move out of station $s(q, \Psi)$ at time $t(q, \Psi)$, forward to $m(q, \Psi)$ and then into $s_1(q, \Psi)$. The detailed procedure is discussed in Figure 9. The procedure assigns train q to use control points $p_1(q, \Psi)$ and $p_2(q, \Psi)$, sections in segment $m(q, \Psi)$, and station $s_1(q, \Psi)$.

- *wait*(q, t, Ψ) makes train q wait at station $s(q_2, \Psi)$ between time $t(q_2, \Psi)$ and $t-1$. The detailed procedure is presented in Figure 10. The procedure assigns train q to use of one the wait tracks in train q.
- *movable*(q, t, Ψ) checks if train q can move from station $s(q, \Psi)$ to $s_1(q_2, \Psi)$ at time t. Figure 11 shows the detailed procedure. Specifically, the procedure checks if the control points $p_1(q, \Psi)$ and $p_2(q, \Psi)$, sections in segment $m(q, \Psi)$, and station $s_1(q, \Psi)$ can be used.
- *waitable*(q, t_1, t_2, Ψ) checks if train q can wait at station $s(q, \Psi)$ between time t_1 and t_2. Figure 12 describes the procedure. The procedure checks if there is a wait section available for train q to use between time t_1 and t_2.
- *reverse*(q, τ, Ψ) reverses or backtracks train q to its previous station and waits additional τ periods at previous station. Figure 13 describes the detailed procedure. The procedure first reverses train q to its previous station, denoted as s, calculates the time that q should wait in station s, and calls the procedure *delay* to delay train q at station s.
- *delay*(q, t, Ψ) makes train q either wait or reversed to the previous station so that it leave $s(q, \Psi)$ no earlier than time t. The detailed procedure is presented in Figure 14. The procedure first checks if train q can wait at station $s(q, \Psi)$: if so, train q is made to wait at station $s(q, \Psi)$ for time t; otherwise, the procedure reverse is called so that train q enters station $s(q, \Psi)$ when $s(q, \Psi)$ has available track.
- *select_active_train*(Ψ) selects an active train to serve as the candidate train to move forward in the dispatching stage. Figure 15 presents the detailed procedure. The procedure first finds trains, if any, in stations that are fully occupied and identifies the earlier such train; if no such train exists, the train in $Q(\Psi)$ with earliest

$t(q, \Psi)$ is chosen. As we will discussed later, this procedure can help reduce deadlocks.

`Procedure: move`(q, Ψ)
Input: train q and system state Ψ
Step 1: assign q to use control point $p_1(q, \Psi)$ at $t(q, \Psi)$;
Step 2: assign q to use control point $p_2(q, \Psi)$ at $t(q, \Psi) + \tau^q_{m(q,\Psi)}$;
Step 3: assign q to use section g time between $t(q, \Psi) + \sigma^q_g$ and $t(q, \Psi) + \lambda^q_g - 1$ for each $g \in G_{m(q,\Psi)}$
Step 4: assign q to use one track in $s_1(q, \Psi)$ during $t(q, \Psi) + \tau^q_{m(q,\Psi)}$ and $t(q, \Psi) + \tau^q_{m(q,\Psi)} + \tau^q_{s_1(q,\Psi)} - 1$;
Step 5: $t(q, \Psi) \leftarrow t(q, \Psi) + \tau^q_{m(q,\Psi)} + \tau^q_{s_1(q,\Psi)}$ and $s(q_2, \Psi) \leftarrow s_1(q_2, \Psi)$.

Figure 9. Procedure to move a train forward

`Procedure wait`(q, t, Ψ)
Input: train q, time to end waiting t, and system state Ψ
Step 1: assign q to use one track in $s(q_1, \Psi)$ between $t(q, \Psi)$ and $t - 1$;
Step 2: updating $t(q, \Psi) \leftarrow t(q, \Psi) + t$

Figure 10. Procedure to make a train wait

`Procedure waitable`(q, t_1, t_2, Ψ)
Input: train q, time to start waiting t_1, time to end waiting t_2, and system state Ψ
Output: <true> if q can wait in $s(q, \Psi)$ between t_1 and $t_2 - 1$, and <false> otherwise
Step 1: If there is available wait section in $s(q, \Psi)$ between time t_1 and $t_2 - 1$
Return <true>;
Else
Return <false>;

Figure 11. Procedure to check if a train can wait in current station

`Procedure movable`(q, t, Ψ)

Input: train q, time to leave $s(q, \Psi)$ t, and system state Ψ

Output: <true> if train q can leave $s(q, \Psi)$ and move to $s_1(q, \Psi)$, and <false>otherwise

Step 1: If $p_1(q, \Psi)$ is used or unavailable between $t-\theta+1$ and $t+\theta-1$

Return <false> and terminate the procedure

Step 2: If $p_2(q, \Psi)$ is used or unavailable between $t+\tau^q_{m(q,\Psi)}-\theta+1$ and $t+\tau^q_{m(q,\Psi)}-\theta+1$

Return <false> and terminate the procedure

Step 3: If there are trains traveling in $m(q, \Psi)$ in the opposite direction between t and $t+\tau^q_m-1$

Return <false> and terminate the procedure

Step 4: If there exists a $g \in G_{m(q,\Psi)}$ that is unavailable or is used by other trains between $t+\sigma^q_g$ to $t+\lambda^q_g-1$

Return <false> and terminate the procedure

Step 5: If there is no available track in $s_1(q, \Psi)$ between time $t+\tau^q_{m(q,\Psi)}$ and $t+\tau^q_{m(q,\Psi)}+\tau^q_{s_1(q,\Psi)}-1$

Return <false> and terminate the procedure

Step 6: Return <true>

Figure 12. Procedure to check if a train can move forward at a certain time

`Procedure reverse` (q, τ, Ψ)

Input: train q, additional time to wait in previous station τ, and system state Ψ

Step 1: If $s(q, \Psi) = o^q$ and o^q is not a terminal

Terminate and the procedure cannot find a solution;

Else

Get station s and segment m such that $a^q_s = m$ and $a^q_m = s(q, \Psi)$.

Step 2: $t(q, \Psi) \leftarrow t(q, \Psi) - w(q, s(q, \Psi), \Psi) - \tau^q_{s(q,\Psi)} - \tau^q_m$;

Step 3: Clear the track assignment to q in $s(q, \Psi)$, m, entry and exit control points to m;

Step 4: $s(q, \Psi) \leftarrow s$;

Step 5: delay(q, $t(q, \Psi) + \tau$, Ψ)

Figure 13. Procedure to backtrack the train to its previous station

```
Procedure delay(q, t, Ψ)
Input: train q, time to end delay t, and current state Ψ
Step 1: t1 = t(q, Ψ);
        While t1 ≤ t and not waitable(q, t1, t, Ψ)
            t1←t1+1;
Step 2: If t1 == t(q, Ψ)
            wait(q, t, Ψ);
        Else
            reverse(q, t1 − t(q, Ψ) + w(q, s(q,Ψ), Ψ), Ψ);
```

Figure 14. Procedure to delay a train

```
Procedure select_active_train(Ψ)
Input: system state Ψ
Output: the active train
Step 1: Q' = ∅
        For each station s that are fully occupied in state Ψ
            Q' ← Q'∪Q(s, Ψ);
        If |Q'|==0
            Go to Step 2;
        Else if
            Sort Q' increasingly according to t(q, Ψ);
            Return the first train in Q'.
Step 2: Sort Q(Ψ) increasingly according to t(q, Ψ);
            Return the first train in Q(Ψ).
```

Figure 15. Procedure to select the active train

3.5.2. Conflict resolution

The procedure *move*(q, Ψ) of the active train may be incompatible with other trains since they may need to use the same segment at the same time or use the same control point within θ periods. Such incompatibilities are called *conflicts*. The conflict is called a *meet* conflict if it is between two trains traveling in the same direction, and *pass* conflict otherwise. The detailed procedures to detect these conflicts are described

in Figure 16 and Figure 17. Both procedures check if two trains (traveling in the same direction for meet conflicts and traveling in opposite directions for pass conflicts) will appear at the same section or use the same control point at the same time; if so, there is a conflict. Figure 18 and Figure 19 describe the procedure to resolve the pass and meet conflicts respectively. As illustrated in Figure 18, faster trains are granted the right of way in any pass conflict so that the fast train does not need to wait for slow trains in all subsequent stations. On the other hand, when a meet conflict arises, as illustrated in Figure 19, the right of way is first given to a train that is currently at a fully occupied station, then given to the train that will cause a hard deadlock if the other train moves, and assigned according to the cost. Note that the value of the threshold probability ρ , a parameter in the procedure resolve_meet(q_1, q_2, ρ, Ψ), controls how often the active train is moved forward; specifically when ρ = 0, the procedure resembles a greedy scheme that looks myopically at the total cost in the next iteration; larger values of ρ indicate granting right of way to the active train more frequently. As discussed later, this parameter can enable creating different movement plans and searching for plans with lower cost.

`Procedure isMeet`(q_1, q_2)
Input: trains q_1 and q_2
Output: <true> if train q_1 and q_2 have a meet conflict, and <false> otherwise.
Step 1: If q_1 and q_2 are traveling in the same direction or $m(q_1, \Psi) \neq m(q_2, \Psi)$
Return <false>;
Else if $\{t: t(q_1, \Psi) \leq t \leq t(q_1, \Psi) + \tau_m^{q_1} - 1\}$
$\cap \{t: t(q_2, \Psi) \leq t \leq t(q_2, \Psi) + \tau_m^{q_2} - 1\} \neq \emptyset$
Return <true>;
Else if $| t(q_1, \Psi) - t(q_2, \Psi) - \tau_m^{q_2} | < \theta$ or $| t(q_1, \Psi) + \tau_m^{q_1} - t(q_2, \Psi) | < \theta$
Return <true>;
Else
Return <false>.

Figure 16. Procedure to check if two trains have a meet conflict

`Procedure isPass`(q_1, q_2)
Input: trains q_1 and q_2
Output:<true> if train q_1 and q_2 have a pass conflict, and <false> otherwise.
Step 1: If q_1 and q_2 are traveling in the opposite directions or $m(q_1, \Psi) \neq m(q_2, \Psi)$
Return <false>;
Else if $\{t: t(q_1, \Psi) + \sigma_g^{q_1} \leq t \leq t(q_1, \Psi) + \lambda_g^{q_1} - 1\} \cap \{t: t(q_2, \Psi) + \sigma_g^{q_2} \leq t \leq t(q_2, \Psi) + \lambda_g^{q_2} - 1\} \neq \emptyset$ for some section $g \in G_m$
Return <true>;
Else if $|t(q_1, \Psi) - t(q_2, \Psi)| < \theta$ or $|t(q_1, \Psi) + \tau_m^{q_1} - t(q_2, \Psi) - \tau_m^{q_2}| < \theta$
Return <true>;
ELSE
Return <false>.

Figure 17. Procedure to check if two trains have a pass conflict

`Procedure resolve_pass`(q_1, q_2, Ψ)
Input: Trains q_1 and q_2, system state Ψ
Step 1: Set $m = m(q_1, \Psi) = m(q_2, \Psi)$;
If $\tau_m^{q_1} > \tau_m^{q_2}$ or $\tau_m^{q_1} == \tau_m^{q_2}$ and $t(q_1, \Psi) > t(q_2, \Psi)$
$q = q_2$;
Else
$q = q_1$;
Step 2: move(q, Ψ);

Figure 18. Procedure to resolve a pass conflict

`Procedure resolve_meet`(q_1, q_2, ρ, Ψ)
Input: active train q_1, meeting train q_2, threshold probability ρ, and state Ψ
Step 1: If $s(q_1, \Psi)$ is fully occupied
move(q_1, Ψ); terminate the procedure;
Else if $s(q_2, \Psi)$ is fully occupied
move(q_2, Ψ); terminate the procedure;
Else
Go to Step 2;
Step 2: $t_1 = t(q_2, \Psi) + \tau_{m(q_2,\Psi)}^{q_2} + \theta - 1$;
$t_2 = t(q_1, \Psi) + \tau_{m(q_1,\Psi)}^{q_2} + \theta - 1$;
If waitable(q_1, $t(q_1, \Psi)$, t_1, Ψ) and not waitable(q_2, $t(q_2, \Psi)$, t_2, Ψ)
wait(q_1, t_1, Ψ); terminate the procedure;
Else if not waitable(q_1, $t(q_1, \Psi)$, t_1, Ψ) and waitable(q_2, $t(q_2, \Psi)$, t_2, Ψ)
wait(q_2, t_2, Ψ); terminate the procedure;
Else
go to step 3;
Step 3: //move trains that result in higher value in next iteration
Let Ψ_1 be the state after executing wait(q_1, t_1, Ψ) and move(q_1, Ψ);
Let Ψ_2 be the state after executing wait(q_1, t_2, Ψ) and move(q_2, Ψ);
Generate a random number r between 0 and 1;
If $c(\Psi_1) < c(\Psi_2)$ or $r < \rho$
move(q_1, Ψ); terminate the procedure;
Else
move(q_2, Ψ); terminate the procedure;

Figure 19. Procedure to resolve a meet conflict

3.5.3. Deadlock prevention and resolution

As in discrete-event heuristics for train dispatching, deadlocks may occur when we progressively route trains. A deadlock is a system state that no train is able to move forward without reversing at least one of them. Depending on the states of trains in the deadlock, deadlock can be either *hard* or *soft*.

In hard deadlocks, a train can neither move forward nor wait in current station. For example, if both movable(q, $t(q, \Psi)$, Ψ) and waitable(q, $t(q, \Psi)$, $t(q, \Psi) + 1$, Ψ) return false value, the deadlock arises since train q can neither move forward nor wait in $s(q, \Psi)$ at time $t(q, \Psi)$. In such a situation, we need to apply reverse(q, τ, Ψ) with $\tau \geq 1$ so that we will not get into state Ψ later. Although reversing trains can be resolved the hard deadlocks, it usually gives solutions with lower quality. To reduce applications of reverse(q, τ, Ψ), our heuristic adopt the following two strategies to prevent hard deadlocks in the pre-dispatch stage.

- Apply procedure movablize(q, Ψ) to delay train q accordingly if it is not movable in current state. Figure 21 presents the procedure movablize(q, Ψ). Specifically, the procedure checks if train q can move forward at time $t(q, \Psi)$. If not, procedure delay is called to make train q either wait at $s(q, \Psi)$ or reverse.
- Apply procedure look_ahead(q, Ψ) to delay train q appropriately if it is neither movable or waitable after every train q moves forward to its next station. Figure 22 presents the procedure look_ahead(q, Ψ). Specifically, the procedure checks whether train q, after moving to the next station $s_1(q, \Psi)$, can wait at $s_1(q, \Psi)$ or move forward to the next station of $s_1(q, \Psi)$. If not, procedure delay is called to make train q wait at $s(q, \Psi)$ or reverse.

In soft deadlocks, trains can wait, but not move forward. Figure 20 shows such a deadlock that commonly incurres. In Figure 20, trains q_1 and q_2 are traveling in

direction +, and trains q_3 and q_4 are traveling in direction −. No trains are able to move forward without violating the capacity constraint of stations s_1, s_2 or segment m. The only option left may be to reverse one of the trains. When combined with unavailability of track resources, the situation can get more complicated and identifying all possible variants could require considerable effort. For example, with the siding of station s_2 is under MOW, deadlock arises even when only trains q_1 and q_2 are in station s_1 and train q_3 is in station s_2 in Figure 20. As we know, a soft deadlock arises since the tracks inside stations are fully occupied. For example, stations s_1 and s_2 in Figure 20 are fully occupied. Thus, we can avoid all soft deadlocks by making sure that no stations are fully occupied. Specifically we apply the following soft deadlock prevention strategies:

- when selecting the active train in procedure select_active_train(Ψ), as presented in Figure 15, we give higher priority to trains in stations that are fully occupied, as illustrated in Figure 15;
- for an active train, we always apply resolve_pass(q_1, q_2, Ψ), as presented in Figure 18, before resolve_meet(q_1, q_2, Ψ), as presented in Figure 19, since a pass conflicts implies that the station is fully occupied;
- when resolving the conflict between two meeting trains in resolve_meet(q_1, q_2, Ψ), as presented in Figure 19, we always make the train in fully occupied stations move forward and the other train wait.

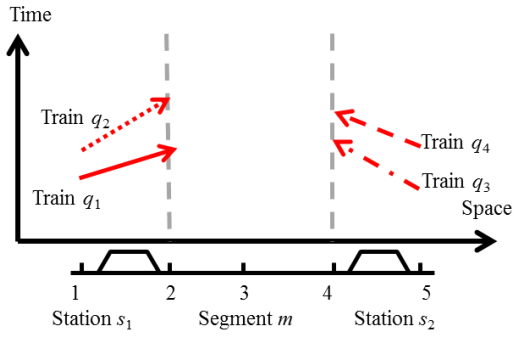


Figure 20. Example of a soft deadlock

```
Procedure movablize(q, Ψ)
Input: train q and system state Ψ
Output: return <true> if the system state is updated, and <false> otherwise
Step 1: Set t = t(q, Ψ);
        While not movable(q, t, Ψ)
            t←t + 1;
Step 2: If t == t(q, Ψ)
            Return <false>;
        Else
            delay(q, t, Ψ);
            Return <true>;
```

Figure 21. Procedure to movabilize trains

```
Procedure look_ahead(q, Ψ)
Input: train q and system state Ψ
Output: return <true> if the system state is updated, and <false> otherwise
Step 1: Assume the system state after perform move(q, Ψ) is Ψ̄
        Set t = t(q, Ψ̄);
        While not movable(q, t, Ψ̄)
            t←t + 1;
        If t == t(q, Ψ̄)
            Return <false>;
        Else
            go to step 2;
Step 2: t1= t(q, Ψ̄)
        While t1 < t and not waitable(q, t1, t, Ψ̄)
            t1←t1 + 1;
        IF t1 == t(q, Ψ̄)
            Return false;
        Else
            delay(q, t1− t(q, Ψ̄), Ψ)
            Return true;
```

Figure 22. Procedure to look head one station

3.5.4. Basic Procedure

Figure 23 shows how the dispatching problem can be solved by our sequential routing heuristic. In the initialization phase, the system state Ψ is updated to capture the track unavailability and to ensure that all trains are at stations. Each iteration starts with checking if the stopping criteria are met (Step 1); the procedure terminates when all trains are at their destinations or when maximum number of iterations is reached. Next, the pre-dispatch stage (Step 2) is repeated until the following two conditions are met: 1) each train is movable to next station and 2) each train can move or wait at its next station. In Step 3, we pick an active train and either move it forward or delay it, depending on the type of conflicts and possibly on the threshold probability ρ.

```
Procedure sequential_routing( n*, ρ)
Input:  Maximum number of iterations N
        Threshold probability ρ
Output:the system state Ψ with a meet and pass plan
Step 0: Let Ψ be the initial state;
        Update the track unavailability of Ψ;
        For each train q ∈ Q
            t(q, Ψ) = t^q, s(q, Ψ) and = o^q;
        n = 0;
Step 1: If n >n*
            Return null;
        Else if |Q(Ψ)|==0
            Return current solution;
Step 2  continue_update = true;
        FOR each train q ∈ Q(Ψ)
            While not ( movablize(q, Ψ) and look_ahead(q, Ψ) )
                continue_update = true;
        If continue_update
            Go to step 1;
        Else
            Go to step 2;
Step 3: q = select_active_train(Ψ);
        If there is a train q2 so that isPass(q, q2) is true
            resolve_pass(q, q2, Ψ);
        Else if there is a train q2 so that isMeet(q, q2) is true
            resolve_meet(q, q2, ρ, Ψ);
        Else
            move(q,  Ψ )
        Go to step 1;
```

Figure 23. Procedure general heuristic

3.5.5. Random search

If we select $0< \rho <1$ in sequential_routing(n^*, ρ), the procedure would end with different final state Ψ in different runs. This enables a random search procedure by running sequential_routing(n^*, ρ) multiple times and picking the final state with smallest

cost. As illustrated in Figure 24, the random search starts with a greedy sequential routing procedure with $\rho = 1$, followed by n random sequential routing runs with $0 < \rho < 1$. Such a random procedure can help generate a set of feasible solutions and picking the best one is usually better than the greedy heuristic.

`Procedure random_search`(N, n^*, ρ)
Input: number of replications N, maximum number of iterations n^* and random threshold replication ρ
Output:feasible solution Ψ or null if no feasible solution found
Step 1: Ψ = sequential_routing(n^*, 1)
Step 2: For $i = 1$ to $N - 1$
    Ψ_1 = sequential_routing(n^*, ρ);
    If $c(\Psi_1) > c(\Psi)$
        $\Psi = \Psi_1$;
    Return Ψ

Figure 24. Random search scheme

3.6. COMPUTATIONAL RESULTS

We tested the proposed models and solution techniques using data from two territories of a Class I railroad company in the US. Our models are implemented in JAVA, using CPLEX 12.4. The instances were tested on Dell Poweredge T610 running Ubuntu Linux, with 2 sixcore hyperthreading 3.33 GHz Xeon processors and 24 GB of shared memory. We terminate the procedure when the runtime reached 5 minutes or if the solver can find a solution with optimality gap of 2% or less.

We tested the data on Territory B and Territory M, whose lengths are 100 miles and 140 miles respectively. The planning horizon is 12 hours, time period length is 90 seconds, and control point headway is 7 minutes.

We use the base model to denote the model discussed in Section 3.4; the strengthened model incorporates the following improvements:

- replacing constraints (3.9) and (3.10) in base model with non-concurrency constraints (3.16), (3.19), and (3.20)
- refining the section definitions using the techniques discussed in Section 3.4.2,
- using the heuristic solution as the warm-start to the branch and cut
- applying the following valid inequalities as cuts: train-based non-concurrency inequalities (3.21) and (3.22), pairwise unidirectional inequalities (3.23), (3.26), and (3.31).

Table 5 shows the computational results of the base model and strengthened model. The strengthened model can get a near-optimal solution (within a 2% MIP gap) for 11 instances within 300 seconds, many more than the 3 instances in the base model. Besides, the strengthened model performs better than the base model in all cases; the strengthened model either reduces the solution time or gets lower MIP Gap value. This can be attributed to good feasible solutions obtained by the heuristic procedures and better lower bounds achieved by our effective user cuts.

To compare the effectiveness of our valid inequalities, we define the percentage of gap closed as $(LB-LI)/(LB-LS)$, where *LI* is the upper bound obtained after solving the root node, *LB* is the upper bound obtained after solving the root node, and *LS* is the objective function value of the best solution found. Table 6 summarizes the percentage of gap closed and compares the number of valid inequalities by CPLEX and our separation procedures. We can see that our separation procedure found many more cuts than CPLEX and closes the gap effectively.

Table 5. Comparing Base Model and Improved Model for train dispatching

Instance	# of Trains	Base Model		Improved Model	
		Time(secs)	MIP Gap	Time(secs)	MIP Gap
B1347	13	26	1.89%	8	1.09%
B1013	16	164	1.91%	8	1.88%
B0908	16	600	4.65%	9	1.82%
B0942	17	600	4.79%	15	1.95%
B1107	18	600	4.10%	15	1.92%
M1955	13	600	2.71%	24	1.98%
M1443	16	16	1.53%	25	1.62%
B1037	15	48	1.76%	32	1.99%
M1356	13	600	4.17%	52	1.93%
M0957	19	600	6.88%	96	1.99%
B0954	21	600	4.11%	107	0.88%
B1243	16	600	7.89%	115	1.81%
B1311	22	600	6.24%	600	2.35%
B0914	19	600	20.60%	600	2.58%
M1728	18	600	11.54%	600	4.18%
B1358	30	600	30.81%	600	4.22%
B1020	27	600	28.76%	600	4.80%
B0806	18	600	12.89%	600	4.94%
B1155	25	600	10.37%	600	5.77%
M1225	21	600	28.87%	600	8.79%
M1203	24	600	41.07%	600	12.64%

Table 6. Impact of valid inequalities

	Base Model	Improved Model		Percent of Gap
Instance	# of CPX Cut	# of CPX Cut	# of User Cut	Closed
B1347	100	94	282	54.55%
B1013	144	90	409	51.92%
B0908	172	73	835	52.81%
B0942	186	109	962	58.28%
B1107	260	121	969	63.64%
M1955	174	66	955	41.78%
M1443	37	189	1197	35.47%
B1037	97	60	1349	30.05%
M1356	76	69	1153	74.93%
M0957	298	90	1993	58.58%
B0954	256	100	4965	74.54%
B1243	179	111	1566	63.92%
B1311	251	162	11055	85.85%
B0914	205	182	12057	60.79%
M1728	178	66	3074	52.73%
B1358	314	239	8326	70.44%
B1020	273	157	2867	50.71%
B0806	187	168	16903	42.95%
B1155	275	146	15816	49.69%
M1225	439	194	4599	32.85%
M1203	338	259	9195	17.73%

3.7. CONCLUSIONS

To help train dispatching in single-track territories, we proposed an integer programming model for the train dispatching problem that takes into account various operational considerations including trailing of trains, minimal headway between trains, track unavailability and train priorities. We developed a section-based non-concurrency constraint that can prevent the meeting and overtaking of trains inside a segment, as well as the crossing of trains at the section bound point. We also proposed a train-based non-concurrency inequality that can strengthen the model. Besides strengthening the

pairwise unidirectional inequalities in Cacchiani et al.(2010), we extended the inequality to more than one segment and generalized it to incorporate more trains.

Our study is the first to explore and improve modeling and algorithmic strategies to solve real-life train dispatching problems to optimality or near-optimality by using discrete time formulation. Both Caprara et al. (2002) and Şahin et al. (2008) solved their problems by heuristics, and Harrod (2011) only applies his model to a small territory with simplified territory data. Our computational results show that our solution method can obtain an optimal or near-optimal solution to many real-world problem instances within a reasonable amount of time.

Chapter 4. Route Design for Delivery Vehicles with Backhauling

4.1. INTRODUCTION

Major grocery chains require daily deliveries from one or more warehouses to restock their inventory and replenish perishable items. On the return trip, it is common for a subset of the vehicles to pick up salvage items (or "returns") from the stores and bring them back to the warehouse. The resulting problem, which we call the retail route design problem (RRDP) with backhauls extends the capacitated vehicle routing problem with time windows (VRPTW) by incorporating context specific considerations. As is in the traditional VRPTW, a vehicle corresponds to a truck-trailer combination that is assigned to a route for delivery of replenishment orders and for pickup of salvage orders. In our context, it is assumed that there are a sufficient number of drivers, trucks and trailers available on any given day to meet the demand within the given time windows. The costs associated with a vehicle route consist of two components: (1) the travel cost incurred by each route on a per mile basis, and (2) the driver idle time cost incurred when arriving early at store and having to wait for the start of the delivery and pick up windows.

In addition to the requirements in the traditional VRPTW, the RRDP investigated here has a number of practical and unique constraints. First, there are three types of delivery orders and one type of pickup order. The delivery orders are required to be loaded onto the truck in certain sequences, and the truck can only start pick up orders after its deliveries are made. Second, the truck has pre-specified weight and volume limits. Separation curtains must be placed between different delivery order types and between different stores. The weight of the curtains can be ignored but not the volume, which reduces the capacity of a truck on that dimension. Third, the loading capacity at the warehouse must be taken into account on a 30-minute basis, so only a limited number

of vehicles can be loaded in each time slot. Lastly, the total time of a route and the actually time a driver is behind the wheel must adhere to certain legal restrictions and union regulations.

To model the RRDP, we first construct a route diagram in the form of a directed network in which each order is represented as a node and where the arcs capture possible transitions between the nodes. Based on this diagram, we develop a mixed-integer program (MIP) for the problem. However, because it was not possible to obtain solutions for realistic instances with a commercial code, we developed a greedy randomized adaptive search procedure (GRASP) following the work Kontoravdis and Bard (1995) and Solomon (1988). In the construction phase, GRASP exploits the unsurprising observation that orders for the same store are usually served by the same route in high-quality solutions. In the improvement phase, we initially relied on tabu search to find local optima; however, we discovered that every solution is likely to have many degenerate neighbors, i.e., neighbors with the same cost as the current solution. This limits the extent to which the feasible region can be locally explored. To overcome this difficulty, we implemented a randomized variable neighborhood search as well as several augmented versions of tabu search.

Extensive testing was done to determine the best combination of procedures. The results showed that GRASP with tabu search in phase II edged out pure tabu search with random variable neighborhood search when both procedures were run for 30 minutes. In a second set of tests, we compared the GRASP solutions with those provided by Kroger, the sponsoring company, and found that cost reductions averaging $2737 or almost 3% per day can be obtained with our methods.

With this in mind, the contributions of the paper are fourfold: (1) we study a new version of a pickup and delivery VRPTW in which vehicle capacity is order dependent;

(2) we develop several solution methodologies that integrate a number a metaheuristic ideas; (3) we provide extensive comparisons of alternative implementation approaches; and (4), we compare the solutions obtained with our best algorithm with those actually used and with those obtained with an experimental set-partitioning code.

The rest of the chapter is organized as follows. Section 4.2 presents the literature review and Section 4.3 gives a formal description of the RDDP. This is followed by our MIP formulation in Section 4.4 and the development of alternative solution methodologies in Section 4.5. In phase I of the GRASP, feasible solutions are constructed by sequentially inserting orders into existing routes, and in phase II we propose a variety of local improvement methods. Test results are included in Section 4.6 using seven days of data provided by Kroger, one of the largest grocery chains in the U.S. We close in Section 4.7 with some insights and suggestions for future research.

4.2. RELATED LITERATURE

The past two decades have witnessed an outsized interest in the VRPTW and its variants. Both branch and cut (e.g., see Bard et al. 2002, Kohl et al. 1999, Lysgaard 2004) and branch and price (e.g., see Bard et al. 2014, Desaulniers et al. 2008, Azi 2010) have been applied successfully to find exact solutions to instances with a 100 or more nodes. As a variation, Prescott-Gagnon et al. (2009) developed a large neighborhood search algorithm that takes advantage of the power of branch-and-price. For more information about the exact solution method, see the surveys by Baldacci et al. (2012) and Kallehauge (2008).

Various heuristics have also been proposed for the VRPTW. Bräysy and Gendreau (2005a, 2005b) present an extensive survey of related research that covers route construction algorithms, local search algorithms, and metaheuristics. Solomon

(1987) discusses and compares several solution-construction approaches including saving heuristics, a nearest-neighbor heuristic, and insertion heuristics. He found that an insertion-type heuristic consistently gave good results. Various local search methods have also been developed to improve the solution. Rochat and Semet (1994) present an insertion procedure to construct an initial solution followed by tabu search to improve the incumbent. Taillard (1995) developed two partition methods to speed up the tabu search for VRPs. Other variants of local search applied to these problems include granular tabu search (Toth and Vigo 2003), variable neighborhood search (Kytöjoki et al. 2007), and large neighborhood search (Ergun et al. 2006). Also see Kontoravdis and Bard (1995) for a GRASP to solve the VRPTW and Nagata et al. (2010) for a memetic algorithm.

An expansion of reverse logistics activities has led to a renewed interest in the study of the VRP with pickup and deliveries (PDP), that is, a VRP with demand for two types of services. Berbeglia et al. (2007) and Parragh et al. (2008a and 2008b) present extensive surveys. Depending on the problem context, some studies only allow vehicles to perform pickups after all the deliveries are made (e.g., see Thangiah et al. 1996), while others allow simultaneous pickups and deliveries (e.g., Bianchessi and Righini 2007 and Tasan and Gen 2012). Various heuristics have been proposed to solve the PDP. Bent and Hentenryck (2006) present a two-stage hybrid approach in which a single simulated annealing algorithm is used in the first stage to decrease the number of routes, while large neighborhood search is used in the second stage to decrease total travel cost. Bianchessi and Righini (2007) present and compare construction algorithms, local search algorithms, and tabu search. Their computational results give experimental evidence that local search with complex and variable neighborhoods yields good solutions that are very robust. Recently, Tasan and Gen (2012) developed a genetic algorithm and Goksal et al. (2013) present a heuristic based on particle swarm optimization.

Researchers have also investigated more specialized models designed to accommodate practical restrictions. Using adaptive large neighborhood search, Pisinger and Ropke (2007) solved several variants including VRPs with multiple depots. Lau et al. (2003) provided a computable upper bound on the total number of customers that can be served by a given fleet size and designed a tabu search algorithm to solve the VRPTW with a minimum travel time objective. Penna et al. (2013) considered a VRP in which clients are served by a heterogeneous fleet with distinct capacities and costs. Vidal et al. (2013) proposed a hybrid genetic algorithm for multi-depot and periodic VRPs. For more specialized applications, see Golden et al. (2008). Nowak et al. (2008) demonstrated the benefit of using split loads for the PDP, while Nagy et al. (2013) was concerned with the level of savings that can be achieved by allowing the pickups and deliveries to be served separately as opposed to simultaneously. The VRPTW that we investigate requires all deliveries be made before any pickups, although we do allow split deliveries (cf. Mitra 2008).

In a recent study, Qu and Bard (2013) addressed a VRP in which the interior of the vehicles could be reconfigured to accommodate different structural loads. Their constraints were similar to but less complex than ours. Solutions were found with an adaptive large neighborhood search heuristic for an application in which members of a senior activity center had to be transported to and from the center as well as to secondary facilities for rehabilitative and medical treatment. The number of persons and support equipment that a van could carry was a function of how it was configured.

4.3. PROBLEM DESCRIPTION

The RRDP, as an extension of the traditional VRPTW, requires the scheduling a set of configurable vehicles to deliver orders from a warehouse to geographically

dispersed locations, and then to pick up items at a subset of those locations on the return trip to the warehouse. Each location or store $s \in S$ has a time window $[a_s, b_s]$ during which either loading or unloading can begin. If a driver arrives prior to a_s, he has to wait, thus incurring unit idle time cost of σ. For each day in the planning horizon, the objective is to minimize the total cost of a schedule, which is a function of the total time of each route in the schedule.

Besides the constraints common to the generic VRPTW, the RRDP under consideration must also satisfy the following sets of constraints, which capture the operational requirements associated with retail inventory replenishment.

(1) Loading capacity at the warehouse. Orders must be pulled from the warehouse and delivered to the dock by the material handling equipment. These operations take roughly the same amount of time for each trailer. Since the material handling equipment has limited capacity, only a certain number of trailers can be loaded simultaneously. Accordingly, we divide the available operating time at the warehouse into a set of time periods 30 minutes in length, and restrict the number of routes that can be assigned to each period to some maximum, n^{load} (= 20).

(2) Loading and unloading time at the stores. There are two components to consider: a fixed setup time and order-specific variable time. The fixed setup time is incurred only when a truck starts to deliver orders for a store. The variable time of each order is proportional to its volume.

