Planning the Transportation of Multiple Commodities in Bidirectional Pipeline Networks [1]

Artur Alves Pessoa

artur@inf.puc-rio.br

PUC-RioInf.MCC02/04 January, 2004

Abstract: PTP is a combinatorial model for oil pipeline transportation. It uses a directed graph G and a set of product *batches*, where each batch has a corresponding destination node. The graph G has an associated initial state where each arc contains a non-empty sequence of batches and each node contains a (possibly empty) set of batches. A valid movement is to *shift* the batches of a given arc $a = (i, j)$ in such a way that one batch from the node i is inserted in the beginning of the arc a and another batch, removed from the end of the arc a, is inserted in j. The goal is to reach a state where all batches from a given subset are stored at their destinations. For the general PTP, deciding whether or not a feasible plan exists is proved to be $\mathcal{NP}$-hard.

In this paper, we propose the CBPTP, a new variation of PTP. In this variation, arc contents may be shifted in both directions. Moreover, product batches have no destination nodes assigned. Instead, we assign a commodity to each batch. In this case, the node demands that may be satisfied by any batches of the corresponding commodities. For the general CBPTP, we reduce the instance feasibility to the problem of finding a perfect matching in a bipartite graph. Moreover, we give a polynomial algorithm for constructing feasible plans for the CBPTP, when they exist.

Keywords: Petroleum, Transportation, Oil pipeline, Planning, Complexity

Resumo: PTP é um modelo combinatório para o transporte dutoviário. Este modelo usa um grafo dirigido G e um conjunto de bateladas de produto, onde cada batelada tem um nó de destino associado. O grafo G tem um estado inicial onde cada arco contém a seqüência não-vazia de bateladas e cada nó, um conjunto (possivelmente vazio) de bateladas. Um movimento válido consiste em deslocar as bateladas de um dado arco $a = (i, j)$ de forma que um batelada do nó i seja inserida no início do arco a e uma outra batelada, que é removida do final do arco a, seja inserida em j. O objetivo é atingir um estado onde todas as bateladas de um dado subconjunto estejam nos respectivos nós de destino. Para o PTP genérico, demonstra-se que decidir se existe um planejamento viável ou não é $\mathcal{NP}$-difícil.

Neste artigo, propomos o CBPTP, uma nova variação do PTP. Nesta variação, o conteúdo de um arco pode ser deslocado em ambos os sentidos. Além disso, as bateladas não têm nós de destino associados. Ao invés disso, associamos uma *commodity* a cada batelada. Neste caso, as demandas dos nós podem ser satisfeitas por quaisquer bateladas associadas às *commodities* correspondentes. Para o CBPTP genérico, reduzimos a viabilidade de uma instância ao problema de encontrar um pareamento perfeito num grafo bipartido. Além disso, apresentamos um algoritmo polinomial para a construção de planejamentos viáveis para o CBPTP, quando eles existem.

Palavras-Chave: Petróleo, Transporte, Oleoduto, Planejamento, Complexidade

[1] Financiado pelo CNPq, Bolsa DTI, Projeto 55.2046/2002/7.

1 Introduction

Pipelines play an important role in the transportation of petroleum and its derivatives, since it is the most effective way to transport large volumes over large distances. However, we observe that both the expenses involved in constructing oil pipelines and the energy spent to pump products in them are huge. On the other hand, planning the transportation of petroleum products through a relatively small pipeline network may be a very hard task [6, 8, 10, 11, 7]. As a result, large amounts of money and energy are lost due to the limitations of the existing software tools to do such a planning, which may significantly affect the cost of petroleum derivatives. This scenario makes transportation through oil pipeline networks a problem of high relevance.

The pipeline transportation problem has an unique characteristic, which distinguishes it from other transportation methods: it uses stationary carriers whose (liquid) cargo moves rather than the more usual moving carriers of stationary cargo. Typically, each oil pipeline is a few inches wide and several miles long. As a result, reasonable amounts of distinct products can be transported through the same pipe with a very small loss due to mixing at liquid boundaries. Another important characteristic of oil pipelines is that they must always, for safety reasons, be pressurized. That is, each pipeline must be completely full of liquid. Hence, assuming incompressible fluids, an elementary pipeline operation is the following: pump an amount of product into the pipeline and remove the same amount of product from the opposite side. Observe that the product inserted in one side is not necessarily the same as the one removed from the other side, as shown in Figure 1.(a). This figure shows the content of an oil pipeline that is filled by the two (liquid) products B and A, from left to right. In this case, to insert an amount of the product C in the left side, one must remove the same amount of A from the right side. As a result, product B is *shifted to the right*, inside the oil pipeline. Figure 1.(b) shows the content of the same pipeline after this operation.

