ANALYSIS OF OIL-PIPELINE DISTRIBUTION OF MULTIPLE PRODUCTS SUBJECT TO DELIVERY TIME-WINDOWS

A Dissertation

by

PHONGCHAI JITTAMAI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2004

Major Subject: Industrial Engineering

ANALYSIS OF OIL-PIPELINE DISTRIBUTION OF MULTIPLE PRODUCTS SUBJECT TO DELIVERY TIME-WINDOWS

A Dissertation

by

PHONGCHAI JITTAMAI

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Approved as to style and content by:

Alberto Garcia-Diaz (Chair of Committee)	Cesar O. Malave (Member)
Amarnath Banerjee (Member)	Marietta J. Tretter (Member)
Brett A. Peters (Head of Department)	

December 2004

Major Subject: Industrial Engineering

ABSTRACT

Analysis of Oil-Pipeline Distribution of Multiple Products

Subject to Delivery Time-Windows. (December 2004)

Phongchai Jittamai, B.ENG., Thammasat University, Bangkok, Thailand;

M.S., Texas A&M University

Chair of Advisory Committee: Dr. Alberto Garcia-Diaz

This dissertation defines the operational problems of, and develops solution methodologies for, a distribution of multiple products into oil pipeline subject to delivery time-windows constraints. A multiple-product oil pipeline is a pipeline system composing of pipes, pumps, valves and storage facilities used to transport different types of liquids. Typically, products delivered by pipelines are petroleum of different grades moving either from production facilities to refineries or from refineries to distributors. Time-windows, which are generally used in logistics and scheduling areas, are incorporated in this study.

The distribution of multiple products into oil pipeline subject to delivery time-windows is modeled as multicommodity network flow structure and mathematically formulated. The main focus of this dissertation is the investigation of operating issues and problem complexity of single-source pipeline problems and also providing solution methodology to compute input schedule that yields minimum total time violation from due delivery time-windows. The problem is proved to be NP-complete. The heuristic approach, a reversed-flow algorithm, is developed based on pipeline flow reversibility to

compute input schedule for the pipeline problem. This algorithm is implemented in no longer than $O(T \cdot E)$ time. This dissertation also extends the study to examine some operating attributes and problem complexity of multiple-source pipelines. The multiple-source pipeline problem is also NP-complete. A heuristic algorithm modified from the one used in single-source pipeline problems is introduced. This algorithm can also be implemented in no longer than $O(T \cdot E)$ time.

Computational results are presented for both methodologies on randomly generated problem sets. The computational experience indicates that reversed-flow algorithms provide good solutions in comparison with the optimal solutions. Only 25% of the problems tested were more than 30% greater than optimal values and approximately 40% of the tested problems were solved optimally by the algorithms.

To my dad and mom, my very first and lifelong teachers.

To my three wonderful sisters, who always care and support me.

To my grandaunt, who has boundless energy and persistence to achieve her goals.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest appreciation and gratitude to my advisor, Dr. Alberto Garcia-Diaz. I am thankful to have him providing me valuable support and guidance throughout my years under his advisory. He is more like a father to me and I am grateful to have him inspiring, motivating and leading me the way to this path of success. My work cannot be fulfilled without his valuable counseling, expertise and leadership. He is such a great example for me and I am grateful to follow in his footsteps. I would like to extend my sincere gratitude to Dr. Juan Carlos García Díaz for his suggestion on this research topic and also his valuable guidance and comments. I would like to extend my appreciation to Dr. Amarnath Banerjee, Dr. César O. Malavé and Dr. Marietta J. Tretter for their comments, encouragement and support as members of my graduate committee.

I would like to thank Suranaree University of Technology and the Royal Thai Government, which provided me with the scholarship and support to pursue this degree.

Special thanks are extended to Nantika Smavatkul for being such a good friend and for being someone whom I could always count on.

I would like to show my heartfelt appreciation to my grandaunt who has been such a great example for me, for being persistent and not giving up anything easily regardless of how difficult the tasks will be. I am truly indebted to her support and inspiration.

Finally, I would like to convey my sincere appreciation to my parents and sisters. Dad and mom, I have always been very grateful to be your son. They have always provided me with unlimited love and support. They have been my first and enduring teachers who have guided me to this success. They have always had words of encouragement and valuable suggestions for their children. They have been such a great model for me and provided me footsteps to follow. My dad and mom have been an important part of the fulfillment of my degree. I am also thankful to have my three sisters, my niece and two nephews providing me with their love, endless support and understanding which are greatly essential in helping me to achieve this success.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

Pipeline transportation has played an essential role in supporting the growth of economic prosperity for over decades. Pipelines are widely used to transport different kinds of products, such as water, oil, and gas. In some less common cases, pipelines are also used to deliver coal, ores, aggregates, and other valuable minerals. Compare to the other types of common freight transportation, the transport of products along the pipelines is considered unconventional. Unlike other common transportation modes that usually move carriers of stationary shipment, products in the pipelines are the moving parts that travel along the stationary pipelines.

Despite the fact that pipelines require initial large amount of capital investment, operating costs are much lower compared to other freight modes. There are plenty of benefits due to this distinction. The large amount of products, e.g. crude oil or natural gas, can be sent out for deliveries at destinations in the enormous distance away without sharply escalation of freight costs with the more distance the products travel. This transportation mode also brings about the superiority in maintaining safety, reliability

---

This dissertation follows the style and format of *Annals of Operations Research*.

and monitoring the distribution of products.

The oil pipeline activity is measured in term of trunkline-traffic, i.e. barrel-mile (bbl-mile). Namely, 1 bbl-mile is equal to moving 1 bbl of petroleum product for 1 mile. Based on the data available from Federal Energy Regulatory Commission (FERC), the total US trunkline-traffic in 2002 is 3,562,652 million bbl-miles and the total US interstate pipeline mileages of all pipeline companies in 2002 are 149,619 miles.

In the U.S. alone the petroleum products consumption rate is at almost 20 million barrels per day in 2002 and this rate is likely to be non-decreasing. Thus, pipeline engages in an irreplaceable role in transport petroleum products from supply sources to demand points all over the country that propels the development and the shape up of the petroleum industry. More than 60% of these petroleum products delivered in the U.S. use pipeline as primary mode of transportation. The rest is transported by trucks, rail and water carriers. Pipelines have advantages over other modes of transportation in terms of surpassing the geography limitation and economical freight cost. According to the figure from Association of Oil Pipe Lines (AOPL), the transport of oil along pipelines accounts for more than 17% of the freight transferred all over the country, however it accounts for less than 2% of the national freight cost. In order to obtain the picture of how economical oil pipeline transport is, assuming each truck holds approximately 200 barrels and can travel 500 miles a day, it would require a fleet of 3,000 trucks, with one truck arriving and unloading in every two minutes, to replace a

150,000-barrel a day, on 1,000-mile pipeline. It would take about 75-car train, with a unit train of 2,000-barrel tank car, to replace the same 150,000-barrel per day pipeline[1].

The oil market in the United States is outlined into five regions according to its geography, the Gulf Coast, the East Coast, the Midwest, the Rocky Mountain Region, and the West Coast.

The Gulf Coast is the largest supply area in the U.S. It supplies crude oil to the Midwest and refined products to the Midwest and the East Coast. The highest demand region is the East Coast. Most of imported crude oil is delivered to this area. The Midwest processes crude oil produced locally and sent from the Gulf Coast and also from Canada. The Rocky Mountain Region has the lowest petroleum consumption, but has increased growth in recent years. This area processes crude oil sent from Canada plus local supply. The West Coast is logistically separated from the rest of the nation. Crude oil in this region is supplied mainly from the Alaskan oil fields. The rest of production in this region comes from California. In 2000, 70% of oil shipment among these five regions is conducted via pipelines.

1.1 Introduction to Oil Pipeline Operations

The most important objective of the oil pipeline problem is to transport products from sources to designated destinations within appropriate delivery times. In general, primary components of any pipeline system are pumps, valves, pipes, and storage tanks.

[1] All statistics came from Association of Oil Pipe Lines.

Most petroleum products pipelines have multiple sources and destinations. Oil is commonly traveled along pipelines by the propulsion from centrifugal pumps powered by either electricity or gas turbines. Pumps are located along the pipelines at the approximate interval of 20 to 100 miles, depending on the geography, size of the pipelines and capacity requirements. Multiple petroleum products are transported throughout the pipelines. For example, at Colonial Pipeline, whose pipelines connect from the Gulf Coast to the east Coast (1,500 miles), transports 38 different grades of gasoline, including reformulated gasoline (RFG) and multiple vapor pressures for each grades, and seven grades of home heating oil and diesel fuel.

Pipeline structures can be either: a path (*P*), a tree (*T*), an acyclic graph (*A*), or a general graph (*G*). Flow movement along the graph can be either directed (*D*) or undirected (*U*). If the pipeline network is either a path or a tree, the path between any pair of each source and each destination is unique. Otherwise, there exists more than a single path between any pair of source and destination nodes.

Products transported along the oil pipeline are categorized into two forms, fungible and segregated products. Fungible products are generic products. Shippers will receive equivalent product but may not get back the product actually shipped. Namely, products are interchangeable in a fungible pipeline. Fungible products provide shippers with a significant degree of flexibility for scheduling lifting and delivery times. Generally, gasoline is categorized as fungible product. On the other hand, segregated products are branded products and are not interchangeable. Shippers receive the same product they inject into the pipeline on segregated shipments.

Typically, products are transported in pipelines in form of batches. A batch is a quantity of homogeneous product or grade bundled together and then injected into pipeline. For fungible pipeline, products that meet common requirements can be put together as a batch and shipped through the pipeline. Colonial Pipeline Company defines the meaning of a fungible batch as a batch of petroleum product meeting carrier's established specifications, which may be mixed with other quantities of petroleum product meeting the same specifications. Batch sizes vary according to the sizes of pipelines. Batch sizes can be as small as 2,500 barrels and as big as 3,200,000 barrels depending on requirements of each pipeline company.

On the interface of any pair of batches, the intermixing of two products is unavoidable. Different liquids or fluids of different viscosity tend to mix if they are transported consecutively. One way to prevent such mixing is shipping products in turbulent flow. This method diminishes the settling of heavier fluid. Despite this effort, the interface mixing still arises. If products are of similar gasoline but different grades, the mixture is put in the lower grade product. If two products are not the same, such as gasoline and diesel, the "transmix", which is the hybrid product created by intermixing at the interface, must be channeled to separated storage and reprocessed.

Products in pipelines move at approximately three to five miles per hours in the main pipelines at Colonial Pipeline. The greater the volume being shipped, the more rapidly the product moves. Generally, the transportation of any batch from Houston, Texas to the New York harbor can take from 14 to 24 days via Colonial Pipeline, with the average shipping time of 18.5 days. In order to maintain proper function of the

pipeline transport, it is required that the pipeline be full at all time. In the scenario where the flow direction of pipeline is uphill, if the liquid in the pipeline is not at its full volume, it lacks the propulsion to forward the flow all the way through the end of pipeline. Flow continuity is a crucial property of the function of pipeline transport.

Determine the cost function that truly represents the cost of operating pipeline is cumbersome due to the fact that the energy cost structures of different power companies are not under the same standard. Additionally, the difference in the way batches are sequenced can draw a significant consequence on the operating costs. Poor-sequenced batches can lead to high operating cost. For example, a sequence containing the shipment of a considerable order of small batches causes numerous stoppages along the pipeline to siphon off these orders. Each stoppage immobilizes the downstream movement and also halts the pump operation. Stoppage has financial damage in terms of shorten pump life cycle and additional energy cost. The estimated cost for each stoppage at Colonial Pipeline is $50,000.

Sequencing batches on any pipelines, even on a simple pipeline, is not an easy task to do. There are various constraints that pipeline operators have to take into consideration. Some products are not allowed to travel consecutively. Practically, products may be required to arrive within time-window restrictions. These time-windows restrictions are caused by the availability and/or limitation of storage capacity at delivery points, production process and/or storage requirements at sources and/or destinations, demand by consumers to be delivered at particular range of time, connecting transportation modes arrangement to deliver to final destinations, and some

other logistical reasons. The consequences of being unable to deliver products within time-window requirements may possibly cause several significant effects. These effects can be interpreted in terms of economic losses and financial penalties.

The decisions on pipeline scheduling are likely to have considerable impact on the cost effectiveness of the whole operation. As a result, it is a crucial task for pipeline operators to spend their great effort to produce decent and effective pipeline sequencing and scheduling. This is a challenging task since pipeline scheduling, by its nature, is a difficult optimization problem and pipeline problems are commonly tackled by brute force in conjunction with past experience of pipeline operators as pointed out by Crane et al. (1999). Time-window is a new concept in logistic management used mostly in scheduling and inventory problems. The introduction of this concept to the pipeline problem makes this dissertation more challenging to work on.

1.2 Scope of the Dissertation

In this study, we explore a pipeline scheduling problem in which products are pushed into the pipe at a source moving on a fixed-path pipeline and pushed out of the pipe at a destination node. Delivery time-windows are taken into consideration in this study and products are expected to be delivered within these time-windows. The principal purposes for conducting this research is to provide model formulation and practical solution methodologies in order to find input schedule for the distribution of multiple-product into oil pipeline subjected to delivery time-windows that produces a

minimum value of the total number of delivery time that violates due delivery time-window requirements. We called this problem an "*oil-pipeline distribution of multiple products subject to delivery time-windows*". To accomplish these objectives, the following issues will be considered:

(1) Construct a problem model to represent the scheduling and transportation of multiple products with delivery time restrictions for a single supply source and various delivery points;

(2) Develop solution methodology for single-source, oil-pipeline distribution of multiple products subject to delivery time-windows;

(3) Extend and work on the model from (1) to accommodate the pipeline structure with multiple sources. Develop a problem framework and a guideline for solution methodology based on an approach provided for single-source pipelines;

(4) Provide an analysis of the problem and solution methodology;

(5) Illustrate computational results by working and analyzing on random problem sets.

1.3 Literature Review

The literature survey was conducted in three major areas that were pertinent to our purposed research: oil pipeline scheduling, dynamic programming in scheduling and sequencing areas and time-windows. In the following three subsections are the highlights of the important results found in these areas.

1.3.1 Oil Pipeline Scheduling

To the best of our knowledge, little work has been done and there is not much exploration in the subject of oil pipeline scheduling. There are a small number of publications found related to this area. The highlight of some of the important results published is provided below.

Camacho et al. (1990) used simulation method on oil pipeline networks in search for scheduling and operating pipelines that yielded an optimal operation. The optimal pipeline operation is determined by finding appropriate pump rates that generate the minimization of power costs.

The pivotal study in this area was reported in the publication by Hane (1991). This groundbreaking and inaugural research focused on putting down the extensive fundamental framework of the problem as well as providing the analysis of pipeline scheduling, particularly on the scheduling of a multiple product pipeline. The author examined an optimization strategy in order to minimize the operating cost of the pipeline incurring from energy and maintenance costs of pumping the fluids. Basically, the author focused the work on minimizing pump restart cost of pipeline. The operating cost function is minimized by a combination of local heuristics, greedy procedure and dynamic programming approach. However, the authored concluded that it is still uncertain if there are any polynomial solution procedures for the cost minimization problem and this inquiry remains open. This problem is appealing because it may not be in NP-hard, but does not require enumeration of a solution space.

Hane and Ratliff (1995) also examined the sequencing of multiple commodities in pipelines in order to minimize operating costs. The sequencing problem was decomposed into subproblems and solved by branch-and-bound approach. The solution method was tested on small problems and the results showed that branch-and-bound algorithm was able to explore a small region of the solution space within a reasonable amount of time. Sasikumar et al. (1997) introduced a knowledge-based heuristic search approach to compute a good one-month pumping schedule for a single-source, multiple-destinations, and multiple-products oil pipeline problem. Crane et al. (1999) used genetic algorithms to schedule problem of multiple fungible products pipelines. The objective of this research is to make the shortages at demand points as minimal as possible. The authors tested the genetic algorithms approach on a small problem instance with two petroleum products to transport on eight demand locations. Paolucci et al. (2002) applied simulation approach to work on the problem of crude oil supply allocation from port to refinery tanks through pipeline. The objectives of this problem are twofold: the first and critical one is to minimize tanker service time in order to avoid idle time waiting for tank availability; and the other is to allocate crude oil supplies to appropriate tanks in order to minimize amount of unaccepted crude oil. The authors applied the methodology to the small to medium-sized refinery system.

There are a few publications that highlight more on the mathematical formulation of oil pipeline scheduling problem. Shah (1996) presented a mixed-integer linear programming (MILP) formulation of the crude oil scheduling problem from a supply facility to a refinery, delivering multiple products on single pipeline. The objective of

his research is to minimize the amount of products left in a tank after feeding to a crude distillation unit (CDU). The model was developed for a single refinery problem. Rejowski and Pinto (2002) developed a MILP formulation on systems of multiple-product pipeline with the objective of minimizing operating costs emerged from inventory cost, pumping cost and transition cost. The formulation was tested on small pipeline systems that contain up to five demand depots and four products delivered on the three-day time horizon.

1.3.2 Dynamic Programming and Sequencing

Dynamic programming [Bellman (2003)] is a mathematical procedure designed primarily to improve the computational efficiency of select mathematical programming problems. It is also considered as a computational method that allows us to break up a complex problem into a sequence of easier subproblems. Dynamic programming, created by Richard Bellman, has its uniqueness lies in the principle of optimality and it is this principle that the entire concept of dynamic programming is based on.

Dynamic programming has been prevalently used in many areas; however, it is our intention to focus the literature review in the scope of sequencing problems. Held and Karp (1962) used dynamic programming to work on three sequencing problems: a scheduling problem involving arbitrary cost functions, a traveling salesman problem, and an assembly-line balancing problem. Ibaraki and Nakamura (1994) solved the minimization of weighted sums of earliness and tardiness for single machine scheduling

by using successive sublimation dynamic programming. This technique helps keep the number of states generated to be within manageable level. Mingozzi et al. (1997) worked on the traveling salesman problem with time-window and precedence constraints, which is a NP-hard problem. They proposed an algorithm based on dynamic programming to solve and reduced state space graph by using bounding functions. Ioachim et al. (1998) used dynamic programming to solve shortest path problem with time-windows and linear node costs on the node service start times. The proposed algorithm was proved to perform better empirically than the discretization approach. Lorigeon et al. (2002) scheduled jobs in a two-machine open shop by using pseudo-polynomial time dynamic programming. The authors assumed in the model that a machine is not available at all time and job processing can be resumed after interruption when a machine is available.

1.3.3 Time-Windows

The concept of time-windows is recognizable in scheduling and routing problems. There are numerous literatures in this area as can be seen in the survey reviewed by Solomon and Desrosiers (1988).

Many researchers incorporated time-window concept into vehicle routing problem (VRP) and solved in different ways. It is known that VRP is NP-hard and by further restricting the problem with time-window constraints makes it NP-hard as well. Kolen et al. (1987) used branch and bound to solve vehicle routing problem with time-

windows to minimize total route length. Solomon (1987) provided several algorithms to solve vehicle routing and scheduling problems with time-windows. Fisher Jornsten and Madsen (1997) used lagrangian decomposition method to solve VRP with time-windows. Desaulniers et al. (1998) performed a vehicle fleet scheduling with time-windows to minimize cost by formulating the problem as non-linear multicommodity network flow model with time variables and used column generation approach to solve. Chen et al. (2001) studied the job shop scheduling problem with time-windows in order to minimize the weighted earliness and tardiness cost. The problem was solved by using lagrangian relaxation and algorithm to find a feasible schedule within or approximately within the time-windows.

1.4 Research Objectives

The principal purposes for conducting this research is to provide model formulation and practical solution methodologies for oil-pipeline distribution of multiple products subject to delivery time-windows in order to find an input schedule that minimizes the total number of delivery time that violates time-window requirements. In order to accomplish these objectives, the following issues will be considered:

(1) Construct a problem model to represent the scheduling and transportation of multiple products with delivery time restrictions for a single-source and multiple destinations;

(2) Develop solution methodology for a oil-pipeline distribution of multiple products subject to delivery time-windows;

(3) Extend and work on the model from (1) to accommodate the pipeline structure with multiple sources. Develop problem framework and guideline for solution methodology based on approach provided for single-source pipelines;

(4) Provide analysis of the problem and solution methodology;

(5) Illustrate computational results by working and analyzing on random problem sets.

1.5 Organization of the Dissertation

This dissertation is organized as follows. Chapter II studies the model and provides the mathematical formulation of the oil-pipeline distribution of multiple products subject to delivery time-windows. Chapter III discusses the pipeline scheduling problem on a single-source, simple pipeline (a pipeline with a directed path structure). The solution methodology is also provided in this chapter. The multiple-source pipeline problem is introduced and discussed, based on an extension of the study of a single-source pipeline, in Chapter IV. Computational implementation and results are displayed in Chapter V. Chapter VI concludes this dissertation, provides the contributions of this research and suggests the directions of possible research in the future.

CHAPTER II

MODEL FORMULATION OF OIL PIPELINE DISTRIBUTION OF MULTIPLE PRODUCTS SUBJECT TO DELIVERY TIME-WINDOWS

In this chapter we outline a network model and a mathematical formulation that represent the oil-pipeline distribution of multiple products subject to delivery time-windows. We define the scope of the problem and elementary terms used throughout this study. Principal assumptions adopted in this dissertation are also laid out in this chapter.

We can define the pipeline problem, **P**, by using a set of orders, **O**, and a pipeline structure. The set of orders **O** composes of product information of each batch entering the pipeline, its corresponding source and destination nodes, delivery time-windows. The pipeline structure is formed as a directed network *G(V, E)*, where *V* is a set of nodes and *E* is a set of all edges. The capacity of each edge is defined in term of volume *v(e)*, representing the pipe capacity between any two adjacent nodes. The volume of each edge is assumed to be integral.

This chapter generates the model of oil-pipeline distribution of multiple products subject to delivery time-windows studied in this dissertation. The building of the model

is constructed based on the following assumptions. The structure of a pipeline in this study is assumed to be a path. Namely, each node has only one immediate predecessor and one immediate successor nodes. Products entering the pipeline are assumed to be in form of batches where each batch contains a section of homogeneous product content and has volume of one unit. Each batch is injected into the pipe from the source node, traveled along the pipe and pushed out only at its designated destination node. It is assumed that there is no mixing of products at the interface of any two batches along the pipe. The splitting or merging of any batches at any nodes along the pipeline is prohibited. These assumptions make flow stop downstream when delivery is made. Thus, this forces flow traveling along the pipeline by originating at one source node and departing at one destination node. Furthermore, it is not allowed for any batch to travel beyond its corresponding destination.

2.1 Definitions

A *path* in a graph $G(V, E)$ is a sequence of edges (S, n_1), (n_1, n_2), …, (n_{k-1}, n_k), (n_k, T) such that the nodes S, n_1, n_2, …, n_{k-1}, n_k, T are distinct. A graph $G(V, E)$ is called a path if all of its edges form such a sequence.

A *simple pipeline* is a fixed, directed-path and acyclic pipeline, whose route each order follows is always the same. There exists a unique path for each pair of any source node and any destination node in a simple pipeline.

An *input sequence* is sequence of all batches of products sent into the pipeline with the information on sources, destinations, product types.

A *delivery sequence* is defined on the actual delivery information of products at destinations.

An *input schedule* is an input sequence with the due delivery time-windows information. A *delivery schedule* is a delivery sequence with the actual delivery time-windows information.

Since we assume that the pipeline is a path and each batch has its fixed route to travel from its corresponding source to its corresponding destination, then it is claimed that there must be a 1:1 mapping of input schedules to delivery schedules. Namely, for a pipeline problem **P** with a set of orders **O** entering a simple pipeline, given that the pipeline is empty at the beginning period, then each input schedule generates a unique delivery schedule.

2.2 Feasible Input Schedules

How can we determine a feasible input schedule for a pipeline problem **P** that generates its corresponding delivery schedule? If the set of orders **O** for **P** has the total product volume less than the volume of the whole pipeline, how the products that need delivery at the very end of the pipeline are pushed to that point. This can be problematic

because it is not feasible to find out what the delivery schedule and actual delivery time are.

This difficulty can be resolved by extending the set of orders **O** by either assume the existence of an infinite amount of some filler product or replicate the orders from the original set of orders **O**. It is sensible and practical to assume in the latter case as future demands tend to follow the current demand and the fluctuation of the demand in the future is negligible. Therefore, it is assumed in this study that the set of orders **O** will be serviced repeatedly again and again in the same manner until the delivery schedule has been determined.

Consider a situation where customers have steady demand for delivery requirement. Thus, the repetition of input schedule will substantiate this requirement. Customers would prefer the products be delivered at destinations at the same rate as the product flow rate along the pipeline. We call a schedule that ensures the product delivery at the same rate as the product flow rate a *balanced schedule*. Namely, a balanced schedule enforces the amount of products delivered to be equal to the amount of products ordered. This concept is important for building the pipeline model in this study. With the assumption of a balanced schedule, the delivery schedule in a simple pipeline will be a repetition of the unit volumes from the set of orders **O**. Hence, the delivery schedule will represent the schedule of deliveries in the future.

As described in section 2.1, an input schedule composes of product information, source-destination information and due delivery time-windows. Product of the same type traveling to the same destination must be grouped together in the input schedule.

Splitting batches of the same type that traveling to the same destination in the input schedule is not allowed. Due delivery time-window is indicated in a window of the earliest due delivery time and latest due delivery time representing for the group of batches of the same product type traveling to the same destination, i.e., if there are 10 batches of product A traveling to a destination node *n*, then there is a single due delivery time-window that represents these ten batches for a delivery at node *n*.

Input schedule of order set **O** can be reduced to a sequence form of unit volumes. We already assumed that each batch has a unit volume. Consider a simple pipeline problem with a single source, two delivery nodes. There are three units volume of product going to each delivery node. The feasible input schedule can be represented in a sequence form as [*111222, 111222,* ...] or [*222111, 222111,* ...]. Each number $i \in I$ in the sequence represent one batch of a product traveling to node *i*. As mentioned earlier that splitting orders is forbidden. The number of input batches for each cycle is equal to the total volume of the set of orders **O**. We assume for this study that the schedule is balanced and the product flow rate in the pipeline is one unit of volume per one unit of time. These assumptions play an important role in facilitating the construction of a pipeline model.

2.3 Principal Assumptions

The pipeline scheduling is generally a complex problem and contains many levels of details. To construct a pipeline scheduling model that realistically represents

the practice in the industry, it would make the model too complicated to be able to solve in a reasonable amount time. In order to construct the pipeline model that serves our purposes as indicated in Chapter I, the following assumptions are adopted:

(1) Flow in the pipeline is traveled at a constant speed. Namely, the pipeline problem is assumed to have a balanced schedule. The reasons behind this assumption are two-fold. First of all, it simplifies the complications of the oil pipeline structure and allows us to initially construct a multiple-product pipeline network structure that represents this problem for multiple time-periods. Additionally, the focus of this research is underlined in the area of optimization. If variable flow rates are allowed in our model, we would have to go deep and work into the hydraulic and fluid issues in the pipeline, which are obviously out of bound for the study scope of our area;

(2) All orders are known and available at the beginning of the scheduling process. Orders in the future are assumed not to fluctuate and follow the demand pattern of the current orders; therefore, the repetition of input schedules in the pipeline is justified in this case;

(3) The model is handled deterministically, i.e., stochastic arrivals of orders are disregarded;

(4) Flow in the pipeline network is pushed by pumps and momentum propagates immediately along paths, i.e. for a period of Δt a total quantity of products entering a network is equal to the total exiting quantity at delivery points, namely conservation of flow is observed;

(5) Orders are transported in form of batches and batch interfaces are impassable. Namely there is no product mixing at batch interfaces and merging of products at any node is forbidden;

(6) Some product types are required not to travel adjacent to one another due to their product natures;

(7) Orders in each order set **O** are arranged and modified to be in the form of equal batches. A batch has one unit of volume. Each unit volume of product travels at the rate of one unit distance per one unit time;

(8) Delivery before and/or after the time-window requirement at each destination is permitted but penalty will be enforced for such delivery; namely, soft delivery time-windows are assumed;

(9) Storage capacity at the destinations is assumed to be large enough to accommodate the delivery of all products destined to each delivery node;

(10) Pipeline is assumed to be empty at the beginning period. Once products start entering the pipeline, no gap between batches or interruption of input schedule is allowed. Namely, the pipeline remains completely full for the whole duration of the finite T-period, where T is assumed to be long enough to cover all the necessary input, product flow and delivery operations required for the study;

(11) Batches of same product type and destination node must be grouped together. Splitting batches of this kind in the input schedule is prohibited.

2.4 Multiple-Product Oil Pipeline Problem

Consider a simple pipeline problem with a single source, *S*, and three destination nodes, *1*, *3*, and *4*, as shown in Figure 2.1. Delivery of products is permitted at these three destination nodes. The distances of *(S, 1)*, *(1, 3)*, and *(3, 4)* are equal to one, two, and one unit(s), respectively.

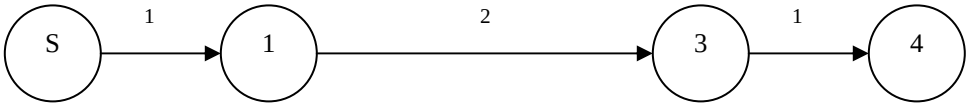


Figure 2.1 Original Pipeline

In order to simplify the formulation of the pipeline network model, the pipeline in Figure 2.1 will be modified to match up with the assumption that we have adopted. Namely, the distance between any pair of nodes in the pipeline will be transformed to be of equal length. Specifically, the length of every pair of nodes is set equal to one unit distance. A unit distance between every edge connecting nodes in the pipeline will synchronize with the flow rate of products in the pipeline assuming that one unit volume of product (one batch) travels at a speed of one unit distance per one time unit. The modified pipeline is constructed and illustrated in Figure 2.2.

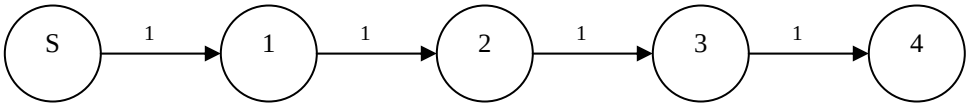


Figure 2.2 Modified Pipeline

It is seen in Figure 2.1 that an edge having the length greater than one unit is the one connecting node 1 and node 3, which has the length of two units. In order to make every edge have the same unit length in distance, the pipeline network structure needs modification. An additional node, node *2*, is inserted between node 1 and node 3 in Figure 2.2 and the unit-length edges connecting between node 1 and node 2 and between node 2 and node 3 are added to this pipeline network. Node 2 is called an *intermediate* node. Batches are allowed to travel pass through this intermediate node; however, delivery or entry of any batches is not allowed at this intermediate node.

2.4.1 Network Representation

It is assumed in this dissertation that we consider a finite T–period pipeline, where $t = 0, 1, \ldots, T$, and the pipeline is empty at the beginning period, $t = 0$. The total

time horizon, *T*, is assumed to be long enough to cover all activities in the pipeline operation, at least until all the products of the first input cycle are completely delivered.

The construction of a multiple time-periods multicommodity network flow is based on the suggestion from the study of Hane (1991) and it will be described as the following.

The network representation of a multiple time-periods multiple-product oil pipeline problem is based on the pipeline structure similar to the one shown in Figure 2.2. Batches enter the pipeline from a source node and continuously travel downstream to their corresponding destination nodes. Each batch moves from one node to another at a rate of one unit distance per one unit time. When a delivery of any batch *i* is performed at any destination node *n*, all batches downstream from this node *n* will be stationary for one time unit and concurrently all upstream batches continue moving and pushing this batch *i* out of the pipe at node *n*.

Let $N_t^D \in I$ be a set of all destination nodes including a source node in period *t*, S_t, and $N_t^+ \in I$ be a set of additional intermediate nodes used to construct the modified pipeline in period *t*. Define S^P as a set of pseudo-source nodes, where the number of pseudo-source nodes is equal to the number of product types transported in the pipeline and also define D^P as a set of pseudo-sink nodes, where the number of pseudo-sink nodes is equal to the number of destination nodes. Therefore, in each time period *t* there is the union of N_t^D and N_t^+ nodes in the network. Since the algorithm allows the

opportunity for flow to change in each time period, the node sets of N_t^D and N_t^+ need to appear for each time period.

A network $G = (V, E)$ is a multiple time-periods multicommodity network flow model where a set of vertex is the union of S^P, D^P, N_t^D, and N_t^+, $t = 0, 1, \ldots, T$, and E is a set of edges that controls how products are allowed to enter, travel through and exit the pipeline network. Each pseudo-source node s_k^P, $k = 1, 2, \ldots, K$, where K is the number of product types delivered in pipeline, is connect to a source node S_t in each time period. Each node in a set of N_t^D excluding S_t, $t = 0, 1, \ldots, T$, for each time period is connected to its correspond pseudo-sink node d_n^P, $n = 1, 2, \ldots, N$, where N is the number of destination nodes in the pipeline. The edges connecting each pseudo-source node and the source node in each time period is assumed to have the length of zero. The same assumption is applied to edges connecting each destination node in each time period and its corresponding pseudo-sink node.

In each time period, the batches occupying an edge of E can either move to any of its downstream neighbors or remain stationary. For convenience of further reference in this study, let's define N be the union of N_t^D and N_t^+. An example of a multiple time-periods multicommodity network with $K = 2$ products and $N = 3$ destination nodes for $T = 3$ periods is exhibited in Figure 2.3. The construction of this model is based on the modified pipeline shown in Figure 2.2. As mentioned earlier, the model is constructed based on the assumption that flow rate is constant throughout the pipeline.

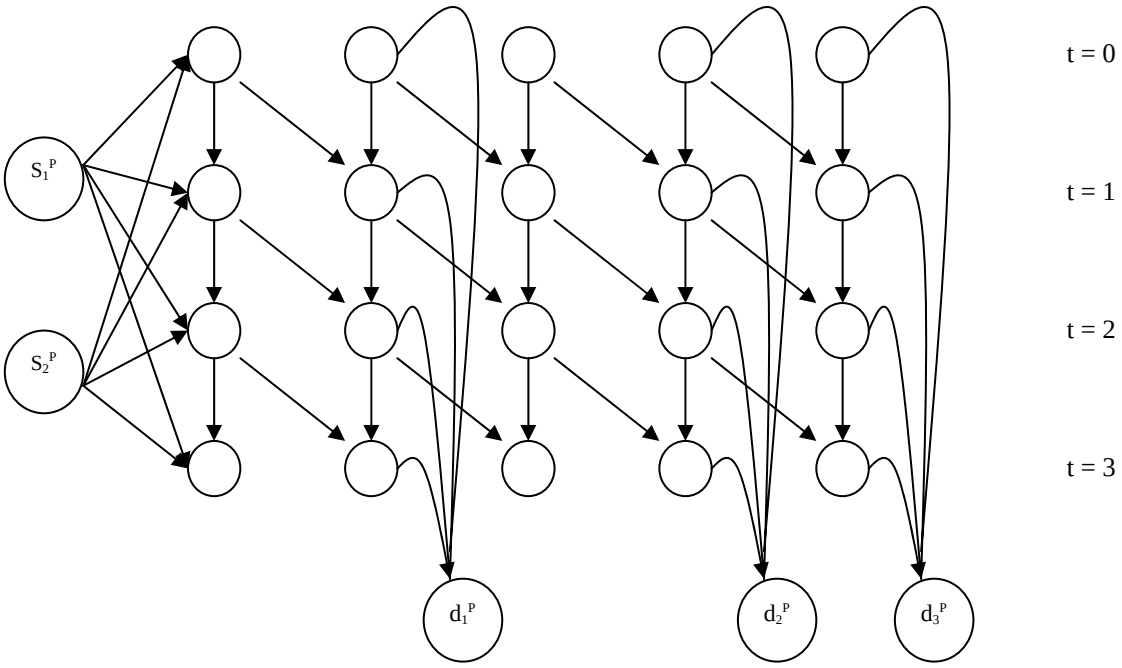


Figure 2.3 Multiple Time-Periods Multicommodity Network Structure for a Single-Source Oil Pipeline Problem

2.4.2 Pipeline Flow Representation

The following example demonstrates the movement of flow and the delivery of batches in a pipeline. Consider a simple pipeline problem with a single source node, *S,* and five destinations, node 1 to node 5. The edge volume of every pair of nodes is equal to one unit and we called this type of pipeline a unitized pipeline. Each destination node demands the delivery of one batch. Therefore, there are a total of five batches of flow in each cycle. Let [b_5 b_4 b_3 b_2 b_1] be an input order set, where b_i is a batch whose destination is node *i*. In this input sequence, b_5 enters the pipeline first. The set of batch may be simplified to be in the form of [*54321*]. The illustration of a pipeline and the representation of product flow simulation are shown in Figure 2.4.

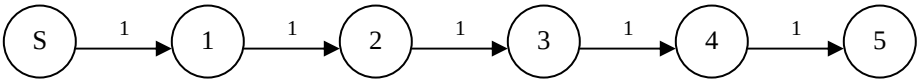
In this example, there are four repetitions of input schedule. The first batch to enter the pipeline is b_5. Then, at time t = 1, this batch b_5 occupies the entire volume of the edge connecting node S and node 1. The next batch to enter the pipeline at node S is b_4. At the time immediately after t = 1, b_4 enters the pipeline and pushes b_5 forward as time elapses. When the time is exactly at t = 2, b_4 occupies the full volume of the edge (*S*, 1) and b_5 moves one unit forward occupying the full volume of the edge (1, 2). The following batches keep pushing the preceding ones forward downstream in the pipeline. Batches continue moving along the pipeline in this manner.

The numerical representation of the flow simulation showing the flow movement along the pipeline as time elapses in Figure 2.4 is called a *flow matrix*. This representation shall be referred throughout this study. The first delivery occurs at node

1. Each delivery is indicated by an up-arrow sign (↑). The delivery of a batch takes one time unit to complete. The delivery of b_1 begins instantaneously right after t = 5 and completes precisely at time t = 6. Namely, the delivery time of a batch is equal to one time unit. At t = 6, batch b_5 is sent from the source node S into the pipeline and occupies the volume of edge (S, 1) as b_1 is pushed out off the pipeline simultaneously. When delivery is conducted at this period, downstream batches from node 1 remain stationary, i.e. from the time immediately right after t = 5 to t = 6 where b_1 is being delivered, downstream batches, b_2, b_3, b_4 and b_5, have no movement.

Since there is the repetition of input schedule in the pipeline, we will consider only the movement of the first cycle. The delivery sequence is obtained after the last batch of the first cycle is delivered and the actual delivery time is recorded.

The actual delivery schedule of products from the input schedule [*54321*] sent into the pipeline shown in Figure 2.4 is as follows: b_1 is delivered at node 1 at t = 6; b_2 is delivered at node 2 at t = 7; b_3 is delivered at node 3 at t = 8; b_4 is delivered at node 4 at t = 9; and b_5 is delivered at node 5 at t = 10. The delivery schedule of the succeeding input schedule cycles follows the pattern of the one from the first cycle.



	S	**1**	**2**	**3**	**4**	**5**
	●	●	●	●	●	
t = 1	5					
t = 2	4	5				
t = 3	3	4	5			
t = 4	2	3	4	5		
t = 5	1	2	3	4	5	
t = 6	5 ↑	2	3	4	5	
t = 7	4	5 ↑	3	4	5	
t = 8	3	4	5 ↑	4	5	
t = 9	2	3	4	5 ↑	5	
t = 10	1	2	3	4	5 ↑	
t = 11	5 ↑	2	3	4	5	
t = 12	4	5 ↑	3	4	5	
t = 13	3	4	5 ↑	4	5	
t = 14	2	3	4	5 ↑	5	
t = 15	1	2	3	4	5 ↑	
t = 16	5 ↑	2	3	4	5	
t = 17	4	5 ↑	3	4	5	
t = 18	3	4	5 ↑	4	5	
t = 19	2	3	4	5 ↑	5	
t = 20	1	2	3	4	5 ↑	

Figure 2.4 Unitized Pipeline and Product Flow Matrix

2.4.3 Mathematical Formulation

This mathematical representation of the multicommodity network shown in Figure 2.3 will be put together in this section. The formulation is based on the generalization of the minimum cost multicommodity network flow formulation with an inclusion of delivery time-window restrictions. Since cost is not the focus in the dissertation, the objective function is modified to correspond to the delivery times [Ahuja et al. (1993), Bazaraa et al. (1990), Chen et al. (2001), Evans (1978), Hartmann (1999), and Phillips and Garcia-Diaz (1990)].

Let f_{ijt}^k be binary variables where *(i, j)* $\in E$, $t = 0, 1, \ldots, T$ and $k = 1, \ldots, K$. f_{ijt}^k is set equal to 1 if there is a single unit of product *k*, travels on edge *(i, j)* at time *t*. Otherwise, f_{ijt}^k is set equal to 0. Define E_j^k as the time that a batch of product *k* is delivered at node *j* before its earliest due time, β_j^k, and also define L_j^k as the time that a batch of product *k* is delivered at node *j* after its latest due time, α_j^k. The goal is to minimize the total time deviation from due delivery time-windows.

Minimize $$\sum_j \sum_k \left\{ E_j^k + L_j^k \right\} \quad (1)$$

Subject to

$$\sum_j \sum_t f_{ijt}^k = A_i^k, \quad \forall k \in K, \forall i \in S \quad (2)$$

$$\sum_i \sum_t f_{ijt}^k = B_j^k, \quad \forall k \in K, \forall j \in D \quad (3)$$

$$\sum_j f_{ijt}^k - \sum_j f_{ji(t-1)}^k = 0, \quad \forall i \in N, \forall k \le K, \forall t \le T-1 \tag{4}$$

$$\sum_j f_{ijT}^k - \sum_j f_{ji(T-1)}^k \le 0, \quad \forall i \in N, \forall k \le K \tag{5}$$

$$\sum_k f_{ijt}^k \le 1, \quad \forall i, j \in N \tag{6}$$

$$E_j^k \ge \beta_j^k - t_s^k \cdot f_{ijt}^k, \quad \forall i \in N, \forall j \in N_t^D, \forall k \le K, \forall t \le T \tag{7}$$

$$L_i^k \ge t_f^k \cdot f_{ijt}^k - \alpha_j^k, \quad \forall i \in N, \forall j \in N_t^D, \forall k \le K, \forall t \le T \tag{8}$$

$$Z_t^k = \sum_{l=0}^{t} \sum_i f_{ijl}^k, \quad \forall k \le K, \forall j \in N_t^D \tag{9}$$

$$p_t^k = \frac{Z_t^k f_{ijt}^k (Z_t^k - 2)(Z_t^k - 3) \cdots (Z_t^k - B^k)}{(-1)(-2) \cdots (1 - B^k)}, \quad \forall k \le K, \forall j \in N_t^D \tag{10}$$

$$t_s^k = \sum_{t=0}^{T} t\, p_t^k \quad , \forall k \le K \tag{11}$$

$$q_t^k = \frac{(Z_t^k f_{ijt}^k)(Z_t^k - 1)(Z_t^k - 2) \cdots (Z_t^k - (B^k - 1))}{(B^k - 1)(B^k - 2) \cdots (B^k - (B^k - 1))}, \quad \forall k \le K, \forall j \in N_t^D \tag{12}$$

$$t_f^k = \{\sum_{t=0}^{T} t q_t^k\} + 1 \quad , \forall k \le K \tag{13}$$

$$p_t^k, q_t^k \in \{0, 1\}, \quad \forall k \le K \tag{14}$$

$$E_j^k \ge 0, \quad \forall j \in N_t^D, \forall k \le K \tag{15}$$

$$L_j^k \ge 0, \quad \forall j \in N_t^D, \forall k \le K \tag{16}$$

$$f_{ijt}^k \in \{0, 1\}, \quad \forall i, j \in N, \forall k \le K, \forall t \le T \tag{17}$$

The objective function (1) indicates the goal of minimizing the total delivery time that exceeds due delivery time-windows requirements. Constraints (2) are set to

control all products supply out of source nodes and constraints (3) force the flows into destination nodes in order to meet the demand requirements. Constraints (4) enforce the conservation of flow; however equalities (4) do not impose in the last time period, i.e., period *T*. Realistically, flows in the pipeline do not stop at period *T* as we assume that pipeline must be full at all time. We consider the problem in the time span *T* that is long enough to cover the interest of our pipeline study. Besides, all nodes in period *T* are not destination nodes. If we impose (4) in the last time period, we would wrongly assume that all nodes *i*, $i \in N_t^D$, in period *T* require delivery at amounts specified by the model. Constraints (5) are formulated to prevent such problem. The upper bound constraints of flow on all edges in the network are formulated in (6). In addition, inequalities (6) are known as the *"bundle constraints"* for the reason that they tie together the commodities on each edge by imposing the total allowable flow of all commodities not to exceed the upper bound limit. The next pair of constraints, (7) and (8), enforces the delivery time-window requirements of product *k* at destination nodes. Constraints (9) check the number of product *k* delivered at destination node up to time *t*. Constraints (10) and (11) compute the start delivery time of product *k* and constraints (12) and (13) compute the finish delivery time of product *k*. Constraints (14) enforce variables p_t^k and q_t^k to be binary. The nonnegativity requirements of E_j^k and L_j^k are enforced in (15) and (16). The last constraints, (17), restrict all flows to have values either 0 or 1. Namely, flows are not allowed for splitting or joining in this model.

2.4.4 Delivery Time-Windows

It is a goal of this study to find the input schedule that yields the minimum time deviation from the due delivery time-windows. However, in practicality finding input schedules that meets the due delivery time-windows of every batch in the sequence may be hard and sometimes infeasible. As a result, we relax this condition by allowing products to be delivered either before or after the due delivery windows. This section examines how the product delivery patterns are and how to compute the time deviation from due delivery windows.

Let define a due delivery time-window at any destination node i as $[\beta_i, \alpha_i]$ and this time-window is represented by the box displayed in Figure 2.5. β_i is the earliest due time for delivery and α_i is the latest due time for delivery. Define t_{si} be the actual start time of batches delivery at node i and t_{fi} be the actual finish time of batches delivery at node i. The patterns of delivery are categorized into four major instances as shown in Figure 2.5. Each delivery pattern has its unique approach to calculate the total delivery time violation.

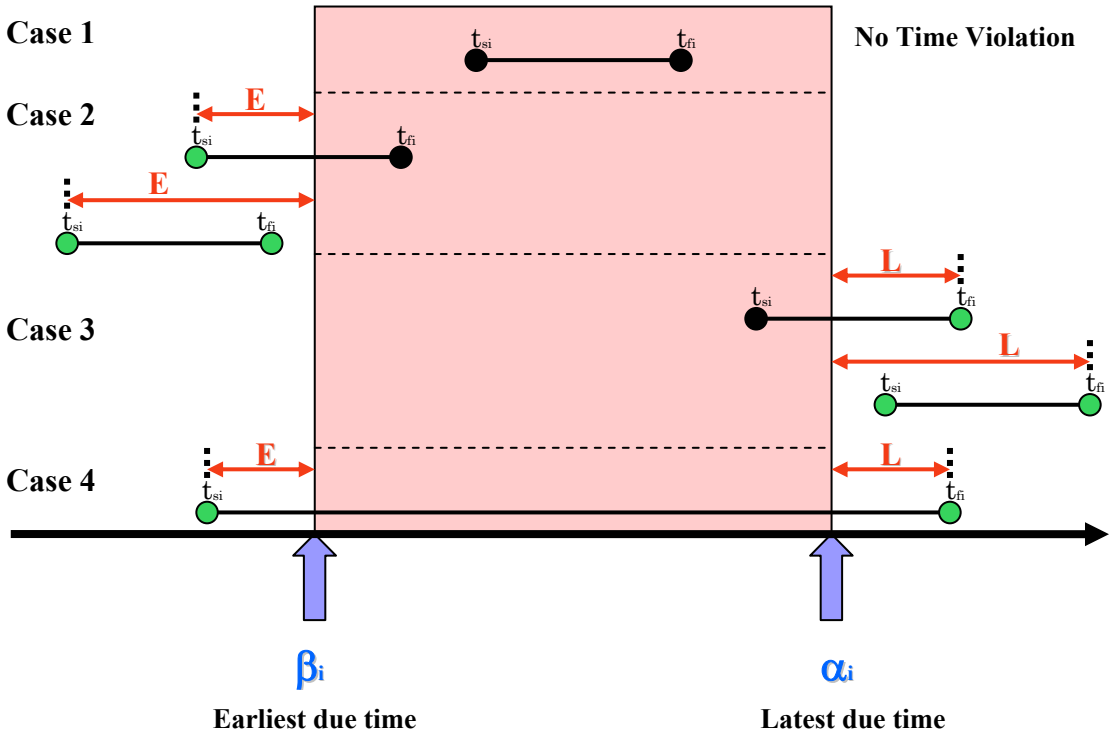


Figure 2.5 Types of Products Delivery

If the actual product delivery starts and finishes within the due delivery time-window, then there is no time violation in this case. Namely, if the actual delivery time stays inside the delivery time-window box shown in Figure 2.5, then no penalty is imposed. Thus, this is a scenario for case 1.

In case 2, the actual delivery starts prior to the earliest due time of the due delivery time-window, β_i. There are two possible situations under this circumstance. The first one is the situation where the actual delivery starts before β_i, i.e. $t_{si} < \beta_i$, and finishes at the time t_{fi} where $\beta_i < t_{fi} < \alpha_i$. The time violation is the part of delivery time that does not stay in delivery time-window box, which is the time difference between β_i and t_{si}. The other situation is the one where the entire actual delivery process starts and finishes entirely before β_i. Then, the total time violation is calculated by the time difference between β_i and t_{si}.

In case 3, the actual delivery finishes after the latest due time of the due delivery time-window, α_i. There are also two feasible situations under this circumstance. The first one is the situation where the actual delivery starts sometime between β_i and α_i, i.e. $\beta_i < t_{si} < \alpha_i$ and the delivery is completely finished after α_i, i.e., $t_{fi} > \alpha_i$. The time violation is the part of delivery time that does not remain in delivery time-window box, which is the time difference between t_{fi} and α_i. The other situation is the one where the whole actual delivery process is in fact conducted after α_i. The time violation for this situation is calculated from the time difference between t_{fi} and α_i.

The final case, case 4, is the circumstance where the actual delivery starts before β_i. The delivery is carried on and completed after α_i. The time violation is computed from the part of actual delivery time that does not remain in the due delivery time-window box. Namely the total time violation is in this case is the addition of ($\beta_i - t_{si}$) and ($t_{fi} - \alpha_i$).

CHAPTER III

SINGLE-SOURCE PIPELINES

In this chapter, we examine the problems of scheduling on a simple pipeline with a single source in order to find an input schedule that minimizes the total time violation from the due delivery time-windows. Recall that product mixing is not allowed in this study; therefore, all orders can be treated as different products. Demands are assumed not to be fungible; thus, each order has a fixed destination.

We begin this chapter with an example of a simple pipeline problem. This example will be referred throughout this chapter. The time-windows are not yet considered in this example and the first few sections of this chapter but they will be introduced in the later sections of this chapter. It is our intention to present this example first without the inclusion of delivery time-windows in order to generate some perception about the pipeline operation. Furthermore, this example will assist us in exploring some important attributes of the single-source pipeline problem. Later in the chapter, we will examine the complexity of a single-source oil pipeline distribution of multiple products subject to delivery time-windows. The reversed-flow algorithm developed based on an important attributes of the pipeline operation is used to determine the input schedule for the single-source pipeline problem will also be introduced later in this chapter.

3.1 Example 1

This section will work on an example of a simple pipeline problem with a single source and five destination nodes. The pipeline structure of this example is illustrated in Figure 3.1. These five destination nodes receive a total of 10 batches for each input cycle; namely, two batches are requested for a delivery at node 1, three batches at node 2, two batches at node 3, two batches at node 4, and two batches at node 5. The volumes of edges connecting nodes (S, 1), (1, 2), (2, 3), (3, 4) and (4, 5) are 3, 5, 3, 1 and 3 units, respectively. Let [*55 4 33 222 11*] be an input sequence for this example. Note that the time-windows are not taken into consideration yet in this section.

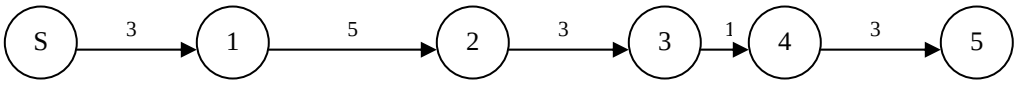


Figure 3.1 Pipeline Structure of Example 1

The delivery sequence for this input sequence is computed by a pipeline flow simulation. The pipeline flow simulation matrix for this example is shown in Figure 3.2. Each number *i* in the flow matrix represents a unit volume of a batch traveling to destination node *i*. A delivery at any destination node is indicated by an up-arrow sign (↑). A large space between batches in the sequence also indicates the position of a destination node.

S 1 2 3 4 5

t = 1	5				
t = 2	5 5				
t = 3	4 5 5				
t = 4	3 4 5	5			
t = 5	3 3 4	5 5			
t = 6	2 3 3	4 5 5			
t = 7	2 2 3	3 4 5 5			
t = 8	2 2 2	3 3 4 5 5			
t = 9	1 2 2	2 3 3 4 5	5		
t = 10	1 1 2	2 2 3 3 4	5 5		
t = 11	5 1 1	2 2 2 3 3	4 5 5		
t = 12	5 5 1↑	2 2 2 3 3	4 5 5		
t = 13	4 5 5↑	2 2 2 3 3	4 5 5		
t = 14	3 4 5	5 2 2 2 3	3 4 5	5	
t = 15	3 3 4	5 5 2 2 2	3 3 4	5	5
t = 16	2 3 3	4 5 5 2 2↑	3 3 4	5	5
t = 17	2 2 3	3 4 5 5 2↑	3 3 4	5	5
t = 18	2 2 2	3 3 4 5 5↑	3 3 4	5	5
t = 19	1 2 2	2 3 3 4 5	5 3 3	4	5 5
t = 20	1 1 2	2 2 3 3 4	5 5 3↑	4	5 5
t = 21	5 1 1	2 2 2 3 3	4 5 5↑	4	5 5
t = 22	5 5 1↑	2 2 2 3 3	4 5 5	4	5 5
t = 23	4 5 5↑	2 2 2 3 3	4 5 5	4	5 5
t = 24	3 4 5	5 2 2 2 3	3 4 5	5↑	5 5
t = 25	3 3 4	5 5 2 2 2	3 3 4	5	5 5 5
t = 26	2 3 3	4 5 5 2 2↑	3 3 4	5	5 5 5
t = 27	2 2 3	3 4 5 5 2↑	3 3 4	5	5 5 5
t = 28	2 2 2	3 3 4 5 5↑	3 3 4	5	5 5 5
t = 29	1 2 2	2 3 3 4 5	5 3 3	4	5 5 5↑
t = 30	1 1 2	2 2 3 3 4	5 5 3↑	4	5 5 5
t = 31	5 1 1	2 2 2 3 3	4 5 5↑	4	5 5 5
t = 32	5 5 1↑	2 2 2 3 3	4 5 5	4	5 5 5
t = 33	4 5 5↑	2 2 2 3 3	4 5 5	4	5 5 5
t = 34	3 4 5	5 2 2 2 3	3 4 5	5↑	5 5 5
t = 35	3 3 4	5 5 2 2 2	3 3 4	5	5 5 5↑
t = 36	2 3 3	4 5 5 2 2↑	3 3 4	5	5 5 5
t = 37	2 2 3	3 4 5 5 2↑	3 3 4	5	5 5 5
t = 38	2 2 2	3 3 4 5 5↑	3 3 4	5	5 5 5
t = 39	1 2 2	2 3 3 4 5	5 3 3	4	5 5 5↑
t = 40	1 1 2	2 2 3 3 4	5 5 3↑	4	5 5 5
t = 41	5 1 1	2 2 2 3 3	4 5 5↑	4	5 5 5
t = 42	5 5 1↑	2 2 2 3 3	4 5 5	4	5 5 5
t = 43	4 5 5↑	2 2 2 3 3	4 5 5	4	5 5 5
t = 44	3 4 5	5 2 2 2 3	3 4 5	5↑	5 5 5
t = 45	3 3 4	5 5 2 2 2	3 3 4	5	5 5 5↑

Figure 3.2 Pipeline Simulation Flow Matrix

In order to obtain the delivery sequence for this problem, the input sequence is repeated at least four times. Thus, the delivery sequence of the first input cycle is [*11 222 33 4 55*].

3.2 Pipeline Decomposition

Decomposition is a fundamental concept behind the formation of the delivery sequence from the known input sequence. The following statement develops the idea for decomposition of single-source pipelines.

"A simple pipeline problem **P** with a single source node and *n* destination nodes can be broken down into a set of *n* simple pipeline problems $\mathbf{P_i}$, $i = 1, \ldots, n$, where each $\mathbf{P_i}$ has a single source node and merely a single destination node."

The pipeline structure in Figure 3.1 represents the pipeline problem of an example 1 in section 3.1. Consider the first edge of **P** connecting (*S*, 1), called edge a_1. Each and every batch from the input sequence must pass through this edge in order to arrive at node 1. At node 1, every batch may either be delivered or pass through this node.

The pipeline problem $\mathbf{P_i}$ has node *i-1* as its source, define this source node as s_{i-1}, node *i* as its destination and a_i as the edge connecting these two nodes. For example, problem $\mathbf{P_1}$ has node *S* as its source and node 1 as its destination. The input sequence of $\mathbf{P_1}$ at node *S* is the same as delivery sequence at node 1. The flow surpassing node 1 behaves like it is a delivery at node 1 and an actual delivery at node 1 would look like

the flow is stationary in edge a_2. If the flow in edge a_2 is stationary, then flow in edges downstream of edge a_2 must also be stationary.

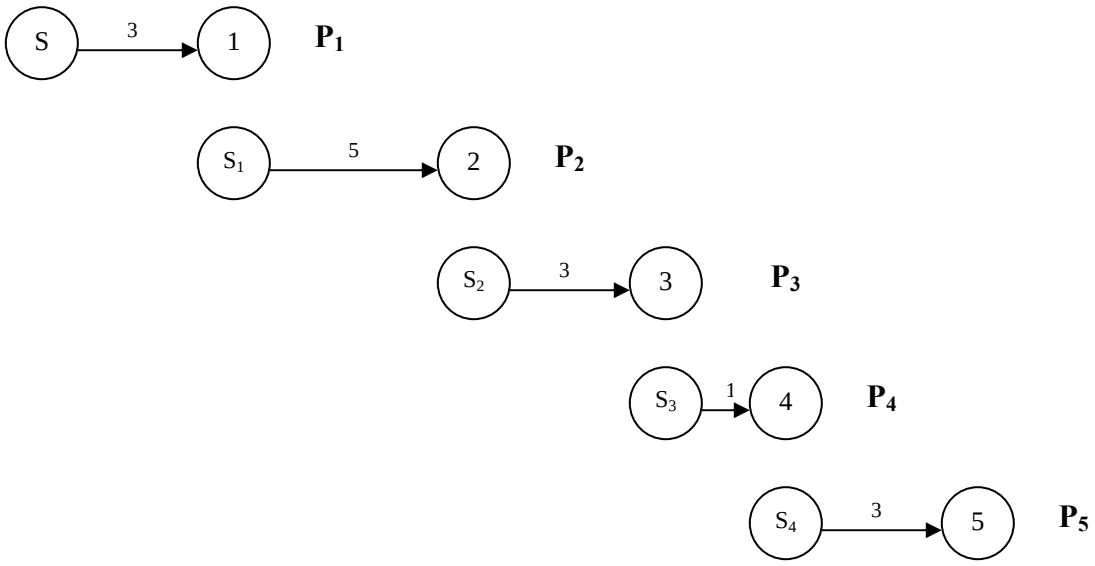


Figure 3.3 Pipeline Decomposition

	S		1		2		3		4		5
t = 1		5									
t = 2		5 5									
t = 3		4 5 5									
t = 4		3 4 5	**5**	5							
t = 5		3 3 4	**5**	5 5							
t = 6		2 3 3	**4**	4 5 5							
t = 7		2 2 3	**3**	3 4 5 5							
t = 8		2 2 2	**3**	3 3 4 5 5							
t = 9		1 2 2	**2**	2 3 3 4 5	**5**	5					
t = 10		1 1 2	**2**	2 2 3 3 4	**5**	5 5					
t = 11		5 1 1	**2**	2 2 2 3 3	**4**	4 5 5					
t = 12		5 5 1 ↑		2 2 2 3 3		4 5 5					
t = 13		4 5 5 ↑		2 2 2 3 3		4 5 5					
t = 14		3 4 5	**5**	5 2 2 2 3	**3**	3 4 5	**5**	5			
t = 15		3 3 4	**5**	5 5 2 2 2	**3**	3 3 4	**5**	5	**5**	5	
t = 16		2 3 3	**4**	4 5 5 2 2 ↑		3 3 4		5		5	
t = 17		2 2 3	**3**	3 4 5 5 2 ↑		3 3 4		5		5	
t = 18		2 2 2	**3**	3 3 4 5 5 ↑		3 3 4		5		5	
t = 19		1 2 2	**2**	2 3 3 4 5	**5**	5 3 3	**4**	4	**5**	5 5	
t = 20		1 1 2	**2**	2 2 3 3 4	**5**	5 5 3 ↑		4		5 5	
t = 21		5 1 1	**2**	2 2 2 3 3	**4**	4 5 5 ↑		4		5 5	
t = 22		5 5 1 ↑		2 2 2 3 3		4 5 5		4		5 5	
t = 23		4 5 5 ↑		2 2 2 3 3		4 5 5		4		5 5	
t = 24		3 4 5	**5**	5 2 2 2 3	**3**	3 4 5	**5**	5 ↑		5 5	
t = 25		3 3 4	**5**	5 5 2 2 2	**3**	3 3 4	**5**	5	**5**	5 5 5	
t = 26		2 3 3	**4**	4 5 5 2 2 ↑		3 3 4		5		5 5 5	
t = 27		2 2 3	**3**	3 4 5 5 2 ↑		3 3 4		5		5 5 5	
t = 28		2 2 2	**3**	3 3 4 5 5 ↑		3 3 4		5		5 5 5	
t = 29		1 2 2	**2**	2 3 3 4 5	**5**	5 3 3	**4**	4	**5**	5 5 5 ↑	
t = 30		1 1 2	**2**	2 2 3 3 4	**5**	5 5 3 ↑		4		5 5 5	
t = 31		5 1 1	**2**	2 2 2 3 3	**4**	4 5 5 ↑		4		5 5 5	
t = 32		5 5 1 ↑		2 2 2 3 3		4 5 5		4		5 5 5	
t = 33		4 5 5 ↑		2 2 2 3 3		4 5 5		4		5 5 5	
t = 34		3 4 5	**5**	5 2 2 2 3	**3**	3 4 5	**5**	5 ↑		5 5 5	
t = 35		3 3 4	**5**	5 5 2 2 2	**3**	3 3 4	**5**	5	**5**	5 5 5 ↑	
t = 36		2 3 3	**4**	4 5 5 2 2 ↑		3 3 4		5		5 5 5	
t = 37		2 2 3	**3**	3 4 5 5 2 ↑		3 3 4		5		5 5 5	
t = 38		2 2 2	**3**	3 3 4 5 5 ↑		3 3 4		5		5 5 5	
t = 39		1 2 2	**2**	2 3 3 4 5	**5**	5 3 3	**4**	4	**5**	5 5 5 ↑	
t = 40		1 1 2	**2**	2 2 3 3 4	**5**	5 5 3 ↑		4		5 5 5	
t = 41		5 1 1	**2**	2 2 2 3 3	**4**	4 5 5 ↑		4		5 5 5	
t = 42		5 5 1 ↑		2 2 2 3 3		4 5 5		4		5 5 5	
t = 43		4 5 5 ↑		2 2 2 3 3		4 5 5		4		5 5 5	
t = 44		3 4 5	**5**	5 2 2 2 3	**3**	3 4 5	**5**	5 ↑		5 5 5	
t = 45		3 3 4	**5**	5 5 2 2 2	**3**	3 3 4	**5**	5	**5**	5 5 5 ↑	

Figure 3.4 Pipeline Flow Simulation for Decomposition

The input sequence for $\mathbf{P_2}$ will be the modification of the delivery sequence of $\mathbf{P_1}$. Idle time will be replaced in the sequence on the position where deliveries corresponding to the original input sequence are made at node 1. The start time for $\mathbf{P_2}$ will be the total volume units of the upstream edge, a_1, plus the total stationary time period(s) in $\mathbf{P_1}$. The rest of the nodes in the pipeline will follow this pattern of structure. Figure 3.3 illustrates the pipeline decomposition structure of an example 1.

The flow simulation matrix for this pipeline decomposition is displayed in Figure 3.4. The columns of numbers printed in bold underneath the position of each destination node *i* represent the delivery sequence for $\mathbf{P_i}$ and the modification of this delivery sequence is the input sequence for the following pipeline problem $\mathbf{P_{i+1}}$. For instance, the delivery sequence for $\mathbf{P_1}$ is the same as its input sequence, which is [*55 4 33 222 11*]. By removing batches delivered at node 1 from this sequence and replacing them with two stationary time periods for downstream flow, then the modified sequence, [*55 4 33 222 ØØ*], is an input sequence for $\mathbf{P_2}$. Note that *Ø* in the sequence represents a stationary period. Since there is no stationary period in $\mathbf{P_1}$ from the time the first batch is pushed into a_1 to the time of the first delivery, then the start time for $\mathbf{P_2}$ is equal to the total volume units of edge a_1. For $\mathbf{P_2}$, the first batch enters node 1 and occupies the first volume unit of a_2 at time $t = 4$. At $t = 12$ and $t = 13$, flow in a_2 remains stationary due to an upstream delivery at node 1. As mentioned earlier that once flow is stationary at any edge at any time period, the rest of the flow in all downstream edges is also stationary at that time period. The delivery sequence for $\mathbf{P_2}$ is [*55 4 ØØ 33 222*]. Delivery starts right after $t = 8$ and finishes at $t = 18$ for $\mathbf{P_2}$. Note that flow in a_2 is stationary in period *t*

= 12 and t = 13. Thus, the modified input for $\mathbf{P_3}$ is [*55 4 ØØ 33 ØØØ*]. The problem is carried on in the same approach until $\mathbf{P_5}$ is obtained.

3.3 Forward-Filling the Pipeline

The data given based on the input sequence can be used to determine the contents in the pipeline. Specifically, we can determine the contents in the pipeline at the period when the line is full for the first time, given that the pipeline is empty at an initial period. We restrict our attention merely to simple pipeline problems.

Products enter a pipeline in form of batches, where each batch travels from source node to destination node, v_i. Some characteristics of products moving along the pipeline can be described as the following. Batches that are allowed to traverse on edge a_i are those whose source-destination route intersects edge a_i. It can be observed from sequencing a simple pipeline that it is easier to start at the most downstream portion of the pipeline because the fewest batches are permitted at the end of the line. Consider a case with a batch b_n going to the last destination node v_n of the pipeline and an incoming edge connecting nodes v_{n-1} and v_n has the length greater than one unit. When this batch b_n occupies the last unit portion in the pipe right before node, the upstream batch immediately adjacent to b_n must be a batch whose destination is the same as batch b_n; namely the adjacent batch must have node v_n as its destination.

If we have a set of orders **O** composing of three batches, b_1 b_2 and b_3, being pushed into the pipeline where b_1 enters first, follows by b_2 and then b_3, and these three

batches travel on the same edge set, there is only one possible way that b_1 can be neighboring with b_3 in the pipeline, which is after b_2 is delivered at a destination node prior to destination nodes of b_1 and b_3.

If we arrange an input sequence in such a way that batches going to the most downstream node enter the pipeline first, then this method is called "*forward-filling*" or "*backfilling*". This method determines the pipeline contents and reduces the transit time in the pipeline of batches in each input cycle. Forward-filling is an simple approach to construct a pipeline operation schedule given an input batches information.

It can be seen in Figure 3.2 that the line is completely filled with products for the first time at $t = 25$. At this period, the line is filled with batches entirely from the input sequence. Instead of simulating product flow for 25 periods, we can use forward-filling technique to determine the pipeline contents when the line is filled for the first time.

By using forward-filling technique, we start filling from the very end of the pipeline and move backward toward the source node. Thus, we start at edge (*4, 5*). Only b_5 can occupy this edge. We have two batches of b_5 in each order cycle and the volume of this edge is three, thus two orders going to node 5 (four batches of b_5) are filled here. Now edges (*3, 4*) and (*4, 5*) are filled and we search for the next input that may travel upstream of node 3.

From the input sequence, we know that b_2 pushes b_3, b_4 is pushed by b_3 and b_5 is pushed by b_4. Then, we continue filling the line with one order set of b_4 (one batch), one order set of b_3 (two batches) and one order set of b_2 (three batches). Up to this point, ten batches fill the pipeline and two units volume of edge (*1, 2*) is left unoccupied. From the

input sequence, b_1 pushes b_2 but the assumption would be violated if we place b_1 on edge (1, 2). Thus, the next possible batch to be placed in this position is an order set of b_5 (two batches). The pipeline is filled according to the input sequence from this point upstream toward the source node. Namely, at the steady state, t = 25, edge (S, 1) is filled with $b_3b_3b_4$, edge (1, 2) is filled with $b_5b_5b_2b_2b_2$, edge (2, 3) is filled with $b_3b_3b_4$, edge (3, 4) is filled with b_5 and edge (4, 5) is filled with $b_5b_5b_5$.

From the flow simulation matrix in Figure 3.2, without the consideration of time-windows, the first delivery made at the last node in the line is defined to be the beginning of the steady state of the pipeline flow. Namely, the steady state starts at t = 29. The steady state is defined as the state where the delivery sequence repeats itself. We can obtain the delivery sequence starting from the first delivery made at node 5 at t = 29 for each input cycle. As the input sequence sent into the pipeline repeats itself, the delivery sequence obtained from the flow simulation also repeats itself. Thus, the input sequence for this example is [*55 4 33 222 11*] and the delivery sequence obtained from Figure 3.2 is [*3 11 4 5 222 5 3*].

3.4 Pipeline Flow Reversibility

As we mentioned earlier in Chapter II and it is seen from an example 1 in section 3.1 that a path pipeline yields a unique delivery sequence for each input sequence given that the pipeline is empty at the beginning. Consequently, we can use the delivery sequence to generate the input sequence by constructing a pipeline and let product flow

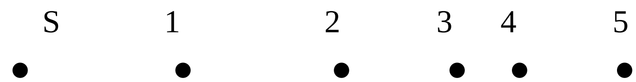
travels in a reverse direction. Namely, in the reversed-flow pipeline problem $\mathbf{P^R}$ the function of each node in the pipe will be changed. The destination nodes are switched to perform as source nodes, a source node is switched to perform as a destination node and the products will move in an opposite direction from the direction of a regular-flow pipeline problem **P**. In $\mathbf{P^R}$, flow travels from multiple sources to a single destination. Therefore, if the problem **P** is a single-source pipeline, then its reversal problem $\mathbf{P^R}$ will be a single-destination pipeline.

Reversed-flow pipeline problem is feasible under the condition that the time horizon must be finite. Let define *T* be the finish time for **P**. The reverse-flow pipeline problem starts with the pipeline contents at time *T* and flow continues to simulate reversely until time 0. Thus, the delivery sequence of **P** from time *T* to 0 is the input sequence for $\mathbf{P^R}$. The pipeline contents of $\mathbf{P^R}$ are simply identical to those of **P** at the same time period. The delivery sequence of $\mathbf{P^R}$ at node *S* is the input sequence for **P**.

The reversed-flow pipeline problem is beneficial in case that the delivery sequence carries more significant influence on the outcome of the pipeline study. In our study we emphasize more on the producing delivery schedules that minimize the total deviation from time-windows; therefore, reversed-flow concept will play a principal role throughout this study.

Consider the pipeline flow matrix in Figure 3.2 assuming *T* equal to 40 and it is known that the delivery sequence is [*3 11 4 5 222 5 3*]. The input sequence for $\mathbf{P^R}$ is obtained by reversing this sequence. The first batch to enter is b_3 follows by b_5, $b_2b_2b_2$, … , b_3. When each batch is inserted into the pipeline at a source node *i*, flow moves

leftward toward node *S*, which performs as a destination node in $\mathbf{P^R}$, and batches on the right of node *i* remain stationary.



t = 40		*1 1 2*	*2 2 3 3 4*	*5 5 3*	*4*	*5 5 5*
t = 39	**1**←	1 2 2	2 3 3 4 5	5 3 3 «	4	5 5 5
t = 38	**1**←	2 2 2	3 3 4 5 5	3 3 4	5	5 5 5 «
t = 37	**2**←	2 2 3	3 4 5 5 2 «	3 3 4	5	5 5 5
t = 36	**2**←	2 3 3	4 5 5 2 2 «	3 3 4	5	5 5 5
t = 35	**2**←	3 3 4	5 5 2 2 2 «	3 3 4	5	5 5 5
t = 34	**3**←	3 4 5	5 2 2 2 3	3 4 5	5	5 5 5 «
t = 33	**3**←	4 5 5	2 2 2 3 3	4 5 5	4 «	5 5 5
t = 32	**4**←	5 5 1 «	2 2 2 3 3	4 5 5	4	5 5 5
t = 31	**5**←	5 1 1 «	2 2 2 3 3	4 5 5	4	5 5 5
t = 30	**5**←	1 1 2	2 2 3 3 4	5 5 3 «	4	5 5 5
t = 29	**1**←	1 2 2	2 3 3 4 5	5 3 3 «	4	5 5 5
t = 28	**1**←	2 2 2	3 3 4 5 5	3 3 4	5	5 5 5 «
t = 27	**2**←	2 2 3	3 4 5 5 2 «	3 3 4	5	5 5 5
t = 26	**2**←	2 3 3	4 5 5 2 2 «	3 3 4	5	5 5 5
t = 25	**2**←	3 3 4	5 5 2 2 2 «	3 3 4	5	5 5 5
t = 24	**3**←	3 4 5	5 2 2 2 3	3 4 5	5	5 5 5 «
t = 23	**3**←	4 5 5	2 2 2 3 3	4 5 5	4 «	5 5 5
t = 22	**4**←	5 5 1 «	2 2 2 3 3	4 5 5	4	5 5 5
t = 21	**5**←	5 1 1 «	2 2 2 3 3	4 5 5	4	5 5 5
t = 20	**5**←	1 1 2	2 2 3 3 4	5 5 3 «	4	5 5 5
t = 19	**1**←	1 2 2	2 3 3 4 5	5 3 3 «	4	5 5 5
t = 18	**1**←	2 2 2	3 3 4 5 5	3 3 4	5	5 5 5 «
t = 17	**2**←	2 2 3	3 4 5 5 2 «	3 3 4	5	5 5 5
t = 16	**2**←	2 3 3	4 5 5 2 2 «	3 3 4	5	5 5 5
t = 15	**2**←	3 3 4	5 5 2 2 2 «	3 3 4	5	5 5 5
t = 14	**3**←	3 4 5	5 2 2 2 3	3 4 5	5	5 5 5 «
t = 13	**3**←	4 5 5	2 2 2 3 3	4 5 5	4 «	5 5 5
t = 12	**4**←	5 5 1 «	2 2 2 3 3	4 5 5	4	5 5 5
t = 11	**5**←	5 1 1 «	2 2 2 3 3	4 5 5	4	5 5 5
t = 10	**5**←	1 1 2	2 2 3 3 4	5 5 3 «	4	5 5 5

Figure 3.5 Reversed-Flow Simulation Matrix for $\mathbf{P^R}$

The delivery sequence at node *S* in $\mathbf{P^R}$ is actually the input sequence for **P**. The matrix of reversed-flow pipeline simulation of example in section 3.1 is shown in Figure 3.5.

We assume *T* equal to 40 and start simulating flow in pipeline $\mathbf{P^R}$. The pipeline contents are exactly the same as the contents in **P** at $t = 40$ as shown in the first row of the flow matrix in Figure 3.5. The entry of each batch from various sources of the input sequence in $\mathbf{P^R}$ is indicated by a symbol "«" located in a space in a flow sequence at the position underneath its corresponding source node. All batches are delivered only at node *S*. As shown in Figure 3.5 that the delivery sequence for $\mathbf{P^R}$ is [*11 222 33 4 55*], which is printed in bolded in the first column. By reversing the delivery sequence obtained from $\mathbf{P^R}$, then this new sequence is actually an input sequence for **P**. Hence, the input sequence for **P** is [*55 4 33 222 11*].

3.5 Pipeline Flow with Delivery Time-Windows

In the previous sections the delivery time-windows are not considered in order to study some fundamental attributes of a simple pipeline problem. We will now include delivery time-windows in the pipeline problem and use the attributes that we just examined to study the problem in the scope of our interests. Generally, the delivery time-window for batches of the same product type destined for the same destination node *i* grouping together is represented in the form of $[\beta_i, \alpha_i]$, where β_i is the earliest due time and α_i is the latest due time for delivery at node *i*.

Consider a pipeline problem in example 1, we assign the due delivery time-windows for batches traveling to each destination node *i* as the following: due delivery window of b_1b_1 at node 1 : [10, 14]; due delivery window of $b_2b_2b_2$ at node 2 : [15, 19]; due delivery window of b_3b_3 at node 3 : [20, 22]; due delivery window of b_4 at node 4 : [23, 25]; and due delivery window of b_5b_5 at node 5 : [28, 32].

As we mentioned earlier that input sequences enters the pipeline in repetitive manner. If we keep track of any batch placed in the i^{th} position of the input sequence entering, traversing and exit the pipeline and record the total traversing time, then the same batch placed in the i^{th} position of the following input cycles will spend the same amount of traversing time in the pipeline and exit at the same destination node in an equal amount of shifting flow time. When delivery time-windows are incorporated, we are interested primarily on the delivery time-windows that batches made at each destination node. Therefore, it is justifiable for us to examine simply just the movement of batches in the first input order cycle starting from the first batch entering the line until the last batch of this cycle exiting the line.

We collect the actual delivery time-windows of the first input cycle from Figure 3.2 as the following: b_1b_1 at node 1 at time [11, 13]; $b_2b_2b_2$ at node 2 at time [15, 18]; b_3b_3 at node 3 at time [19, 21]; b_4 at node 4 at time [23, 24]; b_5b_5 at node 5 at time [28, 35]. The calculation of time violation is described in section 2.3.4 and Figure 2.5. The actual delivery time-windows at node 3 and node 5 do not range within the due time-windows set for these two nodes, and the time violations at each node equal to 1 and 3, respectively. Hence, the total time violation for this input schedule is equal to 4.

There is one observation obtained from pipeline flow simulation of a single-source pipeline. The travel time of each batch in the pipe is greater than or equal to the distance from the source node to its corresponding destination node. Only batches, whose delivery is expected at the first destination node adjacent to the source node, take time to reach this destination node equal to the edge length of this node pair. Most batches takes more time than the actual source-destination edge length to travel to their corresponding destination node(s) due to the fact that the delivery at any destination node upstream in the pipeline causes the downstream flow to be stationary. Consequently, it is not logical to assign any due delivery time-window with β_i to start at a time that is fewer than the source-destination length of their corresponding batch(es) or the one that has α_i end at the time fewer than the length of corresponding source-destination nodes since it is not feasible to find an input schedule that fulfills such due delivery time-window.

3.6 Single-Source Oil Pipeline Distribution of Multiple Products Subject to Delivery Time-Windows Is NP-Complete

We will explain in this section that the single-source oil pipeline distribution of multiple products subject to delivery time-windows is NP-complete by way of a reduction from a satisfiability (SAT, for short) problem. This reduction not only provides us with a measure of complexity for the oil pipeline distribution of multiple

products subject to delivery time-windows, but also gives us an additional insight into the structure of the problem.

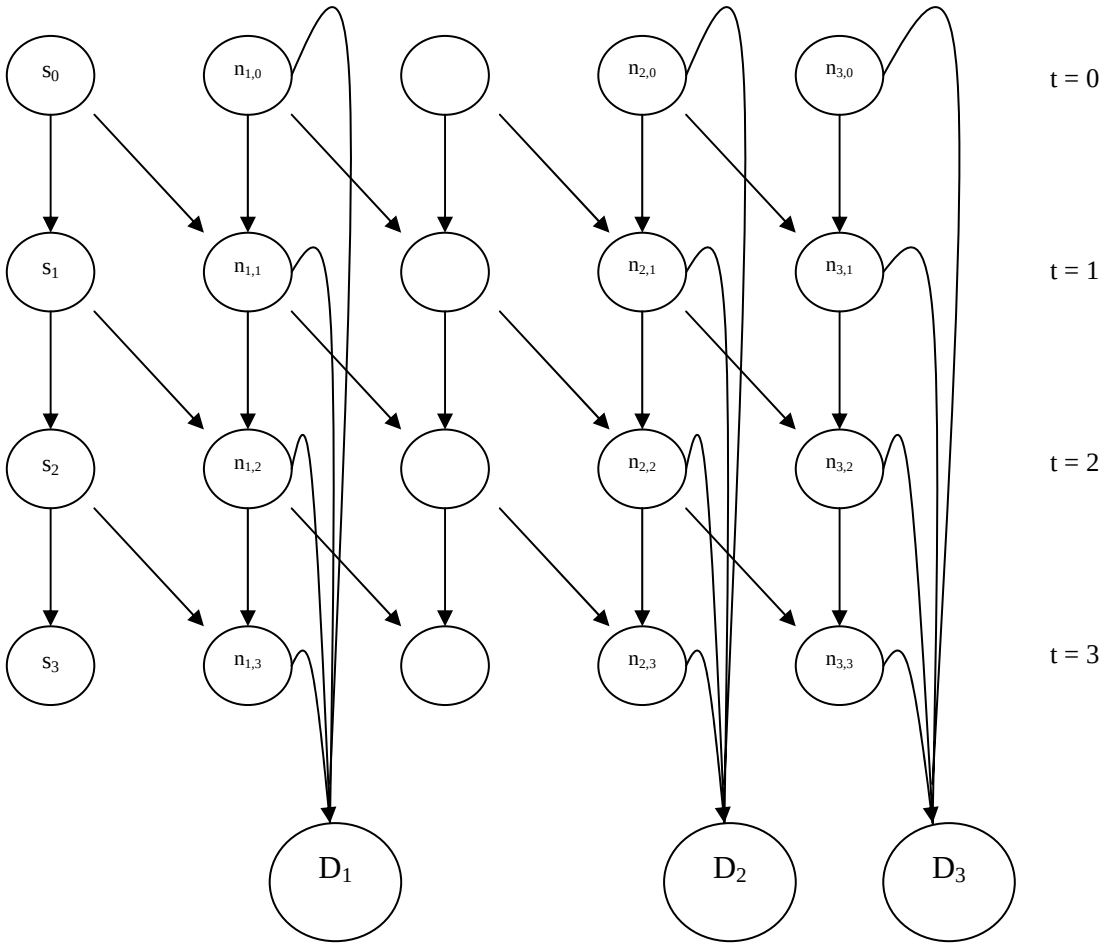


Figure 3.6 Network Structure Used as an Instance

We can express a single-source oil pipeline distribution of multiple products subject to delivery time-windows as a decision problem as the following. Note that the pipeline has a single source and the number of destinations in the pipeline is equal to $|Z|$, where $Z \in I^+$. The network representing the instance stated below is illustrated in Figure 3.6. The instance and question to proof NP-complete for a single-source oil pipeline distribution of multiple products subject to delivery time-windows is explained as the following.

INSTANCE: A directed graph **G** = *(V, E)* ; a set K of $|K|$ unit-batches of input products where each batch has its destination node z; a set $t \in T$ of discrete time points where $T \in I^+$ and $T > |K|$ is large enough to include the specific time periods of pipeline operation; in the multiple time-periods multicommodity network there are $|K|$ source nodes for each time period each batch k is pushed into the line, $s_t \in V$, and $|Z| \times T$ destination nodes, $n_{z,t} \in V$ for each node z and time period t, all edges emanating from nodes $n_{z,t}$ are connected to their corresponding pseudo-destination node $D_z \in E$; capacity $c(e) = 1$ for all edges $e \in E$, requirement $R_z \in I^+$ for each $z \in Z$ and the delivery of a batch k is expected to be within $[\beta_i, \alpha_i]$ where β_i and $\alpha_i \in I^+$.

QUESTION: Are there $|K|$ flow functions $f_1, f_2, \ldots, f_k : E \rightarrow I^+$ such that

(1) for each $e \in E$, $\Sigma_{k \in K} \; f_k(e) \leq c(e)$ (capacity constraints);

(2) for each $v \in V \setminus \{s_k, D_z\}$ where $k \in K$ and $z \in Z$, the conservation of flow is observed at each node v;

(3) the total flow into pseudo-destination node D_z under flow $\Sigma_{k \in K} f_k$, for each $z \in Z$, is at least R_z ; and

(4) for each batch $k \in K$, its delivery is conducted within a time-window of $[\beta_i, \alpha_i]$?

The satisfiability problem [Garey and Johnson (2003)] can be described as follows:

Let a set of Boolean variables $U = \{u_1, u_2, \ldots, u_k\}$ in which each variable can be either true or false. The literals of variable u over U are "u" and "$\bar{u}$". The clause is simply an "or" connection of a set of literals; i.e., $u_1 \cup \overline{u_3} \cup u_4$. A collection of clauses over U with the "and" connection is called *conjunctive normal form expression*; i.e., $\{u_1 \cup \overline{u_3} \cup u_4\} \cap \{u_2 \cup u_4\}$. Each variable is assigned value as either true *(T)* or false *(F)*. Assign the values of these variables as follows: $u_1 = T$, $u_2 = F$, $u_3 = T$, $u_4 = T$. The evaluation of the conjunctive normal form expression shown above is *T*. A conjunctive normal form expression is *satisfiable* if and only if there exists some truth assignment for U that simultaneously satisfies all the clauses. The satisfiability problem is to find out, given an arbitrary conjunctive normal form expression, if the expression is satisfiable.

SATISFIABLE (SAT)

INSTANCE: A set *U* of variables and a collection *C* of clauses over *U*.

QUESTION: Is there a satisfying truth assignment for *C* ?

Theorem 3.1. Single-source oil pipeline distribution of multiple products subject to delivery time-windows problem is NP-complete.

Proof. It suffices to show that satisfiability $\propto$ multicommodity flow problem and multicommodity flow problem $\propto$ single-source oil pipeline distribution of multiple products subject to delivery time-windows. A single-source oil pipeline distribution of multiple products subject to delivery time-windows is simply a restricted form of the multicommodity flow problem.

According to Karp (1975) and Even et al. (1976), satisfiability problem is reduced to a restricted version of multicommodity flow by creating a network in which, for each variable, there exists two possible paths from source and sink, one corresponding to the true value of the variable and the other corresponding to its negation. The network is constructed in such a way that flow can only take place through merely one of the two paths, but not both.

Consider *k* source-destination node pairs of *k* different batches $(s_0^{n_{i_1}}, n_{i_1,t}), (s_1^{n_{i_2}}, n_{i_2,t}), \ldots, (s_{k-1}^{n_{i_k}}, n_{i_k,t})$. This asks whether or not there is a set of *k* flow

paths. We can transform any instance of a satisfiability problem in polynomial time into an equivalent instance of a single-source oil pipeline distribution of multiple products subject to delivery time-windows. Let assume that the source-destination node pairs be a one-to one correspondence with clauses in the satisfiability problem; namely, a pair $(s_{j-1}^{n_{i_j}}, n_{i_j,t})$ will corresponds to a clause c_j.

We can construct a graph to represent each variable u. If literal u appears in clauses $c_1, c_2, \ldots, c_p$ and literal $\bar{u}$ appears in clauses $c_{p+1}, c_{p+2}, \ldots, c_q$, then the graph that represents a variable u is depicted in Figure 3.7. The graph for all variables will be obtained by joining together a number of graphs that correspond to each variable.

In Figure 3.7, it can be seen that it is possible to connect all $s_{c_l}'s$ to their matching $n_{c_l}'s$ by horizontal paths (for literal u) and by vertical paths (for literal $\bar{u}$). Every horizontal path has a node in common with every vertical path. The network of all variables are obtained by identifying, as a single node, all the occurrences of each s_{c_l} (or n_{c_l}) in all graphs.

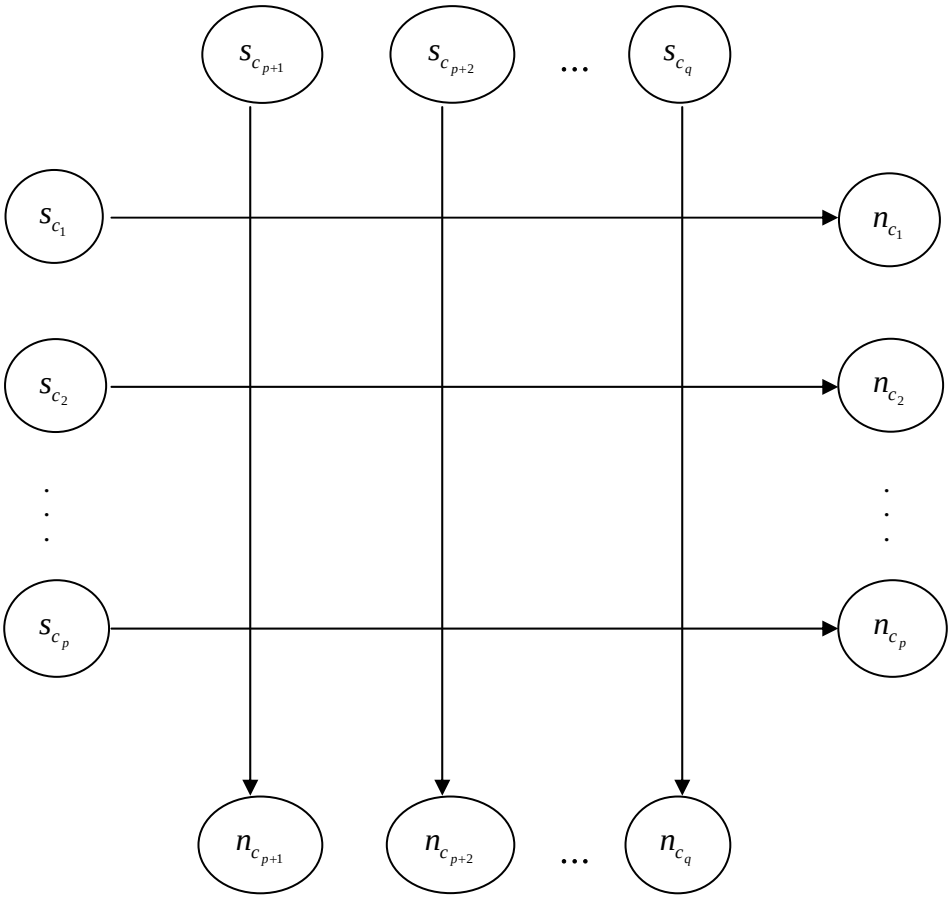


Figure 3.7 A Subgraph for Variable *u*

($\rightarrow$) If there exists a conjunctive normal form expression for a satisfiability problem and this conjunctive normal form expression is satisfiable, then the single-source oil pipeline distribution of multiple products subject to delivery time-windows is feasible. Given that each variable is assigned the truth value ("true" or "false"). The horizontal path in the subgraph for variable u is chosen if u is assigned a "true" value and the vertical path in the subgraph for variable u is chosen if u is assigned a "false" value. If either u or $\bar{u}$ appears in clause c_j and its value is "true", there is a path in the subgraph connecting s_{c_l} and n_{c_l} for variable u.

($\leftarrow$) Assume that the single-source oil pipeline distribution of multiple products subject to delivery time-windows is feasible and there exists a solution to our problem by having $|K|$ flow functions, which satisfy requirements, for $|K|$ batches entering the pipeline. Then each batch is delivered to its designated destination node and exactly one path is taken that corresponds to each clause of the instance of the satisfiability problem. Assign the truth value "true" for each variable u to a path traversed by a batch from source to sink (implicitly assigning "false" value to its negation). Arbitrarily assign "true" value for any unassigned variables and no variable is assigned both "true" and "false" values. As a result, each variable has a single value assigned and the satisfiability problem is satisfiable.

Thus, both requirements are met. **Q.E.D.**

Given the above result, there is no polynomial time optimal algorithm to solve the single-source oil pipeline distribution of multiple products subject to delivery time-

windows unless $P = NP$. However, there are ways to approach single-source oil pipeline distribution of multiple products subject to delivery time-windows if we are willing to relax our need for optimal solutions. We look to construct an approximation algorithm to solve the problem. Approximation algorithm gives up the notion of an optimal solution to an optimization problem but, in turn, guarantees that the quality of the solution of the resulting algorithm is within a factor of the optimal solution for any problem instance.

The approximation algorithm that we construct for a single-source pipeline problem is based on pipeline flow reversibility mentioned in section 3.4. With this technique and without the consideration of delivery time-windows, the optimal sequence can be obtained. However, with the inclusion of delivery time-windows, this method does not guarantee the optimal schedule and minimum total time violation.

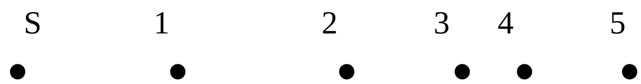
3.7 Reversed-Flow Algorithm for Single-Source Pipeline

This section introduces a constructive algorithm based on the pipeline flow reversibility technique used in order to obtain an input schedule for the single-source oil pipeline distribution of multiple products subject to delivery time-windows. Without delivery time-windows, pipeline flow reversibility technique can generate the input sequence that yields the delivery sequence to the need of the pipeline scheduling planner. However, with the inclusion of delivery time-windows, the problem is so complex that it is very hard to determine the delivery schedule that makes all batches

delivered within their corresponding time-windows. The problem may be infeasible to solve in a number of cases. Therefore, we simplify this complication by assuming that the delivery of batches is allowed at their destination nodes either before or after the delivery time-windows. Nevertheless, it is unattainable to determine a delivery schedule from only just an input schedule without performing a pipeline flow simulation.

The pipeline scheduling problem is complicated because of its unique pattern of flow movement in the pipe. The way each batch is sequenced for the input schedule before being pushed into the pipeline has a tremendous effect on the outcome of actual delivery time at destination nodes. The interchange of some pair(s) of batches in the input schedule may yield a huge different in total time deviation from due delivery time-windows.

The motivation behind the reversed-flow algorithm rests on the observation that obtaining input schedule from generating pipeline flow reversibility would bring a decent and reasonable outcome on the delivery schedule. Since it is not possible to determine the exact delivery time of each batch, we would use the information we have on their due delivery time-windows to establish an input schedule. Recall that batches traveling to destinations downstream that are further away from a source node need more travel time than those traveling to destinations closer to a source node. In general the delivery time-windows for any batches should not have their times overlapped; however, this incident is not forbidden.




```
t = 1                                                        5 «
t = 2                                                      5 5 «
t = 3                                              4 «     5 5
t = 4                                         3 «  4       5 5
t = 5                                       3 3 «  4       5 5
t = 6                            2 «        3 3    4       5 5
t = 7                          2 2 «        3 3    4       5 5
t = 8                        2 2 2 «        3 3    4       5 5
t = 9             1 «        2 2 2          3 3    4       5 5
t = 10          1 1 «        2 2 2          3 3    4       5 5
t = 11          1 1          2 2 2          3 3    4     5 5 5 «
t = 12          1 1          2 2 2        3 3 4    5     5 5 5 «
t = 13          1 1        2 2 2 3        3 4 5    4 «   5 5 5
t = 14          1 1      2 2 2 3 3        4 5 3 «  4     5 5 5
t = 15        1 1 2      2 2 3 3 4        5 3 3 «  4     5 5 5
t = 16   {1}←1 2 2       2 3 3 4 2 «      5 3 3    4     5 5 5
t = 17   {1}←2 2 2       3 3 4 2 2 «      5 3 3    4     5 5 5
t = 18   {2}←2 2 3       3 4 2 2 2 «      5 3 3    4     5 5 5
t = 19   {2}←2 3 1 «     3 4 2 2 2        5 3 3    4     5 5 5
t = 20   {2}←3 1 1 «     3 4 2 2 2        5 3 3    4     5 5 5
t = 21   {3}←1 1 3       4 2 2 2 5        3 3 4    5     5 5 5 «
t = 22     1 ←1 3 4      2 2 2 5 3        3 4 5    5     5 5 5 «
t = 23     1 ←3 4 2      2 2 5 3 3        4 5 5    4 «   5 5 5
t = 24   {3}←4 2 2       2 5 3 3 4        5 5 3 «  4     5 5 5
t = 25   {4}←2 2 2       5 3 3 4 5        5 3 3 «  4     5 5 5
t = 26     2 ←2 2 5      3 3 4 5 2 «      5 3 3    4     5 5 5
t = 27     2 ←2 5 3      3 4 5 2 2 «      5 3 3    4     5 5 5
t = 28     2 ←5 3 3      4 5 2 2 2 «      5 3 3    4     5 5 5
t = 29   {5}←3 3 1 «     4 5 2 2 2        5 3 3    4     5 5 5
t = 30     3 ←3 1 1 «    4 5 2 2 2        5 3 3    4     5 5 5
t = 31     3 ←1 1 4      5 2 2 2 5        3 3 4    5     5 5 5 «
t = 32     1 ←1 4 5      2 2 2 5 3        3 4 5    5     5 5 5 «
t = 33     1 ←4 5 2      2 2 5 3 3        4 5 5    4 «   5 5 5
t = 34     4 ←5 2 2      2 5 3 3 4        5 5 3 «  4     5 5 5
t = 35   {5}←2 2 2       5 3 3 4 5        5 3 3 «  4     5 5 5
t = 36     2 ←2 2 5      3 3 4 5 2 «      5 3 3    4     5 5 5
```

Figure 3.8 Reversed-Flow Simulation of Example 1 to Find *OUTPUT-R*

The algorithm starts by sorting batches according to their due delivery time-windows, [β_i, α_i], starting from the earliest to the latest of β_i. It is logical and justifiable to arrange batches in this manner because batches with early due delivery time-windows should enter pipeline before batches with later due delivery time-windows. Thus, we call the sequence that obtained from arranging batches by their due delivery time-windows an *"INPUT-R"*.

This *INPUT-R* is actually a delivery schedule that we prefer for the final outcome. From an example 1 and due delivery time-windows assigned to this example in section 3.5, an *INPUT-R* is [*55 4 33 222 11*]. If some batches have common β_i's in their due delivery time-windows, then sort batches with earliest α_i first. If some batches have common on both β_i's and α_i's in their delivery time-windows, then sort batches that are away from a source node the most first.

Then, send input batches of *INPUT-R* starting from the end of the sequence into the pipeline and perform reversed-flow pipeline simulation. Those five destination nodes will behave as source nodes for this reversed-flow pipeline simulation. Each batch from the *INPUT-R* enters the pipeline at its corresponding source node and travel to a single destination node.

The delivery schedule from simulating reversed-flow pipeline is called *"OUTPUT-R"*. Recall that we will record the information of the batches only from the first input cycle. The delivery batches that belong to the first input cycle are indicated by the $\mathbf{\{b_i\}}$ in Figure 3.8. The *OUTPUT-R* obtained from simulating reversed-flow

pipeline is [*11 222 33 4 55*]. We can acquire the final input schedule by reversing the sequence in *OUTPUT-R* and let call this new input sequence *"INPUT-F1"*.

INPUT-F1 will be used to compute the delivery schedule by performing a regular pipeline flow simulation (forward-flow). Hence, an *INPUT-F1* obtained by reversing *OUTPUT-R* is [*55 4 33 222 11*]. The delivery schedule of this *INPUT-F1* is identical to the pipeline flow simulation matrix shown in Figure 3.2 and the actual delivery time-windows in section 3.5.

Since we assume in this study that batches of the same product traveling to the same destination node must be grouped together in the input schedule, then if batches order obtained in *INPUT-F1* does not follow this assumption, the modification of *INPUT-F1* to meet this assumption is required.

In a situation where batches of the same product type traveling to the same destination node are not grouped together, for example if *INPUT-F1* is [*55 4* ***2*** *33* ***22*** *11*], then b_2 are not grouped together in this sequence. We can modify this input sequence by moving batches b_2's in the rear of the sequence forward and grouping them together with one batch of b_2 ahead in the sequence. As a result, the new input that satisfies our assumption is [*55 4* ***222*** *33 11*] and we call the modified input sequence that moving ungrouped batches forward an *"INPUT-F2"*. Additionally, we can move the ungrouped batches in the front if the sequence backward and group with two batches of b_2 in the rear of the sequence and we call this modified sequence an *"INPUT-F3"*. Thus, *INPUT-F3* is [*55 4 33* ***222*** *11*]. We run pipeline flow simulation for both *INPUT-F2* and *INPUT-F3* and choose the best delivery schedule from these two modified input

sequences. The reversed-flow algorithm for single-source pipeline is summarized as follows:

Algorithm 1 : Reversed-Flow Algorithm (Single-Source Pipeline)

Step1

Sort batches according to their due delivery time-windows [β_i, α_i]. This sorting process is conducted by heap sorting technique. For n delivery time-windows (i.e., n groups of batches), sort batches by ascending of β_i's, namely $\beta_1 < \beta_2 < \ldots < \beta_n$. Call the input schedule from this sorting *"INPUT-R"*.

If $\beta_i = \beta_{i+1}$, then sort batches with the earliest α_i first, i.e., if $\beta_i = \beta_{i+1}$ and $\alpha_i < \alpha_{i+1}$, then batches of group i are preceding batches of group $i+1$ in the sequence.

If $\beta_i = \beta_{i+1}$ and $\alpha_i = \alpha_{i+1}$, then we break tie by using the length of source-destination nodes for each group of batches. Namely, batches whose length of their corresponding source and corresponding destination nodes is the longest are sorted first.

Step 2

Use *INPUT-R* to simulate reversed-flowed pipeline by sending batches into the pipeline starting from the end of the sequence.

Step 3

Collect the delivery sequence from reversed-flow simulation. Since this is a single-source pipeline, then the reversed-flow problem will be a single-destination problem. So all output information for the delivery sequence will be gathered from a

single point in the pipeline of the reversed-flow simulation. Recall that the information collected is from the first cycle of the input sequence. Call the delivery sequence obtained in this step *"OUTPUT-R"*.

Step 4

Reverse the sequence in *OUTPUT-R* and call this new sequence *"INPUT-F1"*. If each group of batches is sequenced together in *INPUT-F1*, then this *INPUT-F1* is the final input schedule for forward flow simulation. Otherwise, create two new sequences *"INPUT-F2"*, by moving ungrouped batches in the rear of the sequence forward, and *"INPUT-F3"*, by moving ungrouped batches in the preceding of the sequence backward. Hence, *INPUT-F2* and *INPUT-F3* are the final input schedules for forward flow simulation.

Step 5

Use either *INPUT-F1* or *INPUT-F2* and *INPUT-F3* as input schedule(s) for forward flow simulation and then calculate time violation from the actual delivery schedule(s). If *INPUT-F2* and *INPUT-F3* are created, choose the input schedule that yields the minimum total time violation.

3.7.1 Complexity Analysis

It is important for us to determine the time complexity of this reversed-flow algorithm. The next theorem explains the complexity analysis of this algorithm. The

theorem proves that this algorithm can solve the single-source pipeline problem in $O(T \cdot E)$ time.

Theorem 3.2. Reverse-flow algorithm for single-source pipeline can be computed in $O(T \cdot E)$ time.

Proof. Assume that there are N order sets, where each order set $n \in N$ has $b \in I^+$ batches, to be sent into the pipeline. Thus, the number of all batches in the input sequence is equal to $b \cdot N$. Each order set $n \in N$ has its corresponding due delivery time-window. Hence, there are N due delivery time-windows. Let also assume a simple pipeline with total length of E.

The sorting of N sets of due delivery time-windows is done by heap sorting technique and can be computed in $O(N \log N)$ time. There are $b \cdot N$ batches in each input cycle and we perform reversed-flow simulation for time $T \in I^+$ periods, where T is big enough to include all the time periods used to generate delivery schedule and $T > b \cdot N$. In the worst case the total number of input cycles used to simulate flow in order to obtain the delivery schedule of the first input cycle is equal to $\frac{T}{b \cdot N}$. Each batch enters the pipe and travels to a destination node with the length at most E. The time computation for reversed-flow pipeline simulation to obtain an *OUTPUT-R* is $O\ (b \cdot N \cdot \frac{T}{b \cdot N} \cdot E)$. The reverse of an *OUTPUT-R* sequence in order to obtain *INPUT-F1* can be computed

in *O(b·N)* time. The calculation of *INPUT-F2* and *INPUT-F3* are determined in *O(2b·N)* time. The time computation of forward flow pipeline simulation can be calculated in the similar approach to the calculation of reverse flow pipeline simulation, which is equal to $O\ (b \cdot N \cdot \frac{T}{b \cdot N} \cdot E)$. Therefore, the reversed-flow algorithm for a single-source pipeline problem can be implemented in *O(2T·E + 3b·N + N log N+2N) ~ O(2T·E + 4b·N) ~ O(T·E)* time. **Q.E.D.**

3.7.2 Discussion

It is expectable for the reversed-flow algorithm to yield input sequences that give poor delivery time results for some sets of pipeline information since the algorithm tends to sort batches whose destination is further away the most from the source node first into the pipeline. Furthermore, the position of each group of batches in the sequence is dependent of one another so that each unique input sequence could yield a tremendously different result on the actual product delivery times. Specifically, if batches traveling to the destination nodes closer to the source node have earlier due delivery time-windows than ones traveling to nodes further away from the source node, then the results of actual delivery time is likely to be poor.

Since the nature of flow movement in the pipeline is unique and the position of each batch in the sequence is dependent to one another during the movement in the pipeline, then it is quite difficult to determine the input sequence that yields the minimum violation of the actual delivery time from the due delivery time-windows. The possible approach to find the optimal input sequence may be conducted by exhaustive search. If there are *N* groups of batches to be sent into the pipeline, it would take *N!* time to compute the feasible input sequences. Moreover, each input sequence requires $T \cdot E$ times to simulate pipeline flow in order to compute the actual product delivery times.

Consider the following pipeline problem with seven destinations, node *1* to *7*, and fifty input batches. The total length of pipeline is 71 and the lengths of *(S, 1), (1, 2), (2, 3), (3, 4), (4, 5), (5, 6), (6, 7)* equal 8, 12, 9, 11, 11, 11, 9, respectively. The input batches information is as follows: node *1* requires a delivery of 5 batches between time [40, 47]; node *2* requires a delivery of 6 batches between time [62, 69]; node *3* requires a delivery of 11 batches between time [25, 40]; node *4* requires a delivery of 8 batches between time [73, 100]; node *5* requires a delivery of 7 batches between time [109, 125]; node *6* requires a delivery of 8 batches between time [150, 158]; node *7* requires a delivery of 5 batches between time [185, 229]. The input sequence yielded from the reversed-flow algorithm is [*77777 66666666 5555555 44444444 33333333333 222222 11111*] and the total time violation is equal to 134 while the optimal input sequence is [*33333333333 77777 66666666 44444444 11111 5555555 222222*] and minimum total time violation is 1. the results of this example is displayed in Table 3.1.

Table 3.1 Solution of an Example of a Poor Performance Problem

Product Batches	Due Time-windows	Actual Delivery Windows (Algorithm)	Actual Delivery Windows (Optimal)
11111	[40, 47]	[53, 58]	[40, 45]
222222	[62, 69]	[64, 79]	[64, 70]
33333333333	[25, 40]	[62, 79]	[29, 40]
44444444	[73, 100]	[82, 90]	[75, 99]
5555555	[109, 125]	[94, 101]	[110, 123]
66666666	[150, 158]	[109, 142]	[150, 158]
77777	[185, 229]	[158, 203]	[199, 225]

3.7.3 Optimal Solution

As we proved in Theorem 3.1 that single-source oil pipeline distribution of multiple products subject to delivery time-windows problem is NP-complete. So for any problem with a large number of destinations and input batches, the optimal solution cannot be computed in polynomial time. However, the optimal solution for small-sized pipeline problems can be computed by using exhaustive search on the input schedule.

CHAPTER IV

MULTIPLE-SOURCE PIPELINES

We study the oil pipeline distribution of multiple products subject to with delivery time-windows for multiple-source pipelines in this chapter. The multiple-source pipeline is an extension of the single-source pipeline discussed in the previous chapter. The complexity of the problem and the proposed solution methodology will be examined in this chapter. The algorithm proposed to solve the problem in this chapter is modified from the one used to solve the single-source pipeline.

Consider a multiple-source pipeline as a pipeline that connects at least two single-source, simple pipeline problems together as a path. In the single-source pipeline problems, we usually assume that the first node located at the beginning of the pipe is a source node. This assumption is still applicable for the multiple-source pipeline introduced in this chapter. The source node(s) other than the one at the beginning of the pipeline are assumed to be located anywhere downstream of the pipe. The principal assumption of each batch travels one unit distance per one unit time is also applied to the multiple-source problem. Pipeline flow is also assumed to be directed and assumptions outlined in Chapter II are also hold for the pipeline problems in this chapter. Hence, a pipeline with multiple sources is also a path, in which a route of any pair of source-

destination nodes is fixed and unique. Furthermore, some additional assumptions are necessary for examining the multiple-source pipeline problem. Source nodes are solely used for dispensing products into the pipeline. Therefore, it is not allowed for the delivery of products at any source nodes located downstream in the pipe. Each source node is assumed to carry unique products that are uncommon with the ones of other source node(s). For example, if there are four product types, *A*, *B*, *C* and *D*, and there are three source nodes, S_1, S_2 and S_3, then it is valid to our assumption for products *A* and *C* to be released from S_1, and products *B* and *D* to be released from S_2 and S_3, respectively. Namely, any single product type is not sent into the pipeline from two or more source nodes.

The complicating issue in the transition from a single to multiple source pipelines is that input batches can be inserted into the middle of pipeline flow. This obvious difference causes profound changes in the nature of pipeline operations. Thus, there may be a way to exploit the multiple-source pipeline based on the knowledge we developed in single-source pipeline problems.

The multiple-pipeline problem can be better explained by using an example. Thus, the first section of this chapter is the illustration of an example problem of a multiple-source pipeline. Then, it is followed by a study of the attributes and characteristics of multiple-source pipeline problem and also its problem complexity. The reversed-flow algorithm developed in Chapter III will be modified for use in multiple-source pipeline problem.

4.1 Example 2

Consider a simple pipeline problem with two source nodes and three destination nodes as the pipeline structure is illustrated in Figure 4.1. Nodes 1 and 3 are assigned to be source nodes and the other nodes are destination nodes. The edge volume connecting every node pair is set equal to two volume units.

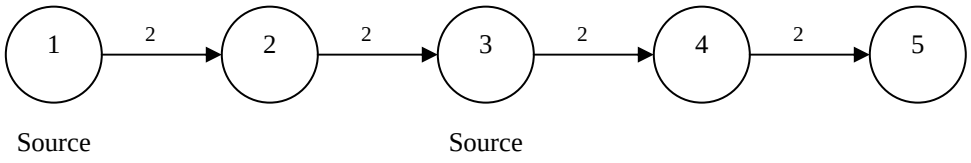


Figure 4.1 A Pipeline with Two Sources and Three Destination Nodes

Let assume that there are three product types to be transported in the pipeline; i.e., products *A*, *B* and *C*. Source node 1 dispenses products *A* and *B* and source node 3 dispenses product *C*. Each destination node requests the product delivery as the following: three units of *A* at node 2, two units of *A* at node 4, three units of *C* at node 4 and two units of *B* at node 5. Namely, each batch requires the input information of its corresponding source and destination nodes as the source node is not fixed to be only at the first node at the start of the pipe like in the single-source pipeline.

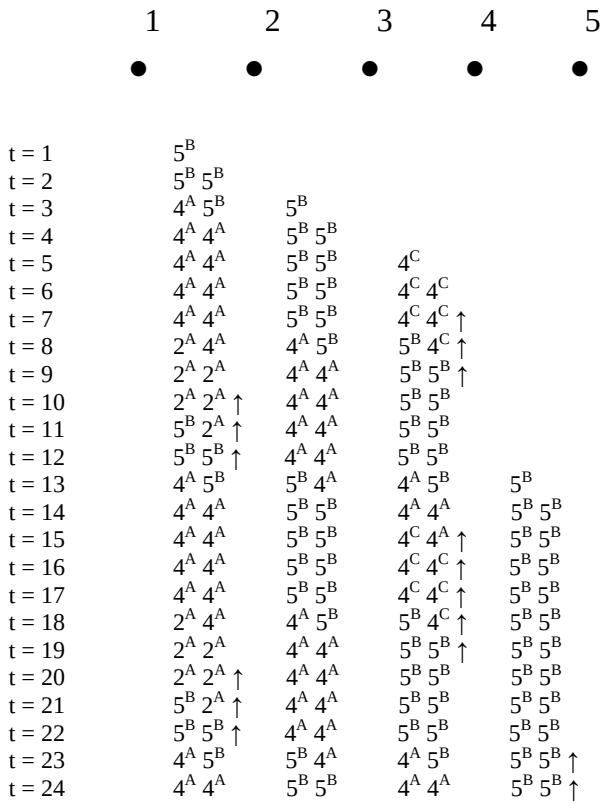


Figure 4.2 Pipeline Flow Simulation of Example 2

Let $[\,b_5^B b_5^B\ b_4^A b_4^A\ b_4^C b_4^C b_4^C\ b_2^A b_2^A b_2^A\,]$ be an input sequence for this example. Note that due delivery time-windows will be introduced later in this section. Pipeline flow simulation matrix of this input sequence is displayed in Figure 4.2. Another assumption required for operating a multiple-source pipeline is when a batch is inserted from any

source node located elsewhere in the middle of the pipe; all upstream batches from that source node remain stationary as that input batch pushes all downstream batches from that source node forward. This assumption makes it possible to generate the pipeline flow simulation. For example in Figure 4.2, at time t = 15 when b_4^C is inserted into the pipe at node 3 then all batches in the pipe upstream of node 3 stay motionless. This assumption is also applied to the case that a batch is sent into a pipeline from a source node located in the middle of the pipeline and the upstream pipe from that source node is not completely filled. Since we do not require the release of input batches to be in any particular order of the source nodes, then each source node may alternately pushes batches into the pipe according to its corresponding input sequence. This scenario leads to an event when there is a gap between batches in the pipe as it is shown in Figure 4.3.

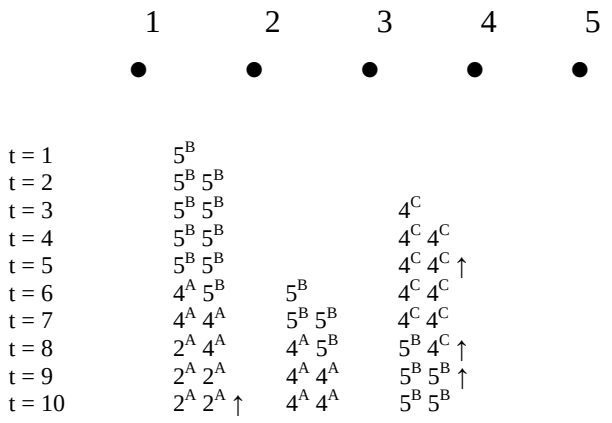


Figure 4.3 Gap Between Batches in the Pipeline

Let assume the input sequence of set of orders **O** for Figure 4.3 be [$b_5^B b_5^B$ $b_4^C b_4^C b_4^C$ $b_4^A b_4^A$ $b_2^A b_2^A b_2^A$]. Two batches $b_5^B b_5^B$ are sent into the pipe from node 1 at the first two periods, following by b_4^C enters the pipe at node 3 at *t* = 3. At this period, $b_5^B b_5^B$ remain stationary by occupying the first two volume units of edge (1, 2), and b_4^C moves downstream and occupies the first one volume unit of edge (3, 4). It can be seen in Figure 4.3 that there exits a gap between batches on an edge connecting node 2 and node 3 from periods *t* = 3 to 6.

At *t* = 6 when b_4^A enters the pipeline at node 1, it pushes the two batches of $b_5^B b_5^B$ that are already in the pipe forward for one unit. However, since there still is a gap between batches entering the pipe from node 1 and batches entering from node 3 at this period and it is not possible to have any adjacent batch to be able to push two batches of $b_4^C b_4^C$ occupying two volume units of edge (3, 4), then these two batches must remain stationary in this period.

Assign the due delivery time-windows for each group of batches for this pipeline problem as follows: $b_2^A b_2^A b_2^A$ at node 2 at time [10, 15]; $b_4^C b_4^C b_4^C$ at node 4 at time [5, 10]; $b_4^A b_4^A$ at node 4 at time [13, 16]; $b_5^B b_5^B$ at node 5 at time [21, 25]. The actual delivery time-windows recorded from the flow simulation shown in Figure 4.2 are as the following: $b_2^A b_2^A b_2^A$ are delivered at time [9, 12]; $b_4^C b_4^C b_4^C$ are delivered at time [6, 9]; $b_4^A b_4^A$ are delivered at time [14, 16]; and $b_5^B b_5^B$ are delivered at time [22, 24]. Therefore, the total time violation equals 1.

4.2 Network Representation for Multiple-Source Oil Pipeline Distribution of Multiple Products Problem

Multiple-source, multiple-product oil pipeline problem can be transformed into multiple time-periods multicommodity network representation in the same approach as the case of a single-source problem. Pipeline structure in Figure 4.1 can be transformed to be a unitized pipeline by adding intermediate nodes to any edges that have a length greater than one unit, like the modified pipeline structure shown in Figure 2.2. Thus, the modified multiple-source pipeline structure of an example shown in Figure 4.1 is constructed by replacing each edge with an intermediate node and reconnecting all nodes in the structure with one-unit directed edges. The modified pipeline structure is shown in Figure 4.4.

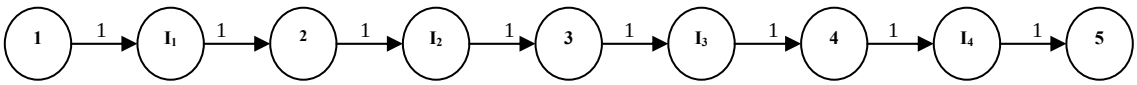


Figure 4.4 Modified Pipeline Structure for Pipeline from Example 2

In Figure 4.4, four intermediate nodes, I_1, I_2, I_3 and I_4, are added in order to make every edge in the pipeline has a unit length. Recall that delivery of products is not allowed at these intermediate nodes.

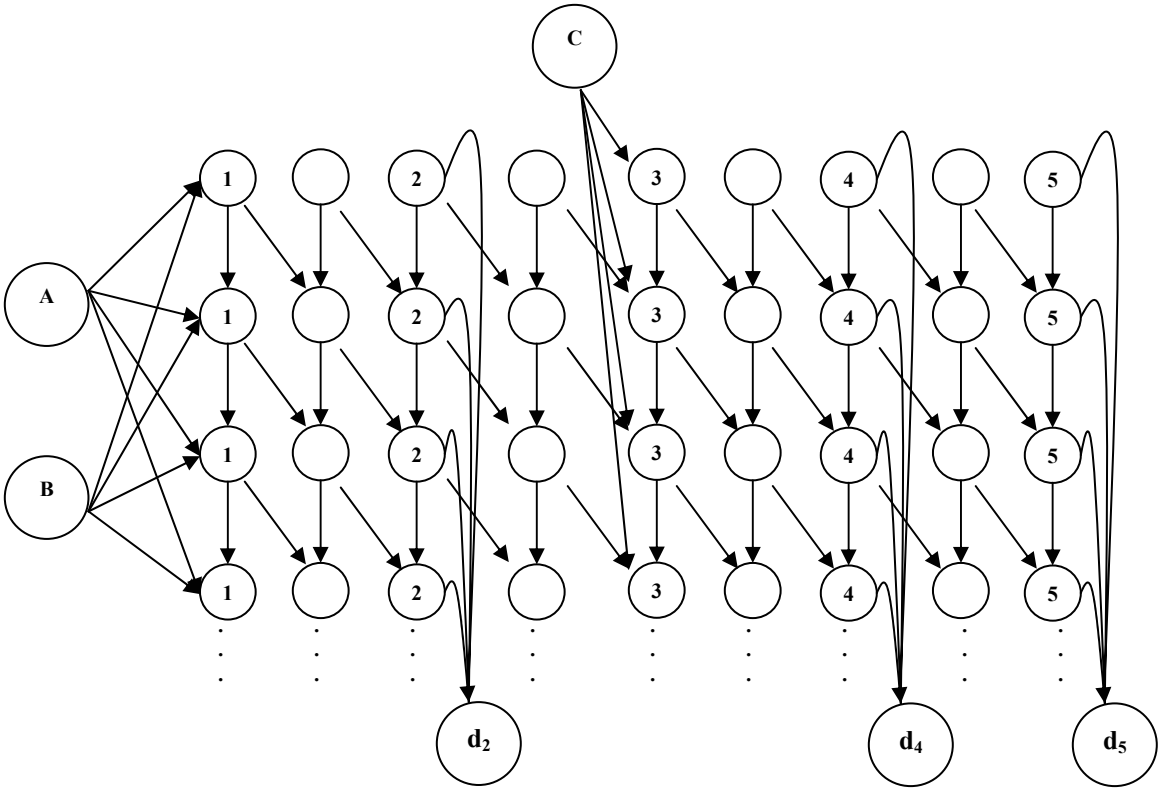


Figure 4.5 Multiple Time-Periods Multicommodity Network for Example 2

The multiple time-periods multicommodity network for oil pipeline problem for multiple-source pipeline is constructed based on the replication of single-source pipeline structure and it is illustrated in Figure 4.4. Each node v in period t except the last node of each row is connected to nodes in the next period, $t+1$, by two edges. One edge connecting to node $v+1$ in period $t+1$ represents the moving of flow in the pipe for one unit distance. The other edge connecting to node v in period $t+1$ represents the stationary of the flow for one unit of time. Each source node in every time period is connected to its corresponding pseudo-source node and each destination node of all time periods is connected to its corresponding pseudo-sink node.

It can be seen in Figure 4.5 that there are three pseudo-source nodes and three pseudo-sink nodes. Each pseudo-source node represents each product type. There are three product types, *A*, *B* and *C*, in example 2. Pseudo-source nodes *A* and *B* have edges connecting to node 1 of every time period and pseudo-source node *C* also has edges connecting to node 3 of every time period.

4.3 Multiple-Source Oil Pipeline Distribution of Multiple Products Subject to Delivery Time-Windows Is NP-Complete

The NP-complete proof for multiple-source oil pipeline distribution of multiple products subject to delivery time-windows can be done in the similar approach to that of a single-source problem. The proof is conducted by means of a reduction from a

satisfiability problem. The instance and question for multiple-source problem are the same as those of a single-source problem described in section 3.6.

Theorem 4.1. Multiple-source oil pipeline distribution of multiple products subject to delivery time-windows problem is NP-complete.

Proof. In a similar claim to a single-source pipeline problem, we can reduce a satisfiability problem into multicommodity flow problem, and then transform into a version of our problem. Regardless of how many source nodes the pipeline has, the transformation of a pipeline problem into a multiple time-periods multicommodity network flow used in the proof is identical to the transformation of a single-source pipeline. Each batch is released into the pipe from its corresponding source node at any possible time period and will be delivered at its corresponding destination node. Therefore, it is able to claim that the proof for multiple-source problem is analogous to the one of the single-source problem shown in Theorem 3.1.

($\rightarrow$) If there exists a conjunctive normal form expression for a satisfiability problem and this conjunctive normal form expression is satisfiable, then the multiple-source oil pipeline distribution of multiple products subject to delivery time-windows is feasible. Given that each variable is assigned the truth value ("true" or "false"). The horizontal path in the subgraph for variable u is chosen if u is assigned a "true" value and the vertical path in the subgraph for variable u is chosen if u is assigned a "false" value. If either u or $\bar{u}$ appears in clause c_j and its value is "true", there is a path in the

subgraph connecting s_{c_i} and n_{c_i} for variable *u*. Note that the subgraph for variable *u* can be seen in to Figure 3.7.

(←) Assume that the multiple-source oil pipeline distribution of multiple products subject to delivery time-windows is feasible and there exists a solution to our problem by having |*K*| flow functions, which satisfy the requirements, for the total number of |*K*| batches entering the pipeline. Each batch is released from its corresponding source node and delivered to its designated destination node. There is exactly one path corresponding to each clause of the instance of the satisfiability problem. Assign the truth value "true" for each variable *u* to a path traversed by a batch from source to destination (implicitly assigning "false" value to its negation). Arbitrarily assign "true" value for any unassigned variables and no variable is assigned both "true" and "false" values. As a result, each variable has a single value assigned and the satisfiability problem is satisfiable.

Thus both requirements are met. **Q.E.D.**

Therefore, there is no polynomial algorithm to solve multiple-source oil pipeline distribution of multiple products subject to delivery time-windows unless *P* = *NP*. The construction of approximation algorithm for multiple-source oil pipeline distribution of multiple products subject to delivery time-windows is modified from the algorithm provided for the single-source problem as described in Chapter III.

4.4 Modified Reverse-Flow Algorithm for Multiple-Source Pipeline

The constructive algorithm for multiple-source pipeline is constructed based on a reverse-flow technique similar to the algorithm of a single source-pipeline. We will implement the algorithm to the problem in example 2 and the summary of the algorithm will be provided later in this section.

We start an algorithm by arranging input batches according to their due delivery time-windows [β_i, α_i], starting from the earliest to the latest of β_i. If some batches have common β_i, then the rule for tie-breaking is the same as the one in a single-source pipeline problem. The sequence obtained after arranging batches by due time-windows is called *"INPUT-R"*. Thus *INPUT-R* for example 2 is [$b_4^C b_4^C b_4^C\ b_2^A b_2^A b_2^A\ b_4^A b_4^A\ b_5^B b_5^B$]. This input sequence can be shorten in the form of [$4^C4^C4^C\ 2^A2^A2^A\ 4^A4^A\ 5^B5^B$]. Then, send input batches into the pipeline starting from the end of *INPUT-R* sequence. Then, perform a reversed-flow pipeline simulation as shown in Figure 4.6. Recall that destination nodes and source nodes in regular-flow pipeline function as source nodes and destination nodes, respectively, in this reversed-flow pipeline. In Figure 4.6, flow travels in reverse direction from three sources, node 2, 4 and 5, to two destinations, nodes 1 and 3. Product deliveries at either node 1 or 3 are indicated in { } sign underneath such nodes.

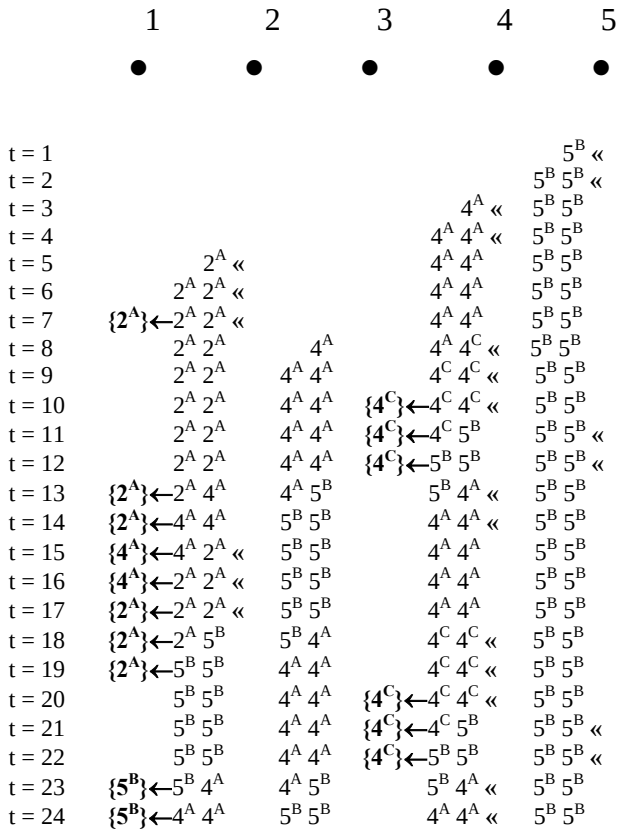


Figure 4.6 Pipeline Reversed-Flow Simulation of Example 2 to Find *OUTPUT-R*

The delivery sequence obtained from simulating reversed-flow pipeline is also called *"OUTPUT-R"*. The information for *OUTPUT-R* is those from the first input sequence. *OUTPUT-R* from Figure 4.6 is [$2^A\ 4^C4^C4^C\ 2^A2^A\ 4^A4^A4^A\ 5^B5^B$]. *INPUT-F1*, [$5^B5^B\ 4^A4^A4^A\ 2^A2^A2^A\ 4^C4^C4^C\ 2^A$], is obtained by reversing *OUTPUT-R* sequence. In case that batches in *INPUT-F1* are not grouped together according to their product types and the destination nodes, then *"INPUT-F2"* and *"INPUT-F3"* will be introduced.

We see that batches 2^A's in *INPUT-F1* are not grouped together. We can modify *INPUT-F1* either by moving one batch of 2^A in the rear forward and grouping it with the ones in the front and define this new sequence *INPUT-F2* or by moving those two batches in the front and group them with the one in the rear and call this new sequence *INPUT-F3*. Therefore, *INPUT-F2* is [5^B5^B 4^A4^A $2^A2^A2^A$ $4^C4^C4^C$] and *INPUT-F3* is [5^B5^B 4^A4^A $4^C4^C4^C$ $2^A2^A2^A$]. These two sequences are used to perform pipeline flow simulation and the best delivery schedule will be selected from these two input sequences. The reversed-flow algorithm for multiple-source pipeline problem can be summarized as the following:

Algorithm 2 : Modified Reversed-Flow Algorithm (Multiple-Source Pipeline)

Step1

Sort batches according to their due delivery time-windows [β_i, α_i] by using heap sorting technique. For n delivery time-windows (i.e., n groups of batches), sort batches by an ascending of β_i's, namely $\beta_1 < \beta_2 < \ldots < \beta_n$. Call the input schedule from this sorting *"INPUT-R"*.

If $\beta_i = \beta_{i+1}$, then sort batches with the earliest α_i first, i.e., if $\beta_i = \beta_{i+1}$ and $\alpha_i < \alpha_{i+1}$, then batches of group i are preceding batches of group $i+1$ in the sequence.

If $\beta_i = \beta_{i+1}$ and $\alpha_i = \alpha_{i+1}$, then batches whose length of their source and corresponding destination nodes is the longest are sorted first. Namely, if batches of group i have node n as a destination node and batches of group $i+1$ have node $n-1$ as

destination node, where node *n–1* precedes node *n* in the pipeline, then batches of group *i* are preceding batches of group *i+1* in the input sequence.

Step 2

Use *INPUT-R* to simulate reversed-flowed pipeline by sending batches starting from the end of the sequence.

Step 3

Collect the delivery sequence from reversed-flow pipeline simulation. Unlike the single-source pipeline problem, the reversed-flow simulation for multiple-source pipeline is a multiple-destination problem. Thus, output information for the delivery sequence will be collected from multiple destination points in the pipeline of the reversed-flow simulation. Recall that the information collected is from the first cycle of the input sequence. Define the delivery sequence obtained from this step *"OUTPUT-R"*.

Step 4

Reverse *OUTPUT-R* sequence and call this new sequence *"INPUT-F1"*. If each group of batches is sequenced together in *INPUT-F1*, then this *INPUT-F1* is the final solution. Otherwise, create new sequences *"INPUT-F2"*, by moving ungrouped batches in the rear of the sequence forward, and *"INPUT-F3"*, by moving ungrouped batches in the preceding of the sequence backward. Hence, *INPUT-F2* and *INPUT-F3* are the final solutions.

Step 5

Use either *INPUT-F1* or *INPUT-F2* and *INPUT-F3* as input schedule(s) for forward flow simulation and then calculate time violation from the actual delivery schedule(s). If *INPUT-F2* and *INPUT-F3* are created, choose the input schedule that yields the minimum total time violation.

4.4.1 Complexity Analysis

The algorithm for multiple-source pipeline is essentially modified from the one of single-source pipeline. The next theorem explains the time complexity of the algorithm provided in section 4.4.

Theorem 4.2. Reverse-flow algorithm for multiple-source pipeline can be implemented in $O(T \cdot E)$ time.

Proof. Basically, the reversed-flow algorithms for single-source and multiple-source problems are similar. The algorithm is composed of one sorting, one reverse of sequences, reversed-flow simulation, two modifications of ungrouped input sequence, *INPUT-F1* to determine *INPUT-F2* and *INPUT-F3*, forward flow pipeline simulation and the computation of time violation. Thus, the reversed-flow algorithm for multiple-source pipeline problem can be implemented in $O(2T \cdot E + 3b \cdot N + N \log N + 2N) \sim O(2T \cdot E + 4b \cdot N) \sim O(T \cdot E)$ time. **Q.E.D.**

CHAPTER V

COMPUTATIONAL IMPLEMENTATION

In the previous chapters, we provided an analysis for the oil pipeline distribution of multiple products subject to delivery time-windows and its solution methodologies. However, it is important to see how well the proposed algorithms perform in practice. In this chapter, we tested and analyzed the reversed-flow algorithms for both single-source and multiple-source pipeline problems on a set of random problems from a uniform distribution. The problems are composed of different pipeline length and destination nodes. All proposed algorithms were coded in Borland JBuilder 9.0.

In each problem, there is just one order per destination. If each destination node is allowed to receive multiple orders, then the problem is equivalent to permit split orders. The input data were randomly generated to approximate the line. For example, Colonial Pipeline has two major pipelines that vary from 32” to 40” in diameter with minimum order of 75,000 barrels. The length of both lines is approximately 2,800 miles with 20 points assigned as delivery depots. Assuming a pipe diameter of 3 feet to simplify the calculations, thus a minimum is slightly more than 420,000 ft^3, or about 9-mile in length. Though the problems randomly generated in this study are not as comparable as the ones used in generally used in the industry, their results yield useful information that could be useful for applying to problems in actual scenarios.

5.1 Single-Source Pipeline

There are four data sets generated for testing the algorithm on single-source pipeline problem. Tables 5.1, 5.2 and 5.3 are the information for data sets 1, 2 and 3, respectively. The number of destinations varies from 5 to 10 nodes, the total edge volumes from 45 to 141 and the total number of input batches from 16 to 130. In data sets 1, 2 and 3 the numbers of destination nodes are U[5, 8], U[5, 10] and U[6,10], respectively. Problems are labeled in numeric form after the letter P. The first two digits are the number of source node, the next two digits are the number of to total nodes in the pipeline, and the next three digits are the total edge volumes of the pipeline.

Data set 4 was generated differently. The problems in data set 4 were constructed in such a way that larger orders were placed at destination nodes downstream of the pipeline and the total number of input batches for each pipeline problem was intended to be larger than the total pipeline volume. The motivation for generating this data set was that they would be harder to solve than the previous sets and they certainly require much longer computing time to find optimal solutions. Data set 4 are displayed in Table 5.4.

Tables 5.5, 5.6, 5.7 and 5.8 show the input sequences and total time violation of each problem calculated from reversed-flow algorithm as well as the optimal input sequence and minimum total time violation.

Table 5.1 Data Set 1

Problem Number	Number of Input Batches	Number of Destination Nodes	Pipeline Length
P0106045	25	5	45
P0106060	32	5	60
P0107087	35	6	87
P0107105	58	6	105
P0108123	50	7	123
P0108114	44	7	114
P0109152	63	8	152
P0109188	89	8	188
P0109120	55	8	120
P0107105	120	6	105

Table 5.2 Data Set 2

Problem Number	Number of Input Batches	Number of Destination Nodes	Pipeline Length
P0106045	32	5	45
P0108069	42	7	69
P0107847	39	7	47
P0107042	49	6	42
P0109074	63	8	74
P0107096	82	6	96
P0110086	47	9	86
P0109052	31	8	52
P0111020	16	10	20
P0109115	130	8	115

Table 5.3 Data Set 3

Problem Number	Number of Input Batches	Number of Destination Nodes	Pipeline Length
P0107127	58	6	127
P0107092	70	6	92
P0107115	81	6	115
P0108141	63	7	141
P0108118	47	7	118
P0108098	39	7	98
P0108130	67	7	130
P0109120	50	8	120
P0110105	35	9	105
P0111100	29	10	100

Table 5.4 Data Set 4

Problem Number	Number of Input Batches	Number of Destination Nodes	Pipeline Length
P0106045	133	5	45
P0107087	250	6	87
P0108123	296	7	123
P0108114	293	7	114
P0108098	342	7	98

Table 5.5 Solutions for Data Set 1

Problem Number	Reversed-Flow Time Violation	Sequence (Reversed-Flow)	Min Time Violation (Optimal)	Sequence (Optimal)
P0106045	95	$[5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	84	$[5^E\ 1^A\ 3^C\ 4^D\ 2^B]$
P0106060	98	$[5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	92	$[5^E\ 3^C\ 4^D\ 2^B\ 1^A]$
P0107087	28	$[6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	18	$[6^F\ 4^D\ 5^E\ 3^C\ 2^B\ 1^A]$
P0107105	19	$[6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	16	$[6^F\ 4^D\ 3^C\ 5^E\ 2^B\ 1^A]$
P0108123	5	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	5	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$
P0108114	33	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	30	$[7^G\ 6^F\ 5^E\ 1^A\ 3^C\ 4^D\ 2^B]$
P0109152	114	$[8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	83	$[8^H\ 6^F\ 7^G\ 3^C\ 5^E\ 4^D\ 2^B\ 1^A]$
P0109188	65	$[8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	34	$[8^H\ 6^F\ 7^G\ 1^A\ 5^E\ 3^C\ 4^D\ 2^B]$ or $[8^H\ 6^F\ 1^A\ 7^G\ 4^D\ 5^E\ 3^C\ 2^B]$
P0109120	18	$[8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	18	$[8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$
P0107105	150	$[5^E\ 6^F\ 3^C\ 4^D\ 2^B\ 1^A]$	82	$[5^E\ 3^C\ 4^D\ 2^B\ 1^A\ 6^F]$

Table 5.6 Solutions for Data Set 2

Problem Number	Reversed-Flow Time Violation	(Sequence) Reversed-Flow)	Min Time Violation (Optimal)	Sequence (Optimal)
P0106045	67	$[5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	66	$[5^E\ 4^D\ 3^C\ 1^A\ 2^B]$ or $[4^D\ 3^C\ 5^E\ 2^B\ 1^A]$
P0108069	23	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	23	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$
P0107847	50	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	45	$[7^G\ 5^E\ 4^D\ 6^F\ 3^C\ 2^B\ 1^A]$
P0107042	21	$[6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	21	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$
P0109074	10	$[8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	6	$[8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 2^B\ 1^A\ 3^C]$
P0107096	28	$[6^F\ 5^E\ 2^B\ 3^C\ 4^D\ 1^A]$	28	$[6^F\ 5^E\ 2^B\ 3^C\ 4^D\ 1^A]$
P0110086	9	$[9^I\ 8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	9	$[9^I\ 8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$
P0109052	61	$[8^H\ 7^G\ 6^F\ 4^D\ 5^E\ 3^C\ 2^B\ 1^A]$	61	$[8^H\ 7^G\ 6^F\ 4^D\ 5^E\ 3^C\ 2^B\ 1^A]$
P0111020	15	$[10^J\ 9^I\ 8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	15	$[10^J\ 9^I\ 8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$
P0109115	135	$[6^F\ 5^E\ 7^G\ 8^H\ 1^A\ 2^B\ 3^C\ 4^D]$	121	$[6^F\ 5^E\ 7^G\ 8^H\ 1^A\ 4^D\ 2^B\ 3^C]$

Table 5.7 Solutions for Data Set 3

Problem Number	Reversed-Flow Time Violation	(Sequence) Reversed-Flow)	Min Time Violation (Optimal)	Sequence (Optimal)
P0107127	76	$[6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	70	$[6^F\ 4^D\ 5^E\ 3^C\ 2^B\ 1^A]$
P0107092	53	$[4^D\ 5^E\ 6^F\ 3^C\ 2^B\ 1^A]$	53	$[4^D\ 5^E\ 6^F\ 3^C\ 2^B\ 1^A]$
P0107115	80	$[6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	80	$[6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$
P0108141	106	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	65	$[7^G\ 6^F\ 3^C\ 5^E\ 4^D 1^A\ 2^B]$
P0108118	110	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	93	$[7^G\ 6^F\ 5^E\ 4^D\ 2^B\ 3^C\ 1^A]$
P0108098	109	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	95	$[7^G\ 6^F\ 2^B\ 4^D\ 5^E\ 3^C\ 1^A]$ or $[7^G\ 2^B\ 6^F\ 4^D\ 5^E\ 3^C\ 1^A]$
P0108130	109	$[7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	76	$[7^G\ 5^E\ 6^F\ 2^B\ 4^D\ 3^C\ 1^A]$
P0109120	105	$[8^H\ 7^G\ 6^F\ 4^D\ 5^E\ 3^C\ 2^B\ 1^A]$	83	$[8^H\ 6^F\ 3^C\ 2^B\ 7^G\ 4^D\ 5^E\ 1^A]$
P0110105	15	$[9^I\ 8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	15	$[9^I\ 8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$
P0111100	49	$[10^J\ 9^I\ 8^H\ 7^G\ 6^F\ 5^E\ 4^D\ 3^C\ 2^B\ 1^A]$	29	$[8^H\ 10^J\ 7^G\ 6^F\ 9^I\ 2^B\ 5^E\ 4^D\ 3^C\ 1^A]$ or $[8^H\ 10^J\ 9^I\ 6^F\ 7^G\ 2^B\ 5^E\ 4^D\ 3^C\ 1^A]$ or $[8^H\ 10^J\ 6^F\ 7^G\ 9^I\ 2^B\ 5^E\ 4^D\ 3^C\ 1^A]$

Table 5.8 Solutions for Data Set 4

Problem Number	Reversed-Flow Time Violation	Sequence (Reversed-Flow)	Min Time Violation (Optimal)	Sequence (Optimal)
P0106045	95	$[2^B\ 1^A\ 4^D\ 5^E\ 3^C]$	70	$[2^B\ 1^A\ 5^E\ 3^C\ 4^D]$
P0107087	234	$[2^B\ 3^C\ 1^A\ 4^D\ 5^E\ 6^F]$	234	$[2^B\ 3^C\ 1^A\ 4^D\ 5^E\ 6^F]$
P0108123	321	$[3^C\ 2^B\ 4^D\ 1^A\ 5^E\ 6^F\ 7^G]$	317	$[2^B\ 4^D\ 1^A\ 3^C\ 5^E\ 6^F\ 7^G]$
P0108114	291	$[3^C\ 2^B\ 1^A\ 4^D\ 5^E\ 6^F\ 7^G]$	291	$[3^C\ 2^B\ 1^A\ 4^D\ 5^E\ 6^F\ 7^G]$
P0108098	306	$[2^B\ 3^C\ 1^A\ 4^D\ 5^E\ 6^F\ 7^G]$	306	$[2^B\ 3^C\ 1^A\ 4^D\ 5^E\ 6^F\ 7^G]$

The results of this reversed-flow algorithm are quite good compared to the optimal solutions. Fourteen of the thirty-five problems were solved optimally by the reversed-flow algorithm, and nine of the overall problems were more than 30% greater than the optimal values. In order to understand the running time, all problems in data set 1, 2 and 3, which have up to 10 destination nodes, 130 input batches and 188 pipeline volume units were solved in less than 3 seconds by the algorithm. However, the running

times for computing the optimal solutions are ranging from a few seconds to more than 17,000 seconds. In data set 4, which has up to 7 destination nodes, 342 batches and 123 pipeline volume units took less than 6 seconds to solve by the algorithm and up to 3,100 seconds to compute the optimal solutions. The proposed algorithm has a considerable advantage of computing an input schedule that yields the value of total time violation closer to the optimal value without consuming much of the running time. Unlike finding an optimal input schedule, the reversed-flow algorithm can determine a solution in a matter of seconds. The proposed algorithm is capable to solve problems with a large number of input batches, destination nodes and pipeline length.

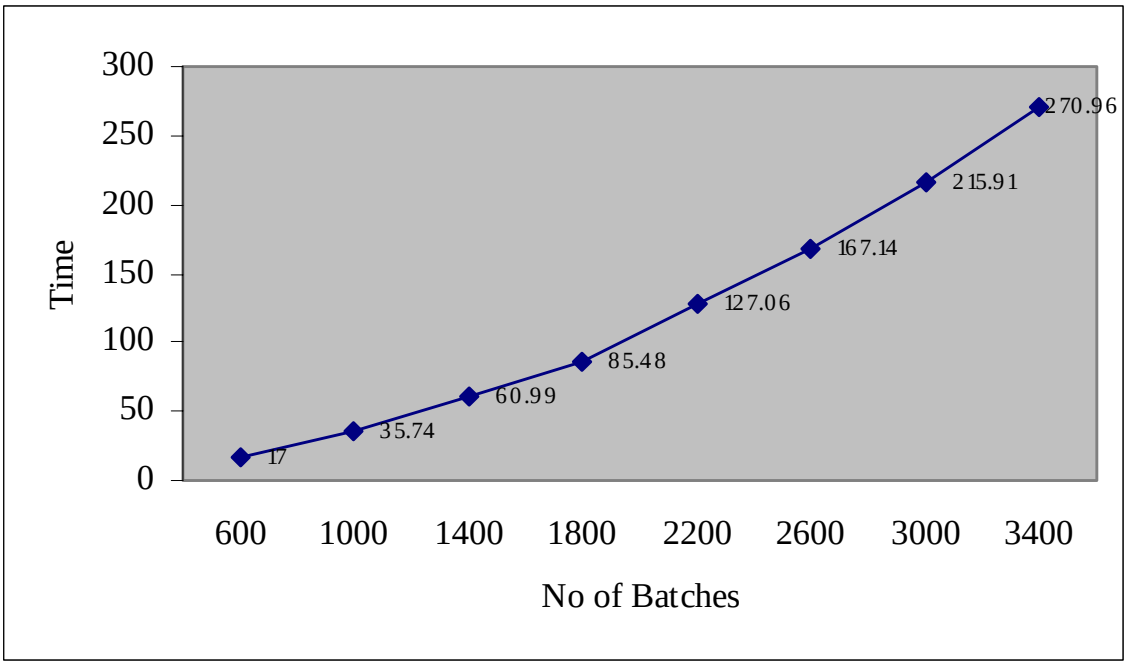


Figure 5.1 Relationship between Number of Input Batches and Computation Time for Single-Source Pipeline

Figure 5.1 illustrates the relationship between computation time and the number of input batches. As the number of input batches increases, the computation time is likely to increase in linear manner. However, the relationship between the number of destination nodes and computation time is likely to be in nonlinear form. The computation time increases exponentially as the number of destination nodes increases. The relationship between computation time and the number of destination nodes are displayed in Figure 5.2.

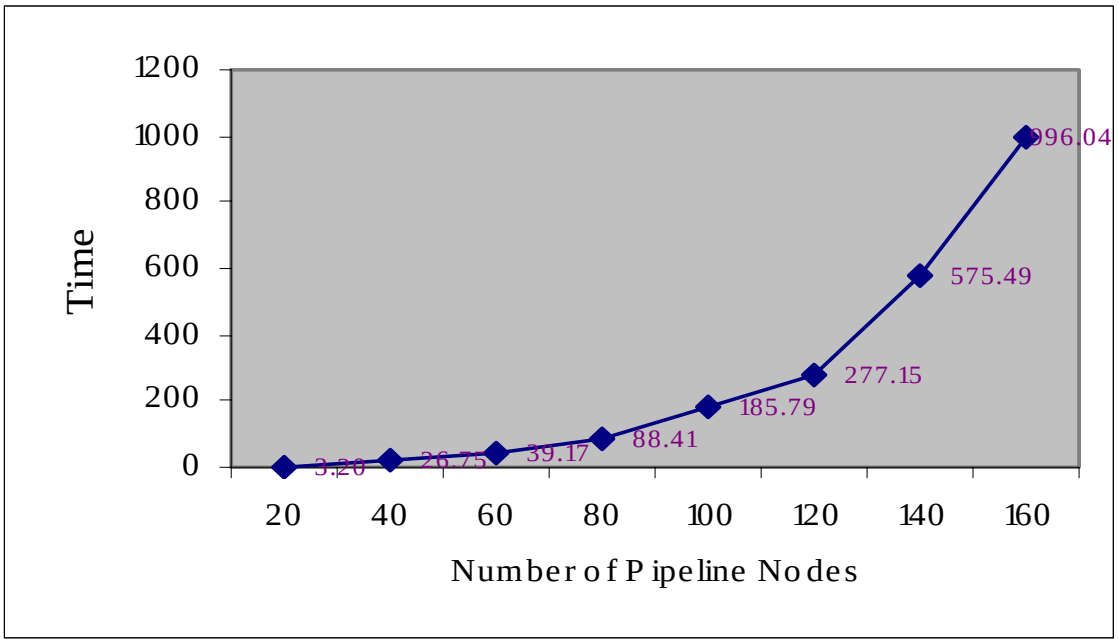


Figure 5.2 Relationship between Number of Destination Nodes and Computation Time for Single-Source Pipeline

5.2 Multiple-Source Pipeline

This section displays the implementation of reversed-flow algorithm to multiple-source pipeline problems. The problem sets are randomly generated and the maximum number of sources in these problems is 4. Each problem is labeled in the same numeric form as the ones in single-source problems presented in section 5.1. The data set 5 are multiple-source pipeline problems as the information of this data set is shown in Tables 5.9 and 5.10.

Table 5.9 Data Set 5

Problem Number	Number of Input Batches	Number of Source Nodes	Number of Destination Nodes	Pipeline Length
P0208064	49	2	6	64
P0210053	17	2	8	53
P0209097	42	2	7	97
P0206046	30	2	4	46
P0312087	56	3	9	87
P0208068	33	3	5	68
P0312099	48	3	9	99
P0412069	40	4	8	69
P0411092	59	4	7	92
P0414111	81	4	10	111

Table 5.10 Source Nodes & Product Types Information for Data Set 5

Problem Number	**Product Types**	**[Source Nodes]-Product Types**
P0208064	*A,B,C,D,E*	[0]-*A,B,C*; [3]-*D,E*
P0210053	*A,B,C,D,E*	[0]-*A,B,E*; [5]-*C,D*
P0209097	*A,B,C,D*	[0]-*A,C,D*; [5]-*B*
P0206046	*A,B,C*	[0]-*A,B*; [2]-*C*
P0312087	*A,B,C,D,E,F*	[0]*A,E*; [3]-*B,C*; [6]-*D,F*
P0208068	*A,B,C,D,E,F*	[0]-*A,B,C,D*; [4]-*E,F*
P0312099	*A,B,C,D,E,F,G*	[0]-*A,B,C,H*; [4]-*D,E*; [7]-*F,G*
P0412069	*A,B,C,D,E,F,G*	[0]-*A*; [2]-*B,C,D*; [5]-*E,F*; [9]-*G*
P0411092	*A,B,C,D,E,F,G*	[0]-*A,B,C*; [4]-*D*; [6]-*E,F*; [9]-*G*
P0414111	*A,B,C,D,E,F,G*	[0]-*A,B*; [3]-*C,D*; [8]-*E,F*; [10]-*G*

The solutions of problems in data set 5 are presented in Table 5.11. Since all *INPUT-F1'*s in this data set do not yield the solutions in the sequence pattern required by this study, then the best solution is selected from the minimum total time violation between *INPUT-F2* and *INPUTF3* for each problem as the best solution is indicated by an asterisk (*).

Table 5.11 Solutions for Data Set 5

Problem Number	INPUT-F1 [Sequence]-Time Violation	INPUT-F2 [Sequence]-Time Violation	INPUT-F3 [Sequence]-Time Violation
P0208064	[6^C 7^E 6^C 4^B 7^E 2^C 1^A 5^D 2^C]-50	[6^C 7^E 4^B 2^C 1^A 5^D]-55*	[6^C 4^B 7^E 2^C 1^A 5^D 2^C]-70
P0210053	[6^B 4^B 9^D 3^A 9^D 2^A 8^D 2^A 8^D 7^C 1^A 7^C]-105	[6^B 4^B 9^D 3^A 2^A 8^D 7^C 1^A]-101 *	[6^B 4^B 3^A 9^D 2^A 8^D 1^A 7^C]-104
P0209097	[7^A 6^C 8^B 6^C 8^B 4^D 8^B 4^D 3^A 2^C 1^D]-29	[7^A 6^C 8^B 4^D 3^A 2^C 1^D]-31*	[7^A 6^C 8^B 4^D 3^A 2^C 1^D]-31 *
P0206046	[5^C 4^A 5^C 4^A 5^C 4^A 3^B 1^A]-28	[5^C 4^A 3^B 1^A]-25*	[5^C 4^A 3^B 1^A]-25*
P0312087	[4^A 11^F 9^B 11^F 10^F 2^A 7^C 10^F 7^C 10^F 8^D 5^B 2^A 1^E 5^B 8^D 5^B]-140	[4^A 11^F 9^B 10^F 2^A 7^C 8^D 5^B 1^E]-180	[4^A 11^F 9^B 7^C 10^F 2^A 1^E 8^D 5^B]-136*
P0208068	[7^F 5^D 7^F 3^C 6^E 3^C 2^B 6^E 2^B 1^A]-61	[7^F 5^D 3^C 6^E 2^B 1^A]-64*	[5^D 7^F 3^C 6^E 2^B 1^A]-68
P0312099	[10^D 5^H 11^G 8^E 11^G 6^D 3^C 2^A 9^F 2^A 1^B 9^F]-79	[10^D 5^H 11^G 8^E 6^D 3^C 2^A 9^F 1^B]-83*	[10^D 5^H 8^E 11^G 6^D 3^C 2^A 9^F]-158
P0412069	[11^E 7^D 8^F 1^A 4^C 8^F 4^C 10^G 4^C 1^A 4^C 3^B 6^E 10^G 6^E]-30	[11^E 7^D 8^F 1^A 4^C 10^G 3^B 6^E]-39	[11^E 7^D 8^F 1^A 4^C 3^B 10^G 6^E]-37*
P0411092	[3^C 5^D 3^C 8^F 3^C 2^B 5^D 1^A 8^F 7^E 10^G 1^A 7^E]-34	[3^C 5^D 8^F 2^B 1^A 7^E 10^G]-45*	[3^C 2^B 5^D 8^F 10^G 1^A 7^E]-66
P0414111	[7^D 6^A 12^E 13^G 7^D 6^A 7^D 4^B 12^E 11^F 13^G 11^F 9^E 4^B 5^C 4^B 2^A 1^A 5^C 2^A]-246	[7^D 6^A 12^E 13^G 4^B 11^F 9^E 2^A 1^A 5^C]-320	[6^A 7^D 12^E 13^G 11^F 9^E 4^B 1^A 5^C 2^A]-131*

Similar to an algorithm for single-source pipeline, the relationship between computation time and number of batches is likely to be linear as shown in Figure 5.3. The computation time is likely to increase exponentially as the number of destination nodes in the pipeline grows as illustrated in Figure 5.4.

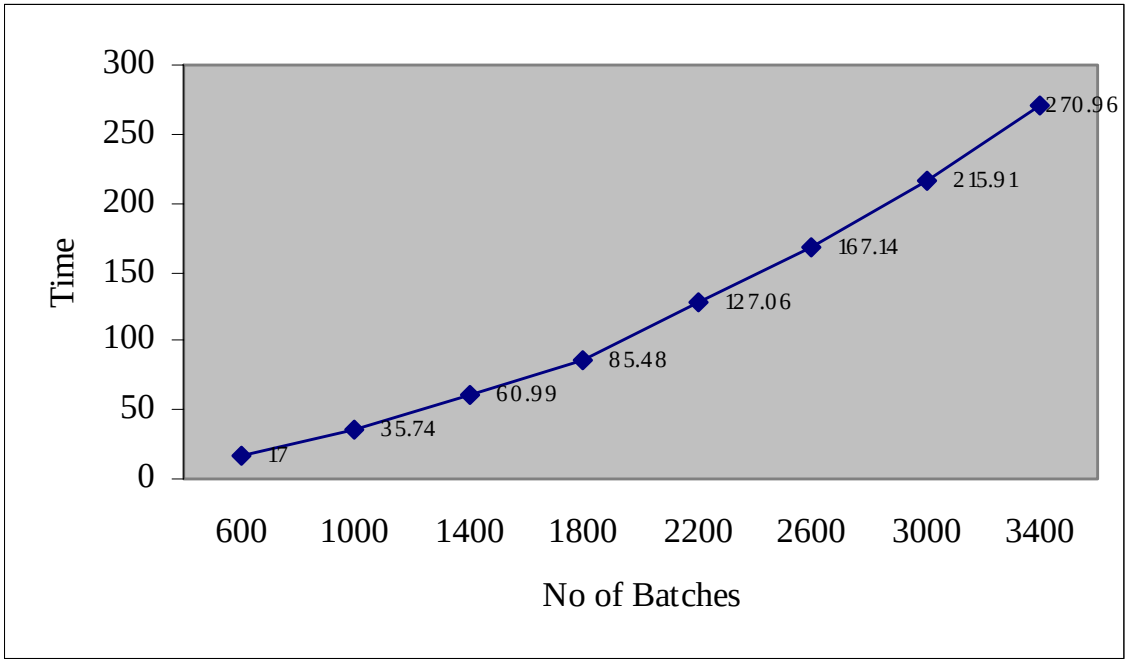


Figure 5.3 Relationship between Number of Input Batches and Computation Time for Multiple-Source Pipeline

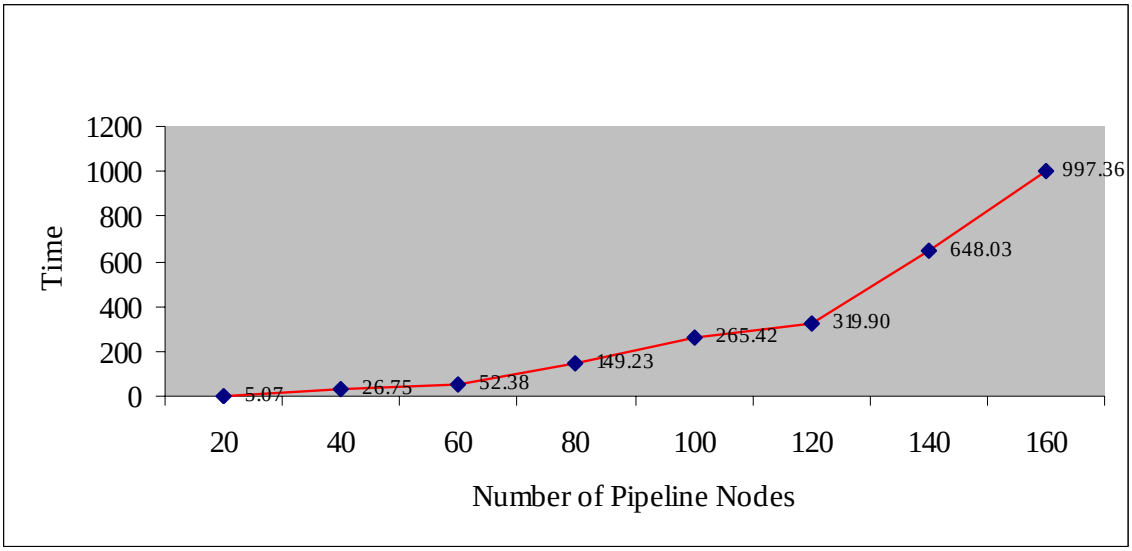


Figure 5.4 Relationship between Number of Destination Nodes and Computation Time for Multiple-Source Pipeline

CHAPTER VI

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

This chapter provides the summary of this dissertation, contributions and suggestions of the possible research works in the future.

6.1 Summary

This dissertation investigates a single-source and a multiple-source oil pipeline distribution of multiple products subject to delivery time-windows. Time-windows are generally used in scheduling and logistic problems and they are introduced to incorporate in pipeline scheduling problem. The principal objective of this dissertation is to construct a model representing the problem and find a solution methodology that calculate the pipeline input schedule that yields delivery schedule with the minimum of total time violation from due delivery time-windows. The investigation in this dissertation accentuates more on the single-source pipeline problem as it is explained in Chapter III; however, this dissertation also discusses in Chapter IV providing a preliminary study on multiple-source pipeline problem based on an extension of the study from Chapter III.

Basically, pipeline scheduling problem without delivery time-windows is a difficult one to solve. The problem is much more difficult by including the delivery

time-windows. This dissertation proved that a single-source oil pipeline distribution of multiple products subject to delivery time-windows is NP-Complete. The algorithm based on flow reversibility approach is introduced and the algorithm is tested computationally. In the same course as the single-source pipeline, the multiple-source oil pipeline distribution problem is also proved to be NP-Complete. The algorithm to solve the pipeline problem with multiple-source is also presented.

The computational results are presented in the last part of this dissertation. Sets of randomly generated problems are used to test the reversed-flow algorithms. The algorithms take running time in a matter of seconds and provide good results compared to the optimal solutions. About a quarter of all the problems tested were more than 30% greater than the optimal values and about 40% of all the tested problems were solved optimally by the algorithm.

6.2 Research Contribution

The principal contribution of this dissertation will be in introducing delivery time-window concept to the oil pipeline distribution of multiple products problem, which is realistic and practical aspects and is a challenging area to work on. The problem is also interesting and challenging due to its complexity and non-traditional nature. This dissertation has laid groundwork to build the model for pipeline scheduling problem with delivery time-windows. We explored the computational complexity of the pipeline problem and provided solution methodologies to solve the problem for both

single-source and multiple-source pipelines. It is our expectation that the effective results from this study will be crucial tools to guide and assist pipeline operators/schedulers to ease the difficulty in planning the pipeline schedule instead of using only the brute force and past experience. It is also our expectation that this research will be part of the initial exploration of using optimization tool to provide a theoretical basis for future work on this area of complicated oil pipeline problem.

6.3 Future Research Recommendations

There are numerous potentials for the future work in this area. Since we assume in this dissertation that all the pipelines in this dissertation are path, it is possible to extend the work and study the pipelines that are in different structure such as tree pipeline. Flow splitting and the study of pipeline with fungible products which are comparable to real-world situation are challenging and interesting topics to be explored in the future. A further investigation on multiple-source pipeline problem is another research direction that needs to be explored. It is encouraged to investigate different approaches to solve the problems.

REFERENCES

Ahuja, R. K., T. L. Magnanti and J. B. Orlin. (1993). *Network Flows: Theory, Algorithms, and Applications*. Upper Saddle River, NJ: Prentice-Hall.

Assad, A. A. (1978). "Multicommodity Network Flows: A Survey." *Networks* 8, 37-91.

Baker, K. R. and G. D. Scudder. (1990). "Sequencing with Earliness and Tardiness Penalties: A Review." *Operations Research* 38, 22-36.

Bazaraa, M. S., J. J. Jarvis and H. D. Sherali. (1990). *Linear Programming and Network Flows*. New York: Wiley.

Bellman, R. (2003). *Dynamic Programming*. Mineola, NY: Dover Publications.

Camacho, E. F., M. A. Ridao, J. A. Ternero and J. M. Rodriguez. (1990). Optimal Operation of Pipeline Transportation Systems. In *IFAC 11th Triennial World Congress*, pp. 455-460.

Chardaire, P. and A. Lisser. (2001). "Minimum-Cost Multicommodity Flow." In P. M. Pardalos and M. G. C. Resende (eds.), *Handbook of Applied Optimizations*, New York: Oxford University Press.

Chen, H., P. B. Luh and L. Fang. (2001). "A Time Window Based Approach for Job Shop Scheduling." In *Proceedings of the 2001 IEEE International Conference on Robotics & Automation*, pp. 842-847.

Crane, D. S., R. L. Wainwright and D. A. Schoenefeld. (1999). "Scheduling of Multi-Product Fungible Liquid Pipelines using Genetic Algorithms." In *Proceedings of*

the 1999 ACM/SIGAPP Symposium on Applied Computing (SAC'99), pp. 280-285.

Dantzig, G. B. and P. Wolfe. (1961). "The Decomposition Algorithm for Linear Programs." *Econometrica* 29, 767-777.

Eiselt, H. A. and C.-L. Sandblom. 2000. *Integer Programming and Network Models*. Berlin: Springer-Verlag.

Evans, J. R. (1978). "On Equivalent Representations of Certain Multicommodity Networks as Single Commodity Flow Problems." *Mathematical Programming* 15, 92-99.

Even, S., A. Itai and A. Shamir. (1976). "On the Complexity of Timetable and Multicommodity Flow Problems." *Siam Journal on Computing* 5, 691-703.

Gerez, J. M. and A. R. Pick. (1996). "Heavy Oil Transportation by Pipeline." In *Proceedings of the International Pipeline Conference*, pp. 699-710.

Hane, C. A. (1991). "Scheduling Multi-Products Flows in Pipelines." Ph.D. Dissertation, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA.

Hane, C. A. and H. D. Ratliff. (1995). "Sequencing Inputs to Multi-Commodity Pipelines." *Annals of Operations Research* 57, 73-101.

Hartmann, S. (1999). *Project Scheduling under Limited Resources: Models, Methods, and Applications*. Berlin: Springer-Verlag.

Held, M. and R. M. Karp. (1962). “A Dynamic Programming Approach to Sequencing Programs.” *Journal of the Society for Industrial and Applied Mathematics* 10, 196-210.

Ibaraki, T. and Y. Nakamura. (1994). “A Dynamic Programming Method for Single Machine Scheduling.” *European Journal of Operational Research* 76, 72-82.

Ioachim, I., S. Gelinas, F. Soumis and J. Desrosiers. (1998). “A Dynamic Programming Algorithm for the Shortest Path Problem with Time Windows and Linear Costs.” *Networks* 31, 193-204.

Karp, R. M. (1975). “On the Computational Complexity of Combinatorial Problems.” *Networks* 5, 45-68.

Kennington, J. L. (1978). “A Survey of Linear Cost Multicommodity Network Flows.” *Operations Research* 26, 209-236.

Lorigeon, T., J-C Billaut and J-L Bouquard. (2002). “A Dynamic Programming Algorithm for Scheduling Jobs in a Two-Machine Open Shop with an Availability Constraint.” *Journal of the Operational Research Society* 53, 1239-1246.

Mingozzi, A., L. Bianco and S. Ricciardelli. (1997). “Dynamic Programming Strategies for the Traveling Salesman Problem with Time Window and Precedence Constraints.” *Operations Research* 45, 365-377.

Minoux, M. (1989). “Network Synthesis and Optimum Network Design Problems: Models, Solution Methods and Applications.” *Network* 19, 313-360.

Paolucci, M., R. Sacile and A. Boccalatte. (2002). "Allocating Crude Oil Supply to Port and Refinery Tanks: A Simulation-Based Decision Support System." *Decision Support Systems* 33, 39-54.

Phillips, D. T. and A. Garcia-Diaz. (1990). *Fundamentals of Network Analysis*. Prospect Heights, IL: Waveland Press.

Rejowski Jr., R., and J. M. Pinto. (2002). "An MILP Formulation for the Scheduling of Multiproduct Pipeline Systems." *Brazilian Journal of Chemical Engineering* 19, 467-474.

Sasikumar, M. P. Ravi Prakash, S. M. Patil and S. Ramani. (1997). "PIPES: A Heuristic Search Model for Pipeline Schedule Generation." *Knowledge-Based Systems* 10, 169-175.

Shah, N. (1996). "Mathematical Programming Techniques for Crude Oil Scheduling." *Computers & Chemical Engineering* 20 Supp, S1227-S1232.

Trench, C. J. (2001). "How Pipelines Make the Oil Market Work-Their Networks, Operation and Regulation." Report, Association of Oil Pipe Lines, Washington, DC. Available: www.aopl.org.

VITA

Phongchai Jittamai was born in Nakhon Pathom, Thailand. He is the eldest of the four children of the family. Upon completion of high school at Triam Udom Suksa School, he attended Thammasat University where he received his Bachelor of Engineering in industrial engineering in 1995. He participated in extra-curricular activities while studying at this university including serving as Treasurer of the Engineering Students Board of Committee and the Vice President of Thammasat University Graduation Committee.

Phongchai received a scholarship awarded by the Royal Thai Government to pursue graduate studies toward a doctoral degree in the area of industrial engineering. In September 1996, he began his studies at Texas A&M University and obtained his Master of Science degree in industrial engineering in August of 1999. He continued his Ph.D. study in the same major at Texas A&M University, fulfilled the requirements of this degree and graduated in December 2004. He will go back to work as a lecturer and researcher at Suranaree University of Technology in Nakhon Ratchasima, Thailand after his graduation.

He can be reached at 63 Charunsanitwong 57/1 Bangplad, Bangkok 10700 THAILAND.