(3) Time limits. Each route must adhere to an upper limit of 10 hours on driving time (which does not include waiting time a store prior to the beginning of the time window or waiting at the warehouse) and total time that a truck is on the road. The Federal Transportation Administration allows up to 14 hours on the road.

(4) Delivery order sequence restrictions. There are three categories of delivery orders, namely, frozen, refrigerated and grocery, and one category for pickups called the salvage order. The sequence in which the delivery orders can be loaded onto a trailer, starting from the front, must be frozen, refrigerated, and grocery. Salvage pickups can only begin after all deliveries are made and the truck is empty. Each order has a weight and volume, but the corresponding values for salvage orders are small in comparison to the size of the truck and therefore are ignored.

(5) Volume reduction. For a trailer carrying grocery orders together with refrigerated or frozen orders, a curtain is necessary between them. Curtains are also placed between orders for different stores. Each curtain reduces the total available volume of a trailer in a nonlinear way.

(6) Capacity. The total volume and weight of the orders on a trailer must not exceed their respective limits.

(7) Maximum number of stores per route. To reduce the excessive traveling between stores, there is a limit on number of delivery stores a truck can visit and a limit on the number of pickup stores a truck can visit.

In practice, some of these constraints are often treated as soft by the scheduling office. For example, we found that the mixed use of two sets of time windows are common at Kroger. "Short" time windows are standard but in some cases, they are extended by moving up a_i to create "long" time windows. Introducing more flexibility may significantly reduce the cost of a schedule. We examined this case as well as an intermediate case in which only a certain portion, γ (= 20%), of the routes are permitted to use the long time windows.

4.4. MATHEMATICAL FORMULATION

The RRDP can be modeled on a directed network with nodes representing the different categories of items for which demand exists at each store, and the salvage orders that are to be picked up on the return to the warehouse. In Section 4.4.1, we define a route diagram $G = (V, A)$ that captures precedence requirements among orders. This is followed in Section 4.4.2 by the presentation of the mixed-integer programming model for the problem.

4.4.1. Route diagram

The sequence in which orders are loaded onto the truck determines the sequence in which deliveries are made. The requirement is that all orders of a particular type be unload before orders of a different type are unloaded. The sequence is groceries followed by refrigerated orders and then with frozen orders. To capture this precedence relationship, we now define the route diagram $G = (V, A)$.

Assume that S is the set of all stores. To facilitate the presentation, we assume that all stores have a grocery order, a refrigerated order, a frozen order, and a pickup order. In the most general case, then, the network has four nodes for each store $s \in S$, denoted by s-G, s-R, s-F, and s-P, corresponding to its grocery, refrigerated, frozen, and pickup orders respectively. Assume that this gives a total of N nodes. The network also contains a starting node, denoted by node 0, to represent the warehouse where the vehicles begin their route and an ending node, N+1, also representing the warehouse but where the vehicles end their route.

The possible sequences of the deliveries are enforced by the arcs in the network. For each store $s \in S$, we create one arc from node 0 to each of its order nodes s-G, s-R, s-F and s-P; similarly, we create one arc from each of its order node to node N+1. For each store $s \in S$, Table 7 identifies all possible arcs emanating from its order nodes to

other nodes in the network. From node *s*-G, arcs connect to grocery nodes at other stores and to refrigerated, frozen and pickup nodes at *s* and other stores, i.e., refrigerated, frozen, and pickup nodes; from node *s*-R, arcs connect to refrigerated order nodes at other stores and to frozen or pickup nodes at *s* and at other stores; from node *s*-F, arcs connect to frozen order nodes at other stores and to all pickup nodes; from node *s*-P, arcs only extend to pickup nodes at other stores.

Table 7. Arcs emanating from order nodes at store *s*

Node	Arcs
s-G	(*s*-G, *r*-G) for $r \in S \setminus \{s\}$ (*s*-G, *r*-R), (*s*-G, *r*-F), and (*s*-G, *r*-P) for $r \in S$
s-R	(*s*-R, *r*-R) for $r \in S \setminus \{s\}$ (*s*-R, *r*-F), and (*s*-R, *r*-P) for $r \in S$
s-F	(*s*-F, *r*-F) for $r \in S \setminus \{s\}$ (*s*-F, *r*-P) for $r \in S$
s-P	(*s*-P, *r*-P) for $r \in S \setminus \{s\}$

Figure 25 depicts an example of possible transitions in the route graph. Panel (a) identifies the nodes associated with a store along with the connecting arcs. The network in panel (b) contains the arcs listed in Table 7, as well as the arcs (dashed lines) between node 0 and order nodes and arcs (dotted lines) between order nodes and node $N + 1$. To avoid clutter, the arcs between pairs of nodes at the same store are not shown. Each path from node 0 and $N + 1$ in the network corresponds to a vehicle route. For example, path (0, *s*-G, *r*-G, *r*-F, *r*-P, *s*-P, $N + 1$), as shown in Figure 26, corresponds to a vehicle that visits the stores in the following sequence: (1) leave the warehouse, (2) visit store *s* to deliver grocery order, (3) visits store *r* to deliver grocery order, (4) remains at store *r* to deliver frozen order, (5) pick up salvage order at store *r*, (6) pick up salvage order at store *s*, and (7) return to the warehouse. Figure 26 illustrates the path in more detail.

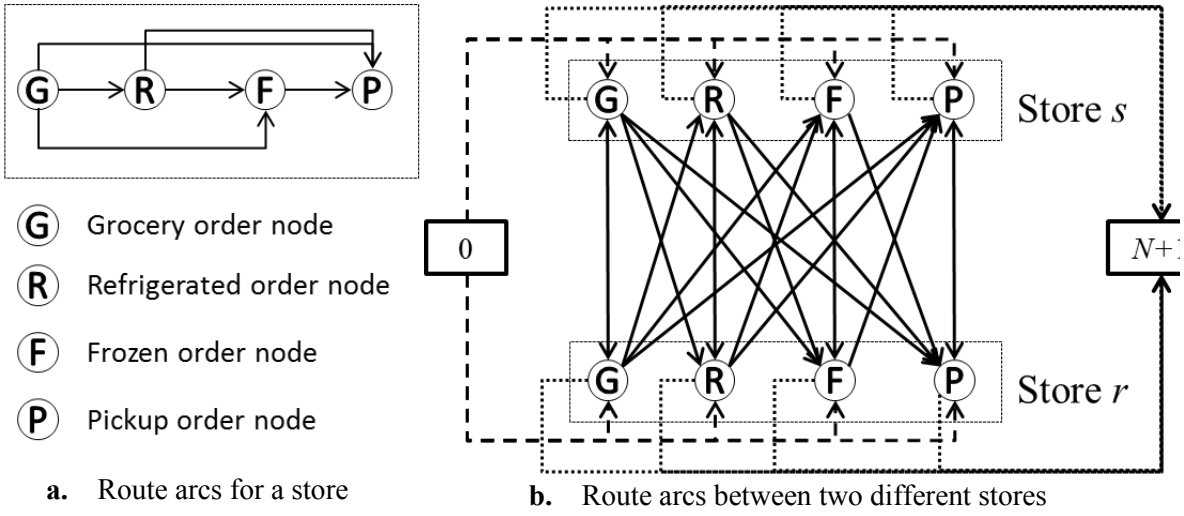


Figure 25. Example route network for two stores

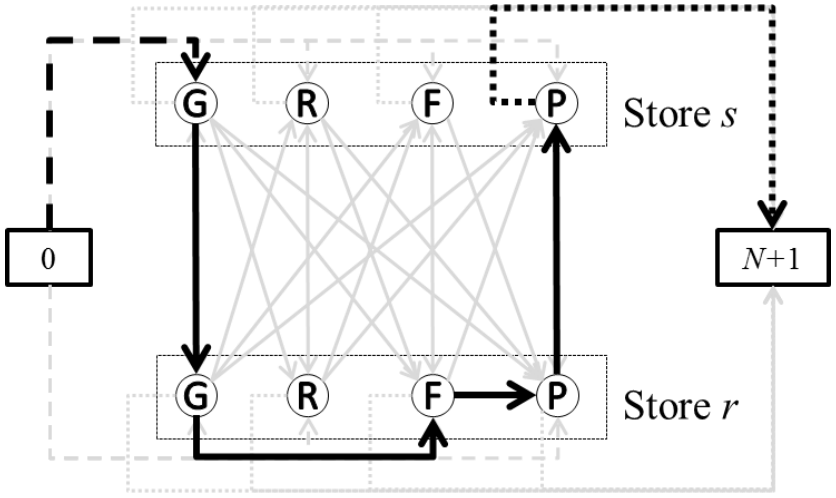


Figure 26. Example path (0, *s*-G, *r*-G, *r*-F, *r*-P, *s*-P, *N* + 1)

4.4.2. Mathematical formulation

In this section, we provide a MIP formulation of the RRDP using the notation listed below.

Indices and sets

i, j	index for nodes in the route network (source, order, and sink)
s	index for stores
k	index for vehicles
p, q	index for vehicle depart time periods from the warehouse
$c(i)$	index for the order type associated with node i
C	set of order types; C = {1: frozen, 2: refrigerated, 3: grocery, 0 pickup}
DO	set of deliver order nodes
K	set of vehicles
O_s	set of order nodes for store s
O	set of order nodes; $O = DO \cup PO$
$O^+(i)$	set of order nodes that can immediately follow node i on a route; for any node $j \in O^+(i)$, we must have $a_i + \hat{\tau}_{ij} \le b_j$ to meet the time window constraint (see below for symbol definitions)
$O^-(i)$	set of order nodes that can immediate precede node i on a route; for any node $j \in O^-(i)$, we must have $a_j + \hat{\tau}_{ji} \le b_i$ to meet the time window constraints
P	set of departure time periods from the warehouse for a vehicle
PO	set of pickup order nodes
S	set of stores

Parameters

a_j	start time of time window for order at node i
b_j	end time of time window for order at node i
h	loading or unloading rate (ft^3 per hour) at the stores
ψ	setup time to start loading or unloading at a store
v_i	volume (ft^3) of the order associated with node i

w_i weight (lb) of the order associated with node i

Δ_{ij} 1 if nodes i and j correspond to different locations, 0 otherwise.

Γ_{ij} 1 if nodes i and j are such that $c(i) = 3$ and $c(j) \in \{2, 1\}$, and 0 otherwise

Φ_{ij} 1 if nodes i and j are such that $c(i) = 2$ and $c(j) = 1$, and 0 otherwise

M_{ij} sufficiently large number associated with arc (i, j)

τ_{ij} travel time between nodes i and j

$\hat{\tau}_{ij}$ total time incurred in traversing arc (i, j), accounting for service time at the location of node i (if any), travel time between nodes i and j, and setup time at the location of node j (if any); specifically, $\hat{\tau}_{ij} = \tau_{ij} + \psi$ if $i = 0$, $\hat{\tau}_{ij} = \rho v_i + \tau_{ij}$ if $j = N+1$, and $\hat{\tau}_{ij} = h v_i + \tau_{ij} + \Delta_{ij}\psi$ otherwise.

n^{load} maximum number of trailers that can be loaded simultaneously ($n^{\text{load}} = 20$)

end_p end time for departure period $p \in P$

$start_p$ start time for departure period $p \in P$

$tlimit_1$ limit on driving time per route (10 hours)

$tlimit_2$ limit on total time per route (14 hours)

$vcap$ volume capacity of a trailer (2000 ft^3)

$wcap$ weight capacity of a trailer (42000 lb)

$dcap$ maximum number of delivery stores allowed per route

$pcap$ maximum number of pickup stores allowed per route

$vred_0$ reduction in volume capacity due to carrying refrigerated commodities in addition to frozen commodities on the same trailer (ft^3)

$vred_1$ reduction in volume capacity due to carrying groceries in addition to frozen and/or refrigerated commodities on the same trailer (ft^3)

$vred_2$ reduction in volume capacity of a trailer for each additional store visited beyond the first store (ft^3)

$\hat{\theta}_{ij}$ travel cost between nodes i and j; note that the cost is 0 if order nodes i and j correspond to the same store

θ_{ij} cost for traversing arc (i, j), consisting of three components: (i) servicing cost at node i (if any), (ii) travel cost going from node i to node j, and (iii) the setup cost at node j

σ unit cost for driver idle time at a store incurred prior to the start of loading/unloading an order (dollars per hour); no cost is incurred at the warehouse if there is idle time before departure

Decision variables

e^k time at which route k returns to the warehouse

s^k time at which route k departs the warehouse

t_i time at which service starts (either unloading or loading) at node i

x_{ij}^k 1 if route k visits node i and j in succession, 0 otherwise

y_p^k 1 if route k is assigned to departure time period p at the warehouse, 0 otherwise

z_i idle time incurred prior to fulfilling the order associated with node i

Model RRDP

$$\text{Minimize } \sum_{k\in K}\left(\sum_{j\in O}\theta_{0j}x_{0j}^k+\sum_{i\in O}\sum_{j\in O^+(i)}\theta_{ij}x_{ij}^k+\sum_{i\in O}\theta_{i,N+1}x_{i,N+1}^k\right)+\sigma\sum_{i\in O}z_i \tag{4.1}$$

subject to

Demand and flow balance constraints

$$\sum_{k\in K}\sum_{j\in O^+(i)}x_{ij}^k+\sum_{k\in K}x_{i,N+1}^k=1 \qquad \forall i\in O \tag{4.2}$$

$$\sum_{j\in O^+(i)}x_{ij}^k+x_{i,N+1}^k-\sum_{j\in O^-(i)}x_{ji}^k-x_{0i}^k=0 \qquad \forall k\in K, i\in O \tag{4.3}$$

Time window constraints

$$t_i + \hat{\tau}_{ij} - M_{ij}(1 - x_{ij}^k) \le t_j \qquad \forall k \in K, i \in O, j \in O^+(i) \qquad (4.4)$$

$$s^k + \hat{\tau}_{0j} - M_{0j}(1 - x_{0j}^k) \le t_j \qquad \forall k \in K, j \in O^+(0) \qquad (4.5)$$

$$t_i + \hat{\tau}_{i,N+1} - M_{i,N+1}(1 - x_{i,N+1}^k) \le e^k \qquad \forall k \in K, i \in O^-(N+1) \qquad (4.6)$$

$$a_i \le t_i \le b_i + \sum_{k \in K} \sum_{j \in O^-(i)} M_{ji}(1 - \Delta_{ji}) x_{ji}^k \qquad \forall i \in O \qquad (4.7)$$

Warehouse handling capacity constraints

$$\sum_{k \in K} \sum_{q=1}^{p} y_q^k \le p \bullet n^{\text{load}} \qquad \forall p \in P \qquad (4.8)$$

$$\sum_{p \in P} start_p y_p^k \le s^k \le \sum_{p \in P} end_p y_p^k \qquad \forall k \in K, p \in P \qquad (4.9)$$

$$\sum_{p \in P} y_p^k \le 1 \qquad \forall k \in K \qquad (4.10)$$

$$\sum_{p \in P} y_p^k \ge \sum_{i \in O} x_{0i}^k \qquad \forall k \in K \qquad (4.11)$$

Trailer capacity

$$\sum_{i \in DO} w_i \left(\sum_{j \in O^+(i)} x_{ij}^k + x_{i,N+1}^k \right) \le wcap \qquad k \in K \qquad (4.12)$$

$$\sum_{i \in PO} w_i \left(\sum_{j \in O^+(i)} x_{ij}^k + x_{i,N+1}^k \right) \le wcap \qquad k \in K \qquad (4.13)$$

$$\sum_{i \in DO} \left(\sum_{j \in O^+(i) \cup \{N+1\}} v_i x_{ij}^k + \sum_{j \in O^+(i) \cap DO} \left(vred_0 \Phi_{ij} + vred_1 \Gamma_{ij} + vred_2 \Delta_{ij} \right) x_{ij}^k \right) \le vcap \quad k \in K \qquad (4.14)$$

$$\sum_{i \in PI} v_i \left(\sum_{j \in O^+(i)} x_{ij}^k + x_{i,N+1}^k \right) \le vcap \qquad \forall k \in K \qquad (4.15)$$

Route time limits

$$\sum_{j \in O} \tau_{0j} x_{0j}^k + \sum_{i \in O} \sum_{j \in O^+(i)} \tau_{ij} x_{ij}^k + \sum_{i \in O} \tau_{i,N+1} x_{i,N+1}^k \le tlimit_1 \qquad \forall k \in K \qquad (4.16)$$

$$e^k - s^k \le tlimit_2 \qquad \forall k \in K \qquad (4.17)$$

Idle time constraints

$$t_i + \hat{\tau}_{ij} + M_{ij}(1 - x_{ij}^k) + z_j \geq t_j \qquad \forall k \in K, i \in O \setminus \{0\}, j \in O^+(i) \tag{4.18}$$

$$s^k + \hat{\tau}_{0j} + M_{0j}(1 - x_{0j}^k) + z_j \geq t_j \qquad \forall k \in K, j \in O^+(0) \tag{4.19}$$

Limit on number of stores

$$\sum_{i \in DO} \sum_{j \in O^+(i) \cap DO} \Delta_{ij} x_{ij}^k \leq dcap - 1 \quad k \in K \tag{4.20}$$

$$\sum_{i \in PO} \sum_{j \in O^+(i) \cap PO} \Delta_{ij} x_{ij}^k \leq pcap - 1 \quad k \in K \tag{4.21}$$

Variable definitions

$$x_{ij}^k \in \{0,1\} \qquad \forall k \in K, i \in O, j \in O^+(i)$$

$$x_{0i}^k,\ x_{i,N+1}^k \in \{0,1\} \qquad \forall k \in K,\ i \in O$$

$$y_p^k \in \{0,1\} \qquad \forall k \in K, p \in P$$

$$s^k, e^k \geq 0 \qquad \forall k \in K$$

$$t_i, z_i \geq 0 \qquad \forall i \in O \tag{4.22}$$

The objective function (4.1) has two terms. The first minimizes the cost of traversing the arcs between each pair of successive nodes in a route. The second penalizes the idle time incurred prior to fulfillment of orders, and indirectly reduces the total time drivers must wait at a node before service can begin.

Constraints (4.2) require that each order node *i* has exactly one successor, while constraints (4.3) enforce flow balance for each route *k* in a solution. That is, for route *k*, order *i* is either succeeded by some other order node *j* or by node *N*+1, and analogously, is either preceded by some order node *j* or by node 0.

The next set of constraints enforces the time windows. Constraints (4.4) ensure that if order node *j* is the immediate successor of order node *i* on route *k* (i.e., $x_{ij}^k = 1$), then service cannot begin for node *j* before the vehicle travels from node *i* to node *j*. The definition of $\hat{\tau}_{ij}$ ensures that $t_j - t_i$ is large enough to (1) unload the order corresponding

to node *i*, (2) transit from *i* to *j*, and (3) unload or unload items at node *j*. When $x_{ij}^k = 0$, implying that *j* is not an immediate successor of *i*, constraints (4.4) becomes redundant for sufficiently large parameter M_{ij}. Here, $M_{ij} = \max\{b_i + \hat{\tau}_{ij} - a_j, 0\}$. Constraints (4.5) and (4.6) are identical to (4.4) but explicitly enforce the first and last orders on route *k*, respectively.

Constraints (4.7) make the time windows requirements redundant for second and third orders on a particular vehicle at the same location *i*. We only want the time window constraints to be active for the first delivery to a store for each vehicle. Consider the situation where $x_{ij}^k = 1$. If $\Delta_{ij} = 0$, i.e., *i* and *j* correspond to the same store, then (4.7) is redundant; otherwise, it is equivalent to $a_i \le t_i \le b_i$. As we know, if $\Delta_{ij} = 1$, then M_{ij} can be any arbitrarily large value. To see the values of M_{ij}, let's see a store with grocery, refrigerated, frozen, and salvage orders, denoted as orders 1, 2, 3, and 4 respectively. Thus, we must have $M_{ji} = \sum_{l=1}^{j-1} \tau_{l,l+1} + \tau_{ij}$. For example, we have $M_{34} = \tau_{12} + \tau_{23} + \tau_{34}$. If a truck *k* pickup order 4 after serving order 3 (frozen order), i.e., $x_{34}^k = 1$, the time to start pickup order 4 can be as late as $b_4 + \tau_{12} + \tau_{23} + \tau_{34}$, which happens when truck *k* delivers orders 1, 2, 3, and picks up 4.

Constraints (4.8)-(4.11) are used to spread out route start times so as not to exceed the retrieval and loading capacity of the material handling equipment at the warehouse. In particular, constraints (4.8) assign the vehicles to the various possible departure periods such that the cumulative number of vehicle departures during the first period through any subsequent period does not exceed the warehouse loading capacity. Written in this fashion, the constraints allow for trailers to be loaded early and then depart at a later time, thus making more efficient use of the material handling capacity. Constraints (4.9) require each route to start within the departure period it is assigned to, while constraints (4.10) require that each route be assigned to no more than one

departure period. Finally, constraints (4.11) ensure that if a route is started, it is assigned to one of the departure periods.

Constraints (4.12)-(4.15) enforce the weight and volume capacities of each trailer. Constraints (4.12) require the total weight of the delivery orders to be within the weight limit for the trailer, while constraints (4.13) do the same for the pickup salvage orders. Constraints (4.14) enforce the volume limit for the delivery orders, while capturing the reduction in trailer capacity due to the need for separation curtains. The second term on the left-hand side is associated with the volume capacity reduction resulting from having groceries in addition to refrigerated and/or frozen items on the same trailer, having refrigerated items in addition to frozen items on the same trailer, and serving two or more stores with the same trailer. Note that for each route $k \in K.$, the sequencing requirement of orders in the trailers guarantee that $\sum_{i \in DO} \sum_{j \in O^+(i) \cup DO} \Delta_{ij} x_{ij}^k \leq 1$. Thus, the volume reduction $vred_0$ and $vred_1$ are each counted at most once in route k. Constraints (4.15) enforce the overall volume limitation for the pickup orders.

Constraints (4.16) and (4.17) impose the driving time and total time limits for each route, respectively. Constraints (4.18) and (4.19) account for driver idle time prior to fulfilling the first order associated with node j when node i is a store and the warehouse, respectively. When $x_{ij}^k = 1$, constraints (4.18) become $t_i + \hat{\tau}_{ij} + z_j \geq a_j$, where z_j represents the idle time before service at node j can begin in a cost-minimizing solution. When $x_{ij}^k = 0$, constraints (4.18) are redundant for sufficiently large M_{ij}. Here, $M_{ij} = \max\{a_j - \hat{\tau}_{ij} + b_i, 0\}$.

Constraints (4.20) and (4.21) enforce the limit on the number of stores that a vehicle can visit for deliveries and pickups, respectively. Constraints (4.20) ensure that the number of transition arcs between delivery order nodes and between different stores

does not exceed *dcap* − 1, thus enforcing the upper bound *dcap*. In the same way, constraints (4.21) limit the number of pickup stores on a route to *pcap*. Finally, variable definitions are given in (4.22).

Note that in model *RRDP*, we assume that the reduction in volume due to the separation of order types for a given store as well as between stores is independent. In our application, volume reduction is a function of the number of order types and the number of stores visited (e.g., see Table 10 in Section 4.6). Modeling this aspect of the problem, however, requires defining new binary variables and enforcing additional constraints, and was omitted to avoid unnecessarily complicating the presentation. A second feature also omitted for the same reason is the option to choose among multiple time windows at each store. As mentioned, longer time windows give more flexibility and are occasionally used by the sponsoring company.

4.5. SOLUTION METHODOLOGY

Model *RRDP* turned out to be unsolvable with commercial software for instances of practical size. A typical realization has more than 50,000 arcs in the route diagram and may require 100 vehicles (or more), giving upwards of 5×10^6 variables and 1×10^7 constraints. Moreover, it is difficult to derive a good upper bound on the number of vehicles required in an optimal solution. The number of vehicles available in practice or the number of vehicles that can be loaded per unit time both give a bad bound. Due to these difficulties, obtaining exact solutions using a standard MIP solver is out of reach. As an alternative, we have developed a GRASP to find high quality solutions, at least from a practical point of view.

4.5.1. GRASP

GRASP is a metaheuristic that combines greedy heuristics, randomization, and local search, and has been widely applied to combinatorial optimization and industry-based problems (Festa and Resende 2009). Each iteration consists of two main phases: solution construction and local search. In phase I, solutions are built iteratively by randomly selecting one or more elements from a candidate list of good choices. Multiple runs lead to different solutions thus allowing a large portion of the feasible region to be explored. In phase II, local search is applied to improve the promising solutions uncovered in phase I. Depending on the size of the problem and how neighborhoods are defined, the algorithm may or may not converge to a local optimum.

The process is repeated many times and the best solution is returned. As noted by a number of researchers, many successfully GRASP applications to variants of the VRP have been reported (e.g., see Kontoravdis and Bard 1995, Carreto and Baker 2001, Nguyen et al. 2012). Its main advantage over other metaheuristics is its ability to generate many good alternative solutions, which is important for routing applications since travel times are often estimates that can vary widely depending on time of day, road structure, speed limits, and weather.

Compared to local search alone, GRASP has two general advantages. First, it can make better use of parallel computing when multiple cores are available as was the case in work. Different cores can run different programs independently, thus speeding up the computations. Specifically, many phase I and phase II iterations can run simultaneously. In contrast, pure local search, which seeks to improve a given solution is sequential in nature and so cannot be effectively parallelized. Second, GRASP investigates a large number of initial solutions rather than just one. In our experience,

where you start is often highly correlated with solution quality (e.g., see Bard et al. 1998, Deng and Bard 2011, Bard et al. 2014).

Similar to Kontoravdis and Bard (1995), our phase I procedure begins by initializing a number of seed routes and then iteratively assigning one delivery or pickup node to one route at a time subject to certain selection rules. If a node cannot be inserted into an existing route, a new route is started. This process is repeated until all nodes are assigned to some route. Phase II tries to improve every solution found in phase I with a local search procedure. To avoid overly complicating the logic, we only consider the short time window option during phase I and ignore the material handling capacity at the warehouse. As a consequence, the solution found may not necessarily be feasible since there may be more routes identified than the material handling equipment at the warehouse can handle in a day. To address this issue, a route elimination procedure is called during phase II that allows for the use of long time windows. In Section 4.5.2, we discuss phase I, the solution construction procedure, and in Section 4.5.3, we discuss phase II, local search.

As implemented, GRASP terminates when one of the following two conditions is met: i) a maximum runtime of 30 minutes is reached; ii) a total of M phase I and phase II iterations are executed. In Appendix B we show that the complexity of the procedure is polynomial under these stopping rules.

4.5.2. Phase I: solution construction

4.5.2.1. Seed route generation

As with many construction heuristics, phase I of our GRASP starts with a set of seed routes which are used to iteratively build a solution based on the opportunity cost of adding an order to a route. To construct seed routes, Solomon (1987) suggests two

possible strategies: maximum distance from the depot and earliest deadline. Kontoravdis and Bard (1995) select seed locations (stores) that are either the most geographically dispersed from each other or the most time constrained. In our context, there are several orders for a store so it would be suboptimal to select two seed routes to serve two different nodes associated with the same store. To avoid this situation, we adopt a strategy that first selects a set of seed stores and creates one seed route for each. When selecting the seed stores, we consider two criteria: (i) geographically separation, and (ii) the volume of the orders – large is better. Kontoravdis and Bard demonstrated that geographically dispersed seeds can generate high quality solutions. Comparing volume to weight, we found that the former is generally the limiting factor so focusing on large volume orders first helps minimize the number of vehicles required to cover demand. This is important in our application because the warehouse can only handle a limited number of vehicles within the 13-hour dispatching time window.

The number of seed routes is calculated using the lower bounding procedures presented by Kontoravdis and Bard. Specifically, we perform the following calculations and take the largest bound as the final number of seeds.

- Volume capacity bound: only the volume of a vehicle is considered; all other constraints, including separations between different orders and stores, are ignored. In this way, the problem is reduced to a bin packing problem. Since no volume separation is considered, the bin capacity corresponds to the capacity of the truck.
- Weight capacity bound: this bound only considers the weight capacity of the vehicle. Similarly, a bin packing problem is solved to get a bound.
- Driving time bound: this bound considers the total driving time limit for a driver. We use ϕ_s to denote the amount of time needed to go from store s to its closest

neighbor. Solving a bin packing problem with bin capacity $tlimit_1$ and items of size ϕ_s, $s \in S$ gives a lower bound on the number of vehicles.

- Total time bound: this bound considers the total time limit from and back to the warehouse. Here, φ_s denotes the minimal amount of time needed to start serving orders in other stores or to return to the warehouse after completing an order at store s. Note that φ_s accounts for both the travel time between stores and any waiting time that is required. Solving the following bin packing problem with bin capacity $tlimit_2$ and items of size φ_s, $s \in S$ gives a lower bound. Since we usually have $tlimit_1 < tlimit_2$, neither time bound dominates the other.

Given that solving the above bin packing problems exactly may be too time consuming, we use the algorithm given by Martello and Toth (1990) to generate approximate solutions.

Our seed selection procedure starts by ranking the stores by the volume of their orders. Those at the top of the list whose total delivery volume cannot be served by a single vehicle are identified and then each is disaggregated into one or two orders whose volume does not exceed the capacity of a truck after reductions are taken into account. These orders form the core seeds and are placed in the set S_E. Other stores are iteratively added with the objective of maximizing the total distance of each candidate from those already selected until n seeds have been chosen, where n is a computed lower bound on the required number of vehicles. Because our data sets do not include exact store coordinates, it was not possible to generate geographically dispersed seeds using the convex hull of the stores. Instead, we use a greedy procedure: at each step a new seed is selected such that it maximizes the minimum distance to those in the set S_E.

In the presentation, we now let τ_{ij} be the travel time between stores r and s rather than the travel time between nodes i and j. Figure 27 highlights the algorithm. Step 1

identifies stores that cannot be served by a single vehicle because of their excessive volume and adds them to the set of seed stores, S_E. Step 2 initializes the set of seeds and the minimum travel time $t(s)$ from store s to the seed stores. If store s is a seed store, we set $t(s) = 0$. Step 3 sequentially adds seed stores based on the value of $t(s)$; the store with maximum $t(s)$ is selected at the current iteration.

```
Procedure_Seed_Store_Selection
Input: Set of unscheduled stores S
       Number of seed routes n (this value is based on lower bound calculations
given below.)
Output:Set of seed stores S_E
Step 1: Set S_E = Ø
       For s ∈ S
           v(s) = volume required for orders at s, including separation between
                  different types
           If v(s) > vcap
               add s to S_E
Step 2: For s ∈ S \ S_E
           t(s) = min{ τ_si : i ∈ S_E} for all s ∈ S \ S_E
Step 3: While |S_E| < n
           q = argmax{t(s) : s ∈ S \ S_E}
           If t(q) ≠ 0, then
               S_E = S_E ∪ {q}
               t(q) = 0
               t(s) = max{t(s), τ_sq} for all s ∈ S \ S_E
           Else
               Terminate
```

Figure 27. Procedure to select seed stores

After selecting the seed stores, we need to construct a route for each seed store. Two strategies were considered. In the first case, we only use one node from each seed store to initialize a route. For store s, we pick the order $i = \text{argmax}\{v_i : i \in O_s\}$, i.e., the order with largest volume. In the second case, we use multiple orders to initialize the routes. For store s, we pick a combination of orders that are within the volume and

weight limit and that uses as much of the available volume capacity as possible while ensuring that the weight limits are not exceeded. Potentially, this strategy has the following advantages.

- Reduce unnecessary traveling between stores. Accordingly, we can save both traveling time and traveling cost.
- Reduce the separation volume between different stores whose orders are on the same vehicle. Since volume is the primary bottleneck, we reduce the possibility that more vehicles will be required than the warehouse can handle.
- Reduce driver idle time. As indicated by constraints (4.7), the time window for a store is only enforced for the first order at that store. When a vehicle finishes unloading an order at a given store, it can continue with any additional orders for that store without considering the time window. In contrast, when a vehicle completes an order at one store and moves to another, the time window must be observed at the latter store.

Our computational study confirmed that it is usually better for a single vehicle to serve multiple orders (nodes) at the same store rather than splitting those orders among vehicles. Figure 28 describes how we generate a seed route with multiple nodes at seed store s. Step 1 sorts the delivery order nodes for store s in decreasing value of volume and initializes the seed route. Since the volume is the main capacity bottleneck, Step 2 iteratively adds order nodes to the route as long as there is sufficient room on the truck.

```
Procedure_Multi-node_Seed_Route_Generation
```
Input: Seed store s
Output:A seed route ρ^*
Step 1: O_s, = the delivery nodes for store s
Sort Q_s in decreasing order of volume.
Create an empty route ρ
$\rho^* = \rho$
Step 2: For each node $i \in O_s$,

Let ρ_1 = the route after adding order node i to route ρ
If the volume of route $\rho_1 \leq vcap$ or weight of route $\rho_1 \leq wcap$
set $\rho^* = \rho_1$

Figure 28. Procedure to generate a multi-node seed route

4.5.2.2. Route construction

After initializing the seed routes, we construct a feasible solution by sequentially assigning other order nodes to either existing routes or to new routes. One key element of this procedure is how to define the opportunity cost of inserting an unassigned order into partially built route. We adopt one of the approaches of Solomon (1987) because it has been seen to generate good initial feasible solutions. To facilitate future discussion, let ρ be the index for a vehicle route that consists of an ordered set of transition arcs which start and end at the warehouse. Also, let Ω be the set of routes in a complete or partial solution, and ρ_{ij} be the route to which arc (i, j) belongs.

We call inserting node k into arc (i, j) feasible if the augmented route is feasible (of course, we mean that node k is being inserted between nodes i and j). Figure 29 describes the feasibility check. It first determines if the weight and volume limits of the vehicle are violated, then determines whether the number of pickup stores and the number of delivery stores are within limits, and finally determines if the time window of each node on the route is met. It is easy to see that the complexity of the procedure is $O(|A_\rho|)$ where $|A_\rho|$ is the number of arcs in route ρ. If inserting node k into arc (i, j) is feasible, we associate an opportunity cost, with it, denoted by $c_{ij,k}$. This cost consists of two parts: (i) opportunity cost due to reduced capacity of the vehicle, denoted by $c^1_{ij,k}$, and (ii) change in travel and waiting cost after inserting the new node, denoted by $c^2_{ij,k}$. Since the volume and weight of pickup orders are negligible and therefore taken to be zero in the model, we define $c^1_{ij,k} = 0$ if k is a pickup order.

```
Procedure_Check_Route_Feasibility
```
Input: Route $A_\rho = (i_1, j_1)$ - ⋯ - (i_K, j_K), where K is the number of arcs in route ρ
Output:<true> if route ρ is feasible, <false> otherwise
Step 1: If total weight of route $\rho >$ *wcap* or total volume of route $\rho >$ *vcap*, then
Return <false> and stop.
Step 2: If number of delivery stores in route $\rho \geq$ *dcap* or number of pickup stores in route $\rho \geq$ *pcap*
Return <false> and stop.
Step 3: //check if the time window of each order node can be satisfied
$t = 0$
For $(i, j) \in A_\rho$
$t = \max(a_k, t + \hat{\tau}_{ij})$
If $t > b_j$, then return <false> and stop.
Return <true>

Figure 29. Procedure to check if a route is feasible

Given that the available volume and weight of a vehicle are highly correlated and that the volume capacity is usually the bottleneck, we use the volume capacity when calculating cost $c^1_{ij,k}$ for delivery order k. As with bin packing heuristics, a rule of thumb in defining $c^1_{ij,k}$ is that we should assign orders with larger volume a smaller opportunity cost so that they are inserted into routes earlier during construction; otherwise, additional routes might be needed at additional expense. Accordingly, we define $c^1_{ij,k} = (vcap - v)/vcap$, where v is the volume of route ρ_{ij} after inserting node k. Cost $c^2_{ij,k}$ measures the change in the actual cost after inserting node k into arc (i, j). If such an insertion is not feasible, we set $c^2_{ij,k} = \infty$; otherwise, we have

$$c^2_{ij,k} = \theta_{ij} + \theta_{kj} - \theta_{ij} + \sigma(z_{ij,k} - z_{ij}),$$

where z_{ij} is the total waiting time in route ρ_{ij}, and $z_{ij,k}$ is the total waiting time after inserting k into arc (i, j). The total cost of an insertion is then $c_{ij,k} = \delta c^1_{ij,k} + c^2_{ij,k}$, where δ is a capacity reduction parameter whose value reflects the relative importance of volume reduction versus the change in cost after node insertion. When δ is large, more emphasis

is placed on inserting large orders into the route earlier, and vice versa when δ is small. Different problem contexts may require different values of δ so parameter tuning is necessary before using the procedure.

We now define the opportunity cost of inserting node k into route ρ, denoted by $c_{\rho, k}$, as the minimum cost that can be achieved by such an insertion:

$$c_{\rho,k} = \min_{(i,j)\in\rho} c_{ij,k}$$

Accordingly, we can find the best route in which to insert node k as follows.

$$\rho^* = \arg\min\{c_{\rho,k} : \rho \in \Omega\}$$

To decide which node should be inserted first, we define the opportunity cost of inserting node k into route ρ^* as

$$\Pi_k = \sum\nolimits_{\rho\in\Omega}\left(c_{\rho,k} - c_{\rho^*,k}\right)$$

To summarize, the above symbols have the following meaning.

ρ	index for routes
Ω	set of routes in a complete or partial solution
ρ_i	route to which node i belongs
$c_{ij,\,k}$	opportunity cost of inserting node k into arc (i, j)
$c^1_{ij,k}$	opportunity cost due to reduced capacity of a vehicle after inserting node k into arc (i, j)
$c^2_{ij,k}$	travel and waiting cost change after inserting node k into arc (i, j)
$c_{\rho,\,k}$	opportunity cost of inserting node k into route ρ
ρ^*_k	best route to insert node k
Π_k	opportunity cost of inserting node k into route ρ^*_k

The procedure to construct an initial solution is summarized in Figure 30. Step 0 fixes the number of seed routes to generate, while Step 1 initializes the seed routes and calculates the opportunity cost of inserting each unassigned node into each seed route. Step 2 checks to see if there is any node that cannot be assigned to an existing route; if such a node exists, a new route is created. Step 3 finds the smallest opportunity cost for each unassigned node. Step 4 randomly selects one unassigned node from the restricted

candidate list consisting of the λ smallest opportunity cost nodes. Step 5 updates the opportunity cost for the new or modified route. In Step 6, if the number of routes generated is greater than the number of seeds, n_S, the process is restarted using the increased number of routes as the new number of seeds; otherwise, the procedure terminates. Step 6 is motivated by the idea that creating more seed routes at the beginning of the procedure will provide more flexibility during solution construction and thus provide a better result.

`Procedure_Generate_Initial_Solution`

Input: Set of unscheduled nodes O_U; integer parameter λ

Output: Set of feasible routes Ω serving all stores and demands

Step 0: n_1 = volume capacity bound on the number of vehicles
n_2 = weight capacity bound on the number of vehicles
n_3 = driving time bound on the number of vehicles
n_4 = total time bound on the number of vehicles
Set number of seed routes $n^* = \max\{n_1, n_2, n_3, n_4\}$

Step 1: Generate n^* seed routes to form the set Ω
For each $k \in O_U$ and each route $\rho \in \Omega$, determine

$$c_{\rho,k} = \min_{(i,j)\in\rho} c_{ij,k}$$

Step 2: If there is a node $k^* \in O_U$ with $c_{\rho,k} = \infty$ for all $\rho \in \Omega$, then
Create a new route serving node k^* and denote as $\rho(k^*)$
Put $O_U \leftarrow O_U \setminus \{k^*\}$ and $\Omega \leftarrow \Omega \cup \{\rho(k^*)\}$
Go to Step 5
Else
Go to Step 3

Step 3: For each unassigned node k
Find the cost $c_{\rho^*,k} = \min_{\rho\in\Omega} c_{\rho,k}$
Find the opportunity cost $\Pi_k = \sum_{\rho\in\Omega_k} \left(c_{\rho,k} - c_{\rho^*,k}\right)$

Step 4: Sort O_U by increasing value of Π_k
Set $\lambda^* = \min\{\lambda, |O_U|\}$
Randomly select a node from the first λ^* elements in O_U
Let k^* be the node selected and $\rho(k^*)$ the corresponding route
Insert node k^* into route $\rho(k^*)$ at the min-cost location
Put $O_U \leftarrow O_U \setminus \{k^*\}$
If $|O_U| > 0$, then
Go to Step 5
Else
Go to Step 6

Step 5 For each $k \in O_U$
Update $c_{\rho(k^*),k} = \min_{(i,j)\in\rho(k^*)} c_{ij,k}$
Go to Step 2

Step 6 If $|\Omega| > n^*$, then
$n^* = |\Omega|$ and go to Step 1
Else
Terminate

Figure 30. Procedure to generate initial routes

4.5.2.3. Loading capacity

The solution generated by the procedure in Figure 30 may not be feasible since it does not consider the loading capacity of 40 vehicles per hour at the warehouse. To address this issue we apply two strategies. The first starts each route as late as possible. Since a vehicle may be loaded any time before its departure time without incurring idle time cost, this strategy will reduce the material handling workload at the beginning of the planning horizon and thus increase the likelihood that routes with customers who have earlier time windows can be accommodated. Figure 31 describes the procedure to determine the latest departure time of a route.

The second strategy is to reduce the number of routes when possible. Figure 32 outlines the procedures. It works by trying to move the nodes in a particular route to other routes, and terminates when a sufficient number of routes are eliminated so that the loading capacity is satisfied. When moving a node to another route, we evaluate all feasible insert locations and pick the one that leads to the least cost for the route. When the procedure terminates, it is possible that the number of routes is still too many to be loaded at the warehouse. In this situation, the route elimination procedure is applied in every iteration of the local improvement heuristic. If the solution is still infeasible at termination, we declare the problem instance to be infeasible.

```
Procedure_Get_Latest_Departure_Time
Input: Vehicle route Ω_ρ = (i_1, j_1) - ⋯ - (i_K, j_K)
Output:Latest departure time of route from the warehouse
Step 1: t = ∞
Step 2: For (i, j) = (i_K, j_K) to (i_1, j_1)
            Set t = min{ t − τ̂_ij , b_j}
        Return t
```

Figure 31. Procedure to determine latest departure time from warehouse for a route

```
Procedure_Reduce_Number_of_Routes
```
Input: Set of routes Ω in the phase I solution
Output:New set of (fewer) routes
Step 1: Sort Ω in decreasing order of the latest departure time from the warehouse
Step 2: For each route $\rho \in \Omega$
    If all nodes in route ρ can be moved to other routes without causing infeasibilities, then
        For each node $l \in \Omega_\rho$
            For each route $r \in \Omega \setminus \{\rho\}$ and arc $(i, j) \in \Omega_r$
                If r is feasible after insertion, then
                    $C_{ij,r}$ = change of cost for r after inserting l into (i, j)
                Else
                    $C_{ij,r} = \infty$
            $(i, j, r) = \text{argmin}\{C_{ij,r} : r \in \Omega \setminus \{\rho\}\ , (i, j) \in \Omega_r\}$
            Insert node l into arc (i, j)
        Put $\Omega \leftarrow \Omega \setminus \{\rho\}$
        If the loading capacity at the warehouse is satisfied
            Terminate

Figure 32. Procedure to reduce the number of routes

4.5.3. Phase II: local improvement heuristics

In phase II of GRASP, an effort is made to improve the solution found in phase I with various exchange techniques. Here, we implemented several variants of tabu search as well as a large neighborhood search. To facilitate the discussion, let GRO, REF, FRO and SAL denote grocery, refrigerated, frozen and salvage orders, respectively, and let x-XXX denote an order node, where x is the index for the store and XXX $\in$ {GRO, REF, FRO, SAL} is the type of order. As mentioned earlier, solution construction does not consider the use of long time windows. Depending on the scenario, though, we now allow for their use on some or all of the routes. For the mixed case, the fraction of routes that are permitted to use the long time window is limited to γ = 0.2.

4.5.3.1. Tabu search

Tabu search has been demonstrated to produce high quality solutions to many complex problems including various versions of the VRP (e.g., see Taillard 1993, Rochat and Semet 1994, Gendreau et al. 1994). In this section, we present the framework for our tabu search algorithm developed to improve the phase I solutions.

Four types of neighborhood are considered that take into account both short and long time window restrictions. In the presentation, let $g_k = 0$ indicate that route ρ_k is governed by its short time window and $g_k = 1$ indicate that route ρ_k is governed by its long time window. If neither option is specified it is assumed that the short time window restriction applies.

- **neighborhood RI[k, (i, j), g_k, g_i]** is constructed from current solution by performing the following operations:
  - remove a node k from route ρ_k and insert it between nodes i and j in route ρ_i
  - use time window option g_k for route ρ_k (after removing node k) and g_i for route ρ_i (after inserting node k)

  Figure 33(a) and Figure 33(b) respectively illustrate the routes before and after these operations.
- **neighborhood S[i, j, g_i, g_j]** is constructed from current solution by performing the following operations:
  - swap node i in route ρ_i with node j in route ρ_j
  - use time window option g_i for route ρ_i (after the swap) and g_j for route ρ_j (after the swap).

  Figure 34(a) and Figure 34(b) illustrate the routes before and after these operations. Note that if both ρ_i and ρ_j only serve one node, then swapping nodes i and j would

not change the solution. Thus, we would not consider such a swap to be in the neighborhood.

- **neighborhood RRI[k, i, (j, l), g_k, g_i, g_j]** is constructed from the current solution by performing the following operations:
  - remove node k from route ρ_k
  - replace node i in route ρ_i with node k
  - insert node i into arc (j, l) in route ρ_j
  - use time window option g_k for route ρ_k (after removing node k), use time window option g_i for route ρ_i (after replacing node i with node k), and use time window option g_j for route ρ_j (after inserting node i)

  Figure 35(a) and Figure 35(b) illustrate the routes ρ_i, ρ_j, and ρ_k before and after these operations.
- **neighborhood SA[(i, j), (k, l), g_i, g_k]** is constructed from the current solution by performing the following operations
  - remove arc (i, j) from route ρ_i and arc (k, l) from route ρ_k
  - add arcs (i, l) and (j, k) to routes ρ_i and ρ_k respectively
  - use time window option g_i for the new route ρ_i and use time window option g_k for the new route ρ_k

  Figure 36(a) and Figure 36(b) illustrate the routes ρ_i and ρ_k before and after these operations.

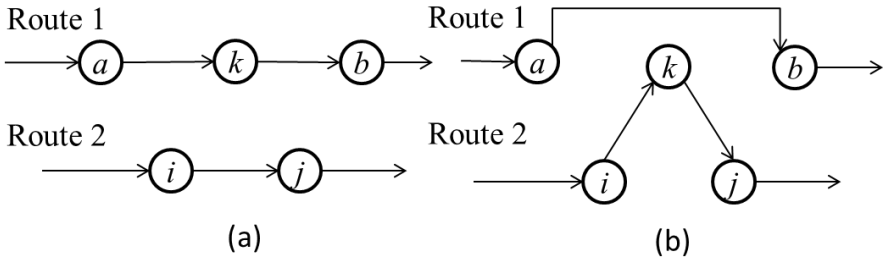


Figure 33. Neighborhood RI[k, (i, j), g_k, g_i]

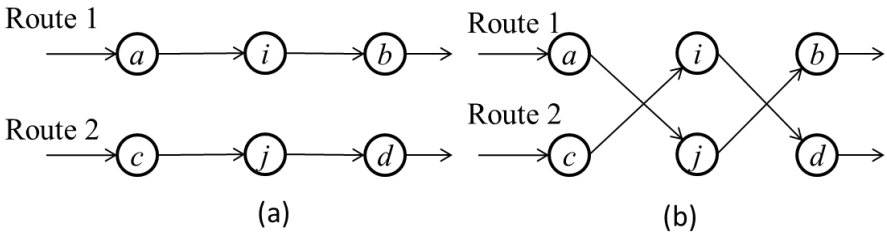


Figure 34. Neighborhood S[i, j, g_i, g_j]

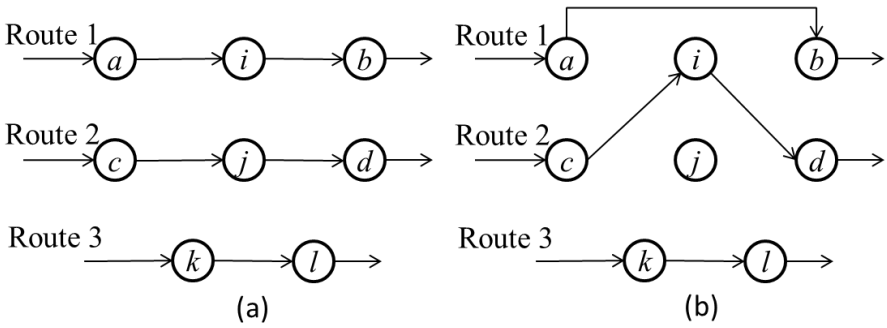


Figure 35. Neighborhood RRI[k, i, (j, l), g_k, g_i, g_j]

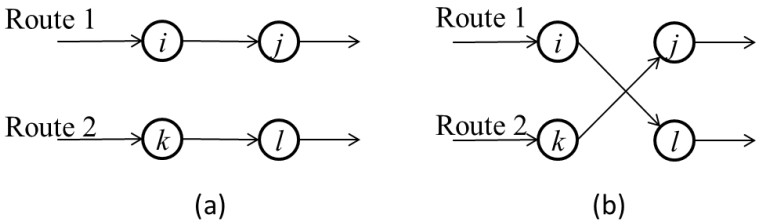


Figure 36. Neighborhood SA[(i, j), (k, l), g_i, g_k]

A neighborhood solution is feasible if all the routes in the solution are feasible, the fraction of routes that use the long time window is within the limit γ, and the warehouse loading capacity is satisfied. Thus, deciding whether a neighborhood is feasible involves three steps: (i) determine whether the ratio of the long-time-window routes is within the given limit γ, (ii) check the feasibility of each route associated with the neighborhood using the procedure in Figure 29 (if any route is not feasible after being updated, the neighborhood is not feasible); and (iii) check to see if the warehouse loading capacity is sufficient for the solution.

As a local search procedure, each iteration of tabu search entails moving from one neighborhood solution to another. In doing so, a tabu list of size N^T is maintained that contains the most recent N^T neighborhoods that have been visited. During each iteration of basic tabu search, only neighborhoods not on the list are considered and the best one is selected, i.e., the neighborhood with minimum cost. As discussed earlier, though, the solution found in the construction phase of the GRASP may not be feasible to the warehouse loading capacity constraint. When this is the case, tabu search calls the route elimination procedure outlined in Figure 32. Terminate occurs when either N iterations are performed or a given time limit is reached.

The general tabu search algorithm is highlighted in Figure 37. The first step initializes the parameters and sets. Step 2 searches for the best neighborhood and Step 3

updates the best feasible solution. Step 4 updates tabu list and checks the termination criteria.

```
Procedure_Tabu_Search
Input: Set of routes Ω in a solution
       Number of tabu iterations N
       Size of tabu list N^T
       Solution time limit T
Output: Improved solution Ω*
Step 1: Tabu list L_T = Ø
       Iteration counter n = 1
       If Ω satisfies the warehouse loading capacity
           Put Ω* ← Ω.
       Else
           Put Ω* = Ø and cost of Ω* = ∞
       Current solution Ω^C ← Ω
Step 2: Find the best neighborhood of Ω^C not in L_T; call it NR and go to Step 3
Step 3: Move to neighborhood NR and denote the resulting solution as Ω^1
       If Ω^1 violates the warehouse loading capacity, then
           Apply the route elimination procedure to Ω^1 in Figure 32
       If Ω^1 satisfies the warehouse loading capacity and cost of Ω^1 < cost of Ω*,
           Put Ω* ← Ω^1
       Put Ω^C ← Ω^1
       Go to Step 4
Step 4: Put L_T ← L_T ∪ NR
       If |L_T| > N^T
           Remove the first element in L_T
       Put n ← n + 1
       If n ≥ N or the solution time ≥ T, then
           Terminate
       Else
           Go to Step 2
```

Figure 37. High-level tabu search procedure

4.5.3.2. Tabu search with generalized neighborhood

Initial testing showed that better solutions were obtained when nodes corresponding to the same store were together in a route. This led to a new set of neighborhoods in which moves and swaps were made on more than one same-store node.