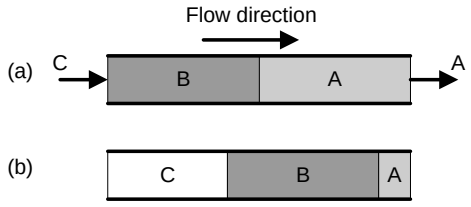

Figure 1: (a) the content of an oil pipeline; (b) the content of the same pipeline after inserting an amount of product C.

Throughout this paper, we use the term batch to denote a contiguous amount of some product in a pipeline network. This term is commonly used in the context of oil pipelines.

1.1 Previous Work

The problem of planning the transportation through oil pipeline networks involves both continuous and discrete information. For example, we have continuous flow rates, batch volumes and time periods. On the other hand, flow directions, product routes and precedencies are discrete decisions. Due to this fact, modelling the problem as Mixed Integer Programming Problem (MIP) seems to be a natural approach. However, previous attempts to do it did not lead to robust solutions to the problem [5, 2, 3, 1]. We point out that the unusual characteristics of this kind of transportation lead to loose LP relaxations, which has a negative impact on the efficiency of MIP methods. Thus, there is an evident need for research on the topic.

The idea to study simplified combinational models for the pipeline transportation problem was introduced by Hane and Ratliff [6]. The authors propose a model under the following assumptions: demands cyclically repeat over the time, the network graph is a directed tree, batches have given destination nodes and no due dates.

The Pipeline Transportation Problem (PTP) is proposed in [9, 10] for general graphs and non-cyclic orders. PTP models a pipeline system through a directed graph G, where each node represents a location and each directed arc represents a pipe, with a corresponding flow direction. As in Hane's model, the flow inside each pipe is assumed to be unidirectional. PTP also defines a set of batches with given destination nodes. The graph G has an associated initial state where each arc contains a non-empty sequence of batches and each node contains a (possibly empty) set of batches. All batches have unitary volumes. Since pipes must always be full, some batches must be used to fill them at the end of the transportation. These batches are not delivered. Due to this fact, PTP defines a subset of batches, called *further* batches, that are not necessarily delivered at the end of a feasible pumping sequence. In this case, a feasible solution to PTP is a pumping sequence that delivers all *non-further* batches.

In [10], the problem of finding a feasible solution to PTP is proved to be $\mathcal{NP}$-hard, even if G is acyclic. Moreover, the authors introduce the Synchronous PTP (SPTP), a special case of PTP where all further batches are initially stored at nodes. The problem of finding a minimum pumping cost solution to SPTP is called SPTOP. In the same work, the authors also introduce the BPA algorithm, that finds feasible solutions to SPTP in

polynomial time. If G is acyclic, then these solutions are also optimal for the SPTOP. The authors also analyze the complexity of finding minimum makespan solution to the SPTP in [11]. This problem is called SPTMP. They prove that, for any fixed $\epsilon > 0$, there is no $\eta^{1-\epsilon}$-approximate algorithm for SPTMP unless $\mathcal{P} = \mathcal{NP}$, where η is the input size. This result also holds if the graph G is both planar and acyclic.

Recently, another combinatorial model based on oil pipelines, called Pipesworld, was proposed for a solver competition [7]. The authors also describe many interesting examples of difficulties introduced by the Pipesworld model. We point out that the Pipesworld model can be viewed as a variation of the PTP model where pipe contents can flow in both directions, nodes have capacity constraints and consecutive batches in pipes have compatibility restrictions.

1.2 The Problem

In this paper, we address a variation of PTP for planning the transportation of multiple commodities in bidirectional pipeline networks. This variation is motivated by both the Pipesworld model and the existing Brazilian oil pipeline networks. We refer to our model as the Commodity Bidirectional PTP (CBPTP). As in the Pipesworld model, CBPTP allows pipe contents to flow in both directions. On the other hand, product batches have no capacity constraints or compatibility restrictions. Moreover, instead of assigning destination nodes to the batches, we use a more general model. Each batch has an associated commodity, and each node i in G has an associated demand $d(i, c)$ for each commodity c. In this case, a feasible solution to the CBPTP is a pumping sequence that delivers at least $d(i, c)$ batches of the commodity c at the node i, for every node i and commodity c. Hence, the model must also determine a destination node for each batch. Observe that we may model the case where the batches' destination nodes are given as a special case of the CBPTP. Let us refer to this last case as the Assigned Bidirectional PTP (ABPTP).

We point out that, since all pipes can flow in both directions, the arc directions are merely a convention to represent the pipe contents. Hence, we define an undirected graph $\bar{G}$ as the graph obtained from G by replacing all its directed arcs with the corresponding (undirected) edges. For the sake of simplicity, let us assume that G has no parallel arcs (even with different directions).

1.3 Results

For the general CBPTP, we reduce the instance feasibility to the problem of finding a perfect matching in a bipartite graph. Moreover, we give a polynomial algorithm for constructing feasible plans for the CBPTP, when they exist.

Our approach to solve the CBPTP is the following. First, we obtain necessary and sufficient feasibility conditions for the ABPTP. Later, we observe that, given a feasible instance I of the CBPTP, we can use the previous conditions to assign destination nodes to the batches of I so that the resulting instance of the ABPTP is feasible. For that, we show that such assignment must correspond to a perfect matching in a bipartite graph.

To solve the ABPTP, we start with a solution for the special case where the graph G has a single arc. This solution is generalized to *handle* the bridges of the graph $\bar{G}$. By handling a bridge e, we mean moving to a state where every non-further batch can reach its destination node without passing by the pipe that corresponds to e. As a result, handling a bridge involves placing all batches that must cross the corresponding pipe at the correct side. Then, we use a decomposition of the graph $\bar{G}$ into bridges and 2-edge-connected components [4]. Whenever the current graph $\bar{G}$ has at least one bridge, we solve the problem as follows: choose a bridge e on $\bar{G}$; handle e, and recursively solve the subproblems that correspond to the two subgraphs of $\bar{G}$ connected by e. Here, we assume that the graph $\bar{G}$ is initially connected. For the 2-edge-connected components of $\bar{G}$, we assign an unique flow direction for each corresponding pipe so that the resulting directed graph is strongly connected. After that, we solve the corresponding subproblem through a generalization of the approach introduced in [10].

1.4 Paper Organization

This paper is organized as follows. In Section 2, we formalize the CBPTP model and introduce some additional notations. In Section 3, we describe our polynomial time planning algorithm. In the last section, we present our final remarks.

2 The CBPTP model

In this section, we describe the CBPTP model and introduce some additional notations for the ABPTP. Our description includes its pipeline system, orders, pipe contents, and allowed operations.

Pipeline System

Let $G = (N, A)$ be a directed graph, where N is the set of n nodes and A is the set of m arcs. Given an arc $a = (i, j) \in A$, we say that i is the start node of a and j is the end node of a. Arcs represent pipes and nodes represent locations. Each arc $a \in A$ has an associated integer capacity $v(a)$. Moreover, we divide each arc a into $v(a)$ pipe positions. We also define the set of all pipe positions $A' = \{(a, l) | a \in A \text{ and } l \in \{1, \ldots, v(a)\}\}$.

Orders

Let L be a set of r unitary volume batches, and C a set of commodities. Each $b \in L$ has an associated commodity $c(b)$. Moreover, each node $i \in N$ has an associated integer demand $d(i, c)$ for each commodity $c \in C$.

For the ABPTP, each batch corresponds to a transportation order which is a commitment to deliver b at $f(b) \in N \cup \{\phi\}$. Each batch b with $f(b) = \phi$ is called a *further* batch. These batches have no destination node assigned and may be used to fill the pipes at the end of the plan. Other batches are called *non-further* batches.

Pipe Contents

Pumping a batch into a pipe requires a non negligible amount of time. However, we only consider the instants where each arc $a \in A$ contains exactly $v(a)$ integral batches. As a result, any solution to this model generates a discrete sequence of states, where the positions of all batches are well-defined.

Let us use $p_t(b)$ to denote the position of batch b at state t. If $p_t(b) = (a, l) \in A'$, then batch b is located at the lth position of arc a at state t. Otherwise, if $p_t(b) = i \in N$, then batch b is stored at node i. Furthermore, the content of a given arc a at a given state t is represented by a list of batches $[b_1, b_2, \ldots, b_{v(a)}]$. If the previous list represents the content of a when the state is t, then we have $p_t(b_l) = (a, l)$, for $l = 1, 2, \ldots, v(a)$.

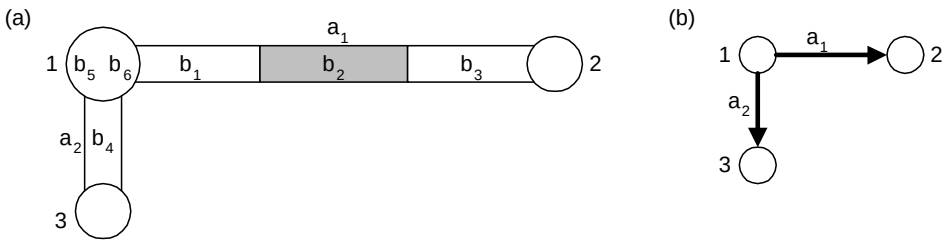

Figure 2: (a) The contents of a pipeline system; (b) the corresponding graph.

As an example, Figure 2.(a) represents the pipe contents corresponding to the graph of Figure 2.(b). Observe that the system has two pipes $a_1 = (1, 2)$ and $a_2 = (1, 3)$. The capacities of a_1 and a_2 are $v(a_1) = 3$ and $v(a_2) = 1$, respectively. Let us assume that Figure 2.(a) corresponds to state t. In this case, we have $p_t(b_1) = (a_1, 1)$, $p_t(b_2) = (a_1, 2)$, $p_t(b_3) = (a_1, 3)$, and $p_t(b_4) = (a_2, 1)$, since the contents of a_1 and a_2 are respectively represented by the lists $[b_1, b_2, b_3]$ and $[b_4]$. Furthermore, we have $p_t(b_5) = p_t(b_6) = 1$ since both b_5 and b_6 are stored at node 1.

At the initial state (state 0), the position $p_0(b)$ of each batch b is given.

Operations

A solution for the model is a list Q of elementary pipeline operations (EPO), defined as follows. Let $a = (i, j)$ be an arc of G, whose content at a given state t is given by the list $[b_1, b_2, \ldots, b_{v(a)}]$. Moreover, let b be a batch stored either at node i or at node j, in this moment. An EPO is a pair (b, a), denoting that b is pumped into a. Suppose that (b, a) is executed after the state t. If b was previously stored at node i, then we refer to this EPO as a *direct* EPO. Otherwise, if b was previously stored at node j, then we refer to this EPO as a *reverse* EPO. In the case of a direct EPO, the contents of a at state $t+1$ are given by the list $[b, b_1, b_2, \ldots, b_{v(a)-1}]$ and $b_{v(a)}$ is stored at the node j. If a reverse EPO was performed, then the contents of a at state $t + 1$ are given by the list $[b_2, b_3, \ldots, b_{v(a)}, b]$ and b_1 is stored at the node i. Let q be the size of a given sequence Q of EPO's. We say that Q is feasible when, for every node $i \in N$ and every commodity $c \in C$, we have at least $d(i, c)$ batches associated to commodity c stored at node i, when the state is q.

For the ABPTP, we say that Q is feasible when every batch b with $f(b) \neq \phi$ is stored at the node $f(b)$, when the state is q.

3 Planning algorithm

In this section, we describe a polynomial algorithm that solves the CBPTP. Given an instance I of the CBPTP, our algorithm either finds a feasible solution I or determines that I is infeasible. Let us refer to this algorithm as the Bridge algorithm, which is motivated by the fact that handling the bridges of the graph $\bar{G}$ is the most difficult part.

As mentioned in the introduction, we assign an unique flow direction for each pipe of a 2-edge-connected component of $\bar{G}$ so that the resulting directed subgraph is strongly connected. In [12], it is shown that such directions can be assigned for any 2-edge-connected undirected graph. In fact, these directions can be obtained in a linear time through a Depth

First Search. For the sake of simplicity, let us assume without loss of generality that the assigned flow directions coincide with arc orientations of G. Moreover, let us refer to any arc of G that corresponds to bridge in $\bar{G}$ as a bridge also.

This section is organized as follows. In Subsection 3.1, we introduce the key concept of excess. In Subsection 3.2, we give our feasibility conditions for the ABPTP. In Subsection 3.3, we use these conditions to propose a reduction from the CBPTP to the ABPTP. In Subsection 3.4, we describe the One-pipe Procedure, that solves networks with a single pipe. In Subsection 3.5, we describe the Selection Procedure. This procedure is used in Subsection 3.6 to extend the One-pipe Procedure to handle bridges in general networks. Finally, in Subsection 3.7, we show how to solve the case where G has no bridge.

3.1 Network Excess

Now, we introduce the concept of excess, which is extensively used throughout this section.

Definition 1 *Given a connected subgraph $G' = (N', A')$ of G, the excess of G' at a given state t is defined as the number of batches stored at the nodes of G', that is, this excess is given by $x_t(G') = |\{b \in L | p_t(b) \in N'\}|$.*

Observe that no EPO can change the excess of G. As a result, this excess is constant over the states generated by any plan. Hence, we define $X = x_0(G) = x_1(G) = \cdots = x_q(G)$, for any plan Q of size q.

Throughout this paper, we may also use the term excess to refer to a set of batches stored at a node. In this case, given an arc $a = (i, j)$, if k batches of i are pumped into a, then we may say that an excess of size k is moved from i to j. Note that the batches received by j may be completely different from the batches pumped from i. Observe also that we can move an excess from j to i, executing reverse EPO's on a.

Now, let us consider a path in G (possibly containing both direct and reverse arcs) that starts at a node i_0 and visits the nodes $i_1, i_2, \ldots, i_l$ in this order. In this case, an excess of size k may be moved from i_0 to i_l as follows: move it from i_0 to i_1, then move it from i_1 to i_2 and so on until i_l receives an excess of size k. Observe that, since G is connected, any excess can be moved from any node to any node in G.

3.2 Feasibility Conditions for the ABPTP

We have already mentioned that we solve the CBPTP through a reduction to the ABPTP. Here, we introduce a necessary and sufficient set of conditions for the feasibility of the

ABPTP. Although the necessity is easy to prove, the sufficiency is only obtained through a planning algorithm. Hence, we start stating these conditions only as necessary conditions. For that, let us introduce some additional definitions.

Given a bridge $a = (i, j)$ of G, let us define $S(a, G)$ and $S'(a, G)$ as the two subgraphs of G connected by a, where i and j are nodes of $S(a, G)$ and $S'(a, G)$, respectively. In this case, we have the following definitions.

Definition 2 *When the state is t, the position of a batch b with respect to a is given by*

$$y_t(b,a) = \begin{cases} l & , \text{ if } p_t(b) = (a, l); \\ 0 & , \text{ if } p_t(b) \text{ is a node of } S(a, G); \\ v(a) + 1 & , \text{ if } p_t(b) \text{ is a node of } S'(a, G). \end{cases}$$

Definition 3 *For $t = 0, 1, \ldots, q$, we define $w'_t(a) = v(a) - x_t(S(a, G)) + 1$ and $w_t(a) = x_t(S'(a, G))$.*

Roughly speaking, if $v(a) \geq X$, then $w'_t(a)$ ($w_t(a)$) is the minimum (maximum) position of a batch b w.r.t. a, when the state is t, such that b can be delivered at a node of $S'(a, G)$ ($S(a, G)$). As we show later, any batch can cross any bridge a such that $v(a) < X$. On the other hand, the next lemma shows that this is not true for bridges with $v(a) \geq X$.

Lemma 1 *If $v(a) \geq X$ and $y_0(b, a) < w'_0(a)$ ($y_0(b, a) > w_0(a)$), then no plan can delivery b at a node of $S'(a, G)$ ($S(a, G)$).*

Proof: We only prove that no plan can delivery b at a node of $S'(a, G)$, when we have both $v(a) \geq X$ and $y_0(b, a) < w'_0(a)$. The other case is symmetric.

To prove this lemma, we must show that, for any plan Q and $t = 0, 1, \ldots, q$, we have $y_t(b, a) < w'_t(a)$ when the state is t. In this case, since $x_t(S(a, G)) \geq 0$, we have by definition of $w'_t(a)$ that $w'_t(a) \leq v(a) + 1$. As a result, we will conclude the $y_t(b, a) \leq v(a)$, which implies that b never reaches $S'(a, G)$.

We prove that $y_t(b, a) < w'_t(a)$ by induction on the value of t. For $t = 0$, we obtain the hypotheses of this lemma. Now, let as assume that it holds for $t = t'$. We shall prove that it also holds for $t = t' + 1$. If no batch is pumped into a, then neither $y_t(b, a)$ nor $w'_t(a)$ changes when t increases from t' to $t' + 1$, and we are done. On the other hand, if a batch b of $S(a, G)$ is pumped into a, then $w'_t(a)$ increases by one unit because $x_t(S(a, G))$ decreases by the same amount. Since $y_t(b, a)$ increases by at most one unit, we are also done in this case. Finally, if a batch b of $S'(a, G)$ is pumped into a, then $w'_t(a)$ decreases

by one unit because $x_t(S(a,G))$ increases by the same amount. Moreover, either $y_t(b,a)$ decreases by one unit or $y_{t'+1}(b,a) = 0$. In the first case, we are done. For the second case, Since $v(a) \geq X \geq x_{t'+1}(S(a,G))$, we have by definition of $w'_t(a)$ that $w'_{t'+1}(a) \geq 1$. Hence, we obtain that $y_{t'+1}(b,a) < w'_{t'+1}(a)$, and the proof is done. ■

Observe that Lemma 1 leads the following necessary feasibility condition associated to each non-further batch $b \in L$: for every bridge a of G, if $y_0(b,a) < w'_0(a)$ $(y_0(b,a) > w_0(a))$ then $f(b)$ must be a node of $S(a,G)$ $(S'(a,G))$. Moreover, since we must have enough further batches to fill all the pipes, we have the following fact.

Fact 1 *A feasible instance of the ABPTP has at least as many further batches as the total number pipe positions in the network.*

Now, we are ready to state the following theorem.

Theorem 1 *Given an instance I of the ABPTP, I is feasible if and only if it satisfies the following two conditions:*

(i) for every non-further batch $b \in L$ and every bridge a of G, if $y_0(b,a) < w'_0(a)$ $(y_0(b,a) > w_0(a))$ then $f(b)$ is a node of $S(a,G)$ $(S'(a,G))$;

(ii) $|A'| \leq |\{b \in L | f(b) = \phi\}|$.

Proof: The necessity to satisfy these conditions is a consequence of both Lemma 1 and Fact 1. On the other hand, the sufficiency is only proved by the algorithm described throughout this section. ■

It is important to note that the condition (ii) of the previous theorem is simpler than the condition (i) because, once we have enough further batches to fill all pipes, Lemma 1 cannot impose a restriction that prevents this filling. For example, if the condition (ii) is satisfied but all further batches are stored in $S(a,G)$ and we have pipes in $S'(a,G)$, then we must have $v(a) < X$.

3.3 Reduction from CBPTP to ABPTP

Here, we show how to use a polynomial algorithm for the ABPTP to solve the CBPTP in polynomial time. Clearly, we can transform any instance I of the CBPTP in an instance I' of the ABPTP by assigning destination nodes to all batches of I so that all demands of I are satisfied. However, we only have a reduction if such an assignment maintains the

feasibility of I, that is, every feasible instance I leads to a feasible instance I'. Next, we describe our reduction.

First, we use the conditions of Theorem 1 to obtain a bipartite graph H that represents all possible destinations for each batch of I. This graph is constructed as follows:

Step 1:	Insert one vertex in the first partition for each batch of I.
Step 2:	Insert $d(i,c)$ vertices in the second partition for each pair $(i,c) \in N \times C$.
Step 3:	Insert additional vertices in the second partition until we have at least r vertices.
Step 4:	Insert one edge connecting every vertex of a batch b to every vertex of a pair (i,c) such that $c = c(b)$ and assigning $f(b) = i$ does not violate the condition (i) of Theorem 1.
Step 5:	Insert one edge connecting every vertex of a batch to every additional vertex.

Then, we find a perfect matching in H. If such a matching does not exist, then I in infeasible because we have no sufficient batches to satisfy all demands. If the number of additional nodes is smaller that $|A'|$, then I is infeasible because any destination assignment that satisfies all demands violates the condition (ii) of Theorem 1. Otherwise, I is feasible, and I' is constructed as follows:

Step 1:	For every edge in the matching that connