The Dissertation Committee for Kursad Derinkuyu certifies that this is the approved version of the following dissertation:

Optimization Models for Transport and Service Scheduling

Committee:

Anant Balakrishnan, Supervisor

Erhan Kutanoglu, Supervisor

Jonathan F. Bard

David P. Morton

S. Travis Waller

Optimization Models for Transport and Service Scheduling

by

Kursad Derinkuyu, B.Sc., M.Sc.

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

May 2011

To my family…

Acknowledgements

I would like to express my sincere gratitude to my advisors, Dr. Anant Balakrishnan and Dr. Erhan Kutanoglu, whose encouragement, guidance, and unconditional support to overcome many personal and professional challenges made this dissertation possible. I am also grateful to the members of my dissertation committee Dr. Jonathan Bard, Dr. David Morton and Dr. Travis Waller for their valuable comments and suggestions. I also would like to thank Dr. Mustafa Pinar of Bilkent University for encouraging me to pursue an academic career.

I am also thankful to my friends in Austin and Bethlehem for supporting me throughout my graduate life. I would like to thank Emrah Zarifoglu, Ali Koc, Emrah Tanriverdi, Fehmi, Burak, Murat, Ferhat, Rasim, Oguz, Ali Pilatin, Gokhan, Ahmet Ali, Bulent, Firat, Adem, Serdal, Tayfun and numerous others.

Last but the most important, I would like to express my deepest gratitude to my wife for her trust and confidence, and to my daughter for bringing light to my life. I would like to thank all the people in my family for their endless support.

Optimization Models for Transport and Service Scheduling

Publication No._____________

Kursad Derinkuyu, Ph.D.

The University of Texas at Austin, 2011

Supervisors: Anant Balakrishnan

Erhan Kutanoglu

This dissertation focuses on service scheduling and transshipment problems. The study of service scheduling is motivated by decisions facing service planners, who must inspect and maintain geographically dispersed infrastructure facilities. We study the problem of deciding which operations a service unit must perform at each customer location, given the sequence in which the unit periodically visits these locations. Each customer requires multiple service operations, and each operation has a time-varying completion or penalty cost that depends on the previous service time. The goal is to schedule the service start time for each customer and select the operations to perform so as to minimize the total completion cost.

We first discuss how to solve a special case of this problem in which each site is visited only once per service cycle. We formulate this problem as a discrete time indexed network flow problem and prove that it is NP-hard in the ordinary sense. Then, we represent the problem as a multidimensional shortest path problem with path-dependent arc lengths. In this structure, arc costs depend on the total time spent for all customers.

The resulting formulation is solvable via algorithms that have pseudo-polynomial run times. Computational results show that the shortest path approach outperformed the general network flow model.

We then analyze the general case of this problem, in which each site can be visited more than once and prove that the problem is *NP*-Hard in the strong sense. We discuss the valid cuts and describe the preprocessor that reduces the problem size. Next, we examine an application to the general case of the problem and develop a fast and effective heuristic procedure that repeatedly applies the shortest path approach to subsequences that do not visit any customer more than once. Computational results for several problem instances show that the proposed heuristic identifies near optimal results very quickly, whereas a general purpose integer-programming solver (CPLEX) is not able to find an optimal solution even after many hours of computational time. Then we focus on techniques such as problem reduction, branching variables, and subdividing problem to smaller problems to get better solution times for the actual problem. Computational results show that these techniques can improve solution times substantially.

Finally, we study a transshipment problem, in which the shipments need to be transported from their origin to destination and are subject to the logical and physical transportation network on which they rely. We consider a space-time network that allows one to formulate the problem as a multi-commodity network flow problem with additional side constraints and show the complexity results. We propose alternative models and propose algorithms for lower and upper bound calculations.

Table of Contents

List of Figures xii

List of Tables

List of Figures

Chapter 1: Introduction

1.1 MOTIVATION

Worldwide competition has been forcing companies to provide better services to their customers. Better services not only increase customer satisfaction but also help companies manage resources effectively. In fact, service management operations are complex processes, and in many businesses, quality of the service depends on a timely response to service needs. The focus of this dissertation is problems with service management under the following service types:

- **One-time required services:** Servicing a number of customers where each customer is considered one at a time. Here the cost of service depends on the amount of time spent with each customer. This type of service usually appears when a service plan is put in effect only after the orders are received, as in after-market and emergency services.
- **Multiple-time required services:** Servicing a number of customers periodically. Each time a customer gets a service, the business incurs a servicing cost that depends on the time since the customer's last service. Services of this type appear in many contexts, such as multi-product lot sizing, vendor managed inventories, machine maintenance and several problems in telecommunications.
- **Transportation services:** Transportation of shipments from their origin to destination under capacity and system constraints. The objective function minimizes the weighted transit times of shipments and/or the cost of used paths. This is a common problem for freight transportation operations as well as applicable to communication networks.

Each of these services requires multi-level hierarchical decisions that lead to difficult problems of resource allocation and scheduling. We will now explain these problems in detail. However, the scope of this study is limited to the operational level problems of each of these service types.

In a typical service management scenario, each customer requires services or *operations*. Although the service provider can choose more than one operation (*opportunistic scheduling*) to perform for each customer, some of the operations cannot be done together for technical reasons. Among the set of various possible combinations of operations, a *task* refers to a combination of *operations* that can be performed together.

One-time required services: In this section, we consider the problem of servicing a set of customers with a service constraint. Namely, only one customer can be serviced at a time. We also assume that the service order of the customers is known. In this situation, resource allocation and order/routing problems could be seen as upper level decisions. At the *strategic level*, the company needs to decide how many resources are needed and how the resources are allocated to customers. The *tactical level* problem aims to solve the routing/ordering problems of customers for each resource. Finally, our problem is an *operational level* problem and entails deciding which among the possible tasks to perform for each customer on a given order. The solution to this problem specifies: (i) what tasks the resource should perform in the given order, and (ii) when the resource performs these tasks.

In this problem, we assume that each customer will appear only once (one-time requirement). Hence, the problem could be seen as a special case of multiple requirement case. However, there are some applications directly related to this service type. The following examples point to this type of service:

- *Roadway snow and ice control*: The streets need clearing after snowy or icy weather. There are some operations to clean streets, such as snowplowing and gritting. Here, the roads/streets are customers and servicing a street means performing winter gritting operations. In this problem, the timing of intervention is of prime importance. That is, if the intervention is too early or too late, the cost sharply increases.

- *After-market repair services*: Consider a case where a service provider is required to repair the product whenever the customer calls. Here, servicing a customer means maintaining the utility of the product. A timely, high-quality response to the service need is a critical element for customer satisfaction. After-market service maintains the utility of the product and helps to increase customer loyalty to the company. Since the service is usually provided by a contract, the quality of this service directly affects the company's profit.

Multiple-time required (periodic) services: This problem is a generalization of the previous one, where a customer can appear multiple times. Again, our study focuses on the operational level problem where customers get service one at a time and the customer order is known. In this problem, the servicing cost depends on the time since the customer's last service. There are several motivating applications related to this service type:

- *Multi-product lot sizing*: The manufacturing plan consists of a cyclic schedule that specifies the sequence in which each product family is produced. In this problem, there are product families, and in each family, there are individual items. The problem of deciding how much of each item in a product family to produce for a given cyclic sequence of family setups is analogous to the problem we are studying. Here, the customers represent the product families, and servicing a customer corresponds to ordering a subset of items belonging to one family or replenishing the subset of the inventory of an item belonging to one family. The cost of servicing a product family may include a fixed ordering cost, inventory carrying charges for the items over the interval until the next service, and possibly shortage cost in case the demand exceeds the production before the next service.

- *Preventive machine maintenance*: In this context, there are machines that require periodic maintenance. Here, the machines are the customers. Each machine may have several parts (operations) to be maintained, and a subset of these parts (a task) can be maintained together. In the maintenance problem, the cost of servicing a machine increases up to its next service, whereas in multi-product lot sizing, inventory cost decreases up to its next reorder point.
- *Vendor managed inventory systems*: Vendor managed service refers to a situation in which a supplier replenishes the inventory of its customers. In these systems, servicing a customer means replenishing its inventory. It is clear that the cost depends on the inventory status of customers, and on-time response is an important element for customer satisfaction. This problem appears in many sectors, such as the petrochemical industry, industrial gas industry, automotive (parts distribution) industry and soft drink (vending machines) industry.
- *Telecommunication services*: There are several applications in the telecommunication sector closely related to our problem. Bar-Noy et al. (2002) present many examples, and we describe one of them here. In broadcast disk application, a database contains a number of pages and broadcasts a limited number of these pages at each period. A client who wishes to access any page must wait for the broadcasting time of that page. Here, the customers represent the pages, and servicing a customer corresponds to the broadcasting of pages. This application aims to minimize the expected time spent by clients waiting to access the pages.

Transportation services: Transportation systems are complex dynamic processes and require the management of multiple resources in order to serve customers. We focus on the problem of shipments needing to be transported from their origin to their destination under the limitations of the underlying transportation network. We assume that the higher level decisions on the network, such as capital investment and the schedule of carriers, are given. The objective function minimizes the weighted transit times of shipments and/or the cost of used paths/resources.

In the transportation scheduling problem, the scheduler makes an enormous number of interrelated decisions on strategic, tactical and operational levels. Strategic level decisions involve capital investments, such as getting new planes, trucks, trains or ships, and expanding the transportation network. At the tactical level, we need to schedule carriers/transporters and make maintenance plans. The operational level problems aim to solve short term planning, such as trip planning of shipments from their origin to destination.

A moderate size transportation company transports thousands of shipments on their network everyday. The system managers should ensure that the shipments are getting an appropriate level of service at a low operating cost. At the operational level, the system should solve such big problems very fast and make decisions immediately. In a normal day, new shipments come to the network, and the state of the system changes over time. Clearly, the effective usage of available capacity is a key element of success. A good decision does not only consider the current information, but also considers the possible uncertainties, such as future shipments that may require the resources currently being used by other shipments.

Transportation service problems also appear in communication networks. In those networks, data packages (as shipments) should be transported from their origin to destination by using underlying multi-layer transport network. In these networks, there are (physical) fiber links that make the actual connections at the bottom layer. Then, above layers are logically designed to handle data packages appropriately. Our problem could be seen as trip planning of these data packages using underlying communication network.

1.2 OUTLINE

Our aim in this dissertation is to examine single and multi-visit service scheduling problems, and transshipment problems. We analyze the complexity properties, develop models and methodologies for these problems and demonstrate their performance on an application.

The dissertation is organized as follows: In Chapter 2, we present the first service scheduling problem, where each customer is visited only once. In this chapter, we formulate the problem as a discrete time indexed network flow. We analyze the computational complexity of this problem and show that the problem is NP-Hard in the ordinary sense. Then, we concentrate on several special case structures of this service scheduling problem and determine their complexity properties. We propose an alternative formulation for the problem as a shortest path problem with path dependent arc lengths. The resulting formulation is solvable via algorithms that have pseudo-polynomial run times. Our study shows that the formulations are equivalent, so the shortest path approach solves the problem optimally. Finally, computational efforts imply that the proposed shortest path formulation outperformed the general network flow formulation on randomly generated test cases.

In Chapter 3, we focus on the second service scheduling problem where each customer may be visited multiple times. In fact, this problem is an extension of the first one and we formulate this periodic service problem as a network flow problem. We prove that the problem is NP-Hard in the strong sense and no pseudo-polynomial algorithm is available to solve this problem. Furthermore, a performance guaranteed heuristic with (pseudo-)polynomial run time is not possible. We also analyze the several special case structures and show their complexity properties.

In Chapter 4, we focus on the application of the multi visit service problem motivated by the actual problem facing maintenance planners at a large company. This application is based on infrastructure facilities that require periodic inspection and maintenance to ensure uninterrupted service and effective operation. These facilities are geographically dispersed, and the inspection and maintenance operations require on-site visits by a "service" unit, consisting of skilled workers and equipment. At each site, several components need to be serviced; the desired frequency of service varies by component and facility, depending on the location of the facility, its usage, and other factors. Scheduling the service tasks associated with these inspection and maintenance activities is an important and challenging problem facing firms that operate infrastructure facilities. We formulate the problem and develop a fast and effective heuristic procedure. The heuristic is based on the shortest path approach developed in Chapter 2. We apply the shortest path approach repeatedly for the subsequences that do not contain any customer twice. We come up with problem size reduction techniques, and determine several branching strategies to solve actual problems effectively. Finally, we introduce a technique for dividing the original problem into sub-problems, so that each of them could be solved much faster. We compare these techniques and provide computational results for this application.

In Chapter 5, we concentrate on the transshipment problem and give two mathematical formulations. To improve computational performance, we develop three sets of inequalities. We show that none of these formulations is stronger than the other one and test them on the same problem instance. We analyze the computational complexity of this problem and its several special cases. We propose three heuristics for calculating good upper bound for the original problem. We also construct a lower bound calculation based on a shortest path solution. We compare these approaches on the same problem instance.

Finally, we summarize our contribution in Chapter 6, and discuss future research directions.

Chapter 2: Task Assignment Problem

2.1 INTRODUCTION

This chapter focuses on the problem of servicing a number of customers in a discrete time environment. We consider a service scheduling problem in which each customer requires services or *operations,* and we assume that each operation has a time-varying completion cost. Although the service provider can choose more than one operation to perform for each customer, some of the operations cannot be performed together for technical reasons. The problem in question consists of assigning a *task*—or combination of operations—to each customer while minimizing the general cost function. We refer to this problem as a *task assignment* (*TA*) *problem*.

We make the following assumptions to facilitate model development:

- We assume that one customer can be serviced at a time. Customers are allocated to resources whenever higher (strategic) level decisions are made. Then, the decision makers concentrate on each resource and its assigned customers. However, there are some cases in which the "one customer at a time" rule may be present because of accounting, physical space, workforce, or transportation considerations. For instance, in a machine maintenance context, maintaining more than one machine at a time may cause serious interruptions in the systems that depend on these machines.
- We also assume that the order/route of the customer is fixed. Routing/ordering decisions are intermediate (tactical) level decisions and a fixed route/order can be a candidate solution for the tactical level problem. In some applications, the order of the customers may come out naturally. A fixed customer sequence may appear

in real life when the services are handled on a first come first serve basis or when the customers are located along an interstate highway.

- We assume that each customer appears only once in the sequence. The case where customers appear more than once is an extension of this study, and we discuss that problem in the next chapter. However, there are some applications where each customer is considered once. In those business processes, a service plan takes place only after the orders are received, as in emergency situations and corrective maintenance.
- Finally, we assume that no partial service is allowed (no preemption). This is a natural constraint in many applications where interruptions seriously affect the quality of the service.

We consider the operational level task assignment problem, which assigns one task to each customer and needs the following items as input:

- a customer sequence,
- the possible tasks for each customer,
- the processing time for each possible task for each customer,
- the cost function and the time window for each operation.

Given these assumptions, the planner determines which task among the possible choices to perform for each customer. We require a plan that visits all the customers in the given sequence while minimizing the general cost function for all operations.

What makes this task assignment problem unique is that the each customer requires multiple services with different time windows and general cost functions. In the task assignment problem, an assignment chosen for one customer may affect the feasibility of assignments for other customers. Furthermore, the cost of the service for any customer depends on not only the duration spent on earlier customers but the duration that will be spent on later customers because of the structure of the cost functions.

In this chapter, we make several contributions to the task assignment problem. First, we analyze several fundamental properties of the task assignment problem. We prove that the problem is *NP*-Hard and show the computational complexity of some special cases. Then, we approach the problem two different ways. The first approach formulates the problem as a discrete time indexed network flow problem and solves the problem by using the commercial software CPLEX. The second approach represents the problem as a multidimensional shortest path problem with path-dependent arc lengths. In this structure, arc lengths depend on the total time spent on all the customers. We convert our problem to the shortest path problem by considering its special network structure. The resulting formulation is solvable via algorithms that have pseudo-polynomial run times. We also compare the computational effort required by these two approaches based on randomly generated test cases. As a result of these computations, we show that the shortest path approach outperformed the general network flow model.

The remaining part of the chapter is organized as follows: Section 2 gives the related literature review, and Section 3 formulates the problem as a discrete time indexed network flow. In Section 4, we prove that the task assignment problem is *NP*-Hard, and Section 5 concentrates on the special case structures. In Section 6, we develop the modified shortest path approach and adopt the well-known shortest path algorithms to our problem. The computational results of these two approaches based on randomly generated data are reported in Section 7. Finally, we offer a conclusion and discuss future extensions in Section 8.

2.2 LITERATURE REVIEW

In service management problems, some services take place only after the orders are received, and some other services require periodic executions. We will discuss the first type of services here and periodic services in the next chapter. In the case where a service takes place only after the orders are received, the customers usually get a service only once, as in emergency services and corrective maintenance. Many papers in the literature also consider routing as a part of the decision process whereas the task assignment problem has a fixed customer sequence.

One emergency service problem is winter gritting operations, where the timing of an intervention is of prime importance (Campbell and Langevin, 2000; Li and Eglese, 1996) Tagmouti et al. (2007) study an arc routing problem with capacity constraints and time-dependent service costs. This problem is motivated by winter gritting applications, where a subset of arcs must be serviced at a cost that depends on the timing of the service. Here, the streets are the customers, and servicing a customer means performing winter gritting operations. There is a single operation required for each street, and routing is a part of the decision process in Tagmouti et al.'s paper. The authors report the exact problem-solving approach that first transforms the arc routing problem into an equivalent node routing problem. Then, a column generation scheme is used to solve the latter. The resulting node routing problem is a vehicle routing problem with time-dependent service costs. To the best of our knowledge, Tagmouti et al. is the only work that deals with time-dependent service costs in the arc routing literature, although some variants of the vehicle routing problem with time windows may be related to it (Ibaraki, et al. 2005; Ioachim, et al. 1998; Taillard, 1997). Desrosiers et al. (1995) provide a good review of time-constrained vehicle routing problems.

Similar applications are municipal waste (Stricker, 1970), waste collection (Beltrami and Bodin, 1974; Bodin, 1990), sanitation operations (Riccio, 1984; Riccio and Litke, 1986; Ball, 1988) and postal delivery (Bodin, Fagan and Levy, 1992). Bodin and Kursh (1978) study street sweeping, and Levy and Bodin (1988) concentrate on postal delivery. An excellent survey of these applications can be found in Assad and Golden (1995). In these papers, the process times for all the operations are equal. Therefore, the researchers concentrate on the routing decisions.

Single-visit services are also present in a corrective maintenance context. Consider a firm providing repair services for a certain type of equipment over some area. Typically, the area is divided into service territories, and in each territory one repairman (server or service representative) is responsible for the repair and maintenance. Here, the machines (equipment) are the customers, and servicing customers means maintaining machines. According to Agnihothri and Karmarkar (1992), the customer calls are serviced according to an FCFS (first-come, first-serve) dispatching policy, so the routing decision is given by default. This so-called *machine repairman* (*or interference*) *problem* gets especial attention in the queuing literature. For instance, Agnihothri and Karmarkar (1992), and Jamil et al. (1994) use queuing models to work on approximating the waiting times for repair services under given probability distributions of equipment failures and the FCFS rule. Here, waiting time can be seen as a service time dependent cost function. Almost all of the related papers in the queuing area have the FCFS rule, but they consider different failure distributions or service availability. Excellent surveys can be found in Stecke and Aronson (1985), and Haque and Armstrong (2007). There are also a few papers that consider the machine repairman problem without any stochastic information about the data. In Abdekhodaee et al. (2006) and Koulamas (1996), there are two parallel machines with a single repairman who is required for setup. The machines have to wait for the repairman before processing any task. These papers define the problem under various objective functions, such as makespan and total completion time, and propose heuristics that run in polynomial time.

Finally, Armstrong et al. (2008) study a problem with a single transporter and a fixed sequence of customers. The production facility has a limited production rate, and the delivery truck has non-negligible traveling times between locations. Each customer requests a delivery quantity and a time window for receiving the delivery. The problem chooses a subset of customers from the given sequence to receive the deliveries in order to maximize the total demand satisfied. Here, servicing customers corresponds to delivering the order to the customer. The problem has a single operation (customer order) for each customer, and the decision maker decides which customer will get service. The problem batches the customer orders before the shipment, whereas there is no batching in the task assignment problem. Armstrong et al. (2008) propose a heuristic, and branch and bound procedure for practical problems.

There are also related problems in preventive maintenance and the multi-item replenishment context. By their nature, these services are required periodically. Therefore, we will discuss them in the next chapter.

2.3 PROBLEM DEFINITION

This study concerns an operational level problem in service management. In particular, we consider a service facility with a single service resource and a *fixed* sequence of customers, denoted as $I = \{1,\ldots,i,\ldots,n\}$ where n denotes the total number of customers. Let i be the index of customers in the visitation order, $i = 1, 2 \ldots n$ with a dummy customer at the end. The last customer n does not have any requirements and has one task with zero duration time. Also, t_{max} refers to the latest possible start time for the execution of the task that belongs to the last customer (i.e. customer n), and T is the ordered set of time periods to be considered, $T = \{1 \ldots t_{max}\}$. In the task assignment problem, completion time (start, also end, time for the execution of the last customer's task) is not fixed. Among the possible time periods in T, let h be the possible completion time and H be the set of possible completion times. In this notation, H is the subset of T at the tail and $h \leq t_{max}$.

Each customer $i \in I$ requires a set of operations $KI(i)$, and each operation, indexed as k, has a specified time window $[\gamma_k, \beta_k]$ within which the start of service is feasible. The time window for operation k only sets the feasible time range for beginning this operation, but the operation k does not have to be completed unless the completion time is greater than β_k. Also, each customer $i \in I$ has an available set of tasks $JI(i)$, and each task, indexed as j, includes a set of operations $KJ(j)$ and requires duration δ_j to be performed. That is, if task j is selected for customer i, then whenever the customer i gets service, the resource spends δ_j unit time and satisfies the time window $[\gamma_k, \beta_k]$ for every operation $k \in KJ(j)$. In this paper, we are interested in hard time window constraints, but in practice, a violation of the time window constraints may be acceptable with a high penalty. In addition, we assume that no partial services are allowed (no preemption). Finally, we assume that each customer appears only once in the sequence.

For each customer, the planner tries to honor the time window requirements of every operation. Let K be the set of all operations for all customers. Each operation $k \in K$ minimally has the following attributes:

γ_k	Earliest start time for the execution of operation k
β_k	Latest start time for the execution of operation k if the completion time is greater than β_k

c_{kt} Total cost until time t for performing operation k at time t. If the operation is not done during the sequence, cost c_{kh} will occur for the operation k if the completion time is equal to h

f_{kth} Total cost of performing operation k at time t to the end of horizon h. If the operation is not done during the sequence, this cost does not appear.

In some applications, f_{kth} equals zero, as in emergency maintenance problems. Once the item is repaired, there is no problem left. In other situations, f_{kth} value should be considered, as in replenishment problems. This cost represents the inventory cost until the end of horizon.

The planner accomplishes the operations by executing the available tasks for each customer. Let $JK(k)$ be the set of all tasks that contain operation k and let $JI(i)$ be the set of all alternative tasks that can be done for customer i. Each task $j \in JI(i)$ has the following characteristics:

Γ_j Earliest start time for the execution of task j, if it is selected. That is, $\max\{\gamma_k \mid k \in KJ(j)\}$

B_j Latest start time for the execution of task j, if it is selected. That is, $\min\{\beta_k \mid k \in KJ(j)\}$

δ_j Duration for performing task j, for all $j \in JI(i)$, $i = 1, 2 \ldots n$

It is clear that the decision maker cannot choose a task earlier than the earliest start time for any operation contained in that task. Similarly, she cannot start a task later than the latest start time of any operation contained in that task. In the light of Γ_j and B_j parameters, let $TJ(j)$ represent the time window for task j where $TJ(j) = \{\Gamma_j, \ldots, B_j\}$. However, time window calculations for tasks are valid only if the customers appear once in the sequence. If the customers appear multiple times, more complicated techniques are needed for preprocessing. We will explain these techniques in the next chapter.

The task assignment problem contains the time windows for each operation. We can use these time windows (and processing times of tasks) to develop time windows for customers in order to reduce the problem size. We refer *TI*(*i*) as the time window of customer $i \in I$ and calculate it as follows:

Let $TI(i)_L$ and $TI(i)_U$ be the lower and upper bound on the time window *TI*(*i*) of the customer $i \in I$. Also, let $\delta_{min}(i)$ and $\delta_{max}(i)$ be the minimum and maximum durations of the tasks that belongs to customer $i \in I$. Then, the lower bound (earliest start time) $TI(i)_L$ can be calculated as follows:

$$TI(i)_L = \max\left\{TI(i-1)_L + \delta_{\min}(i-1), \min_{j \in JI(i)}\left\{\Gamma_j\right\}\right\}$$

where $TI(0)_L = 0$. The earliest start time for the customer $i \in I$ cannot be earlier than the earliest start time for the execution of its tasks. Also, it cannot be earlier than the earliest start time of the previous customer plus the minimum duration that should be spent for the previous customer. At the end, $TI(n)_L$ gives the lower bound of the completion time.

As an improvement step, if the lower bound on the completion time violates the latest start time of some operations, we choose the tasks that include those operations in the calculation of the earliest start time for customers.

The upper bound (latest start time) $TI(i)_U$ depends on the *waiting* assumption, which states whether or not there could be waiting time before task executions. If waiting is not allowed, $TI(i)_U$ can be calculated as follows:

$$TI(i)_U = \min\left\{TI(i-1)_U + \delta_{\max}(i-1), \max_{j \in JI(i)}\left\{B_j\right\}\right\}$$

where $TI(1)_U = 0$. When no waiting is assumed, the latest start time for the customer $i \in I$ cannot be later than the latest start time of the previous customer plus the maximum duration that should be spent for the previous customer. Also, it cannot be later than the latest start time for the execution of its tasks. In this calculation, $TI(n)_U$ gives the upper bound on the completion time.

As an improvement step, if the selected task in the latest start time calculation violates the earliest start time of some operations, we exclude that task and reselect it (maximum processing time among the remaining ones).

If waiting is allowed, $TI(i)_U$ can be calculated as follows:

$$TI(i)_U = \min\left\{TI(i+1)_U - \delta_{\min}(i), \max_{j \in JI(i)}\{B_j\}\right\}$$

where $TI(n)_U = t_{max}$. The latest start time for the customer $i \in I$ cannot be later than the latest start time for the execution of its tasks. Also, it cannot be later than the latest start time of next customer minus the minimum duration that should be spent for the current customer.

In this structure, we assume that service for all customers in the sequence must be completed and that the specified maximum number of time periods (t_{max}) for the completion time is sufficient to complete all steps. The optimization problem outlined above can be formulated as a discrete time indexed network flow.

Decision Variables:

X_{jt} = 1 if we *start* task j at time t, and 0 otherwise, for all i = 1 … n, $j \in JI(i)$, $t \in TJ(j)$

U_{kh} = 1 if no tasks containing operation k are performed during the horizon with length h for $k \in K$, $h \in H$

V_{kth} = 1 if operation k is performed at time t and the completion time is h for $k \in K$, $t \in T$, $h \in H$

Z_h = 1 if the completion time is h, and 0 otherwise, for all $h \in H$ – called the *exit indicator* variable

The Z_h variables are defined merely for convenience and to simplify the representation. We can equivalently formulate the problem without these variables.

Model Formulation:

$$\text{Minimize } \sum_{k\in K}\sum_{h\in H} c_{kh}U_{kh} + \sum_{k\in K}\sum_{j\in JK(k)}\sum_{t\in TJ(j)} c_{kt}X_{jt} + \sum_{k\in K}\sum_{t\in T}\sum_{h\in H} f_{kth}V_{kth} \qquad (1)$$

subject to:

Task assignment for first step

$$\sum_{j\in JI(1)}\sum_{t\in TI(1)} X_{jt} = 1 \qquad (2a)$$

Flow conservation constraints

$$\sum_{j\in JI(i-1)} X_{j,t-\delta_j} = \sum_{j\in JI(i)}\sum_{t'\geq t} X_{jt'} \qquad \text{for } i = 2 \ldots n,\ t \in TI(i) \qquad (2b)$$

Exit indicator

$$\sum_{j\in JI(n)} X_{jh} = Z_h \qquad \text{for } h \in H, \qquad (3)$$

Detection of not done operations

$$Z_h - \sum_{j\in JK(k)}\sum_{t\in TJ(j)} X_{jt} \leq U_{kh} \qquad \text{for } k \in K,\ h \in H, \qquad (4)$$

Detection of time elapses after performing each operation

$$Z_h + \sum_{j\in JK(k)} X_{jt} - 1 \leq V_{kth} \qquad \text{for } k \in K,\ t \in T,\ h \in H \qquad (5)$$

Integrality

$$X_{jt}, U_{kh}, V_{kth}, Z_h = 0 \text{ or } 1 \qquad \text{for } j \in J,\ t \in T,\ k \in K,\ h \in H. (6)$$

The objective function (1) minimizes the total penalty for three terms. The first term is the penalty for operations that are not performed. The second and third terms hold the penalties for operations that are performed. The second one computes the penalties until the execution time of the operations, and the third one calculates the penalties after the execution time of the operations. Constraint (2a) assigns the task for the first customer, and (2b) is a flow conservation constraint. If there is a *no-waiting* assumption, we can write the right hand side of these constraints as $\sum_{j\in JI(i)} X_{jt}$. Constraint (3) determines the exit time and constraint (4) detects incomplete operations. Constraint

(5) calculates the time that elapses after performing each operation. Finally, constraint (6) is for integrality requirements.

In some applications, there can also be an additional cost for waiting times. We did not consider waiting time costs here for the sake of simplicity, but they can be easily incorporated to our formulation.

In the next two sections, we deal with the computational complexity and the special case structures of the task assignment problem.

2.4 NP HARDNESS OF THE TASK ASSIGNMENT PROBLEM

Task assignment [TA] problems are in the category of difficult problems, so called NP-Hard problems. In fact, the well-known knapsack problem can be written as an instance of the TA problem. (See Karp (1972) for the knapsack problem.)

Knapsack Problem: Let N be the number of items and i be the index of each item, $i = 1, 2 \ldots N$. Each item i has the following attributes:

c_i	Cost of item i if it is selected, for all $i = 1, 2 \ldots N$
a_i	size of item i for all $i = 1, 2 \ldots N$

Let b represent the limit that we need to satisfy, i.e. capacity of knapsack. Each item i has the following decision variable:

X_i = 1 if item i is selected and 0 otherwise, for all $i = 1, 2, \ldots, N$

The *Knapsack* problem can be formulated as an integer program:

[KP] Maximize $\sum_{i=1}^{N} c_i X_i$

$$\text{s.t.:} \quad \sum_{i=1}^{N} a_i X_i \le b$$

$$X_i = 0 \text{ or } 1, \quad \text{for all } i = 1, 2 \ldots N$$

Proposition 2.1: *The knapsack problem [KP] is polynomially reducible to the task assignment problem [TA].*

Proof: If we convert the general knapsack problem to a task assignment instance, the construction of the TA instance for the indices and sets is as follows:

- A route consisting of $N + 1$ customers. In other words, $n = N + 1$
- Each customer i requires only two operations, say v_i and w_i, for $i = 1 \ldots N$, and customer $N + 1$ requires only one operation v_{N+1}
- Two tasks are available for each customer $i = 1 \ldots N$. Each task contains one operation. For simplicity's sake, say task v_i includes operation v_i, and w_i includes operation w_i for $i = 1 \ldots N$. Customer $N + 1$ has one task called v_{N+1}

The construction of the TA instance for the parameters is as follows:

- Duration time for performing task v_i is a_i and zero for task w_i, for $i = 1, 2 \ldots N$. Also, the duration time for performing task v_{N+1} is zero
- $\gamma_k = 0$ for operation $k = v_i$ and w_i, for $i = 1, 2 \ldots N$ (earliest start time)
- $\beta_k = b+1$ for operation $k = v_i$ and w_i, for $i =1, 2 \ldots N$ (latest start time)
- $\gamma_k = \beta_k = b$ for operation $k = v_{N+1}$
- $c_{kb} = c_i$ for operation $k = v_i$ at time b and $c_{kt} = 0$ for the others
- $f_{kth} = 0$ for all $k \in K$, $t \in T$, $h \in H$

In this instance, the time window of operation v_{N+1} has only one element b and there is only one task available for customer $N + 1$, so this task should be done at time b. Since we know that the last customer should get service at time b, all the time window constraints for all operations are irrelevant for customers $i =1, 2 \ldots N$. Besides, there will be no cost related to w_i in the objective function. Using this structure, we can make the following observations:

- The only feasible time point for step $N + 1$ is period b. Therefore, flow constraints (2a) and (2b) try to reach that period by selecting either task v_i or w_i. Also, they can wait in intermediate steps.
- Duration of tasks v_i are equal to a_i for all steps, and the durations of tasks w_i are zero, for $i = 1 \dots N$. Therefore, the feasible solution selects the subset of tasks v_i, and the duration of selected v_is cannot exceed period b.
- If the solution does not select to do operation v_i, (equivalently selects to do operation w_i) we will pay c_i for the completion time b.
- If the solution does not choose any of the operations v_i, total cost would be $\sum_{i=1}^{N} c_i$.

We can define a new decision variable to capture these observations better:

Y_i = 1 if task v_i is selected and 0 if task w_i is selected, for all $i = 1 \dots N$

Therefore we can rewrite the task assignment problem as:

$$Obj_1 = \text{Minimize } \sum_{i=1}^{N} c_i - \sum_{i=1}^{N} c_i Y_i$$

$$\text{s.t.:} \quad \sum_{i=1}^{N} a_i Y_i \le b$$

$$Y_i = 0 \text{ or } 1, \quad \text{for all } i = 1, 2 \dots N.$$

In the objective function, the first term is constant and does not affect the solution, so we can exclude it during the solution process. Also, recall that $\min -Z = -\max Z$. Therefore, we can equivalently write the above formulation as:

$$Obj_2 = \text{Maximize } \sum_{i=1}^{N} c_i Y_i$$

$$\text{s.t.:} \quad \sum_{i=1}^{N} a_i Y_i \le b$$

$$Y_i = 0 \text{ or } 1, \quad \text{for all } i = 1, 2 \dots N.$$

There is a one-to-one relation between the objectives of these two formulations, which is:

$Obj_1 = -\,(Obj_2 - \sum_{i=1}^{N} c_i)$.

Therefore, an optimal solution to one of them is also an optimal solution to the other one. Finally, observe that the second formulation is equal to the knapsack problem.

Since each in set, the indices and parameters in TA has at most $O(N)$ items, the reduction will take polynomial time. □

Corollary 2.2: *The following problems are NP-Hard:*

a) Task assignment problem

b) Task assignment problem with waiting time costs

c) Task assignment problem with negative costs

d) Task assignment problem with customer-wise time window constraints (rather than operation-wise time window constraints)

Proof: a) In the construction in Proposition 2.1, there is an objective function with value Z for the knapsack problem if and only if there is an objective function with value – ($Z - \sum_{i=1}^{N} c_i$) for the task assignment problem with knapsack's parameters where $\sum_{i=1}^{N} c_i$ is constant. That is why we can conclude that the TA problem is NP-Hard.

b) The TA problem is special case of this problem with zero waiting costs. Therefore, the result immediately comes from the part (a). If waiting costs should be nonzero, then we can select the per period waiting costs bigger than the total possible penalty cost $\sum_{i=1}^{N} c_i$. Then the equality knapsack problem (knapsack problem with equality constraint) is polynomially reducible to the TA problem with waiting time costs. (See Kaufman et al. (1985) for equality knapsack problem.)

c) We assign negative knapsack cost parameters as cost parameters for the TA problem. This construction of the problem instance is similar to Proposition 2.1 except for the duration parameters: the duration of tasks v_i are equal to zero for all steps, and the duration of tasks w_i are a_i, for i =1 … N. Since $\min -Z = -\max Z$, the optimal objective value for the TA problem is the negative of the optimal objective value for the knapsack problem.

d) The construction of the TA instance with knapsack parameters in Proposition 2.1 requires only one operation-wise time window constraint (for customer N+1). This time window constraint could be treated as a customer-wise time window constraint. Therefore, the conclusion follows from part (a). □

In the construction of the problem instance in Proposition 2.1, the customers require only two operations. In fact, one of the operations has no duration, time window or nonzero cost parameter requirement. The cost parameter for the other operation appears once in time, so we can conclude that any type of cost function other than zero cost function provides an NP-Hard result.

In the below proposition, we prove that even the problem with zero cost function is NP-Hard for the TA problem that has a *no-waiting* assumption.

Proposition 2.3: *Consider the task assignment problem under the no-waiting assumption. If there are only two operations (and their corresponding tasks) for each customer, the resulting problem is NP-Hard with zero cost function.*

Proof: Since there is no cost at all, the problem is a feasibility problem, and we have to satisfy the time window requirements of the operations.

We show that the 2-Partition problem is polynomially reducible to this problem. (See Karp, 1972 and Garey and Johnson, 1979.) In the 2-partition;

- Data: a finite set I and a size $a_i \in Z_+$ for $i \in I$
- Question: Is there a subset $I' \subseteq I$ such that $\sum_{i \in I'} a_i = \sum_{i \in I \setminus I'} a_i$?

We can construct the desired instance with $|I| + 2$ customers. In this construction, assume that each customer i requires only two operations (and has their corresponding tasks), say v_i and w_i for $i = 1 \ldots |I| + 2$. The construction of the TA instance with a *no-waiting* assumption is as follows:

- Durations are $\delta_{v(i)} = a_i$ and $\delta_{w(i)} = 0$ for $i = 1 \ldots |I|$. Furthermore, $\delta_{v(I+1)} = \delta_{w(I+1)} = 0$. Finally, the last customer has durations $\delta_{v(I+2)} = \delta_{w(I+2)} = \frac{1}{2}\sum_{i\in I} a_i + 1$.
- $\gamma_k = 0$ and $\beta_k = \infty$ for all customers except for the customer $I + 1$.
- $\gamma_{v(I+1)} = \beta_{v(I+1)} = \frac{1}{2}\sum_{i\in I} a_i$

Using this structure, we can make the following observations:

- The last customer requires $\frac{1}{2}\sum_{i\in I} a_i + 1$ amount of duration. Therefore, the total duration time in the sequence is equal to or greater than $\frac{1}{2}\sum_{i\in I} a_i + 1$.
- Every feasible solution should select $v(I+1)$ to satisfy the latest start time requirement of that operation, because $\beta_{v(I+1)} < \frac{1}{2}\sum_{i\in I} a_i + 1$. This selection can only be done if the task $v(I+1)$ starts at time $\frac{1}{2}\sum_{i\in I} a_i$ because of the earliest start time requirement.
- Hence, the problem is feasible only if there is a subset $I' \subseteq I$ such that $\sum_{i\in I'} a_i = \sum_{i\in I\setminus I'} a_i$. In the feasible solution, selected $v(i)$ tasks from $i = 1 \ldots |I|$ give the subset I'. We conclude that the problem is NP-Hard. □

In the construction of the problem instance in Proposition 2.3, one of the operations has only a duration requirement and does not have any time window requirement.

Corollary 2.4: *Consider the task assignment problem under a no-waiting assumption and with only two operations (and their corresponding tasks) for each customer. The following problems are NP-Hard:*

a) The problem that has no latest start time requirements and has non-decreasing cost functions with time for all operations

b) The problem that has no time window (earliest or latest time) requirements and has general cost functions for all operations

c) Task assignment problem with waiting and waiting time costs under the restriction of part (a) or part (b)

Proof: a) The construction of the problem instance in Proposition 2.3 does not require time window constraints, except for the operation of customer $|I| + 1$. We can replace the latest start time requirement of this operation with a big penalty cost in case of violation. Hence, this operation acts like there is a latest start time requirement. Therefore, the result immediately comes from the Proposition 2.3.

b) Similar to part (a), we can replace the time window requirement of the operation of customer $|I| + 1$ with big penalty costs in case of violation. Hence, this operation acts like there is a time window requirement. Therefore, the result immediately comes from the Proposition 2.3.

c) We can assign big penalties for waiting time costs to the instance constructed in Proposition 2.3. As a result, this problem acts like the problem under a no-waiting assumption. Therefore, the results in part (a) and (b) are also true for this problem.

In the next section, we concentrate on some special case structures of the task assignment problem that are polynomially solvable. We propose algorithms that have polynomial run times to solve these special cases.

2.5 SPECIAL CASE STRUCTURES

The difficulty of the task assignment problem may arise from many parameters, such as the number of operations, the number of tasks, and the time window parameters. In the previous section, we saw that even the task assignment problem with two operations and a single time window requirement for one of them is NP-Hard (without any cost function). In this section, we begin with this specific problem and consider two cases. The first case considers the earliest start time requirement without the latest one, and the second case deals with the latest start time requirement without the earliest one. Then, we examine the same problems under multiple operations ($k > 2$). Lastly, we discuss another special case that assumes equal process times for all tasks. Since the start times of the tasks are fixed, there is no point for the time window requirements. Instead, we consider the general cost function for this special case.

2.5.1 TA problem: Two operations and single time window

Here, we assume that each customer requires only two operations and has two corresponding tasks. In addition, only one of the operations has either the earliest start time or the latest start time requirement. The service planner chooses one of these two operations for each customer. For this subsection, we use $v(i)$ to represent the operation (and task) with the time window for customer i and $w(i)$ to represent the operation (and task) without the time window for customer i. Finally, we consider the problem with a zero cost function. In other words, feasibility is an issue here.

Proposition 2.5: *Consider the task assignment problem under zero cost function with n customers. Assume that each customer i requires two operations, v(i) and w(i), and only one of the operations, v(i), is restricted with the earliest start time. Under these conditions, the following statements are true:*

a) If waiting is not allowed:

a1. Selection of the tasks without time window restrictions, w(i), for all customers gives a feasible solution

a2. If the tasks without time window restrictions, w(i), are not available for some customers, Algorithm 1 (its logic is given below) solves the problem in O(n)

b) If waiting is allowed:

b1. Selection of the tasks without time window restrictions, w(i), for all customers gives a feasible solution

b2. If the tasks without time window restrictions, w(i), are not available for some customers, waiting as much as the latest earliest start time of v(i) at the beginning and then performing the operation with the earliest start time, v(i) for all customers gives a feasible solution.

Logic of Algorithm 1: The algorithm begins with the first customer and follows the same customer order in the sequence. If the service resource comes to any customer before the earliest start time of the operation *v*, and operation *w* is not available, then the problem is infeasible. Otherwise, the algorithm selects the task that has the maximum duration. The detailed pseudo code is given in the appendix.

Proof: (a1 and b1) Since there is no time window restriction on any operation *w*, and there is no latest start time requirement for any operation *v*, selection of *w* for all customers gives a feasible solution.

(a2) Given in the appendix.

(b2) If the tasks without time window restrictions, *w*(*i*), are not available for some customers, waiting as much as the latest earliest start time at the beginning of the sequence will prevent any violation. Then, performing operation *v*(*i*) for all customers gives a feasible solution. □

Proposition 2.6: *Consider the task assignment problem under zero cost functions with n customers. Assume that each customer i requires two operations, v(i) and w(i), and only one of the operations, v(i), restricted with the latest start time. Then, the following statements are true:*

a) Algorithm 2 (its logic is given below) solves the problem under a no-waiting assumption in O(n^2)

b) The problem under a no-waiting assumption is infeasible if and only if the problem under the waiting allowance is infeasible.

Logic of Algorithm 2: In the initialization part, the algorithm finds the minimum possible completion time for the given customer sequence (by selecting tasks with minimum duration). The algorithm begins with the first customer and follows the same customer order in the sequence. At customer *i*, if the completion time is smaller than the latest start time of the operation *v*(*i*), the resource performs a task with minimum duration for that customer. Otherwise, the algorithm checks the time that the resource is available for that customer. If the time is earlier than the latest start time of the operation *v*(*i*), the algorithm chooses the operation *v*(*i*); otherwise, the problem is infeasible. The algorithm updates the candidate completion time and rescans all the customers. It stops if either infeasibility is found or if there is no change in the candidate completion time. The detailed pseudo code is given in the appendix.

Proof: (a) Given in the appendix.

(b) It is clear that if the problem under a waiting allowance is infeasible, then the problem under no-waiting assumption is infeasible, because the later problem's feasible set is a subset of the earlier one.

If the problem under no-waiting assumption is infeasible, then there should be an operation for some customer in which the resource cannot come to that customer before the latest start time of its operation. Waiting anywhere will not fix this issue. Hence, the problem under a waiting allowance is infeasible. □

2.5.2 Task assignment problem with multiple operations and tasks

- When customers require multiple operations, the task assignment problem gets more complicated for the following reasons: Each customer requires operations that have different time windows. The service manager should choose an appropriate task at an appropriate time in order to balances time windows for each customer.
- For each customer, not all of the operation combinations (i.e. tasks) may be available. The problem needs to select appropriate tasks that cover required operations.

Recall that *KI*(*i*) represents the set of operations for customer $i \in I$. Also, let *JI*(*i*) be the set of all alternative tasks that can be done for customer *i*, and let *JK*(*k*) be the set of all tasks that contain operation *k*.

Proposition 2.7: *Consider the task assignment problem under zero cost functions with n customers. Let* $k^{\max}$ *be the maximum number of operations required by any customer (i.e.,* $\max_i\{|KI(i)|\}$*) and* $j^{\max}$ *be the maximum number of available tasks for any customer (i.e.,* $\max_i\{|JI(i)|\}$*). If there are only earliest start time restrictions for all of operation k, the following statements are true:*

a) If waiting is not allowed, Algorithm 3 (its logic is given below) solves the problem in $O(nk^{max}j^{max})$

b) If waiting is allowed, waiting as much as the latest earliest start time of all operations at the beginning, and then performing any operation gives a feasible solution

Logic of Algorithm 3: The algorithm begins with the first customer and follows the customer order in sequence. If the service resource comes to a customer before the earliest start time of some operations, the algorithm excludes the tasks that contain those operations. If there are no tasks left, then the problem is infeasible; otherwise, it selects the task (among available ones) that has the maximum duration. The detailed pseudo code is given in the appendix.

Proof: (a) Given in the appendix.

(b) Similar arguments as in Proposition 2.5. □

Proposition 2.8: *Consider the task assignment problem under zero cost functions with n customers. Let* $k^{\max}$ *be the maximum number of operations required by any customer (i.e.,* $\max_i\{|KI(i)|\}$*) and* $j^{\max}$ *be the maximum number of available tasks for any customer (i.e.,* $\max_i\{|JI(i)|\}$*). If there are only latest start time restrictions for all operation k, the following statements are true:*

a) Algorithm 4 (its logic is given below) solves the problem under a no-waiting assumption in $O(n|K|k^{max}j^{max})$

b) The problem under a no-waiting assumption is infeasible if and only if the problem under a waiting allowance is infeasible.

Logic of Algorithm 4: In the initialization part, the algorithm finds the minimum possible completion time for the given customer sequence (by selecting tasks with minimum duration) and assigns this length as a candidate completion time. The algorithm begins with the first customer and follows the customer order in the sequence. For each customer, it finds the required operations. (An operation is required if the latest start time is earlier than the candidate completion time.) The algorithm checks the time that the resource is available for that customer. If the time is earlier than the latest start time of all required operations, the algorithm chooses a minimum duration task that contains all of the required operations; otherwise, the problem is infeasible. The algorithm updates the candidate completion time (by summing up the duration of selected tasks) and rescans all of the customers. It stops if either infeasibility is found or if there is no change in the candidate completion time. The detailed pseudo code is given in the appendix.

Proof: (a) Given in the appendix.

(b) Similar arguments as in Proposition 2.6. □

2.5.3 Task assignment problem with equal process times

In the task assignment problem, a task selection for one customer affects the other ones because of different process times. The start times for each customer are determined by the earlier task decisions. In addition, the completion time of the sequence depends on all of the selections.

However, if all process times are equal, neither the start times of the customers nor the completion time depend on the task selections. Therefore, the problem becomes easy to solve.

Proposition 2.9: *Consider the task assignment problem under a no-waiting assumption with n customers. Let* $k^{\max}$ *be the maximum number of operations required by any customer (i.e.,* $\max_i\{|KI(i)|\}$*) and* $j^{\max}$ *be the maximum number of available tasks for any customer (i.e.,* $\max_i\{|JI(i)|\}$*). If all the process times of all tasks of each customer are equal, the problem is solvable in* $O(nk^{max}j^{max})$ *for general cost functions.*

Proof: Since all the process times of all the tasks for each customer are equal, the start time for each customer and the total completion time are known. Therefore, we can calculate the cost of selecting each task. The optimal solution is the selection of tasks with minimum cost for each customer.

The cost calculation of any task has to consider up to k^{max} operation. Then, we have to choose the task with minimum cost among O(j^{max}) tasks. For n customers, the selection of all tasks with minimum costs will take $O(nk^{max}j^{max})$ time.

2.6 Shortest Path Approach

In this section, we develop an alternative formulation for the task assignment problem by using shortest path algorithms (see Dijkstra, 1959; Dial, 1969; Johnson, 1977; Ahuja et al., 1991). In the classical shortest path problem, we know the arc lengths in advance. However, the arc lengths in this problem are defined only after the total service time (completion time) of the sequence is calculated.

Here, we represent the task assignment problem in the time-space network. In this network, each node denotes the specific customer and that customer's visitation time, and denoted as (i, t) where $i \in I$ and $t \in T$. The first node is denoted as (1, 0) and represents the first customer at time 0. The network also has $(n+1, t)$ and sink nodes for structural purposes.

In this graph, there is an arc from (i, t) to $(i + 1, t + \delta_j)$ where δ_j is the duration of task j for each $j \in JI(i)$ and $i \in I$ and $t \in T$. Moreover, there is an arc from each $(n+1, t)$ node to the *sink* node for each $t \in T$.

In the case where waiting is allowed, there is an additional arc from (i, t) to $(i, t + 1)$ for each $j \in JI(i)$ and $i \in I$ and $t \in T$.

In the following example, the problem has three customers and each customer has two tasks. The durations of the tasks are given in Table 2.1.

		Customers		
		1	**2**	**3**
Tasks		1	1	1
		2	3	1

Table 2.1. Durations of the tasks for each customer

Figure 2.1 represents the time-space network of this instance, assuming that the maximum completion time is 6.

If waiting is allowed, the network has extra arcs and nodes compared to the network under the no-waiting assumption. In Figure 2.1, the arcs from (i, t) to (i, t + 1) are the extra arcs, and the shaded nodes are the extra nodes.

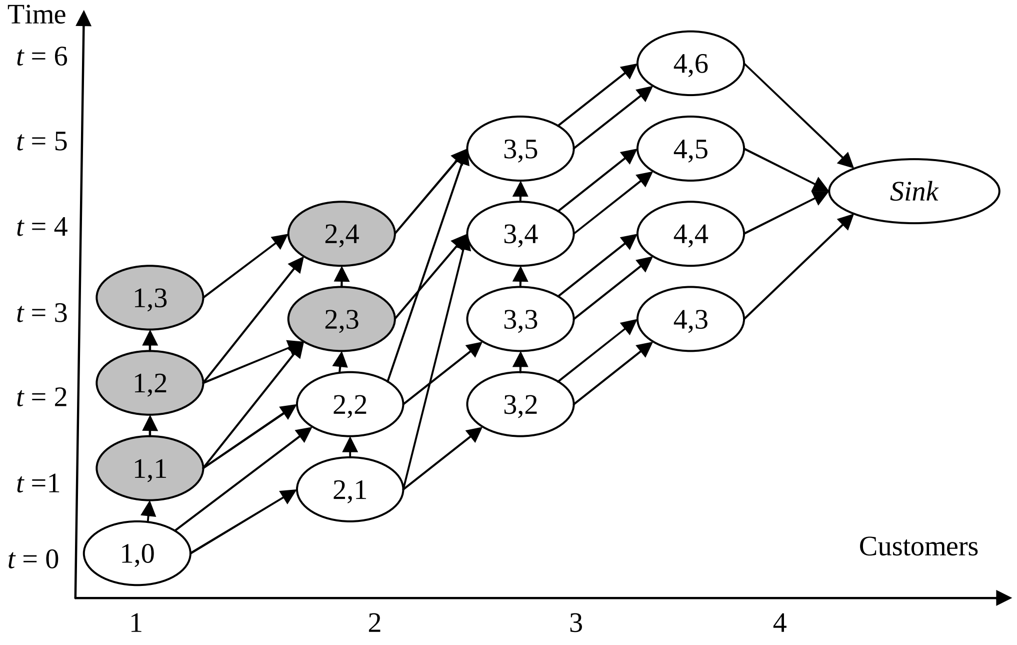


Figure 2.1. Time-space network of given example

In the task assignment problem, the arc from customer i to customer $i + 1$ has a cost which is equal to the total cost of the operations required by customer i. The cost for each operation depends on the completion time and the cost structure for each operation $k \in K$ in the following way for a given completion time h:

$$c(k,t) = \begin{cases} c_{kt} + f_{kth} & \text{If the operation is done at time } t \\ c_{kh} & \text{otherwise} \end{cases}$$

$h \in H$ is a candidate completion time for the task assignment problem. Clearly, the actual completion time is not known at time t.

Let c_{jit} be the cost of an arc from (i, t) to ($i + 1$, $t + \delta_j$) for each $j \in JI(i)$ at time t. This arc represents the selection of task j for customer i at time t. Furthermore, let $KI(i)$ be the set of operations for customer i and $KJ(j)$ be the set of operations for task j. If there are no time window constraints, then the following expression calculates the cost of an arc c_{jit} for a given completion time h:

$$c_{jit} = \sum_{k \in KJ(j)} (c_{kt} + f_{kth}) + \sum_{k \in KI(i) \setminus KJ(j)} c_{kh}$$

The first summation is the cost of the operations that are done at time t, and the second summation gives the cost of the skipped operations. (The costs of arcs from nodes (n + 1, t) to sink node are zero.) For the sake of simplicity, we did not consider waiting time costs here, but they can be easily incorporated by attaching cost to arcs that go from (i, t) to (i, t + 1).

In the case of the time window constraints, the earliest start time constraint for performing operation k is violated if we perform this operation earlier than its earliest start time, γ_k. Similarly, the latest start time constraint for performing operation k is violated if we do not perform this operation within β_k time unit. Therefore, we can delete the arc c_{jit} from the graph under one of the following conditions for a given completion time h:

- If $\gamma_k > t$ for $k \in KJ(j)$
- If $\beta_k < t$ for $k \in KJ(j)$ or $\beta_k < h$ for $k \in KI(i)/KJ(j)$

The task assignment problem finds a path of minimum cost from the node (1,0) to the *sink* node assuming that each arc has an associated cost c_{jit}, where $j \in JI(i)$, $i \in I$ and $t \in T$. Recall that the value of the cost c_{jit} is not known at time t since the actual completion time is not known, whereas it is *a priori* known in the classical shortest path problem.

We can calculate the arc costs if we know the completion time, and if we know the arc costs, we can solve the problem by using the shortest path routine. Therefore, we solve this problem for each candidate completion time h and take the minimum valued shortest path solution. The following is a generic scheme for this algorithm:

Algorithm for the shortest path approach

```
0. Set optimalvalue = ∞, optimalsolution = empty
1. For each candidate completion time $h \in H$ do
2.     Hold the arc originating from node $(n + 1, h)$ to sink node and delete all the
       remaining arcs terminating at sink node
3.     Delete all the arcs that have an incoming node with a time index greater than $h$
4.     Calculate the arc costs with respect to completion time $h$
5.     Solve the shortest path problem for the resulting network
6.     If the current shortest path value is less than optimalvalue
7.         optimalvalue = current shortest path value
8.         optimalsolution = current shortest path solution
9. End For
```

Another way to solve this problem is by duplicating the network for all candidate completion times and solving one big shortest path problem. However, this approach requires more memory than the proposed algorithm above.

The computational performance of the algorithm relies heavily on the number of arcs in the shortest path routine and the number of candidate completion times. Although the algorithm deletes the unnecessary arcs for candidate completion time *h* in step 2 and 3, we can do better than that.

We can improve the computational performance of the algorithm by tightening the time window for each customer. This will decrease the number of arcs so that the shortest path routine becomes faster. If we can tighten the time window of the last customer, this will also decrease the number of candidate completion times. The time window calculations of the customers were described in Section 1.3.

We can also make some improvements to the network for a given completion time. We can delete the unnecessary arcs and nodes in the shortest path graph using the following methods: 1) Changing the direction of the arcs. 2) Finding all the reachable nodes from the *sink* node. 3) Deleting all the nodes that are not reachable from the *sink* node.

This will give us a tighter graph. However, it is an expensive method to consider unless the cost calculations take too much time.

These procedures will decrease the size of the network and hopefully increase the performance of the algorithm.

Proposition 2.10: *Algorithm shortest path approach solves the task assignment problem.*

Proof: In step 2 of the algorithm, there is only one arc left that goes to the *sink* node. If a feasible solution to the task assignment problem has a completion time *h*, this solution could appear as a path in only one shortest path problem, which would contain the arc from node (n + 1, h) to the *sink* node. Therefore, the shortest path value of this solution equals the objective value of the task assignment problem.

If there is a feasible path in any shortest path routine, this path satisfies flow conservation constraints and time window constraints (recall that arcs that have violations are already deleted). Since it only appears in one shortest path routine, the cost functions of the operations are calculated correctly. As a result, it is a feasible solution to the task assignment problem with the same objective value. Similarly, if there is a feasible solution to the task assignment problem with completion time h, this solution will appear as a feasible path in only one shortest path routine, which would contain the arc from node (n + 1, h) to the *sink* node. As a result, it is a feasible path in one shortest path routine with the same objective value. This concludes that the a*lgorithm shortest path approach* solves the task assignment problem. □

When the network is constructed for the shortest path approach algorithm, that network is acyclic, so topological ordering is available. In fact, from the lower time index to the higher time index gives the topological ordering.

According to Ahuja et al. (1993, pp.108), "The reaching algorithm solves the shortest path problem on acyclic networks in $O(m)$ time," where m denotes the number of arcs in the network.

Corollary 2.11: *Let $k^{\max}$ be the maximum number of operations required by any customer (i.e., $\max_i\{|KI(i)|\}$) and t_{max} be the latest time in $t \in T$. The shortest path approach algorithm solves the task assignment problem in $O(nk^{\max}t_{\max}^2)$.*

Proof: There are at most $t_{\max}$ candidate completion times. For each of them, we need to solve the shortest path routine. In each shortest path problem, there are n customers, and each customer has at most $t_{\max}$ arcs. Therefore, there are at most $nt_{\max}$ arcs. Also, the calculation of an arc cost will take $O(k^{\max})$ time. As a result of the proposition in Ahuja et al. (1993), each shortest path problem will take $O(k^{\max}\ n\ t_{\max})$ time. Since we need to solve at most $t_{\max}$ shortest path problems, the *shortest path approach algorithm* solves the task assignment problem in $O(nk^{\max}t_{\max}^2)$. □

2.7 COMPUTATIONAL RESULTS

We calculated the computational performance of the proposed shortest path approach and the IP model formulation through 75 randomly generated test cases. The proposed shortest path approach was programmed in C++, and the IP model uses ILOG OPL Development Studio 5.2. The tests were taken on an Intel Pentium M notebook computer with 1.73 GHz and 1.00 GB of RAM and a Windows XP operating system.

As an objective function, we chose a total weighted tardiness criterion that is commonly used in many areas, such as machine scheduling and after-market repair services. In this criterion, each operation has a due date and a per day penalty for each time period after the due date. The amount of per day penalty for each operation equals its weight, and the weight of the operation is uniformly assigned on a scale of 1 to 40. Since the objective has a specific function, we reformulated the problem (given in Section 3) in a more compact way. (See appendix for details.)

In the numerical analysis, we set the parameters for the number of customers, operations and tasks. The relationship between an operation and a task is randomly assigned, and there is a 50% chance to assign the operation to the task. We ensure that all the operations are assigned to at least one task. The processing time of each task depends on the number of operations included by that task. For each operation included in the task, numbers from 1 to 6 are uniformly assigned, and the summation of these numbers equals the duration time of that task.

The due dates of the operations for a particular customer are uniformly calculated from an interval. The middle point of this interval is equal to the average duration time of all tasks multiplied by the order number of that customer in the sequence. The lower bound of this interval is half of the interval's middle point, and the upper bound of this interval is equal to one and a half times the interval's middle point.

Number of			**Average Number of**		**Average CPU Time (second)**	
Customers	**Operations**	**Tasks**	**Variables**	**Constraints**	**IP Model**	**Shortest Path**
5	**1**	**2**	**93.6**	**76.2**	**0.24**	**0.01**
10	**1**	**2**	**527.4**	**422.6**	**0.42**	**0.01**
15	**1**	**2**	**1143**	**907.8**	**0.76**	**0.01**
20	**1**	**2**	**1963.4**	**1536.6**	**1.72**	**0.01**
25	**1**	**2**	**2973.6**	**2337.4**	**3.54**	**0.02**
5	**3**	**6**	**804**	**534**	**0.67**	**0.01**
10	**3**	**6**	**4270.6**	**2674.6**	**17.68**	**0.04**
15	**3**	**6**	**9987**	**6139**	**121.30**	**0.10**
20	**3**	**6**	**18054.6**	**11020.6**	**630.69**	**0.27**
25	**3**	**6**	**29792.2**	**17973.2**	**2400.36**	**0.59**
5	**5**	**10**	**1718.2**	**1108**	**1.72**	**0.04**
10	**5**	**10**	**8031.8**	**5011.4**	**41.40**	**0.18**
15	**5**	**10**	**22373.8**	**13300**	**981.74**	**0.43**
20	**5**	**10**	**37878.2**	**22931**	**>3600**	**0.77**
25	**5**	**10**	**N/A**	**N/A**	**N/A**	**1.68**

Table 2.2. Comparison of the shortest path approach and IP model formulation

Table 2.2 reports the corresponding CPU times for solving instances via the proposed shortest path approach and the IP formulation with respect to the number of customers, operations and tasks. It also shows the number of constraints and variables in the IP formulation.

Each observation listed in Table 2.2 is the average result from 5 randomly generated test cases. As we can see, the number of operations and tasks has a major impact on the required computational time, both in the proposed shortest path approach and the IP model. However, the shortest path approach requires significantly less time than the IP model. When the number of customers *n* becomes larger, OPL was terminated because of either the one-hour time limit or because of insufficient memory.

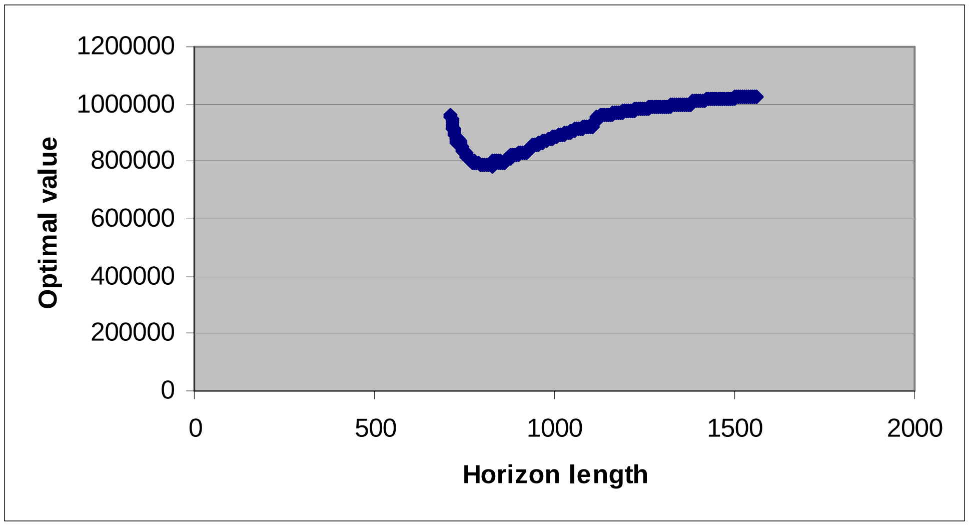


Figure 2.2. Optimal value vs. Completion time graph of a randomly generated instance with 100 customers, 5 operations and 10 tasks

In Figure 2.2, we plot the optimal value vs. completion time graph of a randomly generated instance with 100 customers, 5 operations and 10 tasks. Unfortunately, there are some local optimal solutions in this function that prevent us from performing binary or golden section searches.

2.8 Concluding Remarks

In this chapter, we considered a service management problem with a fixed customer sequence under time window and multiple operation requirements. We proved that this problem is NP-Hard. We analyzed the special case structures and proposed polynomial time algorithms for these special cases. We developed an alternative algorithm based on the shortest path approach and solved the problem effectively. The proposed shortest path approach algorithm is valid for general cost functions, because the algorithm does not make any assumptions on the objective function. Computational results show that this shortest path approach is much faster than the IP formulation solved in OPL.

This work can be extended in several directions. The problem we considered here is an operational level problem in which there are also strategic and tactical level decisions. At the strategic level, we may have multiple resources and want to partition the customers to those resources. At the tactical level, the focus is on finding the optimal route. Another extension would be multiple visitations of the same customer. In that case, task selection does not only affect the other customers, but also affects the other visitations of the same customer.

Chapter 3: Periodic Task Assignment Problem

3.1 INTRODUCTION

In this chapter, we extend the task assignment problem and allow for multiple appearances of the same customer in the given sequence. We consider the service scheduling problem in which each customer requires operations that should be performed periodically, and we assume that performing each operation has a time-varying completion cost that depends on the previous service time. Recall that in the task assignment problem, the service provider can choose more than one operation to perform for each customer, but some of the operations cannot be done together for technical reasons. A *task* refers to a combination of *operations* that can be performed together. We refer to each successive customer visit as a *step* in the sequence and consider a problem that assigns one task to each step, and we refer to the problem as a *periodic task assignment* (*PTA*) *problem*.

The assumptions we make for this problem are similar to those we made for the task assignment problem. We study the periodic task assignment problem with a single resource and a fixed sequence of customers, each of which can appear multiple times in the given sequence. We also assume that no partial service is allowed (no preemption).

In Chapter 2, we studied the problem in which each customer appears once in a sequence. To put it more accurately, the problem considers an *operation type* that should be done only once. In other words, if the same customer appears multiple times in the sequence but the operations in each appearance are different, we can treat these appearances as if they belong to different customers. Hence, we can use the proposed algorithms in Chapter 2 to solve this problem.

In the periodic service type, the customers require the same operations multiple times. We consider the cost function and the time window relative to the last performance of the same operation. Each operation has a so called *relative time window*, which means that the earliest and the latest start times depend on the previous execution time of that operation. Therefore, the time windows are *relative* to the decision maker's previous assignments. If all of the customers appear once in the sequence, this problem coincides with the problem we studied in Chapter 2, and we can use the same techniques to solve the problem. However, shorter sequences without customer repetitions could cause myopic decisions, but longer sequences with customer repetitions prevent us from making decisions that affect later steps undesirably.

In this chapter, we consider the operational level periodic task assignment problem that assigns one task to each step and its corresponding customer and requires the following inputs:

- a sequence with *m* steps,
- the possible tasks for each customer,
- the processing time for each possible task for every customer,
- the cost function and relative time window parameters for each operation, and
- the last execution date for each operation.

The planner determines which among the possible tasks to perform in each step. We require a plan that completes all of the steps in the given sequence while minimizing the general cost function of all operations.

The periodic task assignment problem has unique characteristics. First of all, each customer requires multiple operations with different time windows and general cost functions. The cost of an operation can take any value if it is done within the time window and takes a value of infinity otherwise (time windows are hard). Moreover, the previous decisions of the same customer determine the future time windows. Hence, there is no explicit time window for tasks as there is in the single visit case. In an optimal solution, the assignment would choose a time that balances the cost functions of all operations and does not violate relative time windows. Furthermore, the total cost of a step is affected not only by the decisions made for the earlier steps but also by the decisions made for the later steps because of the structure of the cost function.

In this chapter, we analyze several fundamental properties of the periodic task assignment problem. We prove that the problem is *NP*-Hard in the strong sense and show the computational complexity of some special cases. We formulate the problem as a discrete time indexed network flow.

The remaining part of the chapter is organized as follows: Section 2 gives the relevant literature review, and Section 3 formulates the problem as a discrete time indexed network flow. In Section 4, we prove that the periodic task assignment problem is *NP*-Hard in the strong sense, and Section 5 concentrates on the special case structures. Section 6 discusses the valid cuts and Section 7 describes the preprocessor algorithm that reduces the problem size. Finally, we offer the conclusion and discuss future extensions of this study in Section 8.

3.2 LITERATURE REVIEW

The periodic task assignment problem may appear in many contexts, such as multi-product lot sizing, machine maintenance, and telecommunications. The problem where the order of customers is not given but functions instead as a decision variable has received some attention. In the remainder of this study, we refer to the variant of the PTA problem where the order of customers is considered to be a decision variable as the *sequencing and periodic task assignment* (SPTA) problem. Although the SPTA problem seems to be an extension of the PTA problem, we can write the SPTA problem as a special case of the PTA problem in some circumstances. We will discuss this situation at the end of this section.

Anily et al. (1998) consider the special case of the SPTA problem in the context of scheduling preventive maintenance for a set of machines over an infinite horizon. Here, the machines are the customers, and servicing a machine means performing maintenance. The authors assume that each machine requires a single maintenance operation, and all the processing times are equal. That is, only one machine can receive maintenance in a given period, and the maintenance will be done within the given period. Another application they consider, which falls into the same problem framework, is the multi-item replenishment of stock. In this problem, only one item stock may be replenished at a time. In Anily et al. (1998), the cost of operating a machine in a period is a linear (increasing) function of the number of periods since its last service. They assume no setup cost for performing the maintenance. They show that there is an optimal maintenance schedule that is cyclic, and they present a polynomial time algorithm to compute optimal policies for a two-machine case. They also present heuristics and worst case bounds (2.5-approximation if the linear cost function starts from zero and 2-approximation if the linear cost function starts from one) for a general number of machines. To date, it is not clear whether the problem considered in Anily et al. (1998) is NP-Hard.

In Anily et al. (1999), the authors consider the problem given in Anily et al. (1998) under the additional assumption that there are only three machines. In this work, the authors introduce an algorithm that solves certain instances of the problem optimally, and for other instances, they present a heuristic with a worst case performance ratio of 1.033.

Anily and Bramel (2000) study the problem given in Anily et al. (1998) under convex cost functions. They show that there is an optimal schedule that is cyclic for a general number of machines, and in the case of two machines, they show that there exists an optimal policy, whose closed form can be either predetermined or is one of up to four possible forms.

Grigoriev et al. (2006) work on the problem given in Anily et al. (1998), assuming a finite completion time. They investigate several formulations (linear and nonlinear) and propose a column generation method to solve the problem exactly. They show that the subproblem for the column generation procedure is solvable in polynomial time.

Similar types of problems appear in Holte et al. (1992), Mok et al. (1989), and Wei and Liu (1983). Holte et al. (1992) consider the problem where the length of time without maintenance has an upper bound for each machine. Mok et al. (1989), and Wei and Liu (1983) assume that the exact maintenance intervals for each of the machines are given; the problem is to minimize the number of resources needed for a feasible schedule. Duffuaa and Ben-Daya (1994), and Hariga (1994) study the maintenance scheduling problems that concentrate on the coordination of a common resource to maintain a set of machines. A review of preventive maintenance scheduling problems can be found in Dekker et al. (1997).

Bar-Noy et al. (2002) and Kenyon et al. (2000) generalize the problem given in Anily et al. (1998). They consider that at most *k* items out of the *m* items can be serviced in each period, and they apply the problem to data broadcast scheduling. Broadcasting is an efficient means of disseminating data in asymmetric communication environments, such as satellite access to internet or car navigation systems. Typically, the down link (e.g., from satellite to personal computers) has greater bandwidth and is faster than the up link (e.g., phone lines). In these situations, broadcasting protocols reduce the server load and do not manage the client requests individually. In these protocols, data are scheduled for broadcasting continuously and one (or *k*) of them is broadcasted at a time. The clients wait for the requested data to be broadcast, so the schedule is independent of the incoming requests. Acharya (1998) and Schabanel (2000) present a very complete history of the field.

Bar-Noy et al. (2002) prove that the problem is *NP*-hard, even for $k = 1$, if there is an additional setup cost for maintenance. Further, they investigate lower bounds and propose approximation algorithms for the case $k = 1$, based on the properties of Fibonacci numbers. The worst case bounds of the proposed heuristics are 9/8 in the case when there is no fixed cost, and 1.57 when there is a fixed cost. They also prove that a greedy algorithm used in Anily et al. (1998) has a worst-case bound of 2. In Kenyon et al. (2000), the authors improve the 9/8 result (for no fixed cost case) by giving a polynomial time approximation scheme, which is ε-approximation for any $\varepsilon > 0$. Finally, Kenyon and Schabanel (2003) work on the problem with non-identical service times under no fixed cost. They prove that the problem is *NP*-hard even if the broadcast costs are all zero and give randomized 3-approximation algorithms for the case $k = 1$.

The problems considered as a version of the SPTA problem can be written as a special case of the PTA problem. For instance, the problem in Anily et al. (1998) can be seen as an infinite sequence that consists of only one customer. Here, the set of the machines are operations, and we can only do one operation at a time in each period. That is, all the tasks contain one operation, and their processing times are equal. In the extension where *k* items can be serviced at a time, we can define tasks that consist of, at most, *k* operations. As long as the *k* is given, the transformation takes polynomial time. Therefore, the following observations are true:

Literature results 3.1: *Consider the PTA problem with a cyclic sequence consisting of only one customer that requires a number of operations. Assume that each task contains one operation and each operation is in one task. Under these conditions, the following observations are true:*

- *a) If the cost function increases linearly (cumulative cost function is quadratic) with respect to time since the operation's last service and all the processing times of the tasks are equal,*
  - *a1. If there is no fixed cost, then the heuristic given in Kenyon et al. (2000), has a polynomial time approximation scheme which is ε-approximation for any $\varepsilon > 0$.*
  - *a2. If there is a fixed cost, then the heuristic given in Bar-Noy et al. (2002) has a worst bound of 1.57.*
- *b) If the cost function increases linearly (cumulative cost function is quadratic) with respect to the time since the operation's last service, and the processing times of the tasks can be non-identical, then the heuristic given in Kenyon and Schabanel (2003) has a worst bound of 3.*
- *c) If the cost function is an increasing convex function with respect to time since the operation's last service, and all the processing times of the tasks are equal, then for the case of two operations, there exists an optimal policy, the closed form of which can be either predetermined or is one of up to four possible forms. (Anily and Bramel, 2000.)*

Later on, we will see that polynomial time ε-approximation is impossible for any $\varepsilon > 0$ in the general PTA problem.

3.3 PROBLEM DEFINITION

The problem that we study is a generalized version of the problem in Chapter 2. We consider a single service resource and a *fixed* sequence of customers, denoted as $S = \{1,\ldots,s,\ldots,m\}$ where m denotes the total number of *steps* (visitations) in the sequence with a dummy step at the end. A sequence can consist of multiple cycles or tours through the same customers and as such, can include the same customer multiple times. We refer to each successive visited customer as a *step* in the sequence. The customers form the set $I = \{1,\ldots,i,\ldots,n\}$ where n denotes the total number of customers. Here, each step has an associated customer, but a customer may have more than one associated step if the customer appears more than once in the sequence. Also, t_{max} refers to the latest possible start (and end) time for the last step (i.e. step m), and T is the ordered set of time periods to be considered, $T = \{1 \ldots t_{max}\}$.

Each customer $i \in I$ requires a set of operations $KI(i)$, and each operation, indexed as k, has a specified *relative time window* $[\zeta_k, \eta_k]$ within which the service is feasible. This means that the next execution of the same operation has an earliest start time ζ_k, and the latest start time η_k with respect to the current start time of the same operation. The latest start time is effective only if the completion time is greater than the latest start time. In addition, if this operation was done t units of time before the starting time of the given sequence, then the first execution of the operation has an earliest start time γ_k, and a latest start time β_k, where $\gamma_k = \zeta_k - t$, and $\beta_k = \eta_k - t$.

Figure 3.1 gives the graphical representation of the *relative* time window (separation) parameters. In this figure, the area between $t = 1$ and $t = 3$ represents the desired times at which the operation should be performed, and the areas between $t = 0$ and $t = 1$, and $t > 3$ show the outside of the strict time window.

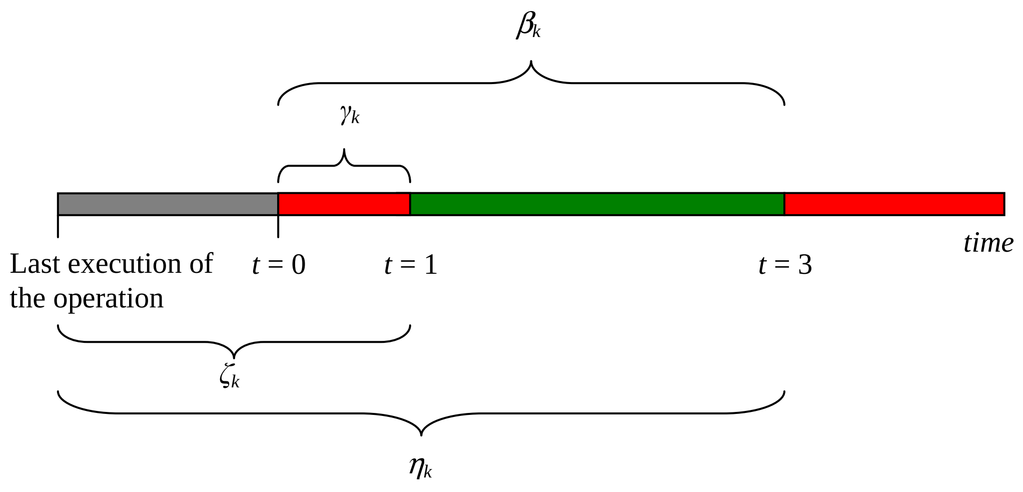


Figure 3.1 Relative time windows (separation) for operation *k*.

Each step allows a set of tasks *JS*(*s*) and each task, indexed as *j*, includes a set of operations *KJ*(*j*) and requires a duration δ_j in which the service is performed. We define tasks for each step rather than for each customer to make the problem more flexible. Doing so, a customer may have different task alternatives in different steps. In this chapter, we are interested in strict time windows, but in practice, a violation of the time window constraints may be acceptable with a high penalty. Depending on the tasks performed at each step, the completion time of the last step may vary. Among the time periods in *T*, let *h* be a possible *cycle completion time* to complete all steps and *H* be the set of possible cycle completion times. In this notation, *H* is the subset of *T* and $h \leq t_{max}$.

For each customer, the planner tries to honor the relative time window requirements of operations and accomplishes the operations by executing the available tasks. Let *K* be the set of all operations for all customers and *JK*(*k*) be the set of all tasks that contain operation *k*. At minimum, each operation $k \in K$ has the following attributes:

γ_k	Earliest start time for the first execution of operation *k*

β_k Latest start time for the first execution of operation k

ζ_k Subsequent earliest start time (minimum separation) will be ζ_k time after the first execution of operation k

η_k Subsequent latest start time (maximum separation) will be η_k time after the first execution of operation k

λ_k Maximum number of executions that can be performed on operation k. One of the upper bounds for this parameter is the total number of visitations of the customer that requires operation k. (We will investigate this parameter further in Section 6.)

$g(k, t)$ The cost between two consecutive executions of operation k, where t is the elapsed time between these executions under the following conditions:

- If the operation is not done during the sequence, then $t = h+\zeta_k-\gamma_k$. Therefore, the cost $g(k, h+\zeta_k-\gamma_k)$ will occur where h represents the completion time, $h \in H$.
- If the operation is executed the first time at time r, then $t = r+\zeta_k-\gamma_k$. Therefore, the cost $g(k, r+\zeta_k-\gamma_k)$ will occur.
- If the operation is executed at time r and the previous execution of this operation is at time r_p, then $t = r-r_p$. Therefore, the cost $g(k, r-r_p)$ will occur.
- If the operation is executed last time at time r, then $t = h-r$. Therefore, the cost $g(k, h-r)$ will occur.

We also calculate the time window of step $s \in S$, represented as $TS(s)$, to reduce the problem size. The periodic task assignment problem defines the relative time windows for each operation. We can use these time windows (and task processing times) to develop time windows for each step. The detailed logic for the time window calculation of steps is given in Section 3.7.

In this structure, we assume that all steps in the sequence must be visited and that the specified maximum number of time periods (t_{max}) in the horizon is sufficient to complete all steps. The optimization problem outlined above can be formulated as a discrete time indexed network flow problem.

Decision Variables:

X_{jt} =1 if we *start* task j at time t, and 0 otherwise, for all $s = 1 \ldots m$, $j \in JS(s)$, $t \in T$

Y_{kpt} = 1 if an operation k is performed for the p^{th} time at time t for $k \in K$, $t \in T$, $1 \leq p \leq \lambda_k$

U_{kh} = 1 if no tasks containing operation k are performed during the horizon with length h for $k \in K$, $h \in H$

W_{kpt} = 1 if t periods elapse from the p^{th} time performing operation k to the $(p+1)^{th}$ time performing operation k for $k \in K$, $t \in T$, $1 \leq p < \lambda_k$

V_{kt} = 1 if t periods elapse from performing operation k the last time to the end of the horizon. $t = h - r$ if the completion time is h and operation k is performed at time r for the last time for all $k \in K$, $t \in T$

Z_h =1 if the last step m starts in period h, 0 otherwise for all $h \in H$

– called the *exit indicator* variable

The Z_h variables are defined merely for convenience and to simplify the representation. We can adequately formulate the problem without these variables.

Model Formulation:

Minimize

$$\sum_{k\in K}\sum_{h\in H} g(k, h+\zeta_k-\gamma_k)U_{kh} + \sum_{k\in K}\sum_{t\in T} g(k, t+\zeta_k-\gamma_k)Y_{k1t} + \sum_{k\in K}\sum_{1\leq p<\lambda_k}\sum_{t\in T} g(k,t)W_{kpt} + \sum_{k\in K}\sum_{t\in T} g(k,t)V_{kt} \tag{1}$$

subject to:

Task assignment for first step

$$\sum_{j\in JS(1)} \sum_{t\in TS(1)} X_{jt} = 1 \qquad (2a)$$

Flow conservation constraints

$$\sum_{j\in JS(s-1)} X_{j,t-\delta_j} = \sum_{j\in JS(s)} \sum_{t'\geq t} X_{jt'} \qquad \text{for } s = 2, \ldots m,\ t \in TS(s) \qquad (2b)$$

Exit indicator

$$\sum_{j\in JS(m)} X_{jh} = Z_h \qquad \text{for } h \in H, \qquad (3)$$

Earliest start time for the first execution of each operation

$$\sum_{j\in JK(k)} \sum_{t'\in\{1,\ldots,t\}} X_{jt'} - \sum_{h\in\{1,\ldots,t\}\cap H} Z_h \leq 0 \qquad \text{for } k \in K,\ t \in \{1,2, ..\ \gamma_k\text{-}1\}, \qquad (4a)$$

Latest start time for the first execution of each operation

$$\sum_{j\in JK(k)} \sum_{t'\in\{1,\ldots,t\}} X_{jt'} + \sum_{h\in\{1,\ldots,t\}\cap H} Z_h \geq 1 \qquad \text{for } k \in K,\ t \in \{\beta_k, ..\ t_{max}\}, \qquad (4b)$$

Subsequent earliest start time (minimum separation) for each operation

$$\sum_{j\in JK(k)} \sum_{t'\in\{t-\zeta_k,\ldots,t\}} X_{jt'} - \sum_{h\in\{1,\ldots,t\}\cap H} Z_h \leq 1 \qquad \text{for } k \in K,\ t \in T \setminus \{1,\ldots,\zeta_k\}, \qquad (5a)$$

Subsequent latest start time (maximum separation) for each operation

$$\sum_{j\in JK(k)} \sum_{t'\in\{t-\eta_k,\ldots,t\}} X_{jt'} + \sum_{h\in\{1,\ldots,t\}\cap H} Z_h \geq 1 \qquad \text{for } k \in K,\ t \in T \setminus \{1,\ldots,\eta_k\}, \qquad (5b)$$

Detection of done operations

$$\sum_{j\in JK(k)} X_{jt} = \sum_{1\leq p\leq\lambda_k} Y_{kpt} \qquad \text{for } k \in K,\ t \in T, \qquad (6)$$

Time elapses between consecutive executions of each operation

$$Y_{k(p+1)t} + Y_{kpt'} - 1 \leq W_{kp(t-t')} \qquad \text{for } k \in K,\ t \in T,\ t' \leq t, \qquad (7)$$

Time elapses after last time execution of each operation

$$Z_h + \sum_{1\leq p\leq\lambda_k} \sum_{t'\leq t} Y_{kpt'} - \sum_{1\leq p\leq\lambda_k} \sum_{t'\in T} Y_{kpt'} + \sum_{t'\in T} Y_{kpt'} - 1 \leq V_{k(h-t)}$$

$$\text{for } k \in K,\ h \in H,\ t \in T, \qquad (8)$$

Detection of not done operations

$$Z_h - \sum_{t \in T} Y_{k1t} \le U_{kh} \qquad \text{for } k \in K, h \in H, \qquad (9)$$

Each operation can be performed p^{th} time once

$$\sum_{t \in T} Y_{kpt} = 1 \qquad \text{for } k \in K,\ 1 \le p \le \lambda_k, \qquad (10)$$

p^{th} time is done earlier than $(p+1)^{th}$ time

$$\sum_{t' \le t} Y_{kpt'} \ge \sum_{t'' \ge t} Y_{k(p+1)t''} \qquad \text{for } k \in K, t \in T, 1 \le p < \lambda_k \quad (11)$$

Integrality

$$U_{kh}, V_{kt}, W_{kpt}, X_{jt}, Y_{kpt}, Z_h = 0 \text{ or } 1 \qquad \text{for } j \in J, t \in T, k \in K, h \in H. (12)$$

The objective function (1) minimizes the total penalty with four terms. The first term is the penalty for operations that are not performed, while the second one is the first execution of operations. The third cost holds the consecutive execution of operations. Finally, the last cost calculates the penalties after the last execution of the operation. Constraint (2a) assigns a task for the first step and (2b) shows flow conservation constraints. If there is a *no-waiting* assumption, we can write the right hand side of (2b) constraints as $\sum_{j \in JI(i)} X_{jt}$. Constraint (3) determines the exit time. Although this model is written to complete all the steps in the route and the completion time is varying, we can easily incorporate the fixed time horizon approach (not necessarily complete all steps) with a little modification to the formulation.

Constraints (4a) and (5a) define the first and subsequent earliest start times (minimum separations), whereas constraints (4b) and (5b) define the first and subsequent latest start time (maximum separations), respectively. Constraint (6) detects how many times the operations are done and when they are done. Constraint (7) calculates the time elapsed between consecutive executions of each operation and (8) measures the time elapsed after the last execution of each operation. Constraint (9) detects operations that were not done. Constraints (10) and (11) are technical constraints that set the precedence relations of *Y* variables. Finally, constraint (12) is for integrality requirements. (We did not consider waiting time costs here for the sake of simplicity, but they can be easily incorporated to our formulation.)

In the next two sections, we deal with the computational complexity and the special case structures of the periodic task assignment problem.

3.4 NP Hardness of the Periodic Task Assignment Problem

The Periodic Task Assignment [PTA] problem is in the category of difficult problems, or so-called NP-Hard problems, because it is a generalization of the task assignment problem. Although the task assignment problem could be solvable in pseudo-polynomial time (NP-Hard in the ordinary sense), the PTA problem is even harder than that. In fact, the well-known 3-partition problem can be written as an instance of the PTA problem.

The 3-partition problem is NP-Hard in the strong sense (Karp, 1972; Garey &Johnson, 1979). We will show that the 3-Partition problem can be polynomially reducible to the PTA problem, but first we will give the definition of the 3-partition problem.

3-Partition: *Given positive integers $a_1,\ldots,a_{3q}$, b such that*

$$\frac{b}{4} < a_j < \frac{b}{2}$$

and

$$\sum_{j=1}^{3t} a_j = qb\,.$$

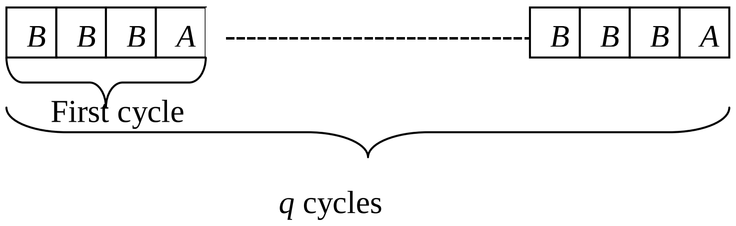
Do there exist q pair-wise disjoint 3 element subsets $S_i \subset \{1,..,3q\}$ *such that*

$$\sum_{j\in S_i} a_j = b \qquad \text{for } i=1,\ldots q?$$

Proposition 3.2: *The 3-Partition problem is polynomially reducible to a periodic task assignment problem [PTA].*

Proof: We represent the 3-Partition problem as an instance of the PTA problem. We take the following sequence as a PTA instance:

- A sequence consisting of *2* different customers: Customer *A* and customer *B*. The smallest cycle in the sequence consists of 3*B* and 1*A* customers (*B* customers are the first ones). The sequence consists of *q* cycles with a number of steps $m = 4q$.



- Customer *A* requires only one operation, and customer *B* requires 3*q* operations.
- Each operation is included in only one task, and each task includes only one operation.

The construction of the PTA instance:

- The duration time for performing task *j* for customer *A* is zero.
- The duration time for performing task *j* for customer *B* is a_j, $j = 1, 2, \ldots, 3q$.
- $\zeta_k = qb+1$ for all operations *k* of customer *B* and *b* for an operation *k* of customer *A*.
- $\eta_k = 2qb+2$ for all operations *k* of customer *B* and *b* for an operation *k* of customer *A*.
- $\gamma_k = 0$ for all operations *k* of customer *B* and *b* for an operation *k* of customer *A*.
- $\beta_k = qb+1$ for all operations *k* of customer *B* and *b* for an operation *k* of customer *A*.

- $g(k, t)$ is zero for all operations k and time t. In other words, the problem is a feasibility problem.

We make the following observations about this PTA problem instance:

- $\gamma_k = \beta_k = b$ for the operation of customer A. Therefore, each cycle length is b in the feasible solution. As such, the completion time will be qb.
- $\zeta_k = qb+1$ for the operations of customer B. Since the subsequent earliest start time for the operations of customer B is greater than the completion time, each task could be selected only once. We have $3q$ customer B in the sequence and $3q$ operations (a one-to-one relationship with their corresponding tasks). This means that each task should be selected once (otherwise, at least one of the tasks will be selected more than once). In other words, each duration time a_j, $j = 1, 2, \ldots, 3q$ appears only once.
- We know that $\sum_{j=1}^{3t} a_j = qb$ and we have q cycles. If the summation of the task durations of 3-customer B in one cycle is less than b, there will be another cycle where the summation of durations of 3-customer B in that cycle is greater than b. However, we have $\eta_k = \beta_k = b$ as the latest start time for the operation of customer A. These constraints are hard constraints and make that solution infeasible. (A similar argument could also be obtained by using the earliest start time for the operation of customer A.) In order to get a feasible solution, the summation of the task durations of 3-customer B in each cycle should be exactly b.
- Since $\sum_{j=1}^{3t} a_j = qb$, there is no waiting in the feasible solution, even if the waiting is allowed.

Therefore, the feasible solution should account for the fact that the summation of the durations of 3-customer *B* in each cycle should be exactly *b* and each duration time a_j, $j = 1, \ldots, 3q$ appears once.

Since each set, index and parameter in the PTA has at most $O(q)$ items, the reduction will take polynomial time.

Corollary 3.3: *The following problems are NP-Hard in the strong sense:*

a) Periodic task assignment problem

a1. Periodic task assignment problem with only latest start time constraints

a2. Periodic task assignment problem with only a general cost function (without time windows)

b) Periodic task assignment problem under a no-waiting assumption

b1. PTA problem under a no-waiting with only earliest start time constraints

b2. PTA problem under a no-waiting with only latest start time constraints

b3. PTA problem under a no-waiting with only a general cost function (without time windows)

c) Periodic task assignment problem with customer-wise time window constraints (rather than operation-wise time window constraints) for both waiting allowance and no-waiting assumption

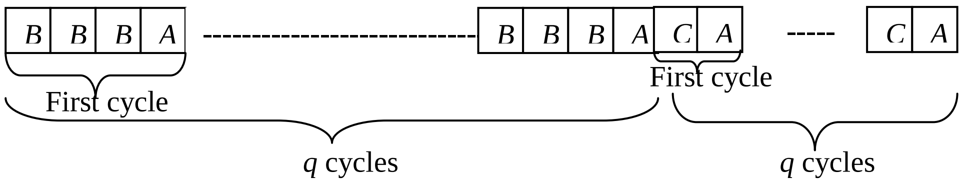
Proof: (a) Based on the construction in Proposition 3.2, there is a feasible solution with an objective value of zero in this PTA problem if and only if the 3-partition has a solution. That's why we can conclude that the PTA problem is NP-Hard in the strong sense.

(b) Since all feasible solutions require no waiting, the result immediately follows from part (a).

(c) In the instance construction, all the operations of each customer have the same time window, so the result immediately follows from part (a) and (b).

(b1) During the construction of the PTA instance with 3-Partition parameters in Proposition 3.2, choose the subsequent earliest start time parameters (ζ) for the operations of customer *B* that are large enough so that each operation of customer *B* will be done just once. This makes the completion time *qb*, and the rest of the proof is similar. The conclusion follows from part (b).

(a1 and b2) In the construction of the PTA instance with 3-Partition parameters in Proposition 3.2, define another customer, say customer *C*, with duration *b*. At the end of the original sequence, add customer *C* and *A* *q* times:



This construction will force each operation of customer *B* to be performed once. (Completion time is at least *qb*, and "not done" operations make the problem infeasible.) The rest of the proof is similar. The conclusion follows from part (a) and (b).

(a2 and b3) The construction of the PTA instance with 3-Partition parameters is the same as parts (a1) and (b2). Instead of the latest start time parameters, assume that there is a positive cost beyond that time. Here the question is whether or not there is an objective function with a value of zero. If the answer is yes, then there will also be a solution for the 3-partition problem. The rest of the proof is similar to the proof of Proposition 3.2.

Corollary 3.4: *There is no ε-approximate heuristic that runs in polynomial time for the problems given in corollary 4.2 unless P = NP for any ε > 0.*

Proof: Since the optimal objective value is zero for the problems given in corollary 3.3, any ε-approximate heuristic should provide a solution that has zero objective value. This means that the heuristic solves the problem in polynomial time, so $P = NP$.

3.5 SPECIAL CASE STRUCTURES

In the previous section, we saw that each of the earliest start times, due dates and latest start time constraints makes the periodic task assignment problem NP-Hard. In this section, we concentrate on the special case in which each customer requires two operations with a time window requirement for one of them (without any cost function).

Secondly, we will examine another special case that assumes equal process times for all tasks. Here, we consider the general cost function and provide a pseudo-polynomial time algorithm to solve this problem.

3.5.1 PTA Problem: Two operations and a single time window

Here, we assume that each customer requires only two operations and each operation is included in its corresponding task. In addition, only one of the operations has either an earliest start time (EST) or latest start time (LST) requirement. The service planner chooses one of these two operations for each customer. For this subsection, we use $v(s)$ to represent the operation (and task) with an EST or LST for step s and $w(s)$ to represent the operation (and task) without the time window for step s. Finally, we consider the problem with a zero cost function. In other words, we concentrate on the feasibility of the problem.

Proposition 3.5: *Consider the periodic task assignment problem under a zero cost function with m steps. Assume that each customer i requires two operations, v(i) and w(i), and only one of the operations, v(i), is restricted by the earliest start time. Under these conditions, the following statements are true:*

a) *If waiting is not allowed:*

   a1. Selection of w(i) tasks for all customers gives a feasible solution

   a2. If w(i) tasks are not available for some customers, the problem is NP-Hard

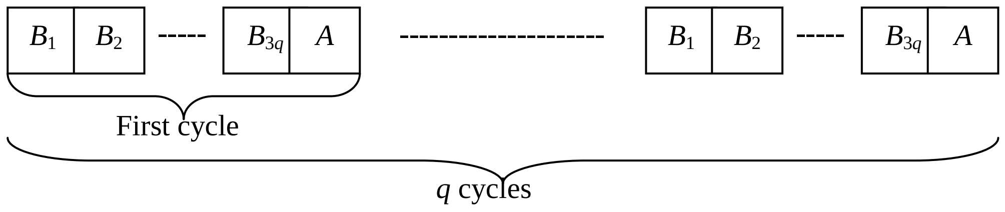
b) *If waiting is allowed:*

   b1. Selection of w(i) tasks for all customers gives a feasible solution

   b2. If w(i) tasks are not available for some customers, then waiting as much as the latest earliest start time of v(i)'s at the beginning and then performing an operation with the earliest start time, v(i), for all customers gives a feasible solution

Proof: (a-b) Arguments are similar as in Proposition 2.5 except part (a2).

(a2) The result comes from the fact that the 3-Partition problem is polynomially reducible to this problem. Consider a sequence consisting of $3q + 1$ different customers: Customer A and customer B_i for $i = 1, \ldots, 3q$. The smallest cycle in the sequence consists of $3q + 1$ customers (B customers are the first ones). The sequence consists of q cycles and a number of steps $m = q(3q + 1)$.



Each customer B_i requires two operations $v(i)$ and $w(i)$, and customer A requires only operation v. For customer B_i, $\delta_{v(i)} = a_i$ and $\delta_{w(i)} = 0$, and for customer A, $\delta_v = 0$.

Moreover, the earliest start times are $\gamma_{v(i)} = 0$ and $\zeta_{v(i)} = q^2b + 1$ for all operations $v(i)$ that belong to customer B_i. Finally, $\gamma_v = \zeta_v = b$ for the operation v of customer A.

We chose large enough subsequent earliest start times $\zeta_{v(i)}$ for the operations of customer B_i, such that each operation $v(i)$ can be done only once. Therefore, the completion time of the sequence is at most qb. On the other hand, the operation v of customer A is available b time later than the operation's last service. That means we have to spend at least b time for each cycle. Hence, the completion time of the sequence is at least qb.

We can conclude that the completion time of the sequence is qb and that each operation $v(i)$ of customer B_i will be done exactly once. Since $\frac{b}{4} < a_j < \frac{b}{2}$, exactly three of the operations $v(i)$ of customer B_i will be done in each cycle. There is a feasible solution to the PTA problem if and only if the 3-partition problem has a solution. That's why we can conclude that the PTA problem with the given restrictions is NP-Hard in the strong sense. Recall that if there is no customer repetition, this problem is solvable in $O(n)$.

Proposition 3.6: *Consider the task assignment problem with m steps. Assume that each customer i requires two operations, v(i) and w(i). Under that assumption, the following problems are NP-Hard both under a waiting and no waiting assumptions:*

a) The problem that has the latest start time for only operation v for every step and has a zero cost function

b) The problem that has no time window (earliest or latest time) requirements and has general cost functions for only operation v

Proof: (a) Here, the latest start times are $\beta_{v(i)} = qb$ and $\eta_{v(i)} = q^2b$ for all operations $v(i)$ that belong to customer B_i. In addition, $\beta_v = \eta_v = b$ for the operation v of customer A. The rest of the parameters are similar to Proposition 3.5(a2). We can define another customer, say customer C, with duration b and add the customer C and customer A q

times at the end of the sequence. This construction will force each operation $v(i)$ of customer B_i to be performed once.

The rest of the proof is similar to Proposition 3.5(a2). Recall that if there is no customer repetition, this problem is solvable in $O(n^2)$.

(b) The construction of the PTA instance with 3-Partition parameters is the same as part (a). Instead of the latest start time parameters, assume that there is a positive cost beyond that time. Here, the question is whether or not there is an objective function with a value of zero. If the answer is yes, then there will also be a solution for the 3-partition problem. The rest of the proof is similar to part (a).

3.5.2 Periodic task assignment problem with equal process times

In the case where process times are equal, neither the start times of customers nor the completion time depends on the task selections. Therefore, the customers do not affect each other. We can partition the problem and solve each customer separately. However, different appearances of the same customer interact with one another, and this makes the problem hard.

Proposition 3.7: *Consider the periodic task assignment problem with n customers. If all the process times of all the tasks of each customer are equal, the problem is NP-Hard both under a waiting and no waiting assumptions.*

Proof: Since all the process times of all tasks of each customer are equal, the start time of each customer and the completion time are known. Therefore, we can partition the original problem for each customer and solve the subproblems separately.

The result comes from the fact that the 3-Partition problem is polynomially reducible to each of these subproblems. Consider a customer that has $3q$ operations and appears q times in the sequence. We construct the tasks so that each task includes only 3 operations and a summation of the attached three a_j parameters is equal to b. In other words, these tasks are valid subsets in the 3-partition problem. The total number of tasks is in $O(q^3)$ and finding valid tasks requires $O(q^3)$ time. Therefore, this construction will take polynomial time.

Let's consider the problem under the no-waiting assumption first. If we select large enough subsequent earliest start times for each operation, each operation cannot be performed more than once. The customer has q visitations, and in each visitation, three operations will be done. Therefore, each operation will be done exactly once. As a result, a feasible solution to this problem is also a feasible solution to the 3-partition problem so the problem is NP-Hard. If waiting is allowed, we can use the latest start time or general cost function structures to prove the same result.

Recall that if there is no customer repetition, this problem is solvable in $O(nk^{max}j^{max})$ time.

3.6 VALID CUTS

The major difficulty in the PTA problem arises when the sequence visits the same customer more than once. This will bring about two important questions to be answered:

- How many times should each operation be done?
- Which tasks should be selected to cover required operations?

We will give the answer of the first question in this section. The second one will be answered in Section 3.7, which describes preprocessor for the problem.

The number of executions required for each operation is determined by the following restrictions:

- **The earliest start time parameters:** This parameter determines the minimum time for the next execution of the operation. Hence, this parameter gives an upper bound for the number of executions required for each operation.
- **The latest start time parameters:** This parameter determines the maximum time available for the next execution of the operation. Hence, the latest start time parameter gives a lower bound for the number of executions required for each operation.

Using these parameters, we can develop the minimum and the maximum requirement constraints that determine the minimum and maximum number of executions that must be performed for each operation in the given planning horizon.

3.6.1 Maximum requirement constraints

The earliest start time parameters γ_k and ζ_k (for the first and subsequent executions) restrict the earliest starting time of the next execution for the same operation. Let h be the completion time and λ_{kh} represent the maximum number of executions of operation k for the completion time h. We observe the following facts:

- If the completion time is $h < \gamma_k$, we cannot perform operation k.
- After the first execution of operation k, we have to wait ζ_k time for each subsequent execution of operation k.

Therefore, we can calculate λ_{kh} as follows:

$$\lambda_{kh} = \left\{ \begin{array}{ll} 1 + \left\lfloor \dfrac{h - \gamma_k}{\zeta_k} \right\rfloor & \text{if } h > \gamma_k \\ 0 & o.w. \end{array} \right\}$$

If the operation cannot be done more than a given number of times, it also cannot be done more than the integral part of that number. Hence, we take the floor of the number. The related maximum requirement constraint is given below:

$$\sum_{j \in JK(k)} \sum_{t \in T} X_{jt} \le \sum_{h \in H} \lambda_{kh} Z_h \qquad \text{for } k \in K,$$

where Z_h refers to the exit indicator. Since the last period (exit time) is varying, the parameter λ_{kh} depends on the exit time h. The constraint sums all the tasks that contain operation k for all times $t \in T$, and makes sure that these tasks are not done more than λ_{kh} for the given planning horizon h.

3.6.2 Minimum requirement constraints

The latest start time parameters β_k and η_k (for the first and subsequent executions) restrict the latest starting time of the next execution for the same operation. Let h be the planning horizon and μ_{kh} represent the minimum number of times that operation k must be performed for the planning horizon h. We have observed the following facts:

- If the completion time $h > \beta_k$, we should perform operation k at least once.
- After the first execution of operation k, we cannot wait more than η_k time for each subsequent execution of operation k.

Therefore, we can calculate μ_{kh} as follows:

$$\mu_{kh} = \left\{ \begin{array}{ll} 1 + \left\lfloor \dfrac{h - \beta_k}{\eta_k} \right\rfloor & \text{if } h > \beta_k \\ 0 & o.w. \end{array} \right\}$$

We take the floor, because we cannot do any operation if the latest start time of that operation is not reached. The related minimum requirement constraint is given below:

$$\sum_{j \in JK(k)} \sum_{t \in T} X_{jt} \ge \sum_{h \in H} \mu_{kh} Z_h \qquad \text{for } k \in K,$$

where Z_h refers to the exit indicator. Since the last period (exit time) is varying, the parameter μ_{kh} depends on the exit time h. The constraint sums all the tasks that contain operation k for all times $t \in T$, and makes sure that these tasks are not done less than μ_{kh} for the given planning horizon h.

3.7 Preprocessor

The difficulty of the problem forces us to find some special structures that can help us in the solution procedure. The most distinctive structure is the fixed sequence that we must follow. With our knowledge of the sequence order, we can tighten the time window of each step, and that will help to reduce the problem size. Recall that s is the index of steps in the visitation order, $s = 1, 2 \ldots m$, and $i \in I$ is the index of customers. Also, it is assumed that t_{max} refers to the latest possible visit time for the last step (i.e. step m), and T is the ordered set of time periods to be considered, $T = \{1 \ldots t_{max}\}$. We define the $TS(s)$ as the set of available time periods to start a task at step s. Our goal in this section is to get a smaller set so that the problem size shrinks. Clearly, the time window $TS(s)$ of each step s heavily affects the number of constraints and variables in our problem.

A primitive method may calculate time windows of steps as in the following fashion: the lower bound of step s can be calculated by summing all of the shortest process times for previous steps s' ($< s$) up to step s. Also, an upper bound can be calculated by summing all the longest process times for previous steps up to step s. However, we can use other information to make the time window $TS(s)$ of each step s smaller.

In the previous section, the minimum μ_{kh} and the maximum λ_{kh} requirements of each operation k were calculated. By using this information, we can determine how much time we will spend at least and at most for each step of the given sequence.

3.7.1 Calculation of the earliest start times for steps

The earliest start times for steps depend on the minimum time that we will spend on each step. Here, we propose a two phase algorithm to calculate those minimum times. In the first phase, we find the minimum duration required for each customer to cover required operations during the planning horizon. In the second phase, we allocate the duration found in the first phase into steps that are visited by the same customer.

The following phases are customer specific, and we need to run these phases for each customer $i \in I$:

Phase I: Minimum duration required for each customer

We know that operation k must be performed at least μ_{kh} and at most λ_{kh} times for the given completion time h. In this section, assume that μ_k represents the lowest μ_{kh} among all h and λ_k represents the highest λ_{kh} among all h, i.e., for $h = t_{max}$.

Let N_j be the decision variable for task j indicating the number of times task j will be performed to cover operations for a given customer. Also, let S_i be the number of visitations of customer i. (Recall that δ_j represents the duration for performing task j, and $KI(i)$ is the set of operations for customer $i \in I$.)

To find the minimum duration requirement for given customer i, we will solve the following IP problem:

$$MTD = \text{Minimize } \sum_{j \in JI(i)} \delta_j N_j$$

subject to:

$$\sum_{j \in JK(k)} N_j \geq \mu_k \qquad \text{for } k \in KI(i)$$

$$\sum_{j \in JK(k)} N_j \leq \lambda_k \qquad \text{for } k \in KI(i)$$

$$\sum_{j \in JI(i)} N_j = S_i$$

$$N_j \geq 0 \text{ and int} \qquad \text{for } j \in JI(i)$$

The first constraint satisfies the minimum requirements for operation $k \in KI(i)$. Similarly, the second one will not to exceed the maximum requirements for operation $k \in KI(i)$. The third constraint sets the number of chosen tasks equals to the number of steps belonging to customer i. The last is for nonnegative integrality.

The IP problem finds the minimum duration that would be spent for a given customer. Since we cannot spend less time without violating requirement constraints, the optimal objective value of this problem could be used to calculate the lower bound of the completion time.

Phase II: Allocation of required duration into steps

In the first phase, we get the minimum time we should spend for each customer. We need to divide this number into corresponding steps for the same customer. Let $\delta_{min}(i)$ and $\delta_{max}(i)$ be the minimum and maximum durations of the tasks that belongs to customer $i \in I$. Also, let $SI(i)$ be the ordered set of steps that correspond to customer i.

The minimum time (found in phase I) allocation procedure for each customer $i \in I$ is given below. (The remaining duration RD refers to the minimum time duration MTD for the given customer subtracted by allocated durations $\sum_{s \in SI(i)} AD_s$):

- Allocate $\delta_{min}(i)$ for all steps except the last one in $SI(i)$.
- Do the following statements for the steps $s \in SI(i)$ in the **reverse order** beginning from the last step in $SI(i)$:
  a) If the remaining duration RD plus already allocated duration AD_s does not exceed the $\delta_{max}(i)$, allocate the remaining duration plus the already allocated duration to the step s ($AD_s = AD_s + RD$) and terminate the procedure.
  b) If the remaining duration exceeds the $\delta_{max}(i)$, allocate $AD_s = \delta_{max}(i)$ to the step s in $SI(i)$. Select the previous step in $SI(i)$ and go to (a).

Since the minimum time we should spend for each customer i is between $\delta_{\min}(i)\,|SI(i)|$ and $\delta_{\max}(i)\,|SI(i)|$, the allocation procedure clearly terminates by allocating at least $\delta_{min}(i)$ time for each step in $SI(i)$.

This allocation procedure runs for each customer and finds how much time we should allocate for each step. Then, the earliest start time of step s can be calculated by summing all the allocations AD_s for previous steps s' ($< s$) up to step s.

The allocation procedure ensures two things: The total time spent in steps of customer $i \in I$ will be equal to minimum time (found in phase I) we should spend for customer $i \in I$. Secondly, the shortest possible times are allocated into earlier steps. Therefore, any solution that satisfies the minimum and maximum requirement constraints will see this allocation as the earliest start times for its steps.

3.7.2 Calculation of the latest start times for steps

There are two cases we should consider for the latest start times with respect to the waiting assumption. If waiting is allowed, then the latest start time for the last customer is at time t_{max}. For each customer from customer $n - 1$ to 1, we find the minimum processing time and subtract this value from the latest start time of the next customer. (These minimum processing times were calculated in the previous section.) This will give us the previous customer's latest start time.

If waiting is not allowed, then the latest start times for each step depend on the maximum time that we should spend for each customer. Similar to the earliest start time case, a two phase algorithm is proposed to calculate those maximum times.

The following phases are customer-specific, and we need to run these phases for each customer $i \in I$.

Phase I: Maximum duration required for each customer

We will solve the same IP problem as we did in the earliest start time calculation section. However in this case, we maximize the same objective function, i.e.;

$$MaxTD = \text{Maximize } \sum_{j \in JI(i)} \delta_j N_j$$

The IP problem finds the maximum duration that would be spent for the given customer by satisfying the maximum requirements of its operations. Since we cannot spend more time without violating requirement constraints, the optimal objective value of this problem could be used to calculate the upper bound of the completion time.

Phase II: Allocation of required duration into steps

In the first phase, we get the maximum time we should spend for each customer. We need to divide this number into corresponding steps for the same customer. Let $\delta_{min}(i)$ and $\delta_{max}(i)$ be the minimum and maximum durations of the tasks that belong to customer $i \in I$. Also, let $SI(i)$ be the ordered set of steps that correspond to customer i.

The maximum time (found in phase I) allocation procedure for each customer $i \in I$ is described below. (The remaining duration RD refers to the maximum time duration $MaxTD$ for the given customer subtracted by allocated durations $\sum_{s \in SI(i)} AD_s$):

- Allocate $\delta_{min}(i)$ for all steps in $SI(i)$.
- Do the followings for the steps $s \in SI(i)$ in **original order** beginning from the first step in $SI(i)$:
  a) If the remaining duration RD plus already allocated duration AD_s does not exceed the $\delta_{max}(i)$, allocate the remaining duration plus the already allocated duration to the step s ($AD_s = AD_s + RD$) and terminate the procedure.
  b) If the remaining duration exceeds the $\delta_{max}(i)$, allocate $AD_s = \delta_{max}(i)$ to the step s in $SI(i)$. Select the previous step in $SI(i)$ and go to (a).

Since the maximum time we should spend for each customer i is between $\delta_{\min}(i)\,|SI(i)|$ and $\delta_{\max}(i)\,|SI(i)|$, the allocation procedure clearly terminates by allocating at least $\delta_{min}(i)$ and at most $\delta_{max}(i)$ time for each step in $SI(i)$.

This allocation procedure runs for each customer and finds how much time we should allocate for each step. Then, the latest start time for step *s* can be calculated by summing up all the allocations AD_s for previous steps s' ($< s$) up to step *s*.

The allocation procedure ensures two things: The total time spent in each step of customer $i \in I$ will be equal to the maximum time we should spend for customer $i \in I$ (found in phase I). Secondly, the longest possible times are allocated into earlier steps. Therefore, any solution that satisfies the minimum and maximum requirement constraints will accept these allocations as the latest start times for the steps.

3.8 Concluding Remarks

In this chapter, we considered the periodic task assignment problem with a fixed customer sequence under the time window and multiple operation requirements. We prove that this problem and almost all the special cases, except perhaps the trivial ones, are NP-Hard in the strong sense. We propose some valid cuts and problem reduction techniques to solve the problem effectively.

In the next chapter, we focus on the field application of this problem and develop techniques to solve it within a reasonable time.

Chapter 4: Maintenance Service Application

4.1 Motivation

This application is motivated by an actual problem faced by maintenance planners at a large company. The company has geographically dispersed infrastructure facilities that require periodic inspection and maintenance to ensure uninterrupted service and effective operation. These maintenance activities require on-site visits by a "service" unit, consisting of skilled workers and equipment. At each site, several components need to be serviced; the desired frequency of service varies by component and facility and depends on the location of the facility, its usage, and other factors. Scheduling the service tasks associated with these inspection and maintenance activities is an important and challenging problem facing this company. In the application that motivated this work, the service planning process begins by deciding a periodic tour for the service unit. This tour specifies the sequence in which the customers will be periodically visited. The tour can visit the same customer multiple times. We address the problem of deciding which task to perform at each site or facility during every visit to that site in order to conform as closely as possible to the desired frequency of service.

Continuing the terminology we established in the previous chapters, we shall refer to these infrastructure facilities to *customers*. Each customer requires multiple service *operations*, but not all operations need to be performed during each visit to the customer, since the desired frequency of service varies by operation. During each visit or step in the sequence, the planner must decide which operations to perform at the customer location. Operations that can be performed together during a visit are grouped together as *tasks*, and each task has a specified duration. For each operation for every customer, we are given a desired frequency or desired time between services for that particular operation. Deviations from this desired frequency are permitted but at a penalty cost. The planner faces the following core tradeoff: performing all or most operations during each visit to a customer helps meet the service frequency requirements for that customer. However, since performing more operations increases the service time at that customer, this strategy delays the time at which the service unit reaches and can begin operations in downstream steps, thereby potentially violating service frequency requirements for these later customers.

In this application, we have lateness costs if the time between services for a particular operation exceeds the desired time interval, as opposed to the time window requirements described in the previous chapter. Therefore, we re-formulate the Periodic Task Assignment problem for this application in a more compact form, which permits us to represent the task selection decisions as a flow of the resource on a time-space network with side constraints to capture penalties if the time between successive executions of each operation exceeds the desired interval. Complexity results both for a “single” visit special case and “multiple” visits are also valid for this application version of the problem (see Appendix B). Since this problem is NP-hard, we develop a fast and effective heuristic procedure that repeatedly applies the shortest path approach developed in Chapter 2 to subsequences that visit each customer at most once. Computational results for several problem instances show that the proposed heuristic identifies near optimal results very quickly, whereas a general purpose integer-programming solver (CPLEX) is not able to solve the problem optimally even after many hours of computational time. Next, we focus on techniques such as problem reduction, branching variables, and subdividing the problem into smaller problems to get better IP solution times for the actual problem. Computational results show that these techniques can improve solution times substantially.

4.2 PROBLEM DEFINITION

For this problem, we will keep the definitions of sets and indices given in the previous chapter. As a reminder, let $I = \{1, 2, \ldots, n\}$ denote the set of distinct service locations or *customers*. A service unit or resource visits these customers in a given periodic sequence or *tour* $S = \{1,\ldots,s,\ldots,m\}$, where m denotes the total number of *steps* (visitations) in the sequence. As a final step of the sequence, we include a dummy step to mark the end of the tour. Each step represents a visit to a particular customer, and the same customer may be visited multiple times in the sequence. Let $i(s)$ denote the customer visited in step $s \in S$. Conversely, $S(i) \subset S$ denotes the subset of steps in the sequence corresponding to visits to customer i. Let $T = \{1, 2, \ldots, t_{max}\}$ denote the planning horizon, i.e., the set of time periods during which the route must be completed, where t_{max} is the latest possible start time for the last step m in the sequence and the first period ($t = 1$) represents the start time of the first step. Depending on the magnitude of service times for various operations (discussed later) and travel times between locations, each period can range from a few hours to days.

Each customer $i \in I$, requires a set of operations $KI(i)$; let K be the set of all operations for all customers. Every operation k for a particular customer i has an associated *relative due date* τ_k that represents the desired time interval between successive executions of operation $k \in KI(i)$. The due date for the first time that operation k must be performed during the planning horizon could be lower than τ_k since this due date depends on the past history of service, i.e., when the operation was last performed before period $t = 1$. Let α_k be the due date for the first execution of operation k. Exceeding the due date incurs a per-period penalty. Let c_{kt} denote the *lateness penalty* if operation k is past due (since its last execution) at period t. Due to technological and policy restrictions, only certain subsets (or combinations) of operations can be jointly performed during a visit. We refer to each such group of operations as a *task j*. Let $JS(s)$ denote the set of permitted tasks during step s for customer $i(s)$, and let $JK(k)$ be the subset of tasks that include operation k. The duration for task j is δ_j time periods; tasks that contain more operations take longer to complete. For convenience, at each step, we also include a "travel" task that corresponds to not performing any task at that step. Depending on the tasks performed at each step, the completion time of the last step may vary. Among the time periods in T, let h be a possible *cycle completion time* to complete all steps and H be the set of possible cycle completion times. In this notation, H is the subset of T and $h \leq t_{max}$.

Performing all the operations that a customer needs during each visit to that customer may be unnecessary (since the relative due dates may not necessitate such frequent service) and may delay operations at other (subsequent) customer sites, thereby incurring lateness penalties. So, the central decision concerns which tasks to perform at each step of the specified tour so as to minimize the total lateness penalties for all customers during the length of the tour. At each step, the service unit must perform one task from among the available tasks $JS(s)$, and we assume that partial services (i.e., fractional tasks) are not permitted. The maximum length of the tour, t_{max}, is the time to complete the tour if the most time-consuming task is done at each step s (i.e., the task with the largest value of δ_j among all tasks that can be performed in step s). Using the problem data (i.e., the smallest and largest task durations at each step), we can determine the interval of time periods in which the service unit will visit each step. Let $TS(s)$ denote the time window for step s, consisting of all periods in which the service unit can arrive and begin its task at step s.

Decision Variables:

X_{jt} = 1 if we *start* task j at time t, and 0 otherwise, for all $s = 1 \ldots m$, $j \in JS(s)$, $t \in TS(s)$;

U_{kt} = 1 if operation k is overdue in period t and cycle is completed after t, and 0 otherwise, for all $k \in K$, $t \in \{\alpha_k, \ldots, t_{max}\}$; and,

Z_h = 1 if the cycle ends in period h, and 0 otherwise, for all $h \in TS(m)$.

We refer to these three variables respectively as *task assignment*, *delay indicator*, and *tour termination* variables. We define the Z_h variables merely for convenience (to make the formulation easier to follow).

Model Formulation for PTA Problem

Minimize $\sum_{k\in K}\sum_{t\in T} c_{kt}U_{kt}$ (1)

subject to:

Task assignment for first step

$$\sum_{j\in JS(1)} X_{j1} = 1 \qquad (2a)$$

Flow conservation constraints

$$\sum_{j\in JS(s-1)} X_{j,t-\delta_j} = \sum_{j\in JS(s)} X_{jt'} \qquad \text{for } s = 2, \ldots m,\ t \in TS(s) \qquad (2b)$$

Exit indicator

$$\sum_{j\in JS(m)} X_{jh} = Z_h \qquad \text{for } h \in H, \qquad (3)$$

Penalties for due date violation of first execution of each operation

$$\sum_{j\in JK(k)}\sum_{t'\in\{1,\ldots,t\}} X_{jt'} + U_{kt} + \sum_{h\in\{1,\ldots,t\}\cap H} Z_h \geq 1 \qquad \text{for } k \in K,\ t \in \{\alpha_k, ..\ \tau_k - 1\}, \qquad (4a)$$

Penalties for relative due date violation of subsequent execution of each operation

$$\sum_{j\in JK(k)}\sum_{t'\in\{t-\tau_k+1,\ldots,t\}} X_{jt'} + U_{kt} + \sum_{h\in\{1,\ldots,t\}\cap H} Z_h \geq 1 \qquad \text{for } k \in K,\ t \in \{\tau_k, ..\ t_{max}\}, \qquad (4b)$$

Integrality

$$X_{jt}, U_{kt}, Z_h = 0 \text{ or } 1 \qquad \text{for } j \in J,\ t \in T,\ k \in K,\ h \in H. \qquad (5)$$

The objective function (1) minimizes the total penalty for due date violations. Constraint (2a) assigns a task to the first step, and constraints (2b) are flow conservation constraints for subsequent steps. These constraints assume that the service unit does not remain idle before any step (with a minor modification, we can incorporate the possibility of waiting before commencing a task at any step). Constraint (3) determines the tour termination time. Although this model requires completing all the steps in the route and treats the tour termination time as a decision variable, we can easily adapt it to situations where the tour duration is specified ahead of time and the tour can terminate before reaching the final step. Constraints (4a) and (4b) serve to identify whether or not each operation k is late at time period t. Constraint (4a) states that if, for $t \in \{\alpha_k, \dots \tau_k\}$, the solution has not performed operation k or completed the tour by period t (since the start of the tour), then the first execution of operation k is overdue. Constraint (4b) captures the relative due date requirement by specifying that if the tour does not end at or before period t and the solution does not perform operation k (or end the tour) within a time interval of τ_k periods prior to period t, then we must incur a lateness penalty for this period. Finally, constraints (5) impose the nonnegativity and integrality requirements.

Proposition 4.1: *For the application version of the periodic task assignment [PTA] problem, the PTA problem is NP-Hard in the strong sense, and the PTA problem where no customer is visited twice is NP-Hard in the ordinary sense.*

Proof: See Appendix B.

4.2.1 Operation level formulation

In the above formulation, constraints (4a) and (4b) contain quite a few numbers of variables in each constraint. We develop an alternative formulation that can represent the objective cost without using the constraint set (4a). In order to do that, we distinguish the variables for the first execution of the operation and the other executions. This formulation is also helpful for constraint (4b) if the number of tasks is much higher than the number of operations. However, we do so at the expense of introducing more variables and relationship constraints.

Additional parameters and indices:

g_{kt} the cost of the first execution of operation k before time τ_k, where t is the first execution time and this cost can be calculated as $\max(0, \min(t-\alpha_k, \tau_k-\alpha_k))$. It represents a penalty for due date violation of first execution of operation k until τ_k.

Decision Variables

X_{jt} =1 if we *start* task j at time t, and 0 otherwise for all $s = 1 \ldots m$, $j \in JS(s)$, $t \in T$

Y_{kst} =1 if we *start* operation k of step s at time t for the first execution, and 0 otherwise for all $s \in FS(s)$, $k \in KS(s)$, $t \in T$

V_{kst} =1 if we *start* operation k of step s at time t for the subsequent executions, and 0 otherwise for all $s \in SS(s)$, $k \in KS(s)$, $t \in T$

U_{kt} = 1 if no tasks containing operation k are performed within the subsequent due date τ_k for operation k, called the *delay indicator* variable for $k \in K$, $t \in T$

Z_h =1 if the last step m starts in period h and 0 otherwise for all $h \in H$, called the *exit indicator* variable

Alternative Model Formulation for PTA Problem

Minimize $\sum_{k\in K}\sum_{t\in T} c_{kt}U_{kt} + \sum_{k\in K}\sum_{s\in S}\sum_{t\in T} g_{kt}Y_{kst}$ (1)

subject to:

Task assignment for first step

$$\sum_{j\in JS(1)} X_{j1} = 1 \qquad (2a)$$

Flow conservation constraints

$$\sum_{j\in JS(s-1)} X_{j,t-\delta_j} = \sum_{j\in JS(s)} X_{jt'} \qquad \text{for } s = 2, \dots m,\ t \in TS(s) \qquad (2b)$$

Job-operation relation constraints

$$Y_{kst} + V_{kst} = \sum_{j\in JK(k)} X_{jt} \qquad \text{for } s \in S,\ j\in JS(s),\ k\in KJ(j),\ t\in TS(s) \quad (2c)$$

Exit indicator

$$\sum_{j\in JS(m)} X_{jh} = Z_h \qquad \text{for } h \in H, \qquad (3)$$

Subsequent due dates for executions of each operation

$$\sum_{s\in SK(k)} \sum_{t'\in\{t-\tau_k+1,\dots,t\}} (Y_{kst'} + V_{kst'}) + U_{kt} + \sum_{h\in\{1,\dots,t\}\cap H} Z_h \geq 1 \qquad \text{for } k \in K,\ t \in \{\tau_k, \dots t_{max}\}, \qquad (4b)$$

Detection of not done operations

$$\sum_{s\in SK(k)} \sum_{t\in T} Y_{kst} + \sum_{h\in H} Y_{ksh} = 1 \qquad \text{for } k \in K, \qquad (5a)$$

$$Y_{ksh} \leq Z_h \qquad \text{for } k \in K,\ h \in H \qquad (5b)$$

Y variables is done earlier than X variables

$$\sum_{s'\leq s} \sum_{t'\leq t} Y_{ks't'} \geq V_{kst} \qquad \text{for } k \in K,\ s \in S,\ t \in T \qquad (6)$$

Integrality

$$X_{jh}, Y_{kst}, V_{kst}, U_{kt}, Z_h = 0 \text{ or } 1 \qquad \text{for } t \in T, j \in J, k \in K, h \in H. \qquad (7)$$

The objective function (1) minimizes the total penalty for due date violations. The second term represents the penalties previously described in constraints (4a). Constraint (2a) assigns a task to the first step, and constraints (2b) are flow conservation constraints for subsequent steps. Constraint (2c) constructs the relationship between job level variables and operation level variables. Constraint (3) determines the tour termination time. Constraint (4b) captures the relative due date requirement by specifying that, if the tour does not end at or before period *t* and the solution does not perform operation *k* (or end the tour) within a time interval of τ_k periods prior to period *t*, then we must incur a lateness penalty for this period. Constraints (5a) and (5b) detect whether the operation is or is not done during the sequence. Constraint (6) ensures that the first execution variables are done earlier in time than subsequent executions. Finally, constraints (7) impose the non-negativity and integrality requirements.

This formulation potentially performs better than the previous one if the number of operations is much smaller than the number of tasks.

4.3 INITIAL SOLUTION

In Chapter 2, we saw that the "single" visit task assignment problem is pseudo-polynomially solvable, whereas the "multiple" visit task assignment problem is NP-hard in the strong sense. The difficulty arises in customer repetition, because in a multi-visit case, the knowledge of the start times of the tasks and the cycle completion time is not enough to give an optimal decision. In other words, the cost structure also depends on the previous decisions for the same customer. Polynomial time algorithms have been developed in special cases of the single-visit problem (Chapter 2, Section 5), but these algorithms do not apply to the multi-visit case (Chapter 3, Section 5).

When there is customer repetition, no (pseudo-)polynomial algorithm can generate an ε-optimal solution for any $\varepsilon > 0$ unless $P = NP$ (see corollary 3.4). Although there is no theoretical bound, (pseudo-)polynomial algorithms may provide good feasible solutions in practice. We take into account the generation of a solution on the basis of the solution of the single visit case. In fact, we can divide the original sequence into parts, so that each part contains a customer only once. These parts are solvable via algorithms developed in Chapter 2.

The heuristics based on dividing the horizon into smaller parts receive attention in the dynamic lot sizing context. Federgruen and Tzur (1994) have demonstrated that for single-item uncapacitated dynamic lot-sizing models, optimal or close-to-optimal initial decisions can be made by truncating the horizon. Stadtler (2003), and Suerie and Stadtler (2003) develop heuristics that solve the multi-item capacitated lot sizing problem over a progressively larger time interval by fixing the variables of a progressively larger number of periods at their optimal values in earlier iterations. The proposed heuristics divide the given horizon of *h* periods into sub-horizons that come one after another. Each sub-horizon is solved optimally after fixing all (or up to some predefined period) variables of previous sub-horizons. The computational tests showed that the heuristics provide promising results. Federgruen et al. (2007) proved that, under some parameter conditions, these heuristics can be designed to be ε-optimal for any desired value of $\varepsilon > 0$ with a running time that is polynomially bounded by the size of the problem. However the theoretical results do not apply to the periodic task assignment problem.

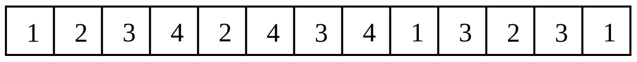
These results suggest that a close to optimal solution may be obtained by partitioning or truncating the customer sequence. In our heuristic, we divide the original sequence into subsequences so that each subsequence contains each customer at most once. An optimal solution to a subsequence can be reached after fixing all task decisions of steps prior to the first step of the current subsequence by solving previous subsequences via algorithms developed in Chapter 2.

We construct the subsequences by using one of these rules:

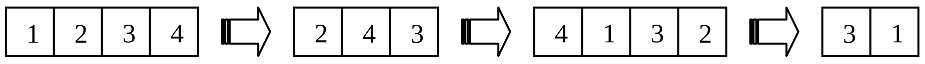
- *Strict partitioning rule*: This rule partitions the original sequence into non-overlapping parts so that each part consists of consecutive steps and contains the same customer only once.

  Example: Suppose that the following is the sequence of customers. (13 steps, 4 customers.) This rule partitions the original sequence into four parts.

Original sequence:



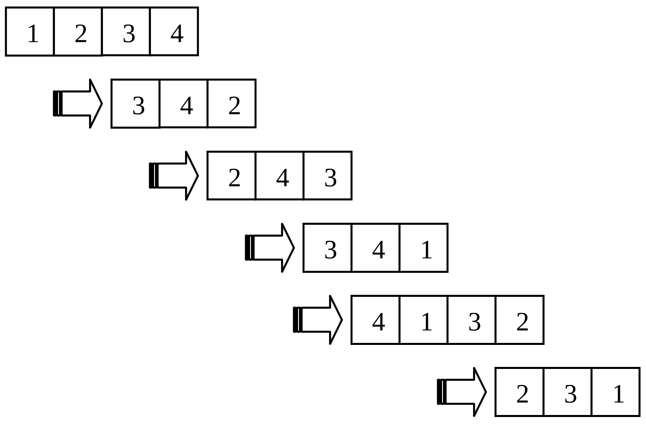
Partitioned into 4 parts:



- *Expanding customer rule*: This rule divides the original sequence into parts, so that each part consists of consecutive steps and contains the same customer only once.
    - o Each subsequence (except the last one) ends if the next step after the last step of the subsequence causes customer repetition.
    - o The next subsequence begins one step after the step that contains the same customer as the next step after the last step of the previous subsequence.

Example: This rule divides the original sequence into six parts.

Divided into 6 parts:



Both strict partitioning and expanding customer rules can provide better solutions with respect to the other one. Dividing the sequence into subsequences in this heuristic looks different than the heuristics developed in the capacitated lot sizing problems. The conditions that can potentially alter the division are as follows:

- The number of customers in each subsequence is not the same. In fact, each subsequence is naturally obtained from the order of the customers in the sequence.
- The division into parts is based on customers, not periods. The time spent on each part depends on the optimal decision, so cannot be fixed a priori.

Our heuristic for the periodic task assignment problem, as discussed in Chaprer 2, is based on the idea of quickly solving each subsequence one after another by applying the pseudo-polynomial (in h, cycle completion time) shortest path method. Although each part of the sequence is solved optimally, the resulting solution might not be optimal for the original problem, because this strategy both in this problem and in the lot sizing context cannot encompass the whole sequence at once and can result in myopic decisions. Using the shortest path method, the algorithm solves each subsequence for each possible subsequence completion time $h \in H$. This method does not predict what will happen after the completion time of the subsequence, which will result in the underestimation of costs in a situation where a customer will not be visited again for a long time. As such, we extend the heuristic to solve this myopia problem.

We modify the shortest path method to improve the heuristic in the following way: assume that a customer i is visited again Δ_i time later than the end of the horizon of the current subsequence. In that case, we use $h + \Delta_i$ instead of h as a subsequence completion time to calculate the arc costs of that customer. Since we do not know how much time will be spent on the later steps, we have to approximate the Δ_i value for each customer i. The approximation assumes a fixed duration time *a priori*, say ε, for each step after the last step of the current subsequence. Hence, if a customer i is revisited a_i

steps later, then the $\Delta_i = a_i\varepsilon$. This ε value could be maximum or average task duration for intermediate steps until next visit.

Finally, the heuristic may generate a solution that is different from the optimal solution for the following reasons:

- the selected task contains an operation which is done early
- the selected task does not contain an operation which is already late
- the selected task contains an operation which is done early, and there is a chance that the operation could be done during that customer's next visit
- the selected task contains an operation which is done late, and there is a chance that the operation could be done during the same customer's previous visit .

Therefore, as a final refinement, we will make the necessary one-task and two-task swaps to get a better solution. In the one-task swap, we change only one task decision of one step at a time. In two-task swaps, two task decisions of two different steps (not necessarily same customer) will be changed at a time and tested for resulting improvement in the objective function. The one-task swap aims to address the first two reasons, and the two-task swap aims to address the last two.

Let ε be the possible approximate duration times (such as minimum, average and maximum duration times) for every step after the last step of any subsequence and E be the set of these ε values. Then, a customer i is visited again $\Delta_i = a_i\varepsilon$ time later than the end of the horizon of the current subsequence where a_i represents the number of steps customer i is revisited after current subsequence. The following is a generic scheme for this heuristic.

Heuristic for periodic task assignment problem

```
0    Set $bestValue = \infty$, $bestSolution$ = empty
```

```
1   Construct the subsequences from the original sequence by using the *strict
    partitioning rule* or *customer expanding rule*. Let $S_p$ represent the $p^{th}$
    subsequence in the order.
2   **For** each $\varepsilon \in E$ **do**
3       **For** each subsequence $S_p$ in the order beginning from $p = 1$ **do**
4           a. Calculate $\Delta_i$ values for each customer $i$ in the subsequence $S_p$
5           b. Solve the problem for the subsequence $S_p$ by using the modified shortest
            path approach problem with $\Delta_i$ values given the assignment for earlier
            subsequences
6           c. Modify the last service date (corresponding $\alpha$ values) of each operation
7       **End For**
8       Calculate the objective function for the original problem (total cost for the
        subsequences could be different than the cost for the original problem)
9       **If** the current objective value of original problem is less than *bestvalue* **then**
10          *bestvalue* = current objective value
11          *bestsolution* = current solution
12      **End if**
13  **End For**
14  **Refinement Step:** Do the one-task or two-task swaps unless there is no
    improvement in the objective function for some predefined number of iterations.
```

The computational performance of the heuristic depends on the set *E*, the set of approximate duration times, and the neighborhood search in the refinement step. There is a tradeoff between the time spent for the heuristic and the quality of the solution. The user can determine the appropriate level by changing the set *E* and using the refinement step.

This heuristic solution provides an initial solution to solve the original problem faster. In the following sections, we develop further methods to speed up this solution process.

4.4 PERTURBATION AND PROBLEM REDUCTION

The special structure of the real problem instances provides us to offer the following perturbation and problem reduction techniques to make these instances faster to solve using CPLEX.

4.4.1 Perturbation

The structure of the objective function contains many equal objective terms, and these terms cause alternative solutions. In fact, the per unit time penalty is equal for all operations and time points for each customer in our application problem instances. This means that the number of distinct objective coefficients for all variables can be as high as the number of customers. Clearly, this causes a huge number of iterations in the LP relaxation solution process.

To decrease the effect of this phenomenon, we made small perturbations in the objective function. We make this perturbation for each customer separately by using following rule:

- For each U_{kt} variable belonging to this customer, rank (arbitrarily) the operations from 1 to $|K|$, denote as r_k, for operation k, where $|K|$ represents the total number of operations for this customer.
- The perturbation value for this variable is $r_k t/|K|t_{max}^2$

Observe that the maximum perturbation for any variable would be $1/t_{max}$, so the total perturbation for any operation will not exceed 1, which will ensure the optimality of the original problem for big enough cost coefficients.

We can also perturb these variables randomly, but the above manual perturbation performed better in our problem instances.

4.4.2 Problem reduction

We develop two methods to reduce the size of the problem. In the first method, we use the following definitions:

Dominated task: For a given task *j*, if there is another task *j'* that includes all the operations that are included by task *j*, and the duration of task *j'* is not greater than task *j*, then task *j* is a *dominated task.*

Proposition 4.2: *There is an optimal solution that does not perform any dominated task in any step.*

Proof: For a given optimal solution, since the problem has penalties only for the lateness of the operations, all the dominated tasks can be replaced by the tasks that dominate them without affecting the optimality of the solution.

We eliminated all the dominated tasks (as many as 20% of all tasks) from our problem instances, which decreased the solution time.

Using the problem data (i.e., the smallest and largest task durations for each step), we can determine the interval of time periods in which the service unit will visit each step. *TS*(*s*) denotes the time window for step *s* consisting of all periods in which the service unit can arrive and begin its task at step *s*. We can narrow this time window further. The next method uses the *nested property* of the tasks in the application instances.

Nested property: For a given task *j*, if there is another task *j'* that includes all the operations that are included under task *j*, then the tasks *j* and *j'* have nested characteristics with respect to each other. If all the task pairs of each customer have nested characteristics, then the problem instance has a *nested property*.

With a nested property in mind, consider a task that contains more operations than any other task for a particular customer. In this case, there is at least one operation in that task that could be performed only by this task. Assume that there is only one operation, operation k, that satisfies this condition. For that operation k, if the difference between the earliest visitation time for the customer to which it belongs in v^{th} visit and maximum cycle completion time t_{max} is smaller than the associated relative due date τ_k for that operation k, then there is no need to perform this operation more than v times as well as the corresponding task. Therefore, we do not always need to use the largest task durations to calculate the upper bound of each time window.

We can generalize this logic to the other operations with a nested property. The following procedure needs to be done for each customer separately, so we drop the customer index from the notation for the sake of simplicity. For each customer, assign an index or order number to each task so that the higher indexed task contains more operations. Let $EST(v)$ be the v^{th} earliest visit time (lower bound of the time window at v^{th} visit) for this customer and n be the total number of visitations for this customer. t_{max} is the current maximum cycle completion time. In the below algorithm, *NumberOfTask* [j] variable holds how many times task j should be performed and *remainingVisit* variable shows how many visitation left to consider for that customer.

Problem size reduction under the nested property for a particular customer

```
0    remainingVisit = n, NumberOfTask [j] = 0
1    For choose task j from highest indexed task to lowest one do
2        For each operation k included in task j but not lower indexed tasks do
3            minVisit = min {v | $\tau_k > t_{max} - \mathrm{EST}(v)$ or v = remainingVisit}
```

```
4               **If** $NumberOfTask\ [j] < minVisit$ **then**
5                   $NumberOfTask\ [j] = minVisit$
6               **End If**
7           **End For**
8       $remainingVisit = remainingVisit - NumberOfTask\ [j]$
9       **End For**
```

This algorithm calculates the number of times each task should be considered in the calculation of upper bounds. The outer "for loop" examines the tasks in decreasing order (longest duration task first, shortest duration task last). The inner "for loop" considers the operations that can only be performed by the current task but not lower indexed tasks. In each iteration of the inner loop, the first visitation where the relative due date of an operation at that iteration is bigger than the difference between lower bound of visitation and t_{max}, is calculated. We know that we do not need to perform that operation after that visit because the operation will not be late at the completion time if it is performed on that visit. Hence, the maximum number of execution for the given task should not be higher than that visitation number.

Previously, we used largest task durations (maximum δ_j values where $j \in JS(s)$ for each step s) to calculate the upper bounds of each time window. The above algorithm provides the number of times we should count each task during the calculation of the upper bounds of each time window using the array *NumberOfTask*. We use the duration of these tasks in decreasing order to calculate the upper bounds of each time window.

For example, if a customer is visited 5 times, and the algorithm provides the 3, 1 and 1 for the tasks that have the longest, second longest and third longest durations respectively, then we use the longest duration for the first three visits, the second longest duration in the 4th visit, and the third longest duration in the 5th visit to calculate the upper bound of each time window for corresponding steps.

This algorithm is performed for all customers, and if t_{max} is decreased at the end, then the process is repeated for the new t_{max}.

Proposition 4.3: *The above algorithm, which accounts for the nested property case, does not eliminate the optimal solution.*

Proof: Step 3 in this algorithm checks the visitation number of the customer, where the selected operation does not need to be done after that visitation. Therefore, we do not need to perform that operation more than the visitation number captured at step 5; therefore, the optimal solution is not cut.

4.5 LAZY CUTS

Lazy cuts are defined as constraints that are part of the original problem but are unlikely to be violated. In the PTA problem, it is unlikely that the largest duration tasks for each step will be performed because the task with a larger duration delays the visitation time of later steps and increases the potential lateness penalties.

Figure 4.1 shows the cycle completion time's cumulative flow distribution for the LP solution of one of the instance we tested with 118 steps. In this instance, the maximum cycle completion time is 524 days, and the optimal solution's (also initial solution's) cycle completion time is 363 days. In its LP solution, the cumulative flow hits its mid point 0.5 at 360 days, and the longest fractional flow ends at 424 days. Therefore, the solution is unlikely to consider all the constraints in (2b), (4a) and (4b) until t_{max}, 524 days.

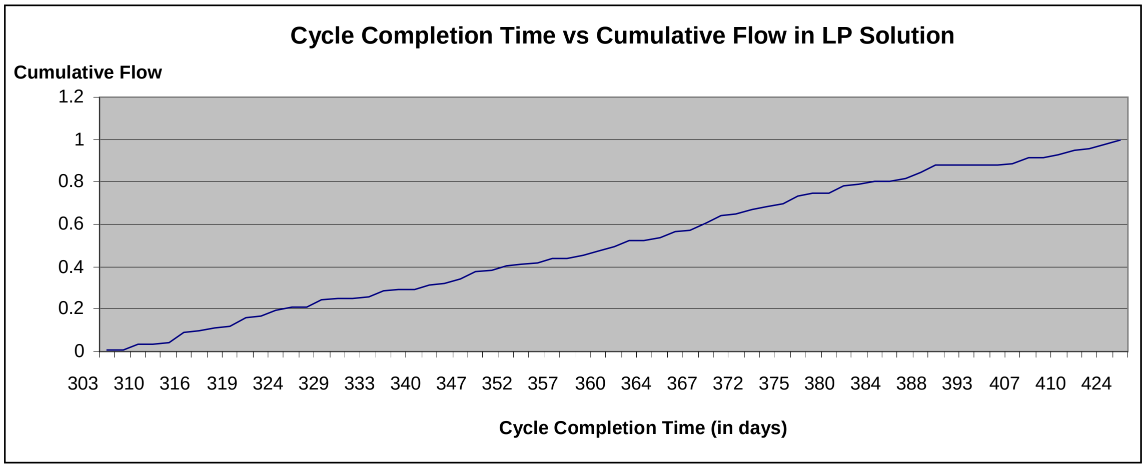


Figure 4.1. Cycle completion time for 118 steps instance in the LP solution

We use the following procedure to declare some constraints to be lazy constraints:

a. Calculate the point at the middle of t_{max} and the cycle completion time of the initial solution, say h^*, which is equal to $h^* + (t_{max} - h^*)/2$

b. Declare all the flow constraints (constraint 2b) to be lazy constraints if the minimum time indexed variable is higher than the value found at (a).

c. Declare all the due date constraints (constraint 4a and 4b) to be lazy constraints if the minimum time indexed variable is higher than the value found at (a).

As an extension, flow constraint declarations could be done separately for each step by considering latest start time of that step and the visitation time of the initial solution at that step.

4.6 BRANCHING STRATEGIES

Branching rules are developed to increase the performance of the solver. The branching strategies are not only based on original variables but also on additional variables that are defined for this purpose. The computational results show that the appropriate selection of branching variables has a significant impact on solution time. We provide these strategies in increasing order of importance below.

4.6.1 Task assignment variables

In an ordinary LP solution, the number of positively valued variables is increasing in later steps of the sequence. Hence, branching on task assignment variables of earlier steps provides a faster solution than branching on task assignment variables of later steps.

To capture this characteristic, the task assignment variables of earlier steps have a higher priority when it comes to branching.

4.6.2 Time variables

When an LP solution selects more than one task with fractional values for a particular step, the completion times of these tasks are different in the next step. We define a new variable set to prevent this phenomenon from happening.

Assume that the minimum task duration time is at least one in a given instance. As such, we cannot start more than one task for a given time. Define the following variables:

W_t = 1 if a task at any step is started at time t, and 0 otherwise, $t \in T$

We refer to these variables as *time* variables. The following constraint relates the time variables to the task assignment variables.

$$\sum_{\{s \mid t \in TS(s)\}} \sum_{j \in JS(s)} X_{jt} = W_t \qquad \text{for } t \in T.$$

The first summation is for steps in which time t is an element of their time windows, and the second summation is for all tasks in those steps. For further improvement, assume that the minimum task duration for all steps appearing in the first summation is $\delta_{\min}$; therefore, the next task cannot start before $t + \delta_{\min}$:

$$\sum_{\{s|t\in TS(s)\}} \sum_{j\in JS(s)} \sum_{t'=t}^{t+\delta_{\min}-1} X_{jt'} = W'_t \qquad \text{for } t \in T,$$

where W'_t equals to one if a task at any step is started between time t and $t + \delta_{\min} - 1$, and 0 otherwise, $t \in T$

Although the LP solution satisfies these constraints anyway, the time variables are useful in branching because number of time variables is much less than the number of task assignment variables. Hence, we give higher priority to time variables over task assignment variables when it comes to branching. In addition, we provide higher priority to earlier time variables compared to the later ones.

4.6.3 Tour termination variables

Although the number of time variables is much less than the number of task assignment variables, we need to define additional variables and constraints. In our problem, we already have tour termination (sink) variables that determine the cycle completion time. Branching on these variables before than time variables has pros and cons:

- **Pros:** We do not need to define additional variables and constraints. The number of sink variables is much less than the number of time variables.
- **Cons:** Even though the solution has integer valued sink variables, it does not guarantee the feasibility of the solution.

Computational results show that sink variables are more powerful tools than time variables, so we give them higher priority than time or task assignment variables. As usual, earlier sink variables have higher priorities compared to the later ones.

4.6.4 Cumulative sink variables

Although the tour termination (sink) variables help us with our branching strategies, every sink variable Z_h appears with *due date* constraints for all operations and times if the time index h of the corresponding sink variable is less than the time t corresponding to the delay indicator variable U_{kt} at that constraint. This fact causes dense columns and rows in our constraint matrix and makes it difficult to conduct matrix operations. Therefore, we define the following variables:

CZ_h = 1 if the last step m starts at or before period h, and 0 otherwise for all $h \in TS(m)$.

Instead of defining a variable Z_h to indicate tour termination at time h, new *cumulative sink* variables CZ_h become one if the cycle completion time of the tour is equal to or less than h. Using these variables, we can change the constraints (3), (4a) and (4b) in our formulation such that:

$$\sum_{j \in JS(m)} \sum_{h' \leq h} X_{jh} = CZ_h \qquad \text{for } h \in H, \qquad (3`)$$

$$\sum_{j \in JK(k)} \sum_{t' \in \{1,\dots,t\}} X_{jt'} + U_{kt} + CZ_t \geq 1 \qquad \text{for } k \in K, t \in \{\alpha_k, .. \tau_k - 1\}, \qquad (4a`)$$

$$\sum_{j \in JK(k)} \sum_{t' \in \{t-\tau_k+1,\dots,t\}} X_{jt'} + U_{kt} + CZ_t \geq 1 \qquad \text{for } k \in K, t \in \{\tau_k, .. t_{max}\}. \qquad (4b`)$$

The due date constraints now contain only one cumulative sink variable instead of the summation of sink variables. This change will reduce the density of the constraint matrix and improve the solution time.

We give cumulative sink variables higher priority than the other variables. Similarly, cumulative sink variables with earlier time index have higher priorities compared to the later ones.

4.7 SUBDIVISION METHOD

In the above strategies, we concentrate on the full problem and try to improve its solution time. However, since our constraint matrix is not sparse, solution times increase rapidly with the size of the problem.

Hence, we generate the subproblems from the original problem, so that each of them can be solved faster. The idea behind subproblem generation is to force a start time of a task for a chosen step, say *division step*, before or after a chosen time, say *division time*. The following scheme shows subproblem generation:

- We select three steps; $\left\lfloor \frac{m}{3} + 0.5 \right\rfloor$, $\left\lfloor \frac{2m}{3} + 0.5 \right\rfloor$ and m, as division steps where m is the last step. For example, if a sequence has 100 steps, then division steps are 33, 67 and 100.
- For division steps, we select division times t_1, t_2, t_3 to start a task. With the 100 step example in mind, let t_1 = 100, t_2 = 200 and t_3=300.
- We generate 8 subproblems (2^3) by declaring that these division steps should start a task before or after those selected division times. (In each subproblem, selected times are included.) Continuing the example, Table 4.1 shows the starting times of tasks for each step in each subproblem.

	Division Steps		
	33	**67**	**100**
Subproblem 1	**≤100**	**≤200**	**≤300**
Subproblem 2	**≤100**	**≤200**	**≥300**
Subproblem 3	**≤100**	**≥200**	**≤300**
Subproblem 4	**≤100**	**≥200**	**≥300**
Subproblem 5	**≥100**	**≤200**	**≤300**
Subproblem 6	**≥100**	**≤200**	**≥300**
Subproblem 7	**≥100**	**≥200**	**≤300**
Subproblem 8	**≥100**	**≥200**	**≥300**

Table 4.1. Starting times of tasks in each subproblem

One can select fewer or more division steps depending on the problem size. The effectiveness of this method relies on the appropriate time selection of the division steps, because bad time selections may generate a subproblem which is as hard as the original problem. Hence, we offer two division methods using either the initial solution or the LP solution.

4.7.1 Division by initial solution

In Section 4.3, we described a procedure to find an initial solution. The task starting times in this initial solution could be used as the selected times for division steps. There are pros and cons for this selection:

- **Pros:** Each subproblem has a valid initial solution generated for the original full size problem. We do not need to deal with the full size problem at all.
- **Cons:** Theoretically, the initial solution might not be close enough to the optimal solution. It may cause bad time selections, and the resulting subproblems could be hard to solve.

In actual computational tests, this method works very well if the initial solution is close to the optimal solution.

4.7.2 Division by LP solution

The LP solution is another alternative for selecting those time points. We can look at the cumulative flows in time for these division steps and select the time at mid flow 0.5 which is expected to be close to the optimal solution. Here are the pros and cons of this selection:

- **Pros:** Selected times are expected to be close to the optimal solution.
- **Cons:** Each subproblem may not have a valid initial solution. Furthermore we at least need to solve the LP relaxation of the full size problem.

In computational tests, this method performed better when we intentionally provided an initial solution that is far away from the optimal solution. However, with the initial solution at hand (from Section 4.3), division by initial solution method performed much better at all instances.

4.8 COMPUTATIONAL RESULTS

We evaluate the performance of our approaches on real data instances. The heuristic was programmed in Java and the IP model uses CPLEX 11.2 via concert technology. The tests are on the 11 Dell Poweredge 2950 workstation with 3.73 GHz Xeon and 24 GB of shared memory under Ubuntu Linux system.

In our experiments, we tested on two maintenance regions, called Region-A and Region-B. Region-A contains 34 facilities (customers) requiring maintenance and these facilities are visited in 58 steps in one full cycle. (The service “unit” repeats the tour after 58 steps.) Region-B also contains 34 maintenance facilities and one full cycle consists of 59 steps. Each full cycle approximately takes 6-months to complete in each region. The durations of the tasks can range from half day to 14 days, and the weights of the operations (depend on workload of facilities) vary between 7 and 189.

We generate 6 instances for each region. All instances of each region has same starting conditions (first customer and customer order in their sequence, and due date of operations of customers) but contains different number of steps in their sequence. In Table 4.2, the first column shows the problem names. The first letter refers to region name and the number indicates the number of steps in that instance. Second column states the number of customers considered on that instance and the other columns give details about the number of visitations. For example, problem A-58 has 34 customers.

13 of these customers are visited once, 18 of them are visited twice and 3 of them are visited three times.

Problem	customers	1 visit	2 visits	3 visits	4 visits	5 visits	6 visits
A-20	20	20	0	0	0	0	0
A-40	30	20	10	0	0	0	0
A-58	34	13	18	3	0	0	0
A-80	34	6	13	12	3	0	0
A-100	34	2	11	11	7	3	0
A-116	34	0	13	0	18	0	3
B-20	20	20	0	0	0	0	0
B-40	30	20	10	0	0	0	0
B-59	34	10	23	1	0	0	0
B-80	34	5	12	17	0	0	0
B-100	34	3	7	13	11	0	0
B-118	34	0	10	0	23	0	1

Table 4.2. Problem instances for region A and region B

In Table 4.3, the details about IP formulation is given for each instance developed in Section 4.2 without initial solution, preprocessing, additional variables/constraints and methods discussed in Section 4.3 to Section 4.7. The strict partitioning rule is used as a subsequence method for the heuristic.

For each instance, number of constraints, variables and nonzero coefficients are given. Then, the IP, LP and heuristic solution times are presented. Finally, the gap (* = (heuristic value – optimal value)/optimal value) is calculated on the last column.

The heuristic finds the optimal solutions in 10 instances and the biggest difference between the heuristic value and optimal solution is 1.4%. As the problem sizes increase, the solution times increase enormously.

Problem	Number of			IP solution time (sec)	Root solution time (sec)	Heuristic solution time (sec)	Heuristic vs. optimal value*
	constraints	variables	nonzeros				
A-20	2K	3K	67K	2	1	6	0.0%
A-40	9K	13K	718K	87	74	20	0.0%
A-58	14K	22K	1.4M	437	400	54	0.0%
A-80	21K	35K	3.1M	6946	2492	287	0.4%
A-100	32K	57K	7.0M	123847	31288	554	0.0%
A-116	39K	70K	9.5M	21274	20931	588	1.4%
B-20	3K	3K	62K	3	1	6	0.0%
B-40	10K	13K	668K	123	83	22	0.0%
B-59	19K	26K	2.0M	2231	2170	25	0.0%
B-80	28K	40K	4.0M	1.3M	7467	101	0.0%
B-100	39K	58K	7.6M	>1.8M	23022	84	0.0%
B-118	49K	76K	11.2M	>1.8M	49835	683	0.0%

* = (heuristic value – optimal value)/optimal value

Table 4.3. CPLEX and heuristic performances

Problem	Initial gap*	Number of branches	Problem	Initial gap*	Number of branches
A-20	8.06%	0	B-20	0.00%	0
A-40	0.00%	0	B-40	31.44%	0
A-58	0.00%	0	B-59	0.00%	0
A-80	15.25%	185	B-80	30.69%	75702
A-100	31.76%	839	B-100	39.61%	>28250
A-116	0.00%	0	B-118	62.88%	>6100

* = (First feasible solution – Root LP solution)/ First feasible solution

Table 4.4. Initial gaps and number of branches

In Table 4.4, initial gap ((= First feasible solution – Root LP solution)/ First feasible solution) and number of branches in the B&B tree are provided for the problems solved in Table 4.3. We did not provide any initial solution for this set.

We use the methods developed in Section 4.3 to Section 4.6 and obtain substantial improvements on solution time seen in Table 4.5.

Problem	**Number of**			**IP solution time (sec)**	**Root solution time (sec)**	**Heuristic solution time (sec)**	**Heuristic vs. optimal value***
	constraints	**variables**	**nonzeros**				
A-20	**2K**	**2K**	**36K**	**2**	**1**	**3**	**0.0%**
A-40	**9K**	**11K**	**381K**	**12**	**9**	**6**	**0.0%**
A-58	**13K**	**16K**	**759K**	**30**	**24**	**10**	**0.0%**
A-80	**20K**	**26K**	**1.7M**	**201**	**112**	**28**	**0.3%**
A-100	**31K**	**41K**	**3.9M**	**978**	**109**	**118**	**0.0%**
A-116	**37K**	**48K**	**5.3M**	**977**	**915**	**128**	**0.0%**
B-20	**3K**	**3K**	**35K**	**1**	**1**	**2**	**0.0%**
B-40	**10K**	**12K**	**390K**	**13**	**10**	**7**	**0.0%**
B-59	**19K**	**24K**	**1.2M**	**101**	**91**	**9**	**0.0%**
B-80	**27K**	**35K**	**2.5M**	**482**	**53**	**29**	**0.0%**
B-100	**38K**	**50K**	**4.8M**	**27051**	**18991**	**106**	**0.0%**
B-118	**48K**	**66K**	**7.4M**	**80405**	**31651**	**424**	**0.0%**

* = (heuristic value – optimal value)/optimal value

Table 4.5. Improved CPLEX performances

The following approaches are considered:

- Use heuristic solution described in Section 4.3 as an initial solution,
- Apply perturbation and problem reduction techniques described in Section 4.4,
- Apply lazy cuts as in Section 4.5,
- Use following branching strategies:

- First, branch on cumulative sink variables (see Section 4.6.4),
- Second, use time variables (see Section 4.6.2),
- Finally, apply task assignment variables (see Section 4.6.1).

In Table 4.5, the number of nonzero coefficients is reduced approximately 40-50% with the help of problem reduction techniques and cumulative sink variables. Furthermore, the solution times are decreased 99% in some of the instances. Besides the solving smaller problem, initial solution and branching strategies, especially cumulative sink variables, provide these good results.

In Table 4.6, initial gap ((= Heuristic solution – Root LP solution)/ Heuristic solution) and number of branches in the B&B tree are provided for the problems solved in Table 4.5.

Problem	Initial gap*	Number of branches	Problem	Initial gap*	Number of branches
A-20	0.18%	0	B-20	0.00%	0
A-40	0.00%	0	B-40	0.14%	0
A-58	0.00%	0	B-59	0.00%	0
A-80	0.65%	10	B-80	0.77%	32
A-100	0.40%	35	B-100	1.44%	91
A-116	0.00%	0	B-118	2.97%	110

* = (Heuristic solution – Root LP solution)/ Heuristic solution

Table 4.6. Initial gaps and number of branches

Despite of major improvements at hand, we can further reduce the solution time by using subdivision method developed in Section 4.7. We generate 8 subproblems following the guidelines of Section 4.7 by using heuristic solution. (We continue to use approaches that are considered in Table 4.5.)

In these subproblems, we concentrate on the instances that have 80 steps or more. Instead of using smaller instances (that are already solved quickly), we generate the additional problems using following logic: First, we solved the instances of two full cycle problems optimally (116-step problem for region A and 118-step problem for region B). Then, we assume that the service planners perform the tasks for one full cycle by using these optimal solutions. We update the due dates of operations based on the task decisions and cycle completion times of these solutions. Finally, we obtain the new instances with the new starting conditions.

In Table 4.7, we see the performance of the subdivision method by using initial solution. The new instances are referred to an additional letter "R". Our heuristic finds optimal solutions for 5 instances and there is a 0.9% difference between the optimal solution of the "B-100-R" problem and the heuristic solution. Table 4.7 also shows the worst initial gap and maximum number of branches needed to solve IP among all subproblems generated by subdivision method.

Problem	IP solution time (sec)	Initial gap*	Number of brances	Problem	IP solution time (sec)	Initial gap*	Number of brances
A-80	169	0.26%	0	A-80-R	219	8.43%	38
A-100	532	0.25%	25	A-100-R	502	1.69%	17
A-116	597	0.00%	0	A-116-R	329	0.00%	0
B-80	416	0.00%	0	B-80-R	357	0.02%	0
B-100	1406	0.55%	6	B-100-R	1265	1.21%	6
B-118	2481	0.06%	0	B-118-R	3949	0.62%	0

* = maximum of {(Heuristic solution – Root LP solution)/ Heuristic solution} among all subproblems

Table 4.7. Performance of subdivision method

Solving problem instance B-118 takes 23 hours without considering subdivision method and now, it is taking less than an hour. The performance of the subdivision method highly depends on the division steps and times. The closer these time points are to their optimal values, the better performance we get. Therefore, it is very important to have a good initial solution to apply this method.

4.9 CONCLUSION

In this chapter, we considered the maintenance service application of the periodic task assignment problem. We develop a heuristic that use shortest path approach given in Chapter 2 as a subroutine. Computational results show that the heuristic can provide near optimal solutions. We also propose problem reduction techniques to solve the problem effectively. We further improve the solution time by investigating on techniques such that lazy cuts, branching variables, and subdividing the problem into smaller problems. We show that these techniques can provide substantial improvements on solution time.

Chapter 5: Shipment Routing Problem with Dispatching Policies

5.1 INTRODUCTION

In transportation systems, planning and executing the transportation of shipments involves many complex decisions and requires the management of multiple resources. Appropriate management of these resources is necessary to improve service quality while ensuring efficient use of resources and satisfy customer orders on time. The decision makers need to solve many interrelated problems, such as the design of the underlying network, the routing and timetabling of the carriers and the transportation of the shipments.

In this chapter, the focus will be on the problem of routing shipments that need to be transported from their origin to their destination. The shipment routing problem determines the path (physical and temporal) that each shipment will use on its journey. We investigate the transportation problem of shipments from their origin to their respective destinations under capacity constraints and dispatching policies. The effective usage of available capacity under a given network decreases costs and increases customer satisfaction. Dispatching policies determine the handling rules of shipments on intermediate stations during their trips. We assume that higher level decisions in the network, such as capital investment and the carrier schedule, are given.

Complex network systems usually consist of multi-layer (physical and logical) networks to handle traffic. In the physical layer, the actual transmitting network is designed such as location of stations/airports and schedule of carriers in transportation network, and location of routers and fiber optic lines in communication network. The logical network is designed over physical network to handle traffic effectively such as IP networks in computer networks.

In transportation network, a shipment may pass through many classification stations on its route from origin to destination. At these stations, the station operators reclassify the incoming traffic to be placed on outgoing carriers. Each reclassification takes time and incurs handling costs. Instead of reclassifying shipments at every station on its route, several shipments may be grouped together to form a *block*, a term barrowed from railway terminology. A block has its own origin–destination pair that may be different from the origin-destination pair of individual shipments contained in the block. Therefore, a shipment may be assigned to more than one block to reach its destination. With this blocking mechanism, the shipments are classified only at the origin of the blocks to which they are assigned. (See Cordeau et al., 1998 and Ahuja et al., 2005 for multi-layer network designs in railway applications). In communication network, different fiber links group together to form a *trunk* to handle data packages. Again, the data is reclassified only at the origin of the trunks. (More information about multi-layer communication networks can be found in Pioro and Medhi, 2004.)

After the design stages of physical and logical networks, the next step determines the possible carrier assignments within the planning horizon for blocks. During the block construction, carrier scheduling and possible carrier assignments, the forecasted shipments are considered. At the final stage, the shipment routing problem determines carrier assignments for actual (not forecasted) shipments among the possible carriers generated in the design stages.

The shipment routing problem has some important practical constraints that the trip planner should consider:

- *Carrier capacities*: The carriers have capacities between two stations they travel. These capacities could be different during the trip of carriers. If the capacity requirements are not considered, the last minute adjustment will change the original routing plan given to the customer, so these changes may affect customer satisfaction. Moreover, myopic decisions may result in ineffective use of resources, especially in congested systems. In our network structure, multiple blocks could be assigned to a single carrier and a single block could be assigned to multiple carriers. If the arcs are considered as carrier to block assignments for a particular shipment in a given network, multiple arcs can share same resource. This structure is different than the standard multi-commodity flow problems where each arc has its own capacity.

- *Dispatching policies*: The other issue concerns rules about shipment interactions on the network. If two shipments are assigned to the same block, then a shipment that comes to a station earlier should not be assigned to a carrier that departs later than the one carrying a shipment that arrived later. This is a practical constraint in many networks, and simply states first-in-first-out (FIFO) rule for that station if the shipments are using same departing block. Clearly, an approach that designs the routing plan for each shipment independently cannot handle this rule.

  Note: A dispatching policy can be designed to handle arbitrary order of the shipments (not necessarily FIFO).

We consider a problem that requires the following items as inputs:

- Shipments with their release times and volumes
- Set of blocks that can carry each shipment
- Legitimate Block-to-Carrier assignments for each block
- Carrier capacities (could vary by location even for same carrier)
- Dispatching policies

The shipment routing problem determines which among the possible Block-to-Carrier assignment should be assigned to each shipment considering capacities and dispatching policies. The objective function is to minimize the total weighted transit times of shipments from their origin to destination.

This chapter focuses on a shipment routing problem under capacity and dispatching policies. We consider a space-time network that allows one to formulate the shipment routing problem as a multi-commodity network flow problem with additional side constraints. We explore alternative models and develop methodologies for routing decisions. We propose algorithms and techniques that can solve real size shipment routing problem to optimality or near-optimality.

The remainder of this study is organized as follows: The next section provides the background and literature review for shipment routing problems and related multi-commodity network flow problems. Section 3 defines the problem and formulates the shipment routing problem, and Section 4 investigates the characteristics of three different dispatching policy constraints. Section 5 proposes an alternative formulation, and Section 6 shows complexity results. Section 7 concentrates on heuristics and lower bounds, and Section 8 provides computational results.

5.2 LITERATURE REVIEW

Shipment routing problems appear in many network applications such as transportation and telecommunication. These problems are operational level problems and handled after designing the physical and logical networks.

Railroad *trip planning* problems are one of the applications of shipment routing problems. In railroad planning and scheduling, Assad (1980) presents the hierarchical structure of decision problems in railroads. Cordeau et al. (1998) and Ahuja et al. (2005) give a recent survey of railroad network design problems.

In railroad planning hierarchy, the railroad blocking problem and the train scheduling problem should be solved before solving the trip planning problem. The blocking problem which involves grouping shipments into blocks is the primary planning problem in the railroad industry for logical network generation. Newton et al. (1998), Barnhart et al. (2000), and Ahuja et al. (2007) work on this problem. After a railroad has developed a blocking plan, designing a train schedule is the next operational planning task. Related works can be found in Farvolden and Powell (1994), Campbell (1996), Kraft (1998), and Brannlund et al. (1998).

To the best of our knowledge, there are few papers in the literature related to the trip planning problem (see Van Dyke 1992, 1994) which corresponds to shipment routing problem in railway applications. Nozick and Morlok (1997) study the shipment-to-train assignment problem and the problem of repositioning empty cars together for a given train schedule without considering any blocking plans. They consider the objective function of minimizing the total movement cost of cars while satisfying due date constraints. They formulate the problem as an integer program over a time-space network, and propose a heuristic based on the linear programming relaxation. The heuristic rounds up or down some of the fractional values and reruns the linear programming relaxation until a feasible integral solution is found.

Kwon et al. (1998) consider the shipment-to-block assignment and the trip planning problems for a given train schedule under train capacity constraints. They formulate the problem as a linear multicommodity flow problem and use column generation as a solution approach. They formulate the multicommodity flow problem using path flows for every shipment from its origin to its destination. During column generation, the restricted master problem is solved for a subset of the paths, and the subproblems are represented as shortest path problems for every shipment from its origin to its destination. In their computations, they use a network with 12 stations and 16 trains.

Jha et al. (2008) deal with the trip planning problem for a given block plan and a train schedule subject to train capacity constraints. They develop arc-based and path-based multicommodity network flow formulations of the problem. In their model, they assume that all the trains run every day, and all the blocks are made every day. The formulations are defined in a time-space network in which every node is distinctly identified by place, time and train within the block. They connect the last node to the first node at the same station and obtain a one-day network with wrap around arcs.

The path based formulation given in Jha et al. (2008) considers the potential paths for each block. This approach is different than Kwon et al. (1998) because Kwon et al. define the potential paths for each shipment. Since a shipment can use multiple blocks in its blocking plan, the formulation in Jha et al. (2008) has fewer path variables than the formulation in Kwon et al. (1998). However, the path based formulation in Jha et al. requires connection arcs between blocks to capture the transit times of the shipments. Jha et al. (2008) do not generate these connection arcs but instead assume that there is only one release time for each block in a day. This assumption reduces the problem to the block level so that the problem assigns one path for each block under train capacity constraints.

Jha et al. propose exact and heuristic algorithms based on the path-based formulation. Their exact algorithm solves an integer programming formulation based on a branch and price approach. The columns are generated either a priori or dynamically. They also develop Lagrangian relaxation-based heuristic method and present computational results using the data provided by a major U.S. railroad. (Data consists of around 1200 blocks and 350 trains.)

The trip planning problem also appears indirectly in the problem structures of a few papers. These efforts determine the routing and frequency of trains (but not the actual departure times of the trains), and the block to train assignments together. The blocking policy may be either determined within the model or given as an input. Thomet (1971) develops a cancellation procedure that gradually replaces direct shipments with a series of intermediate train connections in order to minimize operation and delay costs. Crainic et al. (1984), and Crainic and Rousseau (1986) propose a nonlinear, mixed integer, multi-commodity flow model that deals with the interaction between blocking, block to train assignments and train and traffic routing decisions. The model specifies the feasible routes on which train services may be run and defines a set of feasible trip plans for each origin-destination pair. Haghani (1987, 1989) develops a formulation and heuristic decomposition approach for combined train routing, block-to-train assignment and empty car distribution problems. Keaton (1989) develops a heuristic method based on Lagrangian relaxation for the combined problem of car blocking, train routing, and block-to-train assignment. He obtains subproblems that can be represented as shortest path and knapsack problems. Keaton (1992) additionally considers constraints for blocking and maximum transit time for each origin–destination pair. Martinelli and Teng (1996) propose a neural network approach to solve the train routing and the shipment to train assignment problems. The problem is formulated as a nonlinear integer program that minimizes the total time spent by shipments in the system. Marin and Salmeron (1996) consider the train routing and the shipment-to-train assignment problems and develop three heuristic methods: the descent method, simulated annealing, and tabu search. In their computational tests, simulated annealing obtained the best solutions but required more time than the other heuristics.

In communication networks, data transfer over multi-layer networks. Related problems about multi-layer network design in communication networks can be found in Pioro and Medhi (2004) and Orlowski (2009). After the design of communication network, data packages are routed on this network following network policies.

The shipment routing problem is often formulated as a multicommodity network flow problem with additional side constraints. In fact, the shipment routing problem without capacity and dispatching constraints is simply solvable via shortest path algorithms. However the shipment routing problem differs from the standard multicommodity flow problem when we consider the capacities such that each capacitated resource may be usable by multiple arcs and each arc may be consists of multiple resources in the shipment routing problem.

The multicommodity network flow problem is one of the classical problems in the literature since the publication of Ford and Fulkerson (1958). Many of the approaches were developed in the 1960s and 1970s. Assad (1978) and Kennington (1978) are excellent survey papers that describe several algorithms and standard properties of multicommodity flow problems. Additionally, Ahuja et al. (1993) present several solution procedures. Decomposition techniques have been used extensively in solving large multicommodity flow problems. Barnhart et al. (1995) and Jones et al. (1993) develop column generation models for linear multicommodity flow problems. Barnhart et al. (2000) propose a branch-and-price-and-cut algorithm for integer origin-destination multicommodity flow problems. Crainic et al. (2001) develop the Lagrangian relaxation technique, and their experiments show that the bundle methods appear superior to subgradient approaches. Castro (2000) considers the interior point algorithm to solve linear multicommodity flow problems. There are also multicommodity flow problems with convex costs, and Ouorou et al. (2000) give an excellent survey of this area.

The shipment routing problem is a large scale problem that is hard to solve. To the best of our knowledge, there is no study done in the literature for the shipment routing problem with capacity and dispatching policies. Without dispatching policy constraints, the efforts in the railway literature either solve small problem instances (Kwon et al, 1998) or deal with simplified versions of this problem (Jha et al., 2008). In this chapter, we develop approaches to solve real-size problems in a reasonable time.

5.3 PROBLEM DEFINITION

In this section, we will formulate the shipment routing problem as a multicommodity network flow problem over a time-space network. Traditionally, there are two formulations developed for multicommodity network flow problems: Arc-based and path-based. The arc-based formulation is a standard method for formulating a multicommodity network flow problem. The path-based formulation is commonly used with column generation techniques because its constraint set is smaller than the constraint set of arc-based formulation, but its variable set is far bigger.

Several side constraints in multi-layer networks make the arc-based formulation impractical in solving shipment routing problems. To be able to construct relations between shipments, we need enormous number of constraints in the arc-based formulation. Conversely, the number of variables in the path-based formulation can be in the billions. Therefore, we formulate the shipment routing problem by using a hybrid approach. We expand the arc definition and consider only those stations where the dispatching constraints take place. This approach handles the practical constraints with an affordable number of variables.

Let W be the set of all shipments that are customer orders and should be transported from their *origin* to *destination*. Each shipment $w \in W$ has the following attributes:

o_w	Origin station of shipment w; $w \in W$
d_w	Destination station of shipment w; $w \in W$
rt_w	Release time (available time to use) of shipment w; $w \in W$
v_w	Volume of shipment w; $w \in W$

A shipment travels from its origin to destination using appropriate blocks. Let B be the set of all blocks, and let $BW(w)$ represent the set of all blocks that can carry shipment w. Each block $b \in B$ has the following characteristics:

ob_b	Origin station of block b; $b \in B$
db_b	Destination station of block b; $b \in B$

A block is carried by a sequence of carriers during its path. The route of a carrier between two consecutive stations is called a *resource* that subject to capacity. The volume capacity of each resource is considered as a resource capacity. Let R represent the set of all resources, with each resource indexed as r.

Each possible sequence of carriers for a block from its origin to destination is called a *link*, a term barrowed from communication network. Each link can be represented by the block's origin, the block's destination, the departure time of a carrier that carries the block from its origin, the arrival time of a carrier that carries the block to its destination, and the resources passed on the block's route.

Let P be the set of all links. Each link $p \in P$ can be related to shipments and resources. We define the following sets to represent relationships between shipments and resources:

$PW(w)$	Set of links that can be used by shipment w; $w \in W$

$WP(p)$ Set of shipments that can use link p; $p \in P$

$PR(r)$ Set of links that use resource r; $r \in R$

Figure 5.1 shows the representation of shipments, blocks, links and their relationships discussed so far.

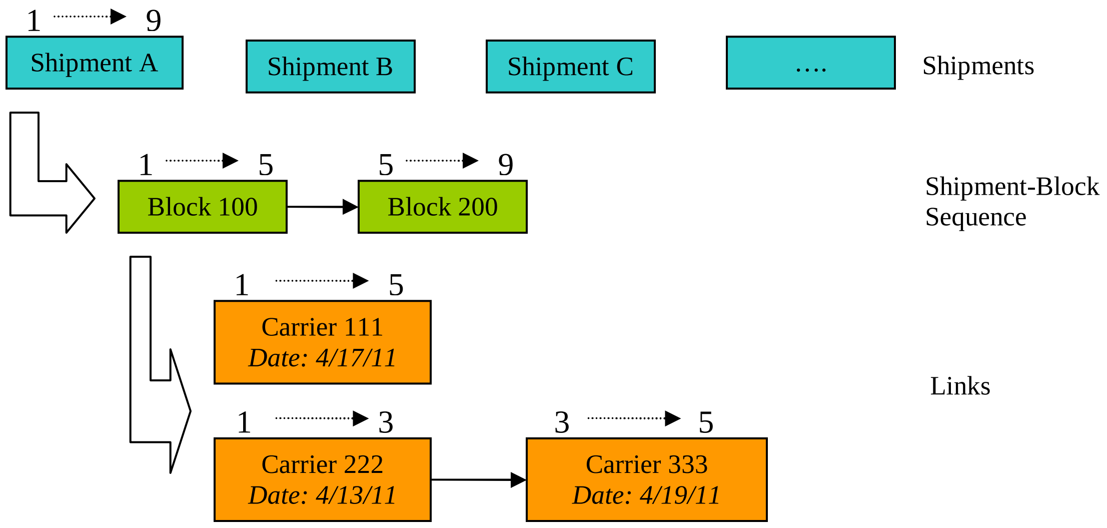


Figure 5.1. Shipments, Blocks and Links

The shipment routing problem uses all of the information given above as input and generates a solution that assigns shipments to links by considering following side constraints:

- **Capacity constraints:** Resource capacity cannot be violated.
- **Dispatching policy constraints:** Among two shipments, a shipment that arrived at the station earlier cannot go after one that arrived later if they are attached to same block for their next trip.

The objective function minimizes total weighted transit times of shipments from their origin to destination. However the formulation we give here can also handle other types of objective criteria such as total waiting times at the stations and total earliness/tardiness (if there are due dates for shipments).

The time-space network of the shipment routing problem is defined as $G = (N, A)$ where N denotes the node set and A denotes the arc set. A node (w, s, t) in the network represents a valid station s at a time t that a link can arrive or depart from that station for shipment w. Two types of arcs are introduced for the arc set A: *Link arcs* and *connection arcs*. A link is designed for each possible carrier sequence from the origin of the block to the destination of the block and a *link arc* is generated for each shipment that can attach to corresponding link. Hence, the link arc may consist of multiple resources.

On the other hand, *connection arcs* connects one link arc to next link arc so they represent waiting time at each station for each shipment, except for the destination station of the shipment. All the nodes for a station are sorted in chronological order by their time attributes, and each node at a station is connected to the next node in this order. We assume that the planning horizon is given and we accept the link arcs within that horizon. Moreover, the latest node at a station is connected to the dummy node at the destination to guarantee feasibility. Otherwise infeasibility may occur either because of capacity constraints or planning horizon is not long enough to complete routings of some shipments. Figure 5.2 illustrates the time-space network for the shipment routing problem for a shipment.

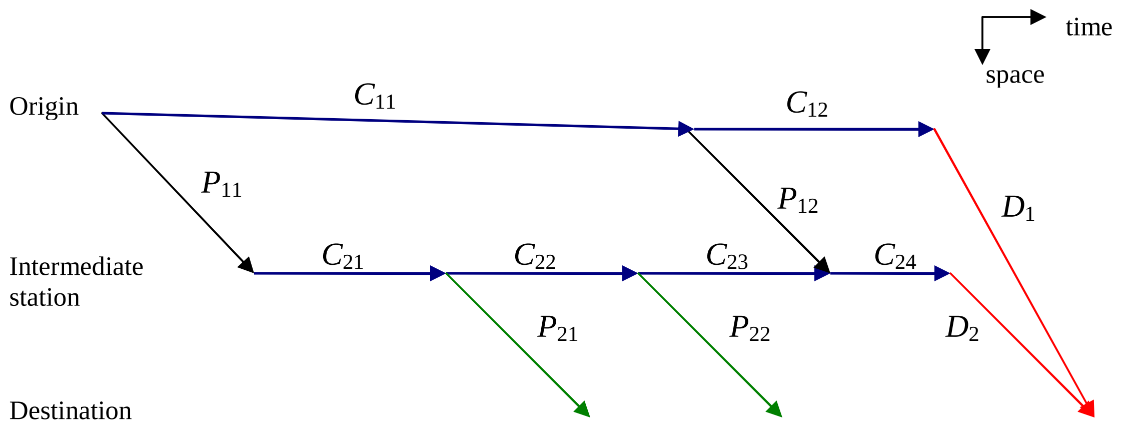


Figure 5.2. Time-space network representation for each shipment

In Figure 5.2, P_{11} and P_{12} are outgoing link arcs from the origin station and incoming link arcs to the intermediate station for given shipment. Similarly, P_{21} and P_{22} are outgoing link arcs from the intermediate station and incoming link arcs to the destination. D_1 and D_2 are dummy link arcs from the origin and the intermediate station to the destination, respectively. These arcs move the shipment from one station to another. C_{11} and C_{12} are connection arcs at the origin, and C_{21}, C_{22}, C_{23} and C_{24} are connection arcs at the intermediate station. These arcs represent the waiting time of the shipments if there is a flow on them.

In this time-space network, shipments flow on "link" arcs and connection arcs, going from their origin to their destination. This formulation is a mixture of pure arc-based and pure path-based formulations. The other two formulations can be described as follows:

- *Pure arc based formulation:* In this time-space network, arcs represent individual resources and waiting times. Although this formulation can potentially reduce the number of arcs in the network, it requires too many constraints to represent shipment-block sequence structure of the network. Besides, two shipments can share same resource by using different blocks, so we need to embed the block information to the formulation anyway to represent dispatching policy constraints.

- *Pure path based formulation:* In this time-space network, paths are generated for all shipments from their origin to destination. Although all of the network structure and practical constraints could be honored during the generation of feasible paths, the number of possible paths can be in the billions (around 2.1 billions in one of our instance we tested) for a moderate size transportation company.

The hybrid formulation holds the number of variables to a manageable size and handles the network structure and practical constraints easily. We use the following sets and indices in the IP formulation.

Sets and Indices:

W	Set of all shipments, indexed by w
P	Set of all links, including dummy links, indexed by p
R	Set of all resources, indexed by r
S	Set of all valid stations, indexed by s
$PW(w)$	Set of links that can be used by shipment w; $\forall\ w \in W$
$POW(w)$	Set of links that can be used by shipment w where the starting location is the origin of w; $\forall\ w \in W$
$PDW(w)$	Set of links that can be used by shipment w where the ending location is the destination of w; $\forall\ w \in W$
$PIW(w)$	Set of links that can be used by shipment w other than the links within the set of $POW(w)$ and $PDW(w)$; $\forall\ w \in W$
$WP(p)$	Set of shipments that can use link p; $\forall\ p \in P$

$PR(r)$	Set of links that use resource r; $\forall\ r \in R$
$SW(w)$	Set of stations that originate links in $PW(w)$; $\forall\ w \in W$
$SIW(w)$	Set of stations (intermediate stations) that originate links in $PIW(w)$; $\forall\ w \in W$
$TSW(s, w)$	Set of departure and arrival times of links in $PW(w)$ that originate and terminate at station s; $\forall\ w \in W, s \in SIW(w)$
$tnext(w, s, t)$	Earliest element of $TSW(s, w)$ later than time t; ; $\forall\ w \in W, s \in SIW(w), t \in TSW(s, w)$
$tprev(w, s, t)$	Latest element of $TSW(s, w)$ earlier than time t; $\forall\ w \in W, s \in SIW(w), t \in TSW(s, w)$
$Porig(w, s, t)$	Set of links in $PW(w)$ that originate at station s and time t; $\forall\ w \in W, s \in SIW(w), t \in TSW(s, w)$
$Pdest(w, s, t)$	Set of links in $P(w)$ that terminate at station s and time t; $\forall\ w \in W, s \in SIW(w), t \in TSW(s, w)$
v_w	Volume of shipment w; $\forall\ w \in W$
u_r	Total volume allowed on resource r; $\forall\ r \in R$
c_{wp}	Cost of using link p for shipment w; $\forall\ w \in W, p \in PW(w)$. **Note:** In our test data, this cost represents weighted transit times; (time to go from origin to destination of link p) times (volume of shipment w).
f_{wst}	Cost of being idle for shipment w at station s from time t to time $tnext(w, s, t)$; $\forall\ w \in W,\ s \in SIW(w), t \in TSW(s, w)$. **Note:** We include the cost of being idle at origin or destination (if any) into the corresponding link to reduce number of variables.

In this structure, we assume that all the shipments should be routed from their origin to their destination without violating capacity or dispatching policy constraints.

The optimization problem outlined above can be formulated as a multicommodity network flow with side constraints.

Decision Variables:

X_{wp} = 1 if shipment *w* uses link *p*, 0 otherwise; $\forall\ w \in W, p \in PW(w)$

Y_{wst} = 1 if shipment *w* idles at station *s* from time *t* to time *tnext*(*w*, *s*, *t*), 0 otherwise; $\forall\ w \in W,\ s \in SIW(w), t \in TSW(s, w)$

Model Formulation:

Minimize $\sum_{w \in W} \sum_{p \in PW(w)} c_{wp} X_{wp} + \sum_{w \in W} \sum_{s \in SIW(w)} \sum_{t \in TSW(s,w)} f_{wst} Y_{wst}$ (1)

subject to:

Flow conservation constraints for origin stations of shipments

$\sum_{p \in POW(w)} X_{wp} = 1$ for all $w \in W$ (2)

Flow conservation constraints for intermediate stations of shipments

$\sum_{p \in Porig(w,s,t)} X_{wp} + Y_{w,s,t} = \sum_{p \in Pdest(w,s,t)} X_{wp} + Y_{w,s,tprev(w,s,t)}$

for all $w \in W, s \in SIW(w), t \in TSW(s, w)$ (3)

Flow conservation constraints for destination stations of shipments

$\sum_{p \in PDW(w)} X_{wp} = 1$ for all $w \in W$ (4)

Resource capacity constraints

$\sum_{w \in WP(p)} \sum_{p \in PR(r)} v_w X_{wp} \le u_r$ for all $r \in R$ (5)

Dispatching policy constraints

First-in-first-out among shipments that use same block (see next section) (6)

Integrality

$X_{wp} = 0 \text{ or } 1$ for all $w \in W, p \in PW(w)$ (7)

$Y_{wst} = 0 \text{ or } 1$ for all $w \in W,\ s \in SIW(w), t \in TSW(s, w)$

The objective function (1) minimizes the total cost of links and connection arcs. The objective function can represent several types of cost structures such as:

- Total weighted transit times of shipments from their origin to destination if the arc costs are equal to time spend on corresponding arcs
- Total waiting times at the stations if only the connection arcs have (time based) costs
- Total earliness/tardiness if there are due dates for shipments and only the link arcs that go to destination of shipment have costs based on due dates.

Constraint (2) and (4) assign shipments to their first and last blocks, respectively. Constraint (3) represents flow conservation constraints for intermediate stations. Constraint (5) indicates the capacity restrictions, and constraint (6) ensures dispatching policy, which will be described in detail in the next section. Finally, constraint (7) is for integrality requirements.

In addition to these constraints, there are also physical constraints, called dispatching constraints, needed to ensure the first in first out rule for shipments at the stations. In the next section, we propose alternative formulations for dispatching constraints.

5.4 DISPATCHING POLICY CONSTRAINTS

In a transportation network, shipments wait in line for their next block connection. When the carrier arrives, it picks up the shipments located at the front of the line and fills to capacity. Therefore, there is a practical constraint which states: Among two shipments, a shipment that arrived at the station earlier cannot go after one that arrived later if they are attached to same block for their next trip.

In Figure 5.3, there are two shipments w and w' at the same station s and they are attached to same block to depart this station. Let shipment w arrives to the station with link p and departs from the station with link q. Then, the dispatching policy rule states that shipment w' cannot come to station s earlier than the arrival time of shipment w and depart from station s later than shipment w. We develop three types of formulations to describe these constraints.

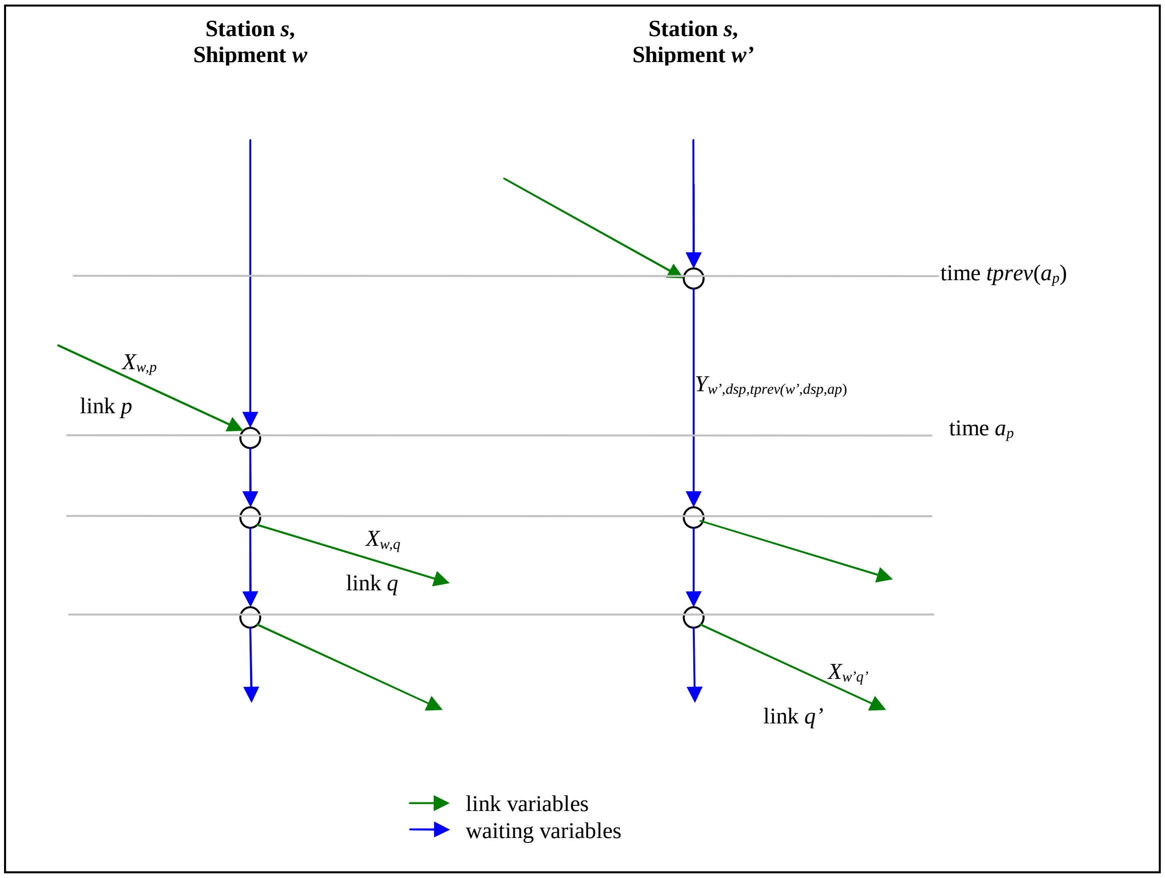


Figure 5.3. Departures of two shipments using the same block

5.4.1 Type-A dispatching constraints

We define the following additional sets and indices:

a_p	Arrival time of link p; $\forall\ p \in P$

d_p	Departure time of link p; $\forall\, p \in P$
ds_p	Destination station of link p; $\forall\, p \in P$
o_w	Origin of shipment w; $\forall\, w \in W$
rt_w	Release time of shipment w; $\forall\, w \in W$
$PW_{dest}(w,p)$	Links usable by shipment w at the destination of link p; $\forall\, p \in P$, $\forall\, w \in W$
$P_{later}(q)$	Links that depart later than link q and belong to same block as link q; $\forall\, q \in P$

The following constraints ensure the dispatching criteria:

both shipments are at their origin

$X_{wq} + X_{w'q'} \le 1$ for all $w \in W$, $q \in POW(w)$, $rt_w > rt_{w'}$ (A1)

$w' \in WP(q)$, $q' \in P_{later}(q)$, $q' \in POW(w')$

shipment w is at the middle station, and w' is at its origin

$X_{wp} + X_{wq} + X_{w'q'} \le 2$ for all $w \in W$, $q \in PIW(w)$, $q \in PW_{dest}(w,p)$ (A2)

$w' \in WP(q)$, $q' \in P_{later}(q)$, $q' \in POW(w')$, $a_p > rt_{w'}$

shipment w' is at the middle station, and w is at its origin

$X_{wq} + Y_{w',o_w,tprev(w',o_w,rt_w)} + X_{w'q'} \le 2$ for all $w \in W$, $q \in POW(w)$, (A3)

$w' \in WP(q)$, $q' \in P_{later}(q)$

both shipments are at their middle station

$X_{wp} + X_{wq} + Y_{w',ds_p,tprev(w',ds_p,a_p)} + X_{w'q'} \le 3$ for all $w \in W$, $q \in PIW(w)$, (A4)

$q \in PW_{dest}(w,p)$, $w' \in WP(q)$, $q' \in P_{later}(q)$

In each of these constraint sets, we are comparing two shipments at each constraint. Constraint (A1) is for the case where both shipments are at the same origin. Constraint (A2) and (A3) are for the cases where only one shipment is at its origin. Constraint (A4) shows the case where both shipments are at their intermediate stations. Suppose shipment w is attached to links p and q. Shipment $w' \neq w$ cannot arrive at station *s* before than link *p* and depart on a link *q'* that departs later than link *q* without violating the dispatching rule. Similarly, if shipment *w'* is waiting at station *s* whenever shipment *w* arrives, then shipment *w* should not depart before shipment *w'*. Therefore, the variables at the left hand side of the constraints cannot be one at the same time.

5.4.2 Type-B dispatching constraints

Type-B dispatching constraints introduce new variables to represent arrival and departure time of a particular shipment at each station as a single variable. We define the following additional sets and indices:

os_p	Origin station of link p; $\forall\ p \in P$
ds_p	Destination station of link p; $\forall\ p \in P$
$P_{later}(q)$	Links that depart later than link q and belong to same block as link q; $\forall\ q \in P$
$TASW(s, w)$	Set of arrival times of links in $PW(w)$ that terminate at station s; $\forall\ w \in W$, $s \in SW(w)$. This also includes release time of shipments that originate at station s.
$PSW(w, s, t)$	Links that are eligible for shipment w and depart from station s later than time t; $\forall\ w \in W, s \in SW(w), t \in TASW(s, w)$

We define the following variables:

Z_{wtq} = 1 if shipment w arrives to station os_q at time t and departs from station os_q by using link q, 0 otherwise; $\forall\ w \in W, t \in TASW(os_q, w), q \in PSW(w, os_q, t)$

The following constraints ensure the dispatching criteria:

$Z_{wtq} + Z_{w't'q'} \leq 1$ for all $w \in W, t \in TASW(os_q, w), q \in PSW(w, os_q, t)$ (B1)

$w' \in WP(q), q' \in P_{later}(q), t' < t, t' \in TASW(os_q, w')$,

$X_{wq} \leq Z_{wtq}$ for all $w \in W, t = rt_w, q \in PSW(w, os_q, t)$, (B2)

$q \in POW(w)$

$X_{wp} + X_{wq} - 1 \le Z_{wtq}$ for all $w \in W$, $t \in TASW(os_q, w)$, $q \in PSW(w, os_q, t)$, (B3)

$q \in PIW(w)$, $t = a_p$

The explanation of constraint (B1) is similar to the explanations for Type-A constraints. The additional Z variables have three indexes to represent both arrival and departure times of a particular shipment. Constraints (B2) and (B3) link the X and Z variables at the origin and intermediate stations.

Clearly, Type-B dispatching constraints use more variables than the Type-A constraints. In addition, Type-B has more constraints than Type-A, because previous constraints check whether shipment w' waits or not at the time shipment w arrives by using one connection variable. However, Type-B constraints look at all the time points smaller than the arrival time of shipment w.

Proposition 5.1: *Neither Type-A nor Type-B dispatching constraints is the stronger than the other.*

Proof: In Figure 5.4, there are two shipments (w and w') that are use the same block to depart station s. Assume that $X_{wp} = X_{wq} = 1$. Then, $Z_{wtq} = 1$ for $t = a_p$.

If $X_{w'p'} = X_{w'p''} = X_{w'q} = X_{w'q'} = 0.5$, then all the Z variables related to w' could be zero. Therefore, this solution satisfies Type-B constraints. However $Y_{w',os_q,tprev(w',os_q,a_p)} = 1$ due to flow constraints. Hence;

$$X_{wp} + X_{wq} + Y_{w',os_q,tprev(w',os_q,a_p)} + X_{w'q'} = 1 + 1 + 1 + 0.5 = 3.5 > 3$$

violates Type-A constraints.

Similarly, if $X_{w'p'} = 1$ and $X_{w'q'} = X_{w'q''} = 0.5$; then $Y_{w',os_q,tprev(w',os_q,a_p)} = 0.5$ due to the flow constraints. Hence, $X_{wp} + X_{wq} + Y_{w',o_q,T_{last}(w',o_q,a_p)} + X_{w'q'} = 1 + 1 + 0.5 + 0.5 = 3$ satisfies Type-A constraints. On the other hand, $Z_{wtq'} = 0.5$ for $t = a_{p'}$. Hence,

$Z_{wtq} + Z_{w't'q'} = 1 + 0.5 = 1.5 > 1$ violates Type-B constraints. Therefore, neither Type-A nor Type-B is the stronger of the two.

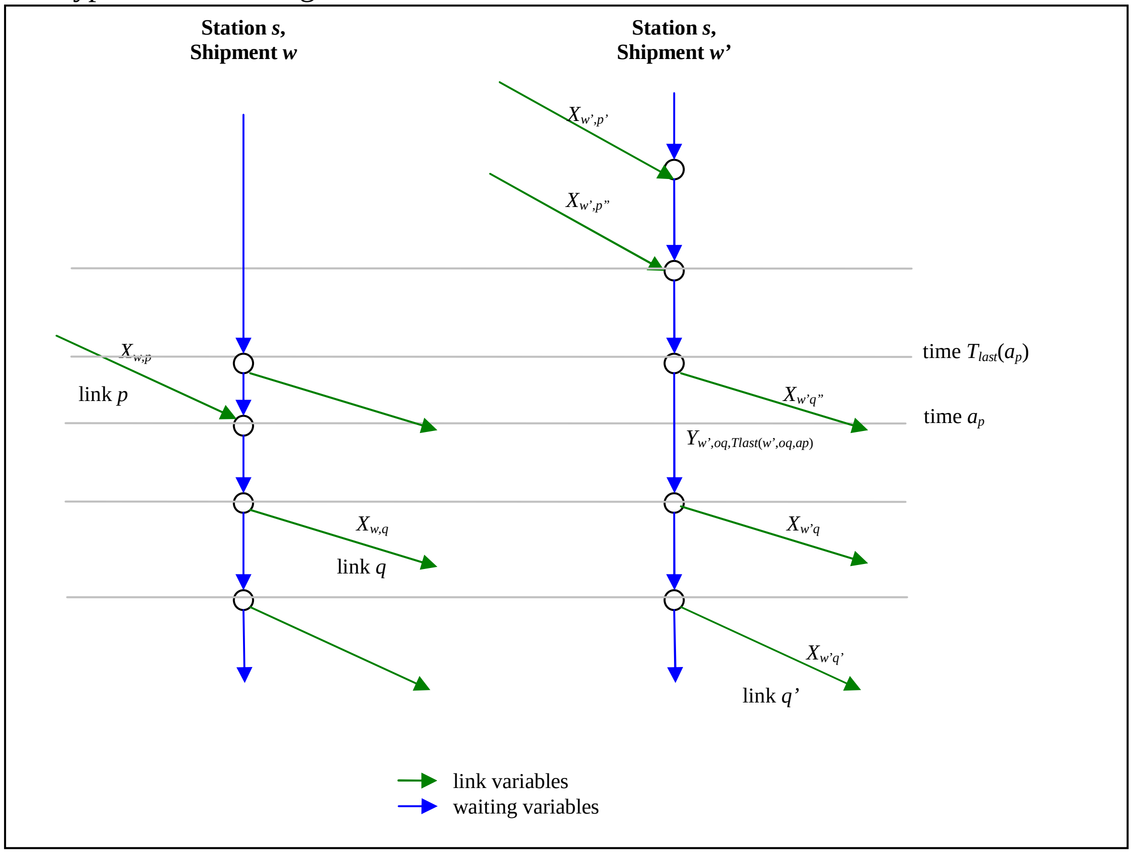


Figure 5.4. Type-A and Type-B constraints where neither of them is stronger

5.4.3 Type-C dispatching constraints

Type-C dispatching constraints require a different approach. In this case, we introduce the variables that determine time window for every link. For each link, there is a time window whose lower and upper bounds are calculated based on the arrival time of the earliest and latest shipments assigned to that link. If the arrival time of the latest shipment assigned to a previous link has a time greater than that of the arrival time of the earliest shipment assigned to a next link of the same block, then we can conclude that there is a dispatching violation between these links.

There are two kinds of shipments that violate the dispatching rule: (i) shipments assigned to a link *p* with arrival times that are later than the shipment with the earliest arrival time in the next link and (ii) shipments with arrival times that are earlier than the shipment with the latest arrival time in the previous link *p*.

Consider the following two links (P_1 and P_2) of the same block:

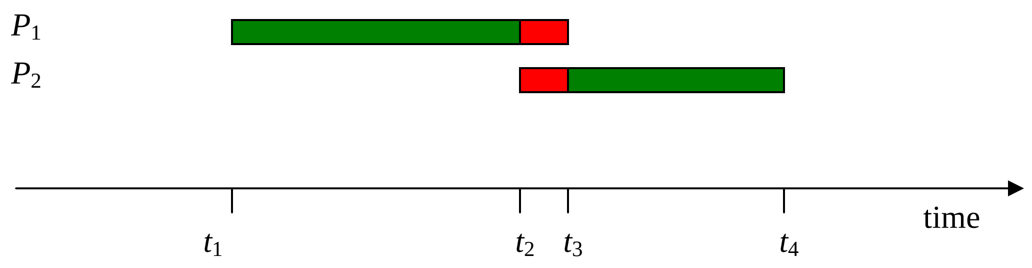


Figure 5.5. Time window violation of links

In Figure 5.5, two links of same block have time windows based on the arrival time of shipments assigned to these links. The first link carries shipments that arrive between t1 and t3, and the second link carries shipments that arrive between t2 and t4. Here we have to reschedule shipments that arrive between t2 and t3. Figure 5.6 shows non-overlapping time windows for links.

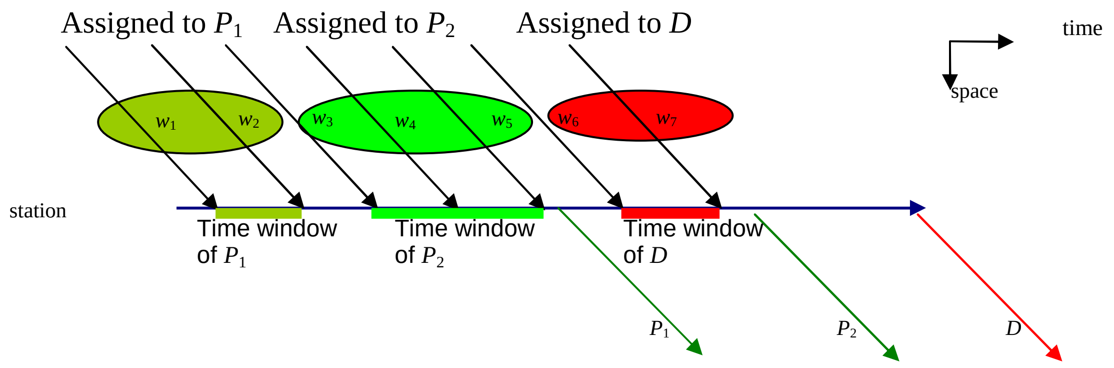


Figure 5.6. Non-overlapping time windows for links

To determine time windows for links, we define the following additional sets and indices:

SIW(*w*)	Set of stations for shipment *w* other than origin and destination; $\forall\ w \in W$
PIOS(*w,s*)	Set of links that can be used by shipment *w* and starting location of links is station *s*; $\forall\ w \in W, s \in SIW(w)$
PIDS(*w,s*)	Set of links that can be used by shipment *w* where its ending location is station *s*; $\forall\ w \in W, s \in SIW(w)$
next(*p*)	Next link in time after link *p* that belongs to same block; $\forall\ p \in P$

We define the following additional variables:

LB_p	Lower bound in time for accepting shipments for link *p*, $\forall\ p \in P$
UB_p	Upper bound in time for accepting shipments for link *p*, $\forall\ p \in P$

The following constraints ensure the dispatching criteria:

$$-M(1-X_{wp})+LB_p \le rt_w \le M(1-X_{wp})+UB_p \qquad \text{for all } w \in W,\ p \in POW(w) \quad \text{(C1)}$$

$$-M(1-X_{wq})+LB_q \le \sum_{p\in PIDS(w,s)} a_p X_{wp} \le M(1-X_{wq})+UB_q \qquad \text{for all } w \in W,\ s \in SIW(w) \quad \text{(C2)}$$

$$q \in PIOS(w,s)$$

$$LB_p \le UB_p \qquad \text{for all } p \in P \quad \text{(C3)}$$

$$LB_{next(p)} = UB_p \qquad \text{for all } p \in P \quad \text{(C4)}$$

where *M* represents a big number and makes a constraint loose if a shipment is not assigned to a link in that constraint. (*M* will be properly calculated in the next section.)

Constraint (C1) is for the origin, and (C2) is for intermediate stations of the shipment. Whenever a shipment is assigned to a link, these constraints ensure that the lower bound of the link is not bigger than the arrival time of the shipment and the upper bound of the link is not smaller than the arrival time of the shipment.

Note: Instead of arrival time of shipments, any kind of ordering criteria could be used to determine dispatching policy.

Although the Type-C constraint set requires fewer constraints than others, it contains big M parameters that weaken the constraints. However, if we choose appropriate big M parameters (calculated in the next section), we can strengthen them.

Proposition 5.2: *Neither Type-A nor Type-B dispatching constraints is stronger than Type-C dispatching constraints and vice versa.*

Proof: In Figure 5.4, there are two shipments (w and w') that use the same block to depart station s. Assume that $X_{wp} = X_{wq} = 1$. Then, $Z_{wtq} = 1$ for $t = a_p$.

- If $X_{w'p'} = X_{w'p''} = X_{w'q} = X_{w'q'} = 0.5$, this solution violates Type-A, and satisfies Type B. Assume that $X_{w'q'''} = 0.5$ instead of $X_{w'q}$, where link q''' departs later than the other links. The solution still violates Type-A, and satisfies Type B. We can choose $M = a_p$, which is the latest arrival time at that station. Therefore,

  $$LB_q \le a_p \le UB_q$$
  $$-0.5a_p + LB_{q'} \le \frac{a_{p'} + a_{p''}}{2} \le 0.5a_p + UB_{q'}$$
  $$UB_q = LB_{q'} \Rightarrow 0.5a_p \le \frac{a_{p'} + a_{p''}}{2}$$

  This solution violates or satisfies Type-C constraints depending on the selection of a_p, $a_{p'}$, and $a_{p''}$.

- If $X_{w'p'} = 1$ and $X_{w'q'} = X_{w'q''} = 0.5$, this solution violates Type-B, and satisfies Type A. We can choose $M = a_p$, then;

  $$LB_q \le a_p \le UB_q$$
  $$-0.5a_p + LB_{q'} \le a_{p'} \le 0.5a_p + UB_{q'}$$
  $$UB_q = LB_{q'} \Rightarrow 0.5a_p \le a_{p'}$$

  This solution violates or satisfies Type-C constraints depending on the selection of a_p, and $a_{p'}$.

This result shows that none of the dispatching constraints are weaker or stronger than any of the others.

5.4.4 Strengthening dispatching constraints

In a capacitated shipment routing problem, we have flow-in-flow-out constraints, and there is one flow for each shipment. This means that a shipment can arrive at or depart from a station using only one link. Using this knowledge, we can find the following constraints, which are a stronger version of the Type-A dispatching constraints:

both shipments are at their origin

$$X_{wq} + \sum_{q' \in P_{later}(q)} X_{w'q'} \leq 1 \qquad \text{for all } w \in W,\ q \in POW(w),\ \ rt_w > rt_{w'} \quad \text{(A1)}$$

$$w' \in WP(q),\ q' \in POW(w')$$

shipment w is at the middle station, and w' is at its origin

$$X_{wp} + X_{wq} + \sum_{q' \in P_{later}(q)} X_{w'q'} \leq 2 \qquad \text{for all } w \in W,\ q \in PIW(w),\ q \in PW_{dest}(w,p) \quad \text{(A2)}$$

$$w' \in WP(q),\ q' \in POW(w'),\ a_p > rt_{w'}$$

shipment w' is at the middle station, and w is at its origin

$$X_{wq} + \sum_{p' \in P_{earlier}(rt_w)} X_{w'p'} + \sum_{q' \in P_{later}(q)} X_{w'q'} \leq 2 \qquad \text{for all } w \in W,\ q \in POW(w), \quad \text{(A3)}$$

$$w' \in WP(q)$$

both shipments are at their middle station

$$X_{wp} + X_{wq} + \sum_{p' \in P_{earlier}(rt_w)} X_{w'p'} + \sum_{q' \in P_{later}(q)} X_{w'q'} \leq 3 \qquad \text{for all } w \in W,\ q \in PIW(w), \quad \text{(A4)}$$

$$q \in PW_{dest}(w,p),\ w' \in WP(q)$$

Similarly, the following constraints are a stronger version of the Type-B dispatching constraints:

$$Z_{wtq} + \sum_{\substack{(t' < t) \\ t' \in TASW(os_q, w')}} \sum_{q' \in P_{later}(q)} Z_{w't'q'} \leq 1 \qquad \text{for all } w \in W,\ t \in TASW(os_q, w), \quad \text{(B1)}$$

$$q \in PSW(w, os_q, t),\ w' \in WP(q),$$

$$X_{wq} \leq Z_{wtq} \qquad \text{for all } w \in W,\ t = rt_w,\ q \in PSW(w, os_q, t), \quad \text{(B2)}$$

$$q \in POW(w)$$

$$\sum_{p \in PAT(w,t)} X_{wp} + X_{wq} - 1 \leq Z_{wtq}$$ for all $w \in W$, $q \in PW_{dest}(w,p)$, $t = a_p$ (B3)

$t \in TASW(os_q, w)$, $q \in PSW(w, os_q, t)$, $q \in PIW(w)$,

where $PAT(w, t)$ is the set of links related to shipment w and arrival times are equal to t. Observe that the proof of Proposition 5.1 does not hold for the strengthened Type-A and Type-B constraints.

Finally, we can choose appropriate values for big M parameters for Type-C constraints. These parameters could be selected separately for each constraint. We make the following observations:

- The lower bound of the link should not be greater than the departure time of an earlier link for the same block, because an earlier link cannot get a shipment after it departs from the station.
- The upper bound of the link could be as low as the earliest release time of the shipments or the earliest possible arrival time of the shipments to that station among the shipments that can also use this link, because that shipment can ride on a later link.

To set the big M values, we define the following additional sets and indices:

rt_w	Release time of shipment w at its origin; $\forall\ w \in W$
$prev(p)$	Previous link in time after the link p that belongs to the same block; $\forall\ p \in P$
$first(p)$	First link in time that arrives among the links of the same block; $\forall\ p \in P$
$last(p, t)$	Last link in time before time t that arrives among the links of same block; $\forall\ p \in P$
$t_{earliest(p)}$	Earliest arrival time or release time among the shipments $w \in WP(p)$; $\forall\ p \in P$

a_p	Arrival time of link p
d_p	Departure time of link p

We obtain the following results for the constraints at the origin (C1):

$$LB_p \le rt_w + M = d_{prev(p)} \Rightarrow M = d_{prev(p)} - rt_w \qquad \text{for all } w \in W, p \in POW(w)$$

$$UB_p \ge rt_w - M = t_{earliest(p)} \Rightarrow M = rt_w - t_{earliest(p)} \qquad \text{for all } w \in W, p \in POW(w)$$

We obtain the following results for the constraints at the origin (C2):

$$LB_q \le \sum_{p \in PIDS(w,s)} a_p X_{wp} + M = d_{prev(q)} \Rightarrow M = d_{prev(q)} - a_{first(p)} \qquad \text{for all } w \in W, s \in SIW(w)$$

$$UB_q \ge \sum_{p \in PIDS(w,s)} a_p X_{wp} - M = t_{earliest(q)} \Rightarrow M = a_{last(p,d_q)} - t_{earliest(q)} \qquad \text{for all } w \in W, s \in SIW(w)$$

Observe that the proof of Proposition 5.2 does not hold for the strengthened Type-A, Type-B and Type-C constraints.

5.5 PATH-BASED ALTERNATIVE FORMULATION

In this formulation, all feasible paths for all shipments, called *shipment-paths*, from origin to destination are created a priori for each shipment. In the construction of shipment-path, one appropriate link is chosen for each block in the shipment-block sequence. The problem is then to optimally assign each shipment to exactly one shipment-path. Observe that the shipment-path definition here is different than the link definition for all shipments that have more than one block in its block sequence.

Sets and Indices:

W	Set of all shipments
P	Set of all paths (from origin to destination)
R	Set of all resources
$PW(w)$	Set of paths that can be used by shipment w; $\forall\, w \in W$
$rWW(w)$	Set of shipments that share at least one block with shipment w; $\forall\, w \in W$
$WP(p)$	Set of shipments that can use path p; $\forall\, p \in P$

$PR(r)$	Set of paths that use resource r; $\forall\ r \in R$
$PI(p, w, w')$	Set of paths that are inconsistent (violate dispatching) with path p, if shipment w uses path p and shipment w' uses the inconsistent path; $\forall\ p \in P, w \in WP(p), w' \in WW(w)$
u_r	Total volume allowed on resource r; $\forall\ r \in R$
v_w	Volume in shipment w; $\forall\ w \in W$
c_{wp}	Cost of using path p for shipment w; $\forall\ w \in W, p \in PW(w)$

Decision Variables:

X_{wp} = 1 if shipment w uses shipment path p, 0 otherwise; $\forall\ w \in W, p \in PW(w)$

Model Formulation:

Minimize $\sum_{w \in W} \sum_{p \in PW(w)} c_{wp} X_{wp}$ (1)

subject to:

Assignment constraints for shipments

$\sum_{p \in PW(w)} X_{wp} = 1$ for all $w \in W$ (2)

Resource capacity constraints

$\sum_{w \in WP(p)} \sum_{p \in PR(r)} v_w X_{wp} \le u_r$ for all $r \in R$ (3)

Dispatching constraints

$X_{wp} + X_{w'p'} \le 1$ for all $w \in W, p \in PW(w)$ (4)

$w' \in \{i \in W: i \neq w\}, p' \in PI(p, w, w')$

Integrality

$X_{wp} = 0 \text{ or } 1$ for all $w \in W, p \in PW(w)$ (5)

The objective function (1) minimizes the total cost of used paths. Constraint (2) assigns shipments to their paths. Constraint (3) determines the capacity restrictions and constraint (4) ensures dispatching policy rules. Finally, constraint (5) is for integrality requirements.

For a stronger version of the dispatching constraints, we can have a set that includes shipment-path pairs in which each of the pairs is inconsistent with another one. Then, we can write a clique constraint for them:

$$\sum_{(w,p)\in Clique(w,p)} X_{wp} \leq 1 \qquad \text{for all } w \in W, p \in PW(w), Clique(w,p) \qquad (4)$$

Although all of the network structure and practical constraints could be considered during the generation of feasible paths, the number of possible paths could be in the billions (2.1 billion for one of our test instance) for a moderate size transportation company.

5.6 COMPLEXITY RESULTS

The capacitated network flow problem is a well-known NP-Hard problem. That is why we will explore the uncapacitated version of the problem with dispatching policy constraints, called USR-DP. We first analyze the conditions that make the solution of the problem integral. Then, we explore the complexity properties of the general shipment routing problem with dispatching constraints.

5.6.1 Integrality conditions

The path-based formulation can be seen as a multicommodity network flow problem with side constraints. In network flow problems, there are many special cases that have an integrality property. We examine this problem to find the same property.

Proposition 5.3: *A shipment routing problem without resource capacity and dispatching policy constraints [USR] has an integer optimal solution even if there is no integrality requirement.*

Proof: If there is no resource capacity or dispatching policy constraints, the problem can be decomposed for each shipment. In each of these subproblems, the corresponding shipment should be carried from its origin to destination with minimum cost. The subproblem is a shortest path problem and always has an integer optimal solution. (See Ahuja et al. (1993) for details about the shortest path problem, such as Dijksta's algorithm for acyclic graphs where the solution time is $O(m)$, m is number of arcs in the graph.) Therefore, the planning problem without resource capacity and dispatching policy constraints has an integer optimal solution.

Remark: In our later discussions, we will refer to the optimal solution of USR problem as the *shortest path solution*.

In the shipment routing problem, one of the most common objective functions is total weighted transit time. After all, the company wants to send their shipments as soon as possible to their destination. This objective is in fact same as minimum weighted arrival time at the destination of shipments because the release times of shipments are fixed and given *a priori*.

The next observation concerns the dispatching policy constraints. The following proposition states that if shared blocks have same shortest paths for all shipments, then the shortest path solution preserves after the addition of dispatching policy constraints. Examples of this type of objective function are total weighted transit time cost and total weighted tardiness cost (if there are due dates for shipments).

Proposition 5.4: *Consider the sub-path for a shipment-pair that shares block(s) where the sub-path begins from the starting station and time of the earliest available link of the first shared block, and ends at the ending station and time of the latest arrived link of the last shared block. If all shipment pairs have same shortest sub-paths for their commonly used blocks, then the uncapacitated shipment routing problem with dispatching policy* *constraints [USR-DP] has an optimal solution which is equal to the shortest path solution.*

Proof: From Proposition 5.3, the USR problem has an integer optimal solution even if there is no integrality requirement. Assume that there is an integer optimal solution to the USR problem, and this solution violates dispatching policy constraints.

In the optimal solution, $w, w' \in W$ use links, respectively, p and p' of the same block. These shipments use the same departing block, and the earlier shipment that comes to that station (say w) uses the later link p for departure. Since shipment w is the earlier shipment, it can also use the earlier link p'. In addition, shipment w' can be assigned to the later link p used by shipment w.

Since both paths are feasible for both shipments, we can assign both of them to the same link with minimum cost. With this assignment, following situations may occur:

- If the new assignment causes an earlier arrival to the ending station of the link, then the remaining downward assignments are still feasible. If the ending station is the destination of one of the shipments and the new assignment allows it to come to that station earlier, then this solution contradicts the optimality of the original solution.
- If the new assignment causes a later arrival to the ending station of the link, then the remaining downward assignments may not be feasible. If it is infeasible, then re-assign the links on the downwards. If it causes a cost increment, then the shortest sub-path assumption in the proposition does not hold. If the ending station is the destination of one of the shipments and the new assignment allows it to come to that station later, then again the shortest sub-path assumption in the proposition does not hold.

Hence, the USR-DP problem has an integer optimal solution which is equal to the shortest path solution. The following result shows some examples of cost functions that satisfy Proposition 5.4. In some of the cost functions, the shipments have due dates, desired time to reach the destination, and if a shipment is late, there could be a tardiness cost which is difference between arrival time to the destination and the due date.

Corollary 5.5: *Results from Proposition 5.4 hold for the following functions:*

- *Total weighted transit time cost,*
- *Total weighted tardiness cost,*
- *Maximum weighted tardiness cost,*
- *Total number of tardy shipments.*

Unfortunately, this result is not true for the general type of objective functions such as total earliness cost.

Proposition 5.6: *Assume that the following assumptions hold:*

- $c_{wp} = g_w h_p$ *and* $f_{wst} = 0$ *for all* $w \in W$,
- *All the shipments have the same destination, but they can have multiple origins,*
- *If two shipments share a block, then they share all blocks after that block.*

Then the multiple origin-single destination uncapacitated shipment routing problem with dispatching policy constraints has an optimal solution which is equal to the shortest path solution.

Proof: Since the shipments are using the same blocks at the tail of the block sequence and go to the same destination, the shortest remaining path after the first common station for one shipment is also the shortest remaining path for the other one.

This proposition shows that the single destination special case could be solvable polynomially even for time independent costs.

Corollary 5.7: *Results in Proposition 5.6 hold for multiple destinations if the shipments that have common blocks also have common destinations.*

Proof: The problem is decomposable for each destination. The remaining shortest path claim is still valid in this case.

Proposition 5.8: *Assume that the following assumptions hold:*

- *c_{wp} and f_{wst} are not restricted for all $w \in W$,*
- *All the shipments have the same origin but can have multiple destinations,*
- *Only the first block is a common block.*

Then the single origin- multiple destination uncapacitated shipment routing problem with dispatching constraints is solvable in polynomial time by using shortest path algorithms.

Proof: If a shipment does not share any block, we can send it directly via its shortest path. Assume that there are n shipments that share their first block and m links that belong to that block. Let release times be $a_1 < a_2 \ldots < a_n$ for these n shipments and m links, and let c_{ij} be the total cost from origin to destination if shipment i is assigned to link j. We will construct the following graph to solve this problem.

At each node (i,j), the outgoing arc represents the flow on link j such that $(i,j) \rightarrow (i+k, j+1)$ assign shipments $i+1,..,i+k$ to link j. The cost of this link is $\sum_{l=i+1}^{i+k} c_{lj}$. Since each link assignment begins with the earliest shipment that is not assigned to previous links, the resulting solution satisfies the dispatching constraints.

The graph is acyclic, so Dijkstra's algorithm could solve it. Under the given release times of shipments, the given links, and the cost from origin to destination for selected links, the problem is solvable in $O(nm)$.

Corollary 5.9: *The result for proposition 5.8 is true for multiple origins if only the first blocks are common.*

Proof: The problem is decomposable for each origin. For each sub-problem, proposition 5.8 holds.

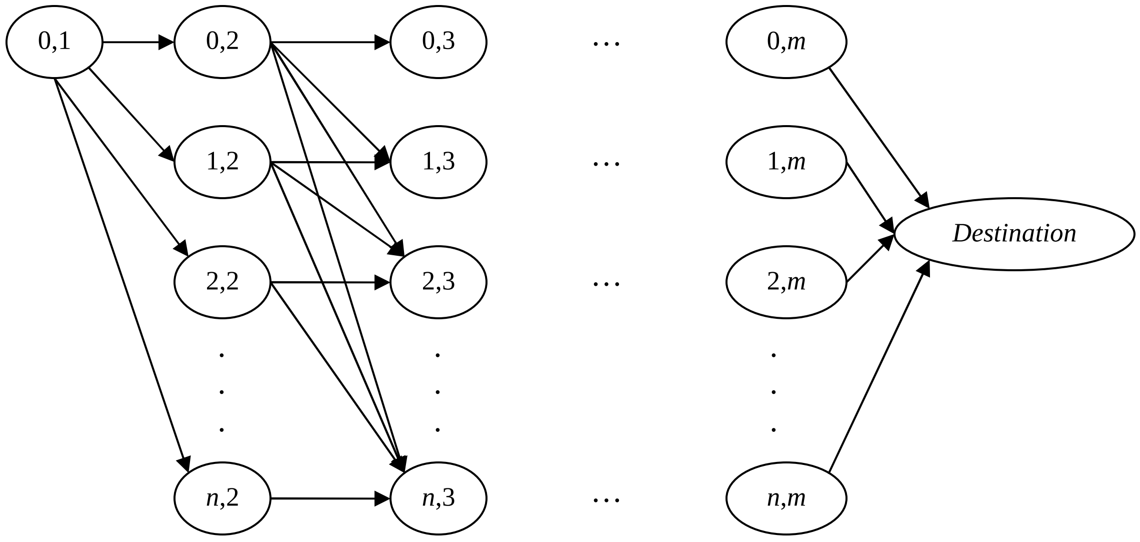


Figure 5.7. Illustration for Proposition 5.8 and Corollary 5.9.

5.6.2 NP-Hard problems

Under the several cost functions (such as earliness cost or time independent cost) given in the shipment-path formulation (see Section 5.5), the uncapacitated shipment routing problem with dispatching policy constraints [USR-DP] is in the category of difficult problems, so called NP-Hard problems. In fact, the well-known 3-satisfiability problem can be written as an instance of the USR-DP problem. (See Karp (1972) for the 3-satisfiability problem.)

3-Satisfiability Problem: Satisfiability is the problem of determining if the variables of a given Boolean formula can be assigned in such a way as to make the formula evaluate to true. For the 3-Satisfiability problem, we are given a Boolean expression *B* such that

$$B = \bigwedge_{i=1}^{n} (a_{i1} \vee a_{i2} \vee a_{i3}) .$$

B is the conjuction of *n* clauses, each of which is the disjunction of 3 literals. A literal a_{ij} represents either a Boolean variable or its negation. *B* is *satisfiable* if the variables can be assigned Boolean values so that *B* is true. In other words, at least one variable should be true in each clause. The 3-Satisfiability problem determines if *B* is satisfiable.

Proposition 5.10: *The 3-Satisfiability problem [3-SAT] is polynomially reducible to the uncapacitated shipment routing problem with dispatching policy constraints [USR-DP].*

Proof: We will use the shipment-path (see Section 4.5) formulation to simplify the proof and convert the general 3-SAT to a USR-DP instance. Assume that we have a Boolean expression $B = \bigwedge_{i=1}^{n} (a_{i1} \vee a_{i2} \vee a_{i3})$. The construction of the USR-DP instance for the indices and sets is as follows:

- There are *n* shipments for each clause.
- Each shipment *w* has 3 paths corresponding to 3 variables in its clause
- The set of pair of paths that are inconsistent contains all Boolean variables and their negations.
- The costs of the paths are zero.

By construction, if there is a solution to the USR-DP instance, that solution cannot contain a Boolean variable and its negation at the same time. Therefore, the solution of the USR-DP instance satisfies the Boolean expression.

Since each set, index and parameter in USR-DP has at most *O*(*n*) items, the reduction will take polynomial time.

Corollary 5.11: *The 3-Satisfiability problem [3-SAT] is polynomially reducible to the uncapacitated shipment routing problem for a single origin and destination with dispatching constraints [USRS-DP].*

Proof: Using the proof of Proposition 5.10, additionally define the paths for each variable and its negation in the Boolean expression. Let all of the available paths for all of the shipments but only the variables in a shipment's corresponding clause have zero costs. The other paths have cost 1 for that shipment.

Then, the question becomes whether there is an optimal solution with a zero objective function or not. If there is, the solution of the USR-DP instance satisfies the Boolean expression. Since each set, index and parameter in USR-DP has at most $O(n)$ items, the reduction will take polynomial time.

Corollary 5.12: *The 3-Satisfiability problem [3-SAT] is polynomially reducible to the uncapacitated shipment routing problem for a single origin and destination with dispatching constraints, and the problem contains only two blocks [USRS2-DP].*

Proof: In the proof of Corollary 5.11, only the Boolean variable and its negation are corresponding inconsistent paths. This inconsistency could be obtained by switching station blocks. Therefore, two blocks are enough to complete the proof of Corollary 5.11.

Corollary 5.13: *The uncapacitated shipment routing problem for a single origin and destination with dispatching constraints where the problem contains only two blocks is NP-hard in the strong sense.*

Proof: From corollary 5.12, proof is clear.

The above complexity results are obtained using a shipment-path formulation and general cost function. The cost function of shipment-path formulation may require a non-linear cost function to represent in the link based formulation. However, we intend to focus on linear cost structures in the link based formulation. As such, we obtained the following result.

Proposition 5.14: *Assume that the following assumptions hold:*

- $c_{wp} = g_w h_p$

- f_{wst} *equals the weighted waiting time at that station for all* $w \in W$,

Then, the 3-Satisfiability problem [3-SAT] is polynomially reducible to the uncapacitated shipment routing problem with dispatching constraints [USRL-DP].

Proof: We will use the link formulation and convert the general 3-SAT to a USRL-DP instance. Assume that we have a Boolean expression $B = \bigwedge_{i=1}^{n} (a_{i1} \vee a_{i2} \vee a_{i3})$, and the expression has m variables. The construction of the USRL-DP instance for the indices and sets is as follows:

- There are n shipments for each clause.
- Each shipment w has 3 blocks to assign, and the intermediate block is the same for all shipments. Shipments have different origins and destinations, but they have the same intermediate stations. Each shipment is released at time 0.
  - First block construction: Each shipment has a different first block, and each of these blocks has 3 links which correspond to variables in that shipment's clause. These links have the same departure times (time 0), but their arrival times are different. In total, $2m$ arrival times are defined for each variable and its negation. Let c be a small number and M be a big number. Then the arrival times are: $\{c, 2c\}$, $\{M+c, M+2c\}$, $\{2M+c, 2M+2c\}$, … , $\{mM+c, mM+2c\}$. There are two numbers in each parenthesis; one for each variable and one for its negation.
  - The second block is a common block and has $2m$ links for each variable and its negation. The departure times of these links are: $\{2c, 3c\}$, $\{M+2c, M+3c\}$, $\{2M+2c, 2M+3c\}$,…, $\{mM+2c, mM+3c\}$. The corresponding arrival times of these links are: M, $2M$, $3M$, … , $2mM$.

- The third block is different for each shipment, and each of these blocks has 3 links. As seen in the above, 2*m* links for the second block have *m* groups, and each group has only two links that have closer arrival and departure times than others. The departure times of the links for the third block are the same as the arrival time of the links for the second block if the second link is in the group where its assigned variable (or its negation) is in the clause of the corresponding shipment.
- There are no costs for links, but there are waiting costs at the stations.

The question is whether there is a feasible solution whose objective value is less than *M*? To arrive at a solution, the shipment should select one of the variables in its clause always based on the links of the third block. Moreover, both of the links in the group cannot be selected at the same time, because there is a conflict in the switch station of the first and second block. Therefore, the solution of the USR-DP instance satisfies the Boolean expression.

Since each set, index and parameter in the USR-DP has at most $O(n)$ items, the reduction will take polynomial time.

Variables	a_1	$\overline{a}_2$	a_3
1st Links	$\{0, c\}$	$\{0, M+2c\}$	$\{0, 2M+c\}$
2nd Links	All of them	All of them	All of them
3rd Links	$\{2M, 2M\}$	$\{3M, 3M\}$	$\{6M, 6M\}$

Table 5.1. Arrival and departure times for the links.

Below there is a graphical illustration of this construction for one shipment. Let the variables be named as $a_1, a_2, \ldots, a_m$ and their negation be named as $\overline{a}_1, \overline{a}_2, .., \overline{a}_m$. The link arrival and departure times are assigned in the same order. Assume that the shipment has a clause that includes the following three variables: $a_1, \bar{a}_2, a_3$. Then, we have the arrival and departure times for each link given in Table 5.1.

Here is the corresponding graph for that shipment:

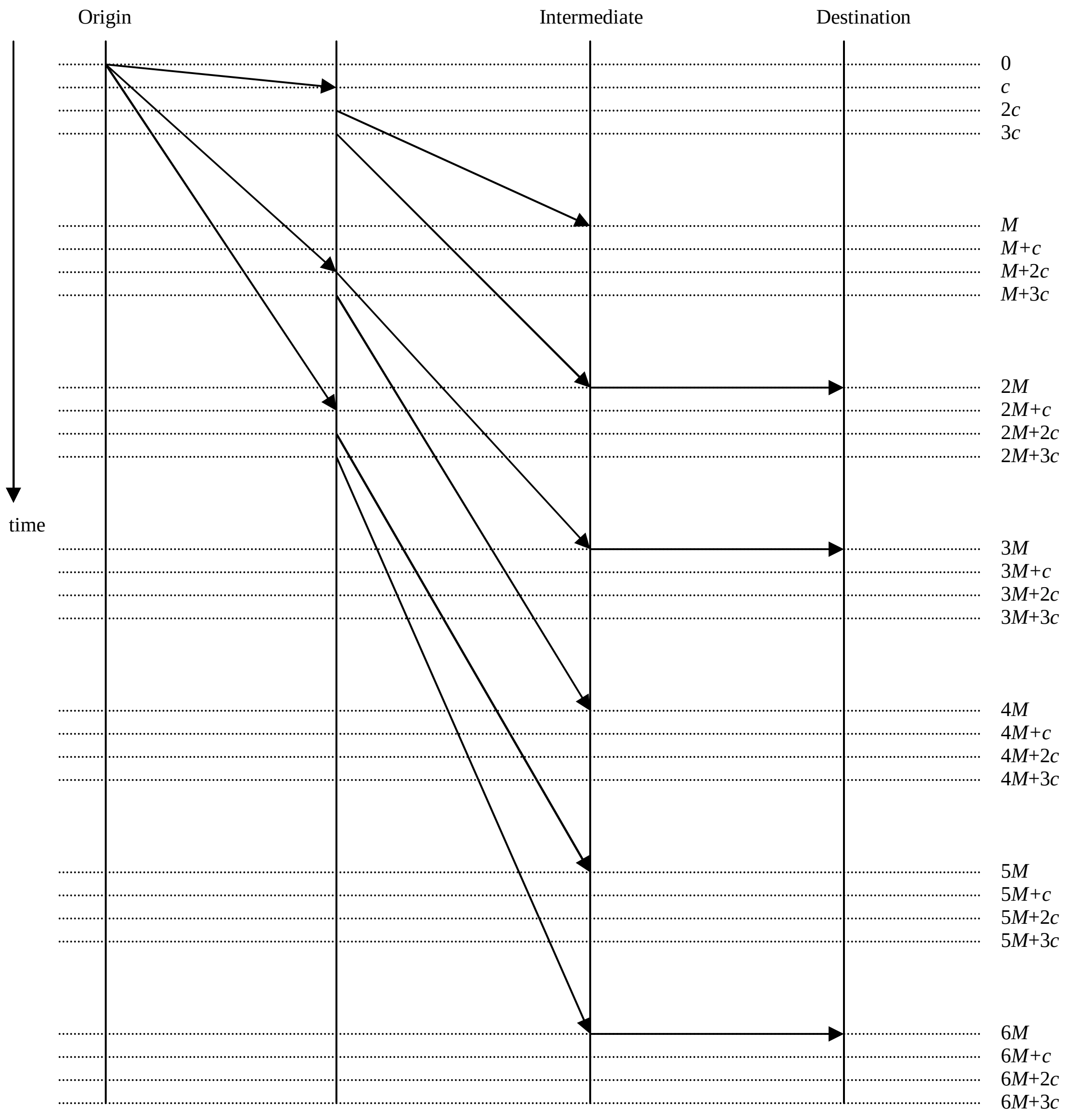


Figure 5.8. Arrival and departure times for the links in Proposition 5.14.

Unfortunately, the USR-DP problem does not have any pseudo-polynomial algorithm to find an optimal solution unless P=NP. We see that solving the USR-DP is not easier than solving the 3-Satisfiability problem. But what is the difficulty level of solving the USR-DP problem? Next, we will transform the USR-DP to a shortest path problem with inconsistent pairs.

Shortest Path with Inconsistent Pairs: Let I be a given a collection of pairs of vertices, referred to as *inconsistent pairs*, under a graph $G = (V, A)$ with two fixed vertices $s,t \in V$. A directed (s-t)-path is called the *I-path* if it contains at most one vertex out of each pair in I. The problem requires finding a shortest path among the I-paths.

Proposition 5.15: *The uncapacitated shipment routing problem with dispatching constraints [USR-DP] is polynomially reducible to the shortest path problem with inconsistent pairs.*

Proof: We will use the shipment-path formulation to simplify the proof. We will convert the general USR-DP instance to a shortest path problem with inconsistent pairs.

Assume that there are n shipments, and each shipment w has p_w paths. We will construct a (n+2) layered graph where each layer represents one shipment and two layers are for origin and destination nodes, as in the following graph:

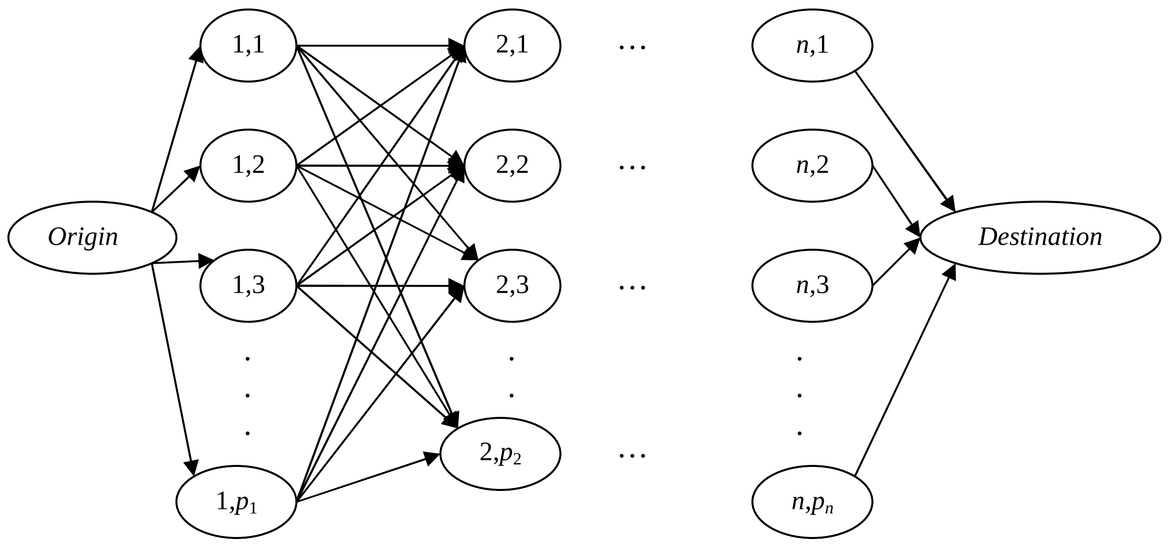


Figure 5.8. Shortest path problem where each layer represents a shipment.

In this graph, node (i,j) represents shipment i and its assigned path j. All the incoming arcs of node (i,j) have a cost of path j that belongs to shipment i. (Incoming arcs of the destination path have no costs.) In this problem, inconsistent pairs are inconsistent paths of the USR-DP problem. By construction, USR-DP is polynomially reducible to the shortest path problem with inconsistent pairs. The following proposition shows a special case of this construction.

Proposition 5.16: *If each shipment appears with only one shipment in the set of inconsistent pairs, the USR-DP problem is solvable in polynomial time via shortest path algorithms.*

Proof: For each shipment in the set of inconsistent pairs, order the shipments such that inconsistent pairs of shipments are ordered consecutively. Construct a graph like the one described in the proof of proposition 5.15. Since each shipment appears with only one shipment and we have ordered them properly, each inconsistent pair has an arc in the shortest path graph. By definition, these arcs cannot be used in the solution, so we delete them. The resulting problem is nothing more than a well known shortest path problem, and it can be solvable in polynomial time (via Dijkstra's algorithm, for instance).

We can generalize this characteristic if inconsistent pairs have a nested structure.

Definition: Assume that i^1, i^2, i^3, i^4 are the layer levels of the shipments in the graph. If there are no (i^1, i^2) and (i^3, i^4) pairs in the set of inconsistent pairs, such that $i^1 < i^3 < i^2 < i^4$, then the inconsistent set has a *nested structure*.

Proposition 5.17: *If the set of inconsistent pairs has a nested structure, the USR-DP problem is solvable in polynomial time.*

Proof: The solution algorithm has two steps:

- In the first step, find a layer that has no inconsistent pairs with other layers. Delete this layer and connect one previous layer and one later layer to each other by summing the related incoming and outgoing arcs of the deleted layer.
- In the second step, find a layer that has only inconsistent pairs with one previous or/and one later layer. Delete the corresponding arcs that are inconsistent.

The algorithm repeats itself until there is no layer left except origin and destination. The algorithm is valid because in each step, we can find a layer that has no inconsistency or inconsistency with only neighbor layers. Otherwise, the nested structure will be violated.

5.7 UPPER AND LOWER BOUNDS

The capacitated shipment routing problem with dispatching policy constraints is hard to solve. Hence, we develop some heuristics to get good feasible solutions which provide upper bounds for the optimal solution. We look at the problem from 3 different perspectives and offer 3 heuristics. We also provide a lower bound that strengthens the uncapacitated shipment routing solution.

5.7.1 Moving-in-Time

In this heuristic, the system has a *state*, which evolves with time, and a state change, which moves the system from one state to another. State changes are called *events*. The process of a system is characterized as a chronological sequence of events. In our problem, the shipment release, departure and arrival links are called *events*, and the time attributes of these events are called *event times*. The heuristic algorithm keeps track of the current time, called *clock*, and the clock moves to the next event start time as the algorithm proceeds.

Our approach constructs a feasible solution by assigning shipments to available links in chronological order. As time goes by, the algorithm assigns the earliest arrived shipment to the earliest available link and changes the arrival time information of the shipment to the arrival time of the assigned link at the next station. The algorithm repeats this process until all shipments reach their destination (by using real or dummy links).

The following is the generic scheme for the Moving-in-Time algorithm.

Algorithm for the Moving-in-Time

```
0    Set eventList = (shipment, currentTime, currentStation) for all shipments where
     (currentTime = shipment release time) and (currentStation = shipment Origin)
1    While eventList is not empty do
2        Set clock to the earliest currentTime in the eventList
3        For all shipments whose currentTime equal to clock do
4            Delete corresponding event from the eventList
```

```
5               Find the earliest available link for the corresponding shipment at
                currentStation and assign that link to the shipment
6               If the destination of shipment is not equal to the destination of the block
                path then
7                   Add (shipment, currentTime, currentStation) event to the
                    eventList:
                        currentTime = arrival time of link at the destination
                        currentStation = destination station of assigned link
8               End If
9           End For
10      End While
```

The algorithm performance could be improved by applying tie-breaking rules for the third step where the shipments have equal *currentTime* attributes. Instead of selecting randomly, choosing a shipment whose earliest available link has minimum capacity could be a better idea.

Although the algorithm works on a first come first served basis, it can find an optimal solution for the uncapacitated shipment routing problem with dispatching constraints.

Proposition 5.18: *Assume that the following assumptions hold:*

- *Objective function is one of the objective functions given in Corollary 5.5,*
- *Earliest available link at step 5 in the Moving-in-Time algorithm is taken as the link for which the arrival time at the destination is the earliest among the available times.*

Then, the Moving-in-Time algorithm provides the optimal solution for the uncapacitated shipment routing problem with dispatching policy constraints [USR-DP].

Proof: Since there are no capacity constraints, all the links that depart after the arrival time of the shipment are available. Choosing the link for which the arrival time is the earliest provides the fastest route to the destination for each shipment. This selection will not violate the dispatching policy constraints, because the earlier shipment can always select the later departing link even if that link arrives the earliest (similar arguments given in the proof of proposition 5.4). The algorithm minimizes the total transit time if the shipment has the unique shipment-block sequence, so the result is true for the objective functions given in corollary 5.5.

5.7.2 Aggregation

The number of constraints and variables in the IP formulation are increasing with the number of shipments in the problem. We can aggregate similar shipments to obtain a smaller problem, but the optimal solution for the aggregated shipments may not be equal to the optimal solution for the actual shipments. However the aggregation could provide a feasible solution close to the optimal solution.

Aggregated Shipment: *An aggregated shipment is a set of similar shipments whose*

- *Origin and destination are the same,*
- *Set of blocks that can carry these shipments is the same,*
- *First available link at the origin that can carry these shipments is the same.*

The volume of the aggregated shipment is the sum of all shipments in that aggregated shipment. By construction, a feasible path for one shipment in the aggregated shipment is also a feasible path for the rest of the shipments in that aggregated shipment.

The following is the generic scheme for the aggregated shipment generation.

Algorithm for the aggregated shipment generation

```
0    Set $W$ = all shipments and set $AW$ = empty set
```

```
1   **While** $W$ is not empty **do**
2       Get $w \in W$ and delete that shipment $w$ from set $W$.
3       Generate an aggregated shipment $aw$ whose origin, destination, volume and
        set of all available blocks are the same as $w$; add it to set $AW$
4       **For** all shipments $w' \in W$ **do**
5           **If** origin, destination, volume, set of all available blocks and the first
            available link at the origin of $w'$ is same as $w$ **then**
6               Add shipment $w'$ to aggregated shipment $aw$
7               Add volume of shipment $w'$ to volume of aggregated shipment $aw$
8               Delete shipment $w'$ from set $W$
9           **End If**
10      **End For**
11  **End While**
```

The heuristic simply solves the original problem optimally by using aggregated shipments and assigns those paths given by the solution to the shipments inside of the corresponding aggregated shipment. Clearly, the aggregation heuristic forces the shipments to use the same path if they are in the same aggregated shipment. In uncapacitated case, this approach satisfies the optimality.

Proposition 5.19: *Assume that the objective function is one of the objective functions given in Corollary 5.5. The following statements are true:*

a) For the capacitated shipment routing problem with dispatching policy constraints [CSR-DP], the problem with aggregated shipments provides a feasible (not necessarily optimal) solution to the problem with normal shipments

b) *For the uncapacitated shipment routing problem with dispatching policy constraints [USR-DP], the optimal solution with aggregated shipments is also optimal with normal shipments*

Proof: (a) Since a feasible path for one shipment in the aggregated shipment is also a feasible path for the rest of the shipments in that aggregated shipment, the solution for aggregated shipments is a feasible solution to the problem for normal shipments.

(b) Without capacity constraints, the shipments use the shortest path. Therefore, the optimal solution with aggregated shipments is also optimal with normal shipments.

5.7.3 LP relaxation correction

We can use the LP relaxation to obtain good feasible solutions. The LP relaxation solution usually contains shipments with integer flows. This heuristic fixes link assignments of those shipments. Then, the remaining problem is resolved as an IP. If the capacity violations are not too high, we hope that the remaining problem is small enough to solve quickly.

The following is the generic scheme for the LP relaxation correction algorithm.

Algorithm for the LP relaxation correction

```
0    Solve the problem as an LP
1    Fix the shipment-link assignments if all the assignments for that shipment are integer
2    Resolve the reduced problem as an IP
3    If the problem is infeasible or takes too much time to solve, terminate the algorithm without solution
```

Remark: *If the problem contains only shipments for which all link assignments are integer in the LP solution, then the LP relaxation is optimal for that problem.*

5.7.4 Strengthening the uncapacitated solution

We also develop some lower bound calculation rules to decrease the optimality gap of feasible solutions obtained in the previous section. The uncapacitated shipment routing problem is a relaxation of the original problem, and the optimal solution to this problem could be obtained easily via shortest path algorithms (or even the Moving-in-Time algorithm described in 5.7.1) under certain cost conditions.

The optimal solution to the uncapacitated shipment routing problem may violate the capacity constraints. Therefore, some of the shipments that use a resource the capacity of which is exceeded should be shipped without using that resource. Let v_w be the volume of shipment w, $\forall\ w \in W$, and u_r be the total volume allowed on resource r, $\forall\ r \in R$. Also, we define set $WR(r, x^*)$ to represent shipments that use resource r for a given solution x^*. Let $cd_{w,r}$ show the cost difference between the shortest path of shipment w that does not use resource r and the current shortest path. Then, the following result holds.

Proposition 5.21: *Let x^* represent the optimal solution of the uncapacitated shipment routing problem with dispatching policy constraints [USR-DP]. Let X_w be one if shipment w will not use the resource r in question. If the optimal value of this problem increased by the optimal value of the following problem;*

$$\left\{ \max_{r \in R} \min \sum_{w \in WR(r,x^*)} cd_{w,r} X_w \mid \sum_{w \in WR(r,x^*)} v_w X_w \geq \sum_{w \in WR(r,x^*)} v_w - u_r ; X_w \text{ is binary for } \forall w \in W \right\}$$

then the resulting value is still a lower bound for the capacitated shipment routing problem with dispatching policy constraints [CSR-DP].

Proof: For each inner minimization problem, the problem finds the minimum possible cost changes to eliminate any capacity violation of the selected resource. Since these re-assignments are necessary anyway, the increment of the objective function is the minimum amount for the problem with capacity constraints. The outer maximization selects the maximum increment among all resources.

There is a close connection between shadow price of capacity constraints and $cd_{w,r}$ parameters. Shadow price is change in the objective function if we change the capacity of corresponding resource with a small amount. If the shipment volumes are small enough, this shadow price could be seen as minimum $cd_{w,r}$ parameter of shipments attached to that resource at the earliest or latest time.

For the problem with a single resource violation and a single path for all shipments (plus dummy paths), the lower bound given in Proposition 5.21 is binding (i.e. gives the optimal value of the capacitated problem). The following proposition strengthens the bound given in Proposition 5.21 even further.

Proposition 5.22: *Let x* represent the optimal solution of the uncapacitated shipment routing problem with dispatching policy constraints [USR-DP]. Let X_w be one if shipment w will not use the resource r in question. Solve the following problem for each $r \in R$:*

$$\left\{ \min \sum_{w \in WR(r,x^*)} cd_{w,r} X_w \mid \sum_{w \in WR(r,x^*)} v_w X_w \geq \sum_{w \in WR(r,x^*)} v_w - u_r ; X_w \text{ is binary for } \forall w \in W \right\}$$

Generate set CD(w) for all $w \in W$ and put $cd_{w,r}$ to corresponding set CD(w) for all positive X_w values. If the optimal value of the uncapacitated shipment routing problem is increased by the amount of $\sum_w \max\{cd_w : cd_w \in CD(w)\}$, then the resulting value is still a lower bound for the capacitated shipment routing problem.

Proof: This proposition is a generalization of the previous one. It adds all shipment path cost differences if the associated shipments are selected for only one problem. If a shipment is selected in more than one problem, then the calculation in this proposition chooses the maximum one among the elements of set *CD*(*w*).

5.8 COMPUTATIONAL RESULTS

We evaluate the performance of our approaches on real data instances. The heuristics were programmed in Java and the IP model uses CPLEX 11.2 via concert technology. The tests are taken on the 11 Dell Poweredge 2950 workstation with 3.73 GHz Xeon and 24 GB of shared memory under Ubuntu Linux system.

In our experiments, we tested on three instances and called these instances 1, 2 and 3. Our objective criterion is total weighted transit times of shipments and our planning horizon is one week. Each instance has over 40000 shipments, over 10000 links and over 8000 resources. In these instances, each shipment may be carried by over 5 blocks. Each block may have over 20 links and each link may use over 10 resources that are subject to capacity.

Using the algorithm in Section 5.7.2, we generate the aggregated shipments by combining at most *k* shipments in one aggregated shipment, where *k* is 2, 5 and *F* (where *F* refers to full aggregation − aggregates all shipments if it can). We test these instances by using one of the three FIFO (dispatching) constraints (Type-A, Type-B and Type-C) developed in Section 5.4. In the problem names of our tables, the first number refers to instance, second one refers to aggregation level and the letter indicates the type of FIFO constraints. In Table 5.2, the details about IP formulation is given for instance 1.

To reduce the problem size, we perform the reachability test (whether the origin node can reach a particular node or not on the time-space network) and eliminate all the unreachable nodes and their potential arcs.

For each problem, number of constraints, variables and nonzero coefficients are given. Then, the IP, LP and heuristic (Moving-in-Time algorithm which performs better) solution times are presented. Finally, the gap (* = heuristic value – best lower bound)/heuristic value) is calculated on the last column.

Problem	Number of constraints	Number of variables	Number of nonzeros	IP solution time (sec)	Root solution time (sec)	Heuristic solution time (sec)	Heuristic vs. best lower bound*
1-F-A	7.1M	216K	48.3M	6753	330	2	6.78%
1-F-B	7.2M	442K	45.2M	82269	269	2	6.84%
1-F-C	350K	227K	1.5M	1925	21	2	5.60%
1-5-A	12.5M	291K	81.9M	11681	583	3	3.53%
1-5-B	12.7M	580K	74.5M	>86.4K	649	3	3.60%
1-5-C	482K	308K	2.0M	3366	37	3	3.14%
1-2-A	33.5M	475K	213.6M	>86.4K	3859	6	3.32%
1-2-B	N/A	N/A	N/A	N/A	N/A	6	N/A
1-2-C	793K	500K	3.3M	3104	91	6	2.54%
1-1-A	N/A	N/A	N/A	N/A	N/A	9	N/A
1-1-B	N/A	N/A	N/A	N/A	N/A	9	N/A
1-1-C	1.3M	846K	5.5M	10450	252	9	2.50%

Table 5.2. CPLEX and heuristic performances for instance 1

We use Moving-in-Time heuristic solution described in Section 5.7.1 as an initial solution. Unfortunately, the lower bounds provided in Section 5.8 are worse than LP solution, so we skip them.

In Table 5.2, the problems generated with Type-C FIFO constraints are solvable much faster than the problems with other FIFO types. Therefore, we use only those ones to obtain the results for instances 2 and 3 in Table 5.3.

The heuristic finds the better solutions as the aggregation level decreases in all of the instances. Other than the non-aggregated problems, we are able to solve all the problems within an hour. The non-aggregate problems are solvable in between 80 minutes and 5 hours.

Problem	Number of			IP solution time	Root solution time	Heuristic solution time	Heuristic vs. best lower bound*
	constraints	variables	nonzeros	(sec)	(sec)	(sec)	
2-F-C	319K	212K	1.4M	803	25	2	15.06%
2-5-C	489K	321K	2.1M	995	42	3	4.30%
2-2-C	850K	552K	3.6M	1812	129	5	3.91%
2-1-C	1.5M	956K	6.2M	5995	318	9	3.66%
3-F-C	354K	235K	1.6M	622	20	2	11.67%
3-5-C	518K	339K	2.2M	797	39	3	3.88%
3-2-C	876K	567K	3.7M	2026	141	6	3.52%
3-1-C	1.5M	971K	6.4M	18925	290	9	3.51%

Table 5.3. CPLEX and heuristic performances for instance 2 and 3

In Table 5.4, the scaled objective values are shown for different level of aggregations with respect to the best lower bound of non-aggregated problem.

Problem	Best lower bound*	Best upper bound*	Heuristic value*
1-F-C	108.18	108.81	116.14
1-5-C	100.05	100.40	103.01
1-2-C	100.04	100.47	102.65
1-1-C	100.00	100.57	102.57
2-F-C	121.42	122.31	142.94
2-5-C	100.02	100.53	104.51
2-2-C	99.99	100.45	104.06
2-1-C	100.00	100.61	103.80
3-F-C	119.39	119.82	135.16
3-5-C	99.98	100.29	104.02
3-2-C	99.98	100.36	103.62
3-1-C	100.00	100.61	103.65

* = 100*(value) / (best lower bound of corresponding non-aggregated problem)

Table 5.4. Scaled objective value changes with different aggregation levels

The objective values are close to each other in the aggregation level of 1, 2 and 5. However, the full aggregation level problems have significantly higher objective values because some of the aggregated shipments may contain huge number of shipments (around 300 in instance 2-F-C). Those aggregated shipment could have volume more than the capacity of some resources and the trip (from one station to another) for those shipments may be impossible.

Chapter 6: Conclusions and Future Work

This dissertation investigates optimization models for transport and service scheduling. The service scheduling problems described here have applications in many decision making activities, such as multi-product lot sizing, telecommunication services and maintenance planning. The transshipment problem focused on in this work is the backbone of many transportation companies.

In the first part, we worked on the problem of deciding which operations a service unit must perform at each customer location given the sequence in which the unit periodically visits these locations. We formulated the problem as an integer program, and proved that it is NP-hard. We discussed the special case in which each site is visited only once per service cycle and showed that it is NP-Hard (in the ordinary sense), and we developed an alternative algorithm based on the shortest path structure.

In Chapter 4, we proposed a heuristic procedure for the general problem for a real-life maintenance application. Computational results for several problem instances show that the proposed heuristic identifies near optimal results very quickly, whereas a general purpose integer-programming solver (CPLEX) is not able to generate an optimal solution even after many hours of computational time. To handle real-life problems, we focused on techniques such as problem reduction, branching variables, and subdividing the problem to smaller problems to get better solution. These strategies improve solution times substantially.

Several opportunities for further research are suggested by this study:

- The problem we considered here is an operational level problem, but there are also strategic and tactical level decisions. At the strategic level, we may have multiple resources and want to partition the customers to those resources. The partition should consider the capacity of the resource, amount of work needed to be done and geographic dispersion of the customers.

- At the tactical level decisions, the focus is on finding the optimal route. The master (full cycle) route should cover all the customers within a reasonable visitation frequencies. The route should balance the visitation requirements of each customer. Furthermore, the relative due date differences between operations for each customer should also be considered.
- This study assumes the full availability of visitation locations or customers. However the customer may not be available at the visitation time. The integration of this service with the other activities operated in those locations is another potential avenue for research.
- Stability analysis of the optimal solution is also important to consider. The additional steps to a sequence should not change the task decisions seriously in the earlier steps. If the solution is stable, the myopic decisions can be obstructed by solving a longer sequence but applying the task decisions on the earlier steps.
- One can also investigate the sensitivity and the robustness of the given solution. In real-life, the calculation of the due date parameters and the duration times of tasks may include uncertainty and is therefore hard to estimate. Understanding the characteristic of the robust solution under uncertainty would be an interesting research direction. This extension can further focus on the price of the robust solution and the value of information.

In the shipment routing problem with dispatching policies, we formulated the problem as a multi-commodity network flow problem with additional side constraints and showed the complexity results. We proposed alternative models and algorithms for lower and upper bound calculations. Computational results show that this problem could be solvable in a reasonable time if we use Type-C constraints. These results also indicate that the optimal objective values of limited aggregation solutions are close to the optimal objective value of non-aggregated solution. Furthermore, Moving-in-Time heuristic provides good initial solutions for the instances we tested on the limited or non-aggregated level.

We also recommend the following potential extensions for future research:

- The methods for lower bound calculation can be further developed. Without dispatching policy constraints, we can obtain a lower bound for the shipment routing problem via Lagrangian relaxation approach by relaxing capacity constraints. The remaining problem can be separable by shipments and solvable via shortest path algorithms. Although there are a few capacity constraints, the number of dispatching policy constraints is huge. If we only relaxed the capacity constraints, the remaining subproblem is NP-Hard for arbitrary Lagrangian multipliers (see Proposition 5.14). The research on uncapacitated shipment routing problem with dispatching policy constraints could be an interesting field.
- Moving-in-Time and aggregation approaches provide good feasible solutions within a reasonable time for tested instances. However there are further improvements one can consider. One of the problems for Moving-in-Time heuristic is tie-breaking rules. Many shipments are released on same time and many arcs are come to their destinations at the same time. Therefore, the selection of shipments for their next trips is highly affected by tie-breaking rules. Eventually, this selection determines the heuristic performance. One can also consider the other heuristic possibilities such as tabu search or LP based heuristics. Especially LP based heuristics could be useful because LP solution times in our problem instances are much smaller than IP solution times.

Effectively correction of LP solution could find a better feasible solution and improve the solution performance.

- The shipment routing problem can be used to analyze the correctness and the effectiveness of the underlying physical network. This research could be developed further to find a stable carrier schedule under demand fluctuations. With the side constraints (capacity and dispatching) in mind, the optimal solution of shipment routing problem deviate from the shortest path solution. The optimal solution for capacitated problem gives an idea about bottlenecks in the transportation system and provides a feedback for carrier scheduling problem. The integration of shipment routing and carrier scheduling problems will lead to better trip plans for shipments and better capacity usage of resources.
- In real life, shipment demands are coming one at a time. Some of the shipments are already in the network and assigned to their trips. However, there are some shipments with uncertain availability and volume at the starting time of the planning horizon. Another direction for research would be making stable shipment routing decisions by using the information on hand. How should planners behave for the shipments that are available at the last minute of their first link departure times? One of the common practices is not using full capacities of resources. Then, how much capacity should be used? What is the cost of not using full capacity? How about the lateness of link arrivals with respect to given schedules? More stable schedule comes at a cost and will give us a benchmark for the routing plan developed in this study.

The importance of transport and service scheduling problems makes them attractive and fruitful research fields even though we have come to the end of our journey: *cursum perficio*.

Appendices

A. ALGORITHMS FOR POLYNOMIALLY SOLVABLE CASES

We dealt with the four polynomially solvable cases and gave their algorithms. Let first examine the sets and parameters we used so far.

Parameter reminder:

K	Set of all operations for all customers
$KI(i)$	Set of all operations for customer i
γ_k	Earliest start time for the execution of operation k
β_k	Latest start time for the execution of operation k
J	Set of all tasks for all customers
$JK(k)$	Set of all tasks that contain operation k
$JI(i)$	Set of all alternative tasks that can be done for customer i
δ_j	Duration for performing task j, for all $j \in JI(i)$, $i = 1, 2 \dots n$

A1. Algorithm 1

Logic of Algorithm 1: The algorithm begins from the first customer and follows the customer order of the sequence. If the service resource comes to any customer before than the earliest start time of the operation v, and operation w is not available, then it returns infeasible otherwise it selects the one of the tasks that has the maximum duration.

Let set A be the set of customers that traveling option is available. The durations are represented with $\delta_{v(i)}$ and $\delta_{w(i)}$ for the task $v(i)$ that has time window and $w(i)$ that has not time window of customer i. Also Δ represents the current time (the time that resource comes to that customer) and $select(1,\dots,n)$ holds the solution vector.

```
1   begin procedure
```

```
$\Delta = 0$
for $i$ = 1 to $n$ do
    if $\Delta \geq \gamma_{v(i)}$ or $i \in A$ then
        select ($i$) = argmax $\{\delta_{v(i)}, \delta_{w(i)}\}$
        $\Delta = \Delta + \max\{\delta_{v(i)}, \delta_{w(i)}\}$
    else
        select ($i$) = infeasible
        exit procedure
    end if
end for
end procedure
```

Proposition 2.5 (related part): *Consider the task assignment problem under zero cost function with n customers. Assume that each customer i requires two operations, v(i) and w(i), and only one of the operations, v(i), restricted with the earliest start time. If waiting is not allowed and the tasks without time window restrictions, w(i), are not available for some customers, Algorithm 1 solves the problem in O(n).*

Proof: The problem has only earliest start time requirements. The algorithm begins from the first customer and follows the customer order of the sequence. It chooses task with maximum duration (among available ones) for each customer. If the resource comes to any customer before the earliest start time of its operation and task w is not available, the problem is infeasible because we cannot come to that step later than that time. Otherwise, selection will give the feasible (so the optimal) solution.

The algorithm has "for" loop for customers. In each iteration, selection, assigning and summation operations are done in $O(1)$. Therefore, the algorithm 1 solves this problem in $O(n)$.

A2. Algorithm 2

Logic of Algorithm 2: In the initialization part, the algorithm finds the minimum possible cycle completion time for the given customer sequence (by selecting tasks with minimum duration). The algorithm begins from the first customer and follows the customer order of the sequence. At customer *i*, if the completion time is smaller than the latest start time of the operation *v*(*i*), the resource performs a task with minimum duration for that customer. Otherwise, the algorithm checks the time that the resource is available for that customer. If the time is earlier than the latest start time of the operation *v*(*i*), the algorithm chooses the operation *v*(*i*); else the problem is infeasible. The algorithm updates the candidate completion time and rescans all the customers. It stops either infeasibility is found or there is no change in the candidate completion time.

Let *T* represents the candidate completion time for the given sequence. The durations are represented with $\delta_{v(i)}$ and $\delta_{w(i)}$ for the task *v*(*i*) that has time window and *w*(*i*) that has not time window of customer *i*. Also Δ represents the current time (the time that resource comes to that customer) and *select*(1,…,*n*) holds the solution vector.

```
1   begin initialization
2   $T = 0$                                              // $T$ represents candidate completion time
3   for $i = 1$ to $n$ do
4       $T = T + \min\{\delta_{v(i)}, \delta_{w(i)}\}$              // minimum possible completion time
5       $select(i) = \arg\min\{\delta_{v(i)}, \delta_{w(i)}\}$      // do the task with minimum duration
6   end for
7   $old\ T = T$
8   end initialization
```

```
9
10  begin procedure
11    flag = false                          // indicator whether candidate completion time
12    while flag = false do                 // changed or not
13       $\Delta = 0$
14       for $i = 1$ to $n$ do
15          if $T < \beta_{v(i)}$ then
16             $\Delta = \Delta + \min\{\delta_{v(i)}, \delta_{w(i)}\}$
17          else
18            if $\Delta \leq \beta_{v(i)}$ then
19               $T = T + \delta_{v(i)} - \delta_{select\ (i)}$
20               select (i) = v(i)          // do the operation related task for customer i
21               $\Delta = \Delta + \delta_{v(i)}$
22            else
23               select (i) = infeasible    // infeasibility detected
24               exit procedure
25            end if
26          end if
27      end for
28      if T = old T then                   // no changes detected
29          flag = true                     // exit while and procedure
30      end if
31      old T = T
32    end while
```

33 **end procedure**

Proposition 2.6 (related part): *Consider the task assignment problem under zero cost function with n customers. Assume that each customer i requires two operations, v(i) and w(i), and only one of the operations, v(i), restricted with the latest start time. Algorithm 2 solves the problem under no-waiting assumption in* $O(n^2)$.

Proof: The problem has only latest start time constraints. The initialization finds the minimum possible completion time T without considering any time window constraints. At line 19, the algorithm updates the candidate completion time T. Observe that if $\delta_{v(i)} \leq \delta_{w(i)}$, T remains same and if $\delta_{v(i)} > \delta_{w(i)}$, T increases. However T will never decrease. Therefore, the algorithm terminates in finite time.

At the beginning, we choose the smallest T and at each iteration, we update T whenever T is greater than $\beta_{v(i)}$ and $v(i)$ is not selected. That means the algorithm increases the T value only if it has to increase it. Therefore, whenever the algorithm comes to step i, the current candidate T cannot be lower. It concludes that if $\Delta > \beta_{v(i)}$, the problem should be infeasible detected at line 23. Otherwise, *select* (1 … n) will give the feasible solution.

The algorithm has for loop with n iterations. In each iteration, comparison, selection, assigning and summation operations are done in $O(1)$. "While loop" occurs whenever T is changed. It changes whenever *select*(i) is assigned to $v(i)$ at step i. Once it is assigned, it would not change anymore for step i. Therefore, the algorithm has at most n iterations in "while" loop. It concludes that the algorithm 2 solves this problem in $O(n^2)$. □

A3. Algorithm 3

Logic of Algorithm 3: The algorithm begins from the first customer and follows the customer order of the sequence. If the service resource comes to a customer before than the earliest start time of some operations, the algorithm excludes the tasks that contain those operations. If there are no tasks left, then the algorithm returns infeasible otherwise it selects the task (among available ones) that has the maximum duration.

Let Δ represent the current time (the time that resource comes to that customer) and *select*(1… *n*) holds the solution vector. Let set *P* represent the available operations whenever the service resource comes to customer location ($\gamma_k \le \Delta$) and *P*` represents the unavailable operations ($\gamma_k > \Delta$). Also, let *Q* holds the tasks that contain only available operations.

```
1  begin procedure
2  $\Delta = 0$
3  for $i$ = 1 to $n$ do
4        $P =: \{k \mid \gamma_k \le \Delta, k \in KI(i)\}$
5        $P` =: \{k \mid \gamma_k > \Delta, k \in KI(i)\}$
6        $Q =: \{j \mid j \in JK(P), j \notin JK(P`)\}$
7        if $Q \ne \varnothing$ then
8            $select\ (i) = \arg\max\{\delta_j \mid j \in Q\}$
9            $\Delta = \Delta + \max\{\delta_j \mid j \in Q\}$
10       else
11           select (i) = infeasible
12           exit procedure
13       end if
14 end for
```

15 **end procedure**

Proposition 2.7 (related part): *Consider the task assignment problem under zero cost function with n customers. Let* $k^{\max} = \max_i \{|KI(i)|\}$ *and* $j^{\max} = \max_i \{|JI(i)|\}$. *If there are only earliest start time restrictions for all operation k and waiting is not allowed, Algorithm 3 solves the problem in* $O(nk^{max}j^{max})$.

Proof: The problem has only earliest start time constraints. The algorithm chooses task with maximum duration among available ones for each customer. Therefore, if $Q = \varnothing$ for some customer *i* (there is no task available at time Δ), clearly the problem is infeasible because we cannot come to that step later than Δ. Otherwise, *select* (1 … *n*) will give the feasible (so the optimal) solution.

The algorithm has “for” loop for customers. The construction of the set *Q* is bottleneck at each iteration. We need to include the tasks of all operations in set *P* and to exclude the tasks of each operation in set *P*`. Each operation may have $j^{\max}$ tasks and set *P* may have $k^{\max}$ operations. Therefore, the algorithm 3 solves this problem in $O(nk^{max}j^{max})$. □

A4. Algorithm 4

Logic of Algorithm 4: In the initialization part, the algorithm finds the minimum possible completion time for the given customer sequence (by selecting tasks with minimum duration) and assigns this length as a candidate completion time. The algorithm begins from the first customer and follows the customer order of the sequence. At each customer, it finds the required operations. (Operation is required if the latest start time is earlier than the candidate completion time.) The algorithm checks the time that the resource is available for that customer. If the time is earlier than the latest start time of all required operations, the algorithm chooses minimum duration task that contains all the required operations; else the problem is infeasible. The algorithm updates the candidate completion time (by summing duration of selected tasks) and rescans all the customers. It stops either infeasibility is found or there is no change in the candidate completion time.

Let T represents the candidate completion time for the given sequence. Let Δ represent the current time (the time that resource comes to that customer) and *select*(1 … *n*) holds the solution vector. Let set $P(i)$ represents the required operations ($\beta_k < T$) for customer i and $P`(i)$ shows the operations of currently selected task for customer i.

```
1   begin initialization
2   $T = 0$                                         // $T$ represents candidate completion time
3   for $i = 1$ to $n$ do
4       $T = T + \min\{\delta_j \mid j \in JI(i)\}$          // minimum possible completion time
5       $select(i) = \arg\min\{\delta_j \mid j \in JI(i)\}$   // do the task with minimum duration
6   end for
7   $old\ T = T$
8   end initialization
9
10  begin procedure
11    $flag$ = false
12    while flag = false do
13        $\Delta = 0$
14        for $i = 1$ to $n$ do
15            $P(i) =: \{k \mid \beta_k < T, k \in KI(i)\}$      //required operations
16            $P`(i) =: \{k \mid k \in KJ(select(i))\}$       //operations in the selected task
```

```
17          **if** $P(i) \setminus P`(i) = \varnothing$ **then**
18              $\Delta = \Delta + \delta_{select(i)}$
19          **else**
// if (current time is less than the latest start time of all required operations) and
// (there is a task that contains all required operations)
20            **if** $\Delta \le \min\{\beta_k \mid k \in P(i)\}$ and $\left\{ j \mid j \in JI(i),\, j \in \bigcap_{k \in P(i)} JK(k) \right\} \ne \varnothing$ **then**
21                $job =: \arg\min\left\{ \delta_j \mid j \in JI(i),\, j \in \bigcap_{k \in P(i)} JK(k) \right\}$
22                $T = T + \delta_{task} - \delta_{select\,(i)}$
23                $select\,(i) = task$
24                $\Delta = \Delta + \delta_{task}$
25            **else**
26                $select\,(i)$ = infeasible
27                **exit procedure**
28            **end if**
29          **end if**
30      **end for**
31      **if** $T = old\ T$ **then**                 // no changes detected
32          $flag$ = true                      // exit while and procedure
33      **end if**
34      $old\ T = T$
35    **end while**
36  **end procedure**
```

Proposition 2.8 (related part): *Consider the task assignment problem under zero cost function with n customers. Let* $k^{\max} = \max_i \{|KI(i)|\}$ *and* $j^{\max} = \max_i \{|JI(i)|\}$. *If there are only latest start time restrictions for all operation k, Algorithm 4 solves the problem under no-waiting assumption in O(n|K|*$k^{max}j^{max}$*).*

Proof: The problem has only latest start time constraints. The initialization finds the minimum possible completion time *T* without considering any time window constraints. At line 22, the algorithm updates the candidate completion time *T*. We come to that step whenever another operation is needed to be done. Therefore the new selected task should have higher duration time than the previous one. In other words, *T* will never decrease. Therefore, the algorithm terminates in finite time.

At the beginning, we choose the smallest *T* and at each iteration, we update *T* whenever another operation is needed to be done. That means the algorithm increases the *T* value only if it has to increase it. Therefore, whenever the algorithm comes to customer *i*, the current candidate *T* cannot be lower. It concludes that if $\Delta > \min\{\beta_k \mid k \in P(i)\}$ the problem should be infeasible detected at line 26. Otherwise, *select*(1 … *n*) will give the feasible solution.

The algorithm has "for" loop with *n* iterations. In each "for" loop iteration, the construction of set $\bigcap_{k \in P(i)} JK(k)$ is bottleneck at lines 20 and 21. Each operation may have $j^{\max}$ tasks and set *P*(*i*) may have $k^{\max}$ operations. Therefore "for" loop runs in *O*(*n*$k^{max}j^{max}$).

"While loop" occurs whenever *T* is changed. It changes whenever an additional operation is needed to be done. Once the operation is an element of the set *P*(*i*), it will always be an element of it (because *T* does not decrease). Therefore, the algorithm has at most $|K|$ "while" iterations. It concludes that the algorithm 4 solves this problem in $O(n|K|k^{max}j^{max})$. □

B. NP HARDNESS OF THE PTA PROBLEM IN APPLICATION

The Periodic Task Assignment [PTA] problem in application described in Chapter 4 is in the category of difficult problems, so called NP-Hard problems. In fact, the well-known 3-partition problem can be written as an instance of the PTA problem.

The 3-partition problem is NP-Hard in the strong sense. (See Karp, 1972 and Garey and Johnson, 1979.) We will show that the 3-Partition problem can be polynomially reducible to the PTA problem. First, we will give the definition of the 3-partition problem.

3-Partition: *Given positive integers $a_1,\ldots,a_{3q}$, b such that*

$$\frac{b}{4} < a_j < \frac{b}{2}$$

and

$$\sum_{j=1}^{3t} a_j = qb\,.$$

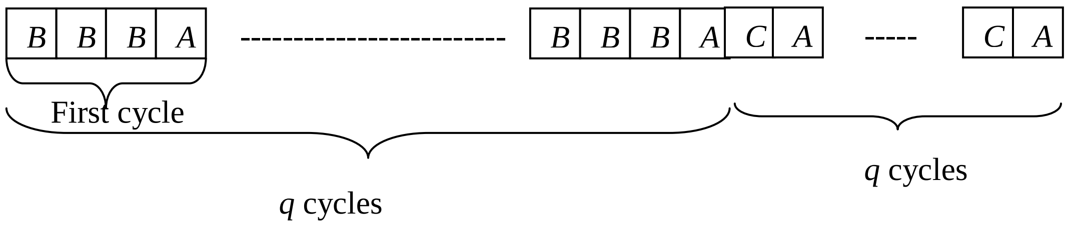
Do there exist q pair-wise disjoint 3 element subsets $S_i \subset \{1,..,3q\}$ such that

$$\sum_{j\in S_i} a_j = b \qquad \textit{for i=1,…q?}$$

Proposition 1: *The 3-Partition problem is polynomially reducible to a periodic task assignment problem [PTA].*

Proof: We represent the 3-Partition problem as an instance of the PTA problem. Our aim is getting an optimal solution for the resulted PTA problem with zero optimal value. We take the following sequence as a PTA instance:

- A sequence consisting of *3* different customers: Customer *A*, *B*, and *C*. First part of the sequence has *q* subcycles that consists of 3*B* and 1*A* customers (*B* customers are the first ones). Second part of the sequence has also *q* subcycles with one C and one A customers (C customers are the first ones). The sequence consists of $m = 6q$ steps.



- Customer A and C require only one operation and customer B requires $3q$ operations.
- Each operation is included in only one task, and each task includes only one operation.

The construction of the PTA instance:

- The duration time for performing task j for customer A is zero and for customer C is b.
- The duration time for performing task j for customer B is a_j, $j = 1, 2, \ldots, 3q$.
- $\alpha_k = qb$ and τ_k is very big number for all operation k of customer B.
- $\alpha_k = \tau_k = b$ for the operation of customer A and for the operation of customer C.
- c_{kt} is one for all operation k and time t in case of due date violation.

We have the following observations in this PTA problem instance:

- Customer C appears q times and each time b units spend on these steps. Then, the completion time should be at least qb.
- Customer B appears $3q$ times and it requires $3q$ operations with $\alpha_k = qb$. Since completion time is at least qb, each operation should be done exactly once to not to get any penalty.
- We know that $\sum_{j=1}^{3t} a_j = qb$ and we have q sub-cycles regarding customer B. If the summation of the task durations of 3-customer B in one sub-cycle is less than b,

there will be another cycle where the summation of durations of 3-customer B in that cycle is greater than b. However, we have $\alpha_k = \tau_k = b$, as the due date for the operation of customer A. To not to pay any penalty, the summation of the task durations of 3-customer B in each cycle should be exactly b.

- Since the operation of customer A should be done in every b units of time, there is no waiting to not to pay any penalty, even if the waiting is allowed.

Therefore, the solution with zero cost function honor that the summation of the durations of 3-customer B in each cycle should be exactly b and each duration time a_j, $j = 1, \ldots, 3q$ appears once. Since each set, indices and parameters in the PTA has at most $O(q)$ items, the reduction will take polynomial time. □

Corollary 2: *The periodic task assignment problem is NP-Hard in the strong sense. Furthermore, there is no ε-approximate heuristic that runs in polynomial time for the PTA problem unless P = NP for any ε > 0.*

Proof: Since the optimal objective value is zero for the PTA problem in the proof of Proposition 1, any ε-approximate heuristic should provide a solution that has zero objective value. This means that the heuristic solves the problem in polynomial time so $P = NP$. □

In the next result, the well-known knapsack problem is written as an instance of PTA problem where no customer is visited twice. (See Karp (1972) for the knapsack problem.)

Knapsack Problem: Let N be the number of items and i be the index of each item, $i = 1, 2 \ldots N$. Each item i has the following attributes:

c_i	Cost of item i if it is selected, for all $i = 1, 2 \ldots N$
a_i	size of item i for all $i = 1, 2 \ldots N$

Let b represent the limit that we need to satisfy, i.e. capacity of knapsack. Each item i has the following decision variable:

X_i = 1 if item i is selected and 0 otherwise, for all $i = 1, 2, \ldots, N$

The *Knapsack* problem can be formulated as an integer program:

[KP] Maximize $\sum_{i=1}^{N} c_i X_i$

s.t.: $\sum_{i=1}^{N} a_i X_i \leq b$

$X_i = 0$ or 1, for all $i = 1, 2 \ldots N$

Proposition 3: *The knapsack problem [KP] is polynomially reducible to the PTA problem where no customer is visited twice [TA].*

Proof: We will convert the general knapsack problem to a task assignment instance. The construction of the TA instance for the indices and sets is as follows:

- A route consisting of $N + 1$ customers. In other words, $n = N + 1$
- Each customer i requires only two operations say v_i and w_i for $i = 1 \ldots N$ and customer $N + 1$ requires only one operation v_{N+1}
- Two tasks are available for each customer $i = 1 \ldots N$. Each task contains one operation. For simplicity, say task v_i includes operation v_i and w_i includes operation w_i for $i = 1 \ldots N$. Customer $N + 1$ has one task called v_{N+1}

The construction of the TA instance for the parameters is as follows:

- Duration time for performing task v_i is a_i and zero for task w_i, for $i = 1, 2 \ldots N$. Also duration time for performing task v_{N+1} is b+1
- $\alpha_k = b$ and τ_k is very big number for all operation k of customer $i = 1, 2 \ldots N$+1
- $c_{kb} = c_i$ for operation $k = v_i$ at time b and $c_{kt} = 0$ for the other times for customer $i = 1, 2 \ldots N$. There will be no cost related to operation w_i for customer $i = 1, 2 \ldots N$.

- $c_{kb} = 1 + \sum_{i=1}^{N} c_i$ for operation $k = v_{N+1}$ at time b and $c_{kt} = 0$ for the other times

In this instance, the cost function value of operation v_{N+1} at time b is very high so this task should be done before than its due date b. Since we know that the duration of the task v_{N+1} is b+1, all the operation v_i for customer i = 1, 2 … N have due and pay penalty c_i if they are not performed. We have the following observations in this TA problem instance:

- The flow constraints (2a) and (2b) try to reach last customer before or at time b by selecting either task v_i or w_i. (Also, the service provider can wait in intermediate steps but the result is also true even if the waiting is not allowed.)
- Duration of tasks v_i are equal to a_i for all steps and duration of tasks w_i are zero, for i =1 … N. Therefore the feasible solution selects the subset of tasks v_i and the duration of selected v_i's cannot exceed period b.
- If the solution does not select to do operation v_i, (equivalently selects to do operation w_i) we will pay c_i at the end for this operation.
- If the solution does not choose any of the operation v_i, total cost would be $\sum_{i=1}^{N} c_i$.

We can define a new decision variable to capture these observations better:

Y_i = 1 if task v_i selected and 0 if task w_i is selected, for all i = 1 … N

Therefore we can rewrite the task assignment problem as:

$$Obj_1 = \text{Minimize } \sum_{i=1}^{N} c_i - \sum_{i=1}^{N} c_i Y_i$$

$$\text{s.t.:} \quad \sum_{i=1}^{N} a_i Y_i \le b$$

$$Y_i = 0 \text{ or } 1, \quad \text{for all } i = 1, 2 \dots N.$$

In the objective function, the first term is constant and does not affect the solution so we can exclude it during the solution process. Also, recall that $\min -Z = -\max Z$. Therefore, we can equivalently write the above formulation as:

$$Obj_2 = \text{Maximize } \sum_{i=1}^{N} c_i Y_i$$

$$\text{s.t.:} \quad \sum_{i=1}^{N} a_i Y_i \leq b$$

$$Y_i = 0 \text{ or } 1, \quad \text{for all } i = 1, 2 \ldots N.$$

There is a one-to-one relation between objectives of these two formulations which is:

$$Obj_1 = -\left(Obj_2 - \sum_{i=1}^{N} c_i \right).$$

Therefore, optimal solution of one of them is also an optimal solution to the other one. Finally, observe that the second formulation is equal to the knapsack problem. Since each set, indices and parameters in TA has at most $O(N)$ items, the reduction will take polynomial time. □

References

Abdekhodaee, A. H., A. Wirth, H. S. Gan. 2006. Scheduling two parallel machines with a single server: the general case. *Computers and Operations Research* **33** 994–1009.

Acharya S. 1998. Broadcast Disks: Dissemination-Based Management for Asymmetric Communication Environments. Ph.D. Dissertation, Brown University. Available at www.bell-labs.com/~acharya.

Agnihothri, S. R., U. S. Karmarkar. 1992. Performance evaluation of service territories. *Operations Research* **40**(2) 355–366.

Ahuja, R. K., K. C. Jha, J. Liu. 2007. Solving real-life railroad blocking problems. *Interfaces* to appear.

Ahuja, R. K., J. Liu, J. B. Orlin, D. Sharma, L. A. Shughart. 2005. Solving real-life locomotive scheduling problems. *Transportation Science* **39** 503–517.

Ahuja, R. K., T. L. Magnanti, J. B. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, New Jersey.

Ahuja, R. K., J. B. Orlin. 1991. Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems. *Naval Research Logistics Quarterly* **38** 413-430.

Anily, S., J. Bramel. 2000. Periodic scheduling with service constraints. *Operations Research* **48**(4) 635-645.

Anily, S., C. A. Glass, R. Hassin. 1998. The scheduling of maintenance service. *Discrete Applied Mathematics* **82** 27-42.

Anily, S., C. A. Glass, R. Hassin. 1999. Scheduling of maintenance services to three machines. *Annals of Operations Research* **86** 375–391.

Armstrong, R., S. Gao, L. Lei. 2008. A zero-inventory production and distribution problem with a fixed customer sequence. *Annals of Operations Research* **159** 395-414.

Assad, A. A. 1978. Multicommodity network flows: A survey. *Networks* **8** 37-91.

Assad, A. A. 1980. Models for rail transportation. *Transportation Research A* **14** 205–220.

Assad, A. A., B. L. Golden. 1995. Arc routing Methods and Applications. M. O. Ball, T. L. Magnanti, C. L. Monma, G. L. Nemhauser, eds. *Handbooks in OR & MS*, Vol. 8. Elsevier Science B.V., The Netherlands, 375-483.

Ball, M. O. 1988. Allocation/Routing: Models and Algoritmhs. B. Golden, A. Assad, eds. *Vehicle routing: Methods and Studies*. North-Holland, New York, NY, 199-221.

Bar-Noy, A., R. Bhatia, J. S. Naor, B. Schieber. 2002. Minimizing service and operation costs of periodic scheduling. *Mathematics of Operations Research* **27** 518–544.

Barnhart, C., C. A. Hane, E. L. Johnson, G. Sigismondi. 1995. A column generation and partitioning approach for multi-commodity flow problems. *Telecommunication Systems* **3** 239-258.

Barnhart, C., C. A. Hane, P. H. Vance. 2000. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research* **48** 318–326.

Barnhart, C., H. Jin, P. H. Vance. 2000. Railroad blocking: A network design application. *Operations Research* **48** 603–614.

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

Berry, P. M. 1993. Uncertainty in scheduling: Probability, problem reduction, abstractions, and the user. *IEE Colloquium on Advanced Software Technologies for Scheduling* Digest no: 193/163.

Bodin, L. D. 1990. Twenty years of routing and scheduling. *Operations Research* **38**(4) 571-579.

Bodin, L. D., G. Fagan, L. Levy. 1992a. The GEOMOD system. *Proc. USPS Advanced Technology Conference*, Vol. 1. United States Postal Service, 413-418.

Bodin, L. D., G. Fagan, L. Levy. 1992b. Vehicle routing and scheduling problems over street networks. *Proc. USPS Advanced Technology Conference*, Vol. 2. United States Postal Service, 625-641.

Bodin, L. D., S. J. Kursh. 1979. A detailed description of a computer system for the routing and scheduling of street sweepers. *Operations Research* **26**(4) 525-537.

Brännlund, U., P. O. Lindberg, A. Nou, J. E. Nilson. 1998. Railway timetabling using Lagrangian relaxation. Transportation Science **32** 358–369.

Campbell, K. C. 1996. Booking and revenue management for rail intermodal services. Unpublished Ph.D. Dissertation, Department of Systems Engineering, University of Pennsylvania, Philadelphia, PA.

Campbell, J. F., A. Langevin. 2000. Roadway snow and ice control. M. Dror, ed. *Arc Routing: Theory, Solutions and Applications*. Kluwer, 389–418.

Castro, J. 2000. A specialized interior-point algorithm for multicommodity network flows. *Siam Journal of Optimization* **10(3)** 852-877.

Cordeau, J. F., P. Toth, D. Vigo. 1998. A survey of optimization models for train routing and scheduling. *Transportation Science* **32** 380–404.

Crainic, T. G., J. A. Ferland, J. M. Rousseau. 1984. A tactical planning model for rail freight transportation. *Transportation Science* **18** 165–184.

Crainic, T. G., A. Frangioni, B. Gendron. 2001. Bundle based relaxation methods for multicommodity capacitated fixed charge network design. *Discrete Applied Mathematics* **112** 73–99.

Crainic, T. G., J. M. Rousseau. 1986. Multicommodity, Multimode Freight ransportation: A General Modeling and Algorithmic Framework for the Service Network Design Problem *Transportation Research B* **20** 225–242.

Dekker, R., F. A. van der Duyn Schouten, R. E. Wildeman. 1997. A review of multi-component maintenance models with economic dependence. *Mathematical Methods of Operations Research* **45** 411–435.

Desrosiers, J., Y. Dumas, M. M. Solomon, F. Soumis. 1995. Time constrained routing and scheduling. M. O. Ball et al., eds. *Network Routing*. Elsevier, 35–139.

Dial, R. 1969. Algorithm 360: Shortest path forest with topological ordering. *Communications of ACM* **12** 632-633.

Dijkstra, E. 1959. A note on two problems in connexion with graphs. *Numeriche Mathematics* **1** 269-271.

Drummond, M., J. Bresina, K. Swanson. 1994. Just-in-case scheduling. *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)* 1098-1104 AAAI Press/MIT Press, Menlo Park, CA.

Duffuaa, S.O., M. Ben-Daya. 1994. An extended model for the joint overhaul scheduling problem. *International Journal of Operations and Production Management* **14** 37–43.

Erera, A. L., J. C. Morales, M. W. P. Savelsbergh. 2005. Robust optimization for empty repositioning problems. Unpublished manuscript.

Farvolden, J. M., W. B. Powell. 1994. Subgradient methods for the service network design problem. *Transportation Science* **28** 256–272.

Federgruen, A., J. Meissner, M. Tzur. 2007. Progressive interval heuristics for multi-item capacitated lot-sizing problems. *Operations Research* **55**(3) 490–502.

Federgruen, A., M. Tzur. 1994a. Minimal forecast horizons and a new planning procedure for the general dynamic lot sizing model: Nervousness revisited. *Operations Research* **42** 456–469.

Federgruen, A., M. Tzur. 1994b. The joint replenishment problem with time-varying parameters: Efficient, optimal and ε-optimal solutions. *Operations Research* **42** 1067–1087.

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

Garey, M. R., D. S. Johnson. 1979. *Computers and Intractability – A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco.

Ghosh., S. 1996. Guaranteeing fault tolerance through scheduling in real-time systems. PhD thesis, University of Pittsburg.

Golden, B. L., R. Wong. 1981. Capacitated arc routing problems. *Networks* **11** 305–315.

Grigoriev, A., J. van de Klundert, F. C. R. Spieksma. 2006. Modeling and solving the periodic maintenance problem. *European Journal of Operational Research* **172** 783-797.

Haghani, A. E. 1987. Rail Freight Transportation: A Review of Recent Optimization Models for Train Routing and Empty Car Distribution. *Journal of Advanced Transportation* **21** 147–172.

Haghani, A. E. 1989. Formulation and solution of a combined train routing and makeup, and empty car distribution model. *Transportation Research B* **23** 433–452.

Haque, L., M. J. Armstrong. 2007. A survey of the machine interference problem. *European Journal of Operational Research* **179** 469-482.

Hariga, M. 1994. A deterministic maintenance scheduling problem for a group of non-identical machines. *International Journal of Operations and Production Management* **14** 27–36.

Hart, E., P. Ross. 1999. An immune system approach to scheduling in changing environments. W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, R. E. Smith, eds. *Proceedings of the Genetic and Evelutionary Computation Conference (GECCO-99)* Morgan Koufman, 1559-1565.

Holte, R., L. Rosier, I. Tulshinsky, D. Varvel. 1992. Pinwheel scheduling with two distinct numbers. *Theoretical Comput. Sci.* **100** 105-135.

Ibaraki, T., S. Imahori, M. Kubo, T. Masuda, T. Uno, M. Yagiura. 2005. Effective local search algorithms for routing and scheduling problems with general time-window constraints. *Transportation Science* **39** 206–232.

Ioachim, I., S. Ge´linas, J. Desrosiers, F. Soumis. 1998. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks* **31** 193–204.

Jamil, M., R. Batta, D. M. Malon. 1994. The traveling repairperson home base location problem. *Transportation Science* **28**(2) 150-161.

Jha, K. C., R. K. Ahuja, G. Sahin. 2008. New approaches for solving the block-to-train assignment problem. *Networks* 48-62.

Johnson, D. B. 1977. Efficient shortest path algorithms. *J. ACM* **24** 1-13.

Jones, K. L., I. J. Lustig, J. M. Farvolden, W. B. Powell. 1993. Multicommodity network flows: The impact of formulation on decomposition. *Mathematical Programming* **62** 95-117.

Karp, R. M. 1972. Reducibility among combinatorial problems. R. E. Miller, J. W. Thatcher, eds. *Complexity of Computer Computations*. Plenum Press, New York.

Kaufman, L., F. Plastria, S. Tubeeckx. 1985. The zero-one knapsack problem with equality constraint. *European Journal of Operational Research* **19** 384-389.

Keaton, M. H. 1989. Designing optimal railroad operating plans: Lagrangian relaxation and heuristic approaches. *Transportation Research B* **23** 415–431.

Keaton, M.H. 1992. Designing optimal railroad operating plans: A dual adjustment method for implementing Lagrangian relaxation. *Transportation Science* **26** 262–279.

Kennington, J. L. 1978. A survey of linear cost multicommodity network flows. *Operations Research* **26** 209-236.

Kenyon, C., N. Schabanel, N. Young. 2000. Polynomial-time approximation scheme for data broadcast. *ACM Symposium on Theory of Computing* **32** 659–666.

Kenyon, C., N. Schabanel. 2003. The data broadcast problem with non-uniform transmission times. *Algorithmica* **35** 146-175.

Koulamas C. P. 1996. Scheduling two parallel semiautomatic machines to minimize machine interference. *Computers and Operations Research* **23** 945–56.

Kraft, E. R. 1998. A reservation-based railway network operations management system. Unpublished Ph.D. Dissertation, Department of Systems Engineering, University of Pennsylvania, Philadelphia, PA.

Kwon, O. K., C. D. Martland, J. M. Sussman. 1998. Routing and scheduling temporal and heterogeneous freight car traffic on rail networks. *Transportation Research E* **34** 101–115.

Lenstra, J. K., A. H. G. Rinnooy Kan. 1981. Complexity of vehicle routing and scheduling problems. *Networks* **11**(2) 221-227.

Leon, V. J., S. D. Wu, R. H. Storer. 1994. Robustness measures and robust scheduling for task shops. *IIE Transactions* **26**(5) 32-43.

Leus, R., W. Herroelen. 2004. Robust and reactive project scheduling: a review and classification of procedures. *International Journal of Production Research* **42** 1599-1620.

Levy, L., L. Bodin. 1988. Scheduling a postal carriers for the United States Postal Service: An application of arc partitioning and routing. B. Golden, A. Assad, eds. *Vehicle routing: Methods and Studies*. North-Holland, New York, NY, 359-394.

Li, L. Y. O., R. W. Eglese. 1996. An interactive algorithm for vehicle routing for winter gritting. *Journal of the Operational Research Society* **47** 217–228.

Marin A., J. Salmeron. 1996. Tactical Design of Rail Freight Networks. Part I. Exact and Heuristic Methods. *European Journal of Operational Research* **90** 26–44.

Marin A., J. Salmeron. 1996. Tactical Design of Rail Freight Networks. Part II. Local Search Methods with Statistical Analysis. *European Journal of Operational Research* **94** 43–53.

Martinelli, D. R., H. Teng. 1996. Optimization of Railway Operations using Neural Networks. *Transportation Research C* **4** 33–49.

McKay, K. N., F. R. Safayeni, J. A. Buzacott. 1998. Task-shop scheduling theory. What is relevant? *Interfaces* **18**(4) 84-90.

Mok, A., L. Rosier, I. Tulshinsky, D. Varvel. 1989. Algorithms and complexity of the periodic maintenance problem. *Microprocessing and Microprogramming* **27** 657-664.

Newton, H. N., C. Barnhart, P. M. Vance. 1998. Constructing railroad blocking plans to minimize handling costs. *Transportation Science* **32** 330–345.

Nozick, K., E. K. Morlok. 1997. A model for medium-termoperations planning in an intermodal rail-truck service. *Transportation Research A* **31** 91–107.

Orlowski, S. 2009. Optimal Design of Survivable Multi-Layer Telecommunication Networks. Unpublished Ph.D. Dissertation, Mathematik und Naturwissenschaften der Technischen, Universitat at Berlin, Germany.

Ouorou, A., P. Mahey, J.-Ph. Vial. 2000. A survey of algorithms for convex multicommodity flow problems. *Management Science* **46(1)**126-147.

Pioro, M., D. Medhi. 2004. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Elsevier, San Francisco.

Riccio, L. J. 1984. Management science in New York's Department of Sanitation. *Interfaces* **14**(2) 1-13.

Riccio, L. J., A. Litke. 1986. Making a clean sweep: Simulating the effects of illegally parked cars on New York City's mechanical street-cleaning efforts. *Operations Research* **34**(5) 661-666.

Sadeh, N., S. Otsuka, R. Schelback. 1993. Predictive and reactive scheduling with the MicroBoss production scheduling and control system. *Proceedings of the IJCAI-93 Workshop on Knowledge-Based Production Planning, Scheduling, and Control* 293-306.

Schabanel, N. 2000. The data broadcast problem with preemption. *17th International Symposium on Theoretical Aspects of Computer Science: Lecture Notes in Computer Science* **1770** 181–192.

Stadtler, H. 2003. Multilevel lot sizing with setup times and multiple constrained resources: Internally rolling schedules with lot-sizing windows. *Operations Research* **51** 487–502.

Stecke, K. E., J. E. Aronson. 1985. Review of operator/machine interference models. *International Journal of Production Research* **23** 129–151.

Stricker, R. 1970. Public sector vehicle routing: The Chinese postman's problem. Unpublished Masters Thesis, Massachusetts Institute of Technology, Cambridge, Mass.

Su, S. I. (I.) 1992. The general postman problem-Models and Algorithms. Unpublished Ph.D. Dissertation, Collage of Business and Management, University of Maryland, Md.

Suerie, C., H. Stadtler. 2003. The capacitated lot-sizing problem with linked lot sizes. *Management Science* **49** 1039–1054.

Tagmouti, M., M. Gendreau, J. Y. Potvin. 2007. Arc routing problems with time-dependent service costs. *European Journal of Operational Research* **181** 30-39.

Taillard, E. D., P. Badeau, M. Gendreau, F. Guertin, J. Y. Potvin. 1997. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science* **31** 170–186.

Thomet, M. A. 1971. A user oriented freight railroad operating policy, *IEEE Transaction Systems Man Cybernetics* **1** 349–356.

Van Dyke, C. D. 1992. Trip planning and seamless rail transportation. *Proceeding of the Transportation Research Forum* **6** 243-263.

Van Dyke, C. D. 1994. Status of trip planning and car scheduling systems in the railroad industry. *Proceedings of the Transportation Research Forum* Daytona Beach 218-235.

Wallace, R. J., E. C. Freuder. 1998. Stable solutions for dynamic constraint satisfaction problems. *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP-98)* 447-461.

Wei, W., C. Liu. 1983. On a periodic maintenance problem. *Operations Research Letters* **2** 90-93.

Wu, S.D., E. Byeon, R.H. Storer. 1999. A graph-theoretic decomposition of the task shop scheduling problem to achieve scheduling robustness. *Operations Research* **47**(1) 113-124.

Vita

Kursad Derinkuyu received his B.Sc. and M.Sc. degrees in Industrial Engineering from Bilkent University, Ankara, Turkiye, in 2002 and 2004. Derinkuyu continued his studies as a graduate student in Industrial and Systems Engineering at Lehigh University and received his second M.Sc. degree in 2006. During his graduate studies at Bilkent and Lehigh, he performed research on robust optimization, S-Procedure and its variants.

In the Fall 2006, he joined the graduate program in Operations Research and Industrial Engineering at the University of Texas at Austin where he pursued his doctoral studies. Under the supervision of Dr. Anant Balakrishnan and Dr. Erhan Kutanoglu, he worked on service scheduling and transshipment problems using mathematical programming and combinatorial optimization. During his doctoral study, he interned at AMD and BNSF companies to improve supporting tools for production planning and logistics.

Kursad joined the Industrial Engineering Department of Hacettepe University as a faculty member in May 2011. He is married and has one daughter.

Permanent address: Hacettepe Universitesi, Endustri Muhendisligi
06800 Beytepe, Ankara, TURKIYE

This dissertation was typed by the author.