We call the resultant algorithm tabu search with generalized neighborhood (tabu search-GN).

To facilitate the discussion, we define the same-store adjacent node list (SANL) as a list of nodes that (i) belong to the same store, and (ii) are adjacent to each other in a route. For example, for route

Warehouse→1-GRO→1-REF→1-FRO→2-FRO→Warehouse,

we have the following SANLs: SANL(1-GRO), SANL(1-REF), SANL(1-FRO), SANL(2-FRO), SANL(1-GRO, 1-REF), SANL(1-REF, 1-FRO), and SANL(1-GRO, 1-REF, 1-FRO). Now, let I be a SANL, and with a slight abuse of notation, let ρ_I denote the route to which SANL(I) belongs.

For tabu search-GN, we consider three types of neighborhoods:

- **neighborhood G-RI[K, (i, j), g_K, g_i]** is constructed from the current solution by performing the following operations:
  - remove a SANL(K) from route ρ_K and insert it between nodes i and j in route ρ_i
  - use time window option g_K for route ρ_K (after removing node k) and g_i for route ρ_i (after inserting node k)
- **neighborhood G-S[I, J, g_I, g_J]** is constructed from the current solution by performing the following operations:
  - swap SANL(I) in route ρ_I with SANL(J) in route ρ_J
  - use time window option g_I for route ρ_I (after the swap) and g_J for route ρ_J (after the swap)
- **neighborhood G-RRI[K, I, (j, l), g_K, g_I, gj]** is constructed from the current solution by performing the following operations:
  - remove SANL(K) from route ρ_K
  - replace SANL(I) in route ρ_I with SANL(K)

- insert SANL(I) into arc (j, l) in route ρ_j
- use time window option g_K for route ρ_K (after removing node k), use time window option g_I for route ρ_I (after replacing SANL(I) with SANL(K), and use time window option g_j for route ρ_j (after inserting SANL(I))

- **neighborhood SA[(i, j), (k, l), g_i, g_k]** is the same as defined in Section 4.5.3.1 for the basic tabu search.

Note that the neighborhood S[i, j, g_k, g_i] is a special case of the neighborhood G-S[I, J, g_I, g_J] with SANL(I) = {i} and SANL(J) = {j}. Similarly, the neighborhood RI[k, (i, j), g_k, g_i] is a special case of the neighborhood G-RI[K, (i, j), g_K, g_i], and the neighborhood RRI[k, i, (j, l), g_k, g_i, g_j] is a special case of G-RRI[K, I, (j, l), g_K, g_I, g_j]. Because Tabu search-GN has strictly larger neighborhoods than tabu search, it has the potential to generate better solutions. The implementation of Tabu search-GN is similar to tabu search but with Step 2 changed to accommodate the new neighborhood structures.

4.5.3.3. Degeneracy of the tabu neighborhood

During our computational experiments, we observed that neighborhoods are highly degenerate, i.e., there are many neighbors that have the same cost as the incumbent. To see this, consider the following examples where it is assumed that the time window option remains the same for all routes so there is no need to specify the values of the parameter g_i.

Example 1. Degenerate RI[k, (i, j)]: Consider the following two routes:

Route ρ_1: Warehouse →1-GRO→1-REF→1-SAL→Warehouse

Route ρ_2: Warehouse →2-GRO→1-FRO→Warehouse

The corresponding routes in the neighborhood RI[1-SAL, (1-FRO, Warehouse)] are:

Route ρ_3: Warehouse →1-GRO→1-REF→Warehouse

Route ρ_4: Warehouse →2-GRO→1-FRO→1-SAL→Warehouse

The total travel costs are the same for both set of routes. Note that since 1-SAL is the last order and follows an order from store 1 in both routes, the total waiting time cost is the same for both solutions. Thus, neighborhood RI[1-SAL, (1-FRO, Warehouse)] is degenerate.

Example 2. Degenerate S[*i*, *j*] and SA[(*i*, *j*), (*k*, *l*)]: Consider the following two routes:

Route ρ_1: Warehouse →1-GRO→1-REF→1-SAL→Warehouse

Route ρ_2: Warehouse →2-GRO→1-FRO→Warehouse

The following routes are in the neighborhood S[1-REF, 1-FRO]:

Route ρ_3: Warehouse →1-GRO→1-FRO→1-SAL→Warehouse

Route ρ_4: Warehouse →2-GRO→1-REF→Warehouse

Similar to Example 1, the total travel and waiting cost is the same for both solutions, and thus the neighborhood S[1-REF, 1-FRO] is degenerate.

Example 3. Degenerate RRI[*k*, *i*, (*j*, *l*)]: In this example, assume that store 1 has an early opening time window no waiting is ever required once a vehicle arrives. Consider the following three routes:

Route ρ_1: Warehouse →2-REF→1-FRO→Warehouse

Route ρ_2: Warehouse →2-GRO→2-SAL→Warehouse

Route ρ_3: Warehouse →3-GRO→2-FRO→Warehouse

Now, evaluating neighborhood RRI[1-FRO, 2-SAL, (2-FRO, Warehouse)] by performing the following moves: (i) remove 1-FRO from route ρ_1, (ii) replace node 2-SAL with node 1-FRO in route ρ_2, (iii) insert node 2-SAL into arc (2-FRO, Warehouse) in route ρ_3, the resulting routes are

Route ρ_4: Warehouse →2-REF→Warehouse

Route ρ_5: Warehouse →2-GRO→1-FRO→Warehouse

Route ρ_6: Warehouse →3-GRO→2-FRO→2-SAL→Warehouse

Since the total travel cost of these routes is the same and there is no waiting time cost incurred for ρ_1, the total cost of both solutions is the same. Thus, neighborhood RRI[1-FRO, 2-SAL, (2-FRO, Warehouse)] is degenerate.

The tabu list has generally proven to be an effective tool for avoiding cycling in a small feasible region and thus allowing for a variety of diversified solutions to be explored before convergence. When degeneracy is present, as in our case, the effectiveness of the tabu list to prevent cycling is reduced and the algorithm tends to get stuck around a small set of degenerate solutions. If every solution has n_Ω degenerate neighbors, it is necessary to make the tabu list larger than n_Ω to assure that the algorithm eventually escapes from the neighborhood. However, if the degerate neighbors themselves have many degenerate neighbors, it would be necessary to use an even larger value of N^T which we now do.

In addition, we have implemented two enhanced strategies. In the first, we penalize solutions in a degenerate neighborhood. If degenerate neighborhood R is visited n_R times during the tabu search, we associate a penalty $n_R \cdot c_d$ with it, where c_d is a cost parameter. When selecting the best neighborhood, we opt for the one with the smallest sum of cost and penalty instead of smallest cost. Such a strategy empirically

restricts degenerate solutions. We call this method Tabu search-DP, where DP stands for degenerate penalty.

The second strategy is to apply randomized variable neighborhood search (Tabu search-RVN). Variable neighborhood search is a metaheuristic that exploits the idea of switching among neighborhoods during local search (Maldenovic and Hasen 1997). Hansen et al. (2010) give an extensive survey of applications. In Section 4.5.3.1, we defined several types of neighborhoods for basic tabu search. To escape from a degenerate neighborhood, we randomize the choice of the one to explore next. Kytojoki et al. (2007) demonstrate this idea for a traditional VRP while Deng and Bard (2011) show its effectiveness when solving capacitated clustering problems.

4.5.3.4. Large neighborhood search

When trying to improve an incumbent, an alternative to tabu search is large neighborhood search (LNS) (Ahuja et al. 2000) which has a strong history of successful implementations (e.g., see Ropke and Pisinger 2006, Pisinger and Ropke 2007). We start with an integer parameter n and define two types of neighborhoods

- **neighborhood n-RI[k_1, k_2, … , k_n, (i, j), g_1, g_2, … , g_n, g_i]** is constructed from the current solution by performing the following operations:
  - remove a node k_1 from route ρ_{k_1}, let node k_l replace node k_{l+1} in route $\rho_{k_{l+1}}$ for $l = 1,\ldots,n-1$, and insert node k_n into arc (i, j)
  - use time window option g_l for route ρ_{k_l} (after the 'remove' or 'replace' operation) for $l = 1,\ldots,n$ and use time window option g_i for route ρ_i
- **neighborhood n-S[k_1, k_2, …, k_n, g_1, g_2, … , g_n]** is constructed from the current solution by performing the following operations:

- let node k_i replace node k_{i+1} in route $\rho_{k_{i+1}}$ for $i = 1, \ldots, n-1$, and let node k_n replace node k_1 in route ρ_{k_1}
- use time window option g_l for route ρ_{k_l} (after the 'replace' operation) for $l = 1, \ldots, n$.

Note that neighborhood RI[k, (i, j), g_k, g_i] is a special case of n-RI[k_1, k_2, … , k_n, (i, j), g_1, g_2, … , g_n, g_i] with $n = 1$ and S[i, j, g_i, g_j] is a special case of n-S[k_1, k_2, …, k_n, g_1, g_2, … , g_n] with $n = 1$. Neighborhood n-RI[k_1, k_2, … , k_n, (i, j), g_1, g_2, … , g_n, g_i] generalizes RI[k, (i, j), g_k, g_i] by introducing a chain of node replacements. As we can see, using n-RI[k_1, k_2, … , k_n, (i, j), g_1, g_2, … , g_n, g_i] and n-S[k_1, k_2, …, i_n, g_1, g_2, … , g_n] greatly expands the size of the original neighborhoods for a given solution and thus increases the possibility of improvement. As a consequence, LNS is often considered an alternative to metaheuristics like tabu search and genetic algorithms, and has been shown to be equally, if not more, effective (Ahuja et al. 2000). Since no tabu list is necessary, the degeneracy in the neighborhood does not affect the performance of LNS.

Figure 38 outlines our LNS. In the procedure, we use $|\Omega|$ to denote the number of routes in solution Ω. The first step initializes the solution. Step 2 identifies the best neighbor with respect to the neighborhoods defined above and reduces the number of routes if the warehouse loading capacity is violated. Step 3 updates the incumbent. Step 4 checks whether the stopping criteria are met: the procedure continues (i) when no solution is found (due to warehouse loading capacity constraint) but it may be possible to further reduce the number of routes or (ii) when the current solution is not a local optimum.

```
Procedure_Large_Neighborhood_Search
```
Input: Set of routes Ω^C in current solution
Neighborhood parameter n
Output:Best solution found Ω^*
Step 1: If Ω satisfies the warehouse loading capacity
Put $\Omega^* \leftarrow \Omega$
Else
Put $\Omega^* = \emptyset$ and cost of $\Omega^* = \infty$
Current solution $\Omega^C \leftarrow \Omega$
Go to Step 2
Step 2: N^* = the best neighbor of Ω^C among n-RI[k_1, k_2, … , k_n, (i, j)] and n-S[k_1, k_2, …, i_n];
Move to neighborhood N^* and let Ω^1 be the corresponding solution
If Ω^1 violates the warehouse loading capacity constraint
Apply the route elimination procedure in Figure 32
Go to Step 3
Step 3: If Ω^1 satisfies the warehouse loading capacity constraint and cost of Ω^1 < cost of Ω^*
Put $\Omega^* \leftarrow \Omega^1$
Go to Step 4
Step 4: If ($\Omega^* = \emptyset$ and $|\Omega^C| > |\Omega^1|$) or (cost of Ω^1 < cost of Ω^C)
Put $\Omega^C \leftarrow \Omega^1$ and go to Step 2
Else
Terminate

Figure 38. Large neighborhood search procedure

4.6. COMPUTATIONAL RESULTS

To test the effectiveness of our algorithms, we use seven data sets provided by Kroger that represent a typical week in the Cincinnati-Columbus region of Ohio. The characteristics of the seven instances are summarized in Table 8. As can be seen, the number of stores range from 120 to 150 and the total number of orders range from 390 to 438. The parameter values used in our computations are summarized in Table 9. Table 10 gives the volume for the separation curtains in a vehicle as a function of the number of stores and number of different order types on a route. All algorithms were implemented

in JAVA and run under Ubuntu Linux on a Dell Poweredge T610 workstation with two 6-core hyperthreading 3.33-GHz Xeon processors and 24 GB of memory.

Table 8. Summary of data set characteristics

Instance	# of stores	# of orders	# of grocery orders	# of refrigerated orders	# of frozen orders	# of salvage orders
0910ST	135	431	149	147	130	5
0911ST	123	401	146	134	121	0
0912ST	130	410	138	139	128	5
0913ST	124	390	134	137	119	0
0914ST	137	424	137	143	135	9
0915ST	150	438	139	156	141	2
0916ST	136	411	130	142	135	4

Table 9. Summary of problem parameters

Name	Value
Set up cost per stop(dollars)	50
Driving cost per mile (dollars)	2.207
Driver idle time cost per hour (dollars)	40
Truck weight limit (pounds)	42000
Truck volume limit (ft^3)	2000
Load and unload rate per hour (ft^3)	1200
Load set up time (minutes)	30
Driver time limit per route (hours)	50
Total time limit per route (hours)	14
Maximum number of trucks loaded at warehouse per 30 minutes	20
Maximum number of stores per delivery route	4
Maximum number of stores per pickup route	4

Table 10. Volume required for separation curtains in a vehicle

Number of order types	One store	Two stores	Three or more stores
One delivery order type	0	100	200
Two delivery order types	160	210	260
Three delivery order types	290	340	390

4.6.1. Results for different construction methods

In this section, we examine the computational results for the two different phase I approaches. When constructing an initial solution using the algorithms in Section 4.5.2.2, we always set $\delta = 0.02$ in the calculation of the opportunity cost, $c_{ij,k}$, of inserting node k in arc (i,i) since it gave the best results on balance. Table 11 compares the initial feasible solutions generated in phase I of the GRASP using multiple nodes vs. single nodes as seed routes. We randomly generated 20 solutions for each instance using each method and report the min cost, the average cost, and total time for the two cases. The results show that the former approach provides initial routes with much lower cost in comparable time. On average, the cost differential for the seven instances is \$16,688 (14.11%). This indicates that generating seed routes with multiple nodes for the same store can help identify better feasible solutions.

Table 11. Comparing different seed route generating methods

Measure	0910ST	0911ST	0912ST	0913ST	0914ST	0915ST	0916ST
Multi-node seed route							
Min cost ($)	106,068	106,634	94,094	100,471	95,014	104,867	94,671
Average cost ($)	107,133	108,079	94,201	102,116	95,521	107,550	95,283
Total time (s)	315	327	380	355	437	503	424
One-node seed route							
Min cost ($)	122,604	123,065	107,243	113,887	105,688	122,372	104,646
Average cost ($)	127,015	124,886	112,926	114,965	112,199	125,654	109,054
Total time (s)	412	392	415	424	417	476	380

To compare the different local search methods considered for phase II, we randomly generated 20 initial solutions with the phase I construction procedure for each instance (a total of 140 solutions), and then tried to improve them. Table 12 reports the minimum cost for each set of the 20 improved solutions, their average cost, and the total runtime. For tabu search and its variants, we set the tabu list size to 200 and the number of iterations to 40,000, and derive an initial solution by running phase I of the GRASP for one iteration. For the LNS, we explored all possible neighborhoods n-RI and n-S with $n \leq 5$.

From the entries in Table 6 we can observe the following:

- LNS is the worst performer, giving the largest minimum cost and largest average cost. In contrast, all variants of tabu search were able to explore a larger portion of the solution space and thus obtain better solutions.
- Tabu search-RVN performed the best amongst its competitors. It produced the lowest average cost and minimum cost for all instances, and incurred the shortest runtime. For instance 0910ST, for example, the average cost of Tabu search-RVN solutions is $1056 (1.04%) lower than the basic tabu search solutions, $793 (0.78%) lower than the Tabu search-GN solutions, $3825 (3.77%) lower than the LNS solutions, and $731 (0.72%) lower than the Tabu search-DP solutions. This

suggests that randomizing the selection of the neighborhood is an effective way to handle degeneracy.

- As discussed earlier, Tabu search-GN examines larger neighborhoods than the other approaches and thus incurs longer runtimes per iteration. In general, this leads to better solutions, which was the case for all instances when compared to the basic tabu search results. For example, for instance 0911ST, the average cost of the Tabu search-GN solutions is $1045 (1.05%) lower than the basic tabu search solutions.
- The penalties included for visiting degenerate neighborhoods improved the solutions obtained with Tabu search-DP when compared to basic tabu search. For example, the average cost of the Tabu search-DP solutions for instance 0911ST is $1176 (1.19%) lower than the basic tabu search solutions.

Table 13 compares the performance of Tabu search-RVN alone and GRASP using Tabu search-RVN in phase II. The cost for each instance is averaged over twenty runs. In each run of Tabu search-RVN, we randomly generate a solution and use Tabu search-RVN to improve the solution, with a runtime limit T = 30 minutes and tabu list size N^T = 200. In each run of the GRASP, we set the total runtime limit to 30 minutes, and we use the tabu list size N^T = 200 number of tabu iterations N = 40,000. Note that the phase II of the GRASP are terminated when number of Tabu iterations reaches 40,000 or the total runtime limit of the GRASP (not the phase II of current iteration) reaches 30 minutes. For GRASP, we used 23 threads so 23 phases I and II GRASP iterations were performed simultaneously. The results show that GRASP performed marginally better in 6 of the 7 instances (except instance 0912ST), leading us to adopt it for the comparisons in the next section.

Table 12. Summary of results for different phase II local search methods

Method	0910ST	0911ST	0912ST	0913ST	0914ST	0915ST	0916ST
Tabu search							
Min cost ($)	101,939	98,144	92,315	93,871	93,733	100,083	91,255
Average cost ($)	102,428	100,167	92,468	94,733	94,062	100,836	91788
Total time (s)	755	623	691	708	848	977	806
Tabu search-GN							
Min cost ($)	101,266	97,801	91,997	93,242	92,624	99,628	91,064
Average cost ($)	102,165	99,122	92,402	94,085	92,941	100,378	91,674
Total time (s)	3337	2645	3052	3128	3650	3777	3023
LNS							
Min cost ($)	103,913	101,929	92,753	97,305	94,130	102,535	94,125
Average cost ($)	105,197	104,629	93,570	98,671	94,689	104,699	95,049
Total time (s)	33	34	19	49	40	56	21
Tabu search-DP							
Min cost ($)	101,300	97,590	91,586	93,335	92,738	99,860	90,361
Average cost ($)	102,103	98,991	92,047	94,261	93,323	100,543	91,383
Total time (s)	927	744	780	1101	1466	1515	1108
Tabu search-RVN							
Min cost ($)	100,694	95,947	90,988	92,872	91,714	99,186	89,565
Average cost ($)	101,372	97,069	91,619	93,334	92,126	99,561	90,288
Total time (s)	639	526	537	591	872	840	795

Table 13. Comparing solutions generated by the tabu search and the GRASP

	0910ST	0911ST	0912ST	0913ST	0914ST	0915ST	0916ST
Tabu search-RVN	100,913	96,420	90,960	92,671	91,843	99,098	90,007
GRASP	100,679	96,277	90,999	92,580	91,665	98,877	89,641

4.6.2. Comparing GRASP solutions with Kroger's solutions

In this section, we compare our GRASP solutions with those provided by Kroger for the seven real instances highlighted in Table 2 and for a second set of results for instances 0911ST and 0196ST only. The solutions for the seven instances were

generated with a commercial software package that is run daily, while the solutions for the two instances were obtained from an experimental set-partitioning heuristic. To make better use of the vehicle capacity when planning, we found that Kroger splits most of the orders greater than 1100 ft^3 into suborders and treats each separately. We adhered to this procedure and did not aggregate orders for the same commodity at the same store.

The number of orders associated with the two cases is summarized in Table 8 andTable 14, respectively. Because the orders correspond to different days of the year, the solutions are not directly comparable. Interestingly, though not uncommon (e.g., see Bard et al. 2014), when examining these solutions, we found many violations of Kroger's operational requirements. Table 15 and Table 16 report the number of routes that violate each requirement in the commercial software solution and set-partitioning heuristic solution, respectively, with short time window violations being the most frequent.

In light of this situation, we calculated two costs for each solution provided by Kroger. In particular, the "Short TW" cost of each route is calculated in one of two ways: (i) if the route satisfies all the short time windows associated the nodes visited, then the total waiting time cost and the transition costs are summed to get the total cost; (ii) if the route violates any short time window of any node, then only the transition cost is reported. In the same way, we calculate the "Long TW" cost. In either case, the true cost of Kroger's solutions are underestimated, considering that all the violations are ignored.

When running the GRASP, Tabu search-RVN again was used in phase II with a list size of 200 and an iteration limit of 40,000. A maximum of 30 minutes was allowed for each run. In the comparisons, we considered four different time window options for the GRASP: long time windows, short time windows, a mix in which at most 20% of the routes were allowed to use the long time windows (20% long TW), and nominally, the

short time windows with the observed parameter values in the Kroger solutions (Short TW-KP). In the latter case, the maximum values of the individual store time windows, vehicle volumes, and vehicle loads were used.

Table 14. Summary of problem characteristics for the set-partitioning solution

Instances	# of stores	# of orders	# of grocery orders	# of refrigerated orders	# of frozen orders	# of salvage orders
0911ST	123	402	154	127	121	0
0916ST	136	418	139	140	135	4

Table 15. Number of violations in the commercial software solution

Instance	# of routes	Total # of violations	Short time window violations	Long time window violations	Weight limit violations	Volume limit violations	Order sequence violations
0910ST	137	28	27	14	1	6	0
0911ST	132	20	20	8	1	3	0
0912ST	128	19	20	7	0	1	2
0913ST	126	27	24	4	1	4	2
0914ST	127	22	21	6	0	1	5
0915ST	137	40	33	14	0	8	0
0916ST	130	32	23	10	0	8	2

Table 16. Number of violations in the set-partitioning solution

Instance	# of routes	Total # of violations	Short time window violations	Long time window violations	Weight limit violations	Volume limit violations	Order sequence violations
0911ST	127	24	17	0	12	0	9
0916ST	124	33	27	0	7	0	2

Table 17 and Table 18 compare the total cost of the GRASP solutions for the different time window options with those obtained with Kroger's commercial package and set-partitioning heuristic, respectively. We can see that even though the Kroger

solutions violate many operational requirements (which implies that their costs are underestimated), they are still inferior to the GRASP solutions in all instances. Nevertheless, the fairest comparison is for the Short TW-KP option in which the parameter values implied by Kroger's solutions are used in the GRASP. Here, the results can be brought into clearer focus. For example, the Short TW cost of the commercial software solutions for instance 0913ST in Table 17 is $2354 (2.48%) inferior to the GRASP Short TW solutions and $3615 (3.82%) worse than the GRASP Short TW-KP solutions. The same pattern is true for the other instances in Table 17 and Table 18. On average, the difference between the cost of the commercial software solutions and the GRASP Short TW-KP solutions is $2727 (2.85%), and the difference between the cost of the set-partitioning solutions and the GRASP Short TW-KP solutions is $1123 (1.21%).

For the GRASP results only, the Long TW solutions dominate the 20% long TW solutions, which in turn dominate the Short TW solutions for all instances. This was to be expected since longer time windows mean a larger feasible region. Relatively speaking, extending the time windows for only 20% of the routes achieved more than half of the cost reduction with respect to the Short TW solutions. To see this, let the cost reduction percentage = $100 \times (C_S - C_P)/C_S$, where C_S denotes the solution with short time windows and C_P denote the costs of solution with 20% long time windows. On average, the cost reduction percentage is 0.78% ($732) for the seven instances in Table 17 and is 1.05% ($969) for the two instances in Table 18.

To evaluate the effect of extending time windows for all routes, we calculate the cost reduction percentage as $100 \times (C_S - C_L)/C_S$, where C_S and C_L denote the solutions with short and long time windows, respectively. On average, the cost reduction percentage is 1.39% ($1324) for the seven instances in Table 17 and is 1.41% ($1278) for the two instances in Table 18. This means that the marginal benefit of extending the

time windows to more routes is a nonlinear decreasing function: go from 20% of the routes can achieve more than half of the cost savings resulting from extending time windows for all routes.

Table 17. Comparing GRASP solution with the commercial software solution

Solution	0910ST	0911ST	0912ST	0913ST	0914ST	0915ST	0916ST
GRASP							
Long TW	98,826	95,037	90,053	91,157	90,091	96,840	88,646
20% Long TW	99,461	95,598	90,565	91,784	90,684	97,721	88,980
Short TW	100,694	95,947	90,988	92,391	91,659	98,674	89,565
Short TW-KP†	99,004	95,527	90,189	91,130	90,809	97,051	88,406
Commercial software							
Short TW Cost	101,354	98,852	92,989	94,745	93,432	98,829	91,074
Long TW Cost	100,190	97,688	91,530	93,524	92,505	98,208	90,424

† Short TW-KP solution is generated by the GRASP using the maximum values of the parameters associated with the commercial software solution.

Table 18. Comparing GRASP solution with the set-partitioning solution

Method	Solution	0911ST	0916ST
GRASP	Long TW	95,149	87,790
	20% Long TW	95,153	88,422
	Short TW	95,947	89,565
	Short TW-KP†	95,527	88,399
Set-partitioning heuristic	Short TW	96,443	89,728
	Long TW	94,968	88,405

† Short TW-KP solution is generated by the GRASP using the maximum value of the parameters used in the set-partitioning solution.

4.7. SUMMARY AND CONCLUSIONS

In this paper, we developed a series of algorithms to help logistics managers construct daily pickup and delivery routes from a central warehouse to outlining retail

stores. The problem was defined by a combination of standard and context-specific constraints, such as order loading restrictions, dynamic vehicle volume limits, weight limits, maximum time on the road, and warehouse time windows. Several of these are new to the VRP literature. To model the problem, we began with a route diagram that captured the possible transitions between order nodes and served as a basis for a mixed-integer programming model.

To find solutions, we designed and tested various heuristics that integrated techniques associated with GRASP, tabu search, and large neighborhood search. In phase I of the GRASP, we found it best to use multiple order nodes to identify seed routes that are then expanded sequentially one node at a time. In phase II four local search methods were considered to improve the initial solutions. A critical observation at this stage was that all neighborhoods were degenerate. This undermined the ability of tabu search, our primary phase II procedure, to find local optima. To resolve this issue, we proposed two enhancements. The first was to greatly expand the neighborhood definition, and the second was to randomize the choice of neighborhoods to explore at each iteration. The latter proved to be the more effective.

Not surprising, extensive testing showed that GRASP with RVN performed best in the vast majority of cases. Comparing the corresponding solutions with those provided by Kroger for the Cincinnati-Columbus region showed that daily savings up to $3615 could be achieved, and perhaps more since the comparisons are based on conservative estimates of Kroger's costs. Basic tabu search did not perform well on these instances primarily due to neighborhood degeneracy.

As extensions of this research, it is worth investigating different strategies for disaggregating the large, same-store orders. In general, split deliveries are not welcome by managers but when the demand at a store exceeds the capacity of the vehicle, there is

a clear need to work with reduced order sizes. In the context of our problem, orders are specified by commodity rather than by store and each is subject a novel set of constraints. This further complicates the routing problem since it leads to an explosion in the number of feasible sequences. A second extension centers on exact solution methods. Although it is unlikely that provably optimal solutions can be derived, decomposing the network into clusters of nearby delivery locations, and then finding solutions for each cluster might be a promising way to start.

Chapter 5. Conclusions

In this dissertation, we study three transportation scheduling problems: the train dispatching problem in railroad industry, the transportation network design problem with service requirements, and the daily service route design problem for major grocery chains.

Each problem is important in their industry and has its industry-specific operational requirements. The train dispatching problem coordinates the movements of trains on railroad tracks so that the average velocity of trains and the throughput of the railroad system are maximized. Various operational requirements are considered, including the separation between adjacent trains, train priorities, and the track unavailability due to the maintenance of way. The transportation network design problem with service requirement builds a subset of arcs in the network and routes commodities on built arcs to minimize the total fixed and variable cost. Besides, each commodity is required to be routed in a simple path that satisfies the total service requirement. The daily service route design problem routes a set of vehicles to deliver orders from the warehouse to stores and pickup salvage orders back to the warehouse. A variety of operational requirements are incorporated, including the warehouse handling capacity, the truck weight and volume limits, and the driver time limits. Moreover, there are four types of orders and these orders must be loaded in a certain sequence in trucks.

Mathematical programming is a useful tool to model these problems. We show that there are many alternative ways to model some requirements that and the strength and size of different models are different. In the train dispatching context, we show that our non-concurrency constraints are more effective in modeling the unidirectional requirement than the pairwise constraints in literature since the number of constraints is

greatly reduced. We compare three formulations, namely the arc-flow, path-flow and hybrid formulations, for the NDSR and show that they are different in terms of strength and model size. Besides, we also develop a set of valid inequalities to strengthen the integer programming model in both the train dispatching problem and NDSR.

We also show that heuristics are helpful in two aspects. First, it can speed up the state-of-the-arc MIP solver by providing an initial feasible solution. We show that an initial warm-start solution, together with model strengthening, can reduce the solution time tremendously. Second, it can provide feasible solutions when the MIP solver fails to find one. In the service route design problem, the mathematical formulation is too large to be solved within reasonable amount of time. Instead, we develop a GRASP to provide a good solution.

This dissertation can contribute to the existing literature in the following ways. First, the modeling and solution strategies of these problems are helpful to both academicians and practitioners in these areas. Besides, it can contribute to the integer programming communities. Our non-concurrency constraints that model the unidirectional requirements in train dispatching and the concept of hybrid formulation for NDSR are novel techniques that can be used in other context. Finally, our work can contribute to the area of meta-heuristic. We show that the Tabu search can be less effective when the neighborhood is highly degenerate. To address the issue, we use the Tabu search with random variable neighborhood. We also show that the GRASP, which improves many solutions with the Tabu search but spends less effort on each solution, can make use of the modern parallel computing technique and are thus more effective than the Tabu search, which improves only one solution with all the effort.

Appendices

APPENDIX A. SEPARATIONS FOR VALID INEQUALITIES IN CHAPTER 3

A.1. Separation procedure for inequality (3.26)

Figure 39 discuss how to find all the violated inequality (3.26) for a given train movement solution vector **x**, a pair of trains q_1 and q_2 traveling in the opposite directions, and segment m. Specifically, Step 1 of the procedure finds the set of time period that should be considered and Step 2 of the procedure goes over each possible pair of time periods to find violated inequalities.

Procedure separate_unidirectional
Input: movement solution vector **x**, train q_1, train q_2 and segment m.
Output: C, set of inequalities (3.26) violated by **x**.
Step 1: $C = \phi$;
$T(q_1) = \{t(q_1, 1), t(q_1, 2), \cdots, t(q_1, K)\} \subseteq T$, where $x^{q_i}_{mt(q,k)} > 0$ and $t(q_1, i) > t(q_1, i+1)$;
Step 2: For $i = 1$ to k
For $j = i$ to k
$t_1 = t(q_1, i)$
$\bar{t}_1 = t(q_1, j)$
$t_2 = \bar{t}_1 - \tau^{q_2}_m - \theta + 1$
$\bar{t}_2 = t_1 + \tau^{q_1}_m + \theta - 1$;
$t(q') = \max\{\bar{t}_1 - \theta + 1,\ \bar{t}_2 - \tau^{q'}_m - \theta + 1,\ t_1\}$
$\bar{t}(q') = \min\{t_1 + \theta - 1,\ t_2 + \tau^{q_2}_m + \theta - 1\}$ for all $q' \in Q^+_m \setminus \{q_1\}$
If $\bar{t}_1 - t_1 \geq \theta - 1$ and $\sum_{t'=t_1}^{\bar{t}_1} x^{q_1}_{mt'} + \sum_{q' \in Q^+_m \setminus \{q_1\}} \sum_{t'=t(q')}^{\bar{t}(q')} x^{q'}_{mt'} + \sum_{t'=t_2}^{\bar{t}_2} x^{q_2}_{mt'} > 1$
add $\sum_{t'=t_1}^{\bar{t}_1} x^{q_1}_{mt'} + \sum_{q' \in Q^+_m \setminus \{q_1\}} \sum_{t'=t(q')}^{\bar{t}(q')} x^{q'}_{mt'} + \sum_{t'=t_2}^{\bar{t}_2} x^{q_2}_{mt'} \leq 1$ to $C(q_1, q_2)$;
Else if $t_2 > \bar{t}_2$
break;

Figure 39. Procedure to separate unidirectional inequalities (3.26)

A.2. Separation procedure for inequality (3.31)

Figure 40 discuss how to find all the violated inequality (3.31) for a given movement and wait solution vector (**x**, **y**), a pair of trains q_1 and q_2 traveling in the opposite directions, and segment m. Specifically, Step 1 of the procedure finds the set of time period that should be considered. Step 2 of the procedure goes over each possible pair of time periods to find violated inequalities by checking condition (1) of Lemma 3.6. Similarly, Step 3 finds violated inequalities by checking condition (2) of Lemma 3.6.

`Procedure separate_unidirectional_across_segment`
Input: solution vector **x** and **y**, train q_1, train q_2 and segment m
Output: C, set of inequalities (3.31) violated by **x**.
Step 1: $C = \phi$;
let s be the station such that $m = e_s^q$;
$T(q_1) = \{t(q_1, 1), \cdots, t(q_1, K)\} \subseteq T$, where $x^{q_i}_{m,t(q,k)} > 0$ or $y^{q_i}_{s,t(q,k)} > 0$;
Step 2: For $i = 1$ to k
For $j = i$ to k
$t = t(q_1, i)$ and $\bar{t} = t(q_1, j)$;
$t_1 = \bar{t} - \tau_m^{q_1} - \theta + 1$ and $\bar{t}_1 = t + \tau_m^q + \theta - 1$;
$\bar{t}_2 = \text{Min}\{t_1 + \tau_{m_1}^{q_1} + \tau_{s_1}^{q_1} - 1, t - \tau_{s_1}^q + \theta - 1\}$
$t_2 = \bar{t} - \tau_{s_1}^q - \tau_{m_2}^q - \theta - \tau_{m_2}^{q_1} + 1$;
If $\sum_{t'=t}^{\bar{t}} x^q_{m_1 t'} - y^q_{s_1(t-1)} + \sum_{t'=t_1}^{\bar{t}_1} x^{q_1}_{m_1 t'} + \sum_{t'=t_2}^{\bar{t}_2} x^{q_1}_{m_2 t'} > 1$
add $(t_1, \bar{t}_1, t_2, \bar{t}_2)$ to C
Step 3: For $i = 1$ to k
For $j = i$ to k
$\bar{t}_2 = t - \tau_{s_1}^q + \theta - 1$ and $t_2 = \bar{t} - \tau_{s_1}^q - \tau_{m_2}^q - \theta - \tau_{m_2}^{q_1} + 1$;
$t_1 = t + \tau_m^q + \theta - 1$ and $t_1 = \text{Max}\{\bar{t}_2 - \tau_{m_1}^{q_1} - \tau_{s_1}^{q_1} + 1, \bar{t} - \tau_m^{q_1} - \theta + 1\}$;
If $\sum_{t'=t}^{\bar{t}} x^q_{m_1 t'} - y^q_{s_1(t-1)} + \sum_{t'=t_1}^{\bar{t}_1} x^{q_1}_{m_1 t'} + \sum_{t'=t_2}^{\bar{t}_2} x^{q_1}_{m_2 t'} > 1$
add $(t_1, \bar{t}_1, t_2, \bar{t}_2)$ to C

Figure 40. Procedure to separate unidirectional inequalities (3.31)

APPENDIX B. COMPLEXITY OF THE GRASP FOR RRDP

In this section, we will show that the complexity of our GRASP is polynomial in the size of the underlying graph for a fixed runtime or fixed number of iterations. In particular, we assume that the number of GRASP replications is M and that the number of tabu iteration is N.

B.1. Complexity of phase I

To analyze the complexity of solution construction, we start with the work required to generate the seed stores, as given in Figure 27. Step 1 requires using the lower bounding technique in Martello and Toth(1990) and thus has complexity $O(|S|^2)$. The complexity of Step 2 is $O(|S|)$. The third step requires $O(|S|)$ time to find a store whose delivery volume is larger than *vcap*, and $O(|S|^2)$ time to initialize $t(s)$ for each $s \in S \setminus S_E$. Step 4 requires at most $O(|S|)$ iterations, where each iteration involves no more than $O(|S|)$ operations, implying that its complexity is $O(|S|^2)$. In summary, forming the set S_E takes $O(|S|^2)$ time. Since the number of orders in each store is not large, we can assume that the time to construct a seed route for a seed store is $O(1)$. Accordingly, the seed route construction procedure runs in $O(|S|^2)$ time.

With respect to the solution generation procedure in Figure 30, we first consider the total number of routes that might be needed. In Step 6, if the number of routes generated is greater than the current number of seed routes, the procedure is reinitialized using the increased number of seed routes. The upper bound on the number of seed routes is $O(|O|)$, which means that the maximum number of solutions that can be generated is $O(|O|)$.

Now let us consider the work required to construct each solution. The complexity of generating seed routes in Step 1 is $O(|S|^2)$, as derived above. The complexity of initializing the opportunity cost in Step 1 is $O(|A|\cdot|O|)$ since it must

consider all arcs in the seed routes and unscheduled node. Given that at most $|O|$ new routes will be created in Step 2, its complexity is $O(|O|)$.

In Steps 3 to 5, an unscheduled node is identified and inserted it into an existing route, where the number of candidate nodes is at most $|O|$. In Step 3, the effort to find the min-cost route is $O(|\Omega|)$ as is the effort to find the opportunity cost Π_k for node k. Because $|\Omega| \leq |O|$, the complexity of Step 3 is $O(|O|)$. The sorting in Step 4 requires $O(|O| \log_2|O|)$ time and all other operations can be performed in $O(1)$ time. Finally, updating the opportunity cost in Step 5 takes at most $O(|A| \cdot |O|)$ time for each iteration.

In putting these results together, we note that $O(|O|) \geq O(|S|)$ since each store has at least one order node, and $O(|A|) \geq O(|O|)$ since the graph is connected. Thus, the total complexity of the algorithm is $O(|O|) \times [O(|S|^2) + O(|A| \cdot |O|) + O(|O|) + O(|O|) \times (O(|O|) + O(|O| \log_2|O|) + O(|A| \cdot |O|))$, or equivalently, $O(|A| \cdot |O|^3)$.

B.2. Complexity of Tabu Search-RVN

Here we only analyze the complexity of Tabu search-RVN, which is the local search method we call in phase II of our GRASP. To begin, we note that the number of transition arcs used in any given solution Ω is bounded by $2|O|$, since at most one arc enters an order node and one arc exits. This observation is used to determine the complexity of neighborhood search. During this process, it may also be necessary to call the route reduction algorithm. In the worst case, we need to check if the nodes on a route ρ can be moved to another route by inserting them, one at a time, into an arc in the existing solution $\Omega \setminus \{\rho\}$. The complexity of route reduction is then $O(N \cdot |O|^2)$.

The work required to explore each of the four tabu neighborhoods is as follows:

- For RI[k, (i, j), g_k, g_i], it is necessary to evaluate all possible combinations of nodes k and arcs (i, j) included in the solution Ω to find a best insert position. The corresponding complexity is $O(|O|^2)$.
- For S[i, j, g_i, g_j], it is necessary to examine all pairs of nodes i and j to find the best swap. This can be done in $O(|O|^2)$ time.
- For RRI[k, i, (j, l), g_k, g_i, g_j], it is necessary to consider all possible combinations of nodes k and i and arcs (j, l) in the solution Ω. This has complexity $O(|O|^3)$.
- For SA[(i, j), (k, l), g_i, g_k], it is necessary to evaluate all possible arc pairs (i, j) and (k, l) in the solution Ω, which requires $O(|O|^2)$ time.

In the worst case, each iteration selects RRI[k, i, (j, l), g_k, g_i, g_j], the neighborhood with the highest complexity given by $O(|O|^3)$. Thus, the total complexity of the local search algorithm is $O(N \cdot |O|^3)$.

B.3. Complexity of the GRASP

Combining the above results, we have that the total complexity of each GRASP replication is $O(N \cdot |O|^3) + O(|A| \cdot |O|^3)$. Because it is typically the case that $N > |A|$, this reduces to $O(N \cdot |O|^3)$. Considering that M tabu search iterations are performed for each feasible solution found in phase I, the total complexity of the GRASP is $O(M \cdot N \cdot |O|^3)$.

Bibliography

Agarwal, Y., and Aneja, Y. (2012). Fixed-charge transportation problem: Facets of the projection polyhedron. *Operations research*, *60*(3), 638-654.

Ahuja, R. K., Magnanti, T. L., Orlin, J. B. (1993). *Network flows: theory, algorithms, and applications*. Prentice Hall, Englewood Cliffs, NJ.

Ahuja, R. K., Orlin, J. B., and Sharma, D. (2000). Very large-scale neighborhood search. *International Transactions in Operational Research*, *7*(4-5), 301-317.

Azi, N., Gendreau, M. and Potvin, J. Y. (2010). An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles. *European Journal of Operational Research*, *202*(3), 756-763.

Balakrishnan, A., and Graves, S. C. (1989). A composite algorithm for a concave-cost network flow problem. *Networks*, *19*(2), 175-202.

Balakrishnan, A., and Altinkemer, K. (1992). Using a hop-constrained model to generate alternative communication network design. *ORSA Journal on Computing*, *4*(2), 192-205.

Balakrishnan, A., Li, G., and Mirchandani, P. (2014). Optimal network design with end-to-end service requirement. *Department of Information, Risk, and Operations Managemetn, The University of Texas at Austin, Austin*.

Balakrishnan, A., Magnanti, T. L., and Mirchandani, P. (1997). Network design. *Annotated bibliographies in combinatorial optimization*, 311-334.

Baldacci, R., Mingozzi, A., and Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, *218*(1), 1-6.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations research*, *46(3)*, 316-329.

Bent, R. and Hentenryck, P. V. (2006). A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, *33*(4), 875-893.

Berbeglia, G., Cordeau, J. F., Gribkovskaia, I., and Laporte, G. (2007). Static pickup and delivery problems: a classification scheme and survey. *Top*, *15*(1), 1-31.

Bianchessi, N., and Righini, G. (2007). Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. *Computers & Operations Research*, *34*(2), 578-594.

Brännlund, U., Lindberg, P. O., Nõu, A., and Nilsson, J. (1998). Railway timetabling using Lagrangian relaxation. *Transportation Science*, 32(4), 358-369.

Bräysy, O., and Gendreau, M. (2005a). Vehicle routing problem with time windows, Part I: Route construction and local search algorithms. *Transportation science*, *39*(1), 104-118.

Bräysy, O., and Gendreau, M. (2005b). Vehicle routing problem with time windows, Part II: Metaheuristics. *Transportation science*, *39*(1), 119-139.

Cacchiani, V., Caprara, A., and Toth, P. (2008). A column generation approach to train timetabling on a corridor. *A Quality Journal of Operations Research*, *6*(2), 125-142.

Cacchiani, V., Caprara, A., and Toth, P. (2010). Scheduling extra freight trains on railway networks. *Transportation Research Part B: Methodological*, *44*(2), 215-231.

Cai, X., Goh, C., and Mees, A. I. (1998). Greedy heuristics for rapid scheduling of trains on a single track. *IIE transactions*, *30*(5), 481-493.

Caprara, A., Fischetti, M, and Toth, P. (2002). Modeling and solving the train timetabling problem. *Operations Research*, *50*(5), 851-861.

Caprara, A., Monaci, M., Toth, P., and Guida, P. L. (2006). A Lagrangian heuristic algorithm for a real-world train timetabling problem. *Discrete Applied Mathematics*, *154*(5), 738-753.

Carey, M. (1994). Extending a train pathing model from one-way to two-way track. *Transportation Research Part B: Methodological*, *28*(5), 395-400.

Carey, M., and Lockwood, D. (1995). A model, algorithms and strategy for train pathing. *Journal of the Operational Research Society*, *46*(8), 988-1005.

Carlyle, W. M., Royset, J. O, and Wood, R. K. (2008). Lagrangian relaxation and enumeration for solving constrained shortest-path problems. *Networks*, *52*(4), 256-270.

Carreto, C., and Baker, B. (2001). An improved GRASP interactive approach for the vehicle routing problem with backhauls. *Essays and Surveys in Metaheuristics*, 185-199.

Cordeau, J. F., Toth, P, and Vigo, D. (1998). A survey of optimization models for train routing and scheduling. *Transportation science*, *32*(4), 380-404.

Corman, F., D'Ariano, A., Pacciarelli, D., and Pranzo, M. (2010). A tabu search algorithm for rerouting trains during rail operations. *Transportation Research Part B: Methodological*, *44*(1), 175-192.

Dahl, G., and Gouveia, L. (2004). On the directed hop-constrained shortest path problem. *Operations Research Letters*, *32*(1), 15-22.

Deng, Y., and Bard, J. F. (2011). A reactive GRASP with path relinking for capacitated clustering. *Journal of Heuristics*, *17*(2), 119-152.

Desaulniers, G., Lessard, F., and Hadjar A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, *42*(3), 387-404.

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.

Dorfman, M. J., and Medanic, J. (2004). Scheduling trains on a railway network using a discrete event model of railway traffic. *Transportation Research Part B: Methodological*, *38*(1), 81-98.

Dumitrescu, I., and Boland, N. (2003). Improved preprocessing, labeling and scaling algorithms for the Weight-Constrained Shortest Path Problem. *Networks*, *42*(3), 135-153.

Ergun, Ö., Orlin, J. B., and Steele-Feldman, A. (2006). Creating very large scale neighborhoods out of smaller ones by compounding moves. *Journal of Heuristics*, *12*(1-2), 115-140.

Festa, P., and Resende, M. G. (2009). An annotated bibliography of GRASP–Part II: Applications. *International Transactions in Operational Research*, *16*(2), 131-172.

GE Reports. (2010). *RailEdge tech: Faster, smarter trains to save millions*. http://www.gereports.com/railedge-tech-faster-smarter-trains-to-save-millions/

Goksal, F. P., Karaoglan, I., and Altiparmak, F. (2013). A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery. *Computers & Industrial Engineering*, *65*(1), 39-53.

Golden, B. L., Raghavan, S., and Wasil, E. A. (2008). *The Vehicle Routing Problem: Latest Advances and New Challenges: latest advances and new challenges*. Springer, New York.

Harrod, S. (2011). Modeling network transition constraints with hypergraphs. *Transportation Science*, *45*(1), 81-97.

Higgins, A., Kozan, E., and Ferreira, L. (1996). Optimal scheduling of trains on a single line track. *Transportation Research Part B: Methodological*, *30*(2), 147-161.

Holmberg, K., and Yuan, D. (2003). A multicommodity network-flow problem with side constraints on paths solved by column generation. *INFORMS Journal on Computing*, *15*(1), 42-57.

Kallehauge, B. (2008). Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, *35*(7), 2307-2330.

Kittelson and Associates. (2003). *Transit capacity and quality of service manual*. U.S. Department of Transportation, Washington DC.

Kohl, N., Desrosiers, J., Madsen, O. B., Solomon, M. M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, *33*(1), 101-116.

Kontoravdis, G. and Bard, J. F. (1995). A GRASP for the vehicle routing problem with time windows. *ORSA journal on Computing*, *7*(1), 10-23.

Kytöjoki, J., Nuortio, T., Bräysy, O., and Gendreau, M. (2007). An efficient variable neighborhood search heuristic for very large scale vehicle routing problems. *Computers & Operations Research*, *34*(9), 2743-2757.

Lau, H. C., Sim, M., and Teo, K. M. (2003). Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research*, *148*(3), 559-569.

Lusby, R. M., Larsen, J., Ehrgott, M., and Ryan, D. (2011). Railway track allocation: models and methods. *OR spectrum*, *33*(4), 843-883.

Lysgaard, J., Letchford, A. N., and Eglese , R. W.(2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, *100*(2), 423-445.

Magnanti, T. L., and Wong, R. T.. Network design and transportation planning: Models and algorithms. *Transportation Science*, *18*(1), 1-55.

Mu, S., and Dessouky, M. (2011). Scheduling freight trains traveling on complex networks. *Transportation Research Part B: Methodological*, *45*(7), 1103-1123.

Nagata, Y., Bräysy, O., and Dullaert, W. (2010). A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, *37*(4), 724-737.

Nagy, G., Wassan, N. A., Speranza, M. G., and Archetti, C. (2013). The vehicle routing problem with divisible deliveries and pickups. *Transportation Science*, *49*(2), 271-294.

Nguyen, V. P., Prins, C., and Prodhon, C. (2012). Solving the two-echelon location routing problem by a GRASP reinforced by a learning process and path relinking. *European Journal of Operational Research*, *216*(1), 113-126.

Nowak, M., Ergun, Ö., and White III, C. C. (2008). Pickup and delivery with split loads. *Transportation Science*, *42*(1), 32-43.

Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2008a). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, *58*(1), 21-51.

Parragh, S. N., Doerner, K. F., and Hartl, R. F. (2008b). A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, *58*(1), 81-117.

Penna, P. H. V., Subramanian, A., and Ochi, L. S. (2013). An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. *Journal of Heuristics*, *19*(2), 201-232.

Pirkul, H., and Soni, S. (2003). New formulations and solution procedures for the hop constrained network design problem. *European Journal of Operational Research*, *148*(1), 126-140.

Pisinger, D., and Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & operations research*, *34*(8), 2403-2435.

Prescott-Gagnon, E., Desaulniers, G., and Rousseau, L. M. (2009). A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, *54*(4), 190-204.

Pugliese, L. D. P., and Guerriero, F. (2013). A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, *62*(3), 183-200.

Randazzo, C. D., and Luna, H.P. L. (2001). A comparison of optimal methods for local access uncapacitated network design. *Annals of Operations Research*, *106*(1-4), 263-286.

Righini, G., and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, *51*(3), 155-170.

Rochat, Y., and Semet, F. (1994). A tabu search approach for delivering pet food and flour in Switzerland. *Journal of the Operational Research Society*, 1233-1246.

Ropke, S., and Pisinger, D. (2006). A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, *171*(3), 750-775.

Şahin, G., Ahuja, R. K., , and Cunha, C. B. (2008). Integer programming based approaches for the train dispatching problem. *Department of Industrial and Systems Engineering, University of Florida, Gainesville*.

Şahin, İ. (1999). Railway traffic control and train scheduling based oninter-train conflict management. *Transportation Research Part B: Methodological*, *33*(7), 511-534.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, *35*(2), 254-265.

Solomon, M. M. and Desrosiers, J. (1988). Time Window Constrained Routing and Scheduling Problems. *Transportation science*, *22*(1), 1-13.

Taillard, É. (1993). Parallel iterative search methods for vehicle routing problems. *Networks*, *23*(8), 661-673.

Tasan, A. S., and Gen, M. (2012). A genetic algorithm based approach to vehicle routing problem with simultaneous pick-up and deliveries. *Computers & Industrial Engineering*, *62*(3), 755-761.

Thangiah, S. R., Potvin, J. Y., and Sun, T. (1996). Heuristic approaches to vehicle routing with backhauls and time windows. *Computers & Operations Research*, *23*(11), 1043-1057.

Toth, P., and Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *Informs Journal on computing*, *15*(4), 333-346.

Törnquist, J. (2006). Computer-based decision support for railway traffic scheduling and dispatching: A review of models and algorithms. In *OASIcs-OpenAccess Series in Informatics* (Vol. 2). Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, *40*(1), 475-489.

Zhou, X., and Zhong, M. (2007). Single-track train timetabling with guaranteed optimality: Branch-and-bound algorithms with enhanced lower bounds. *Transportation Research Part B: Methodological*, *41*(3), 320-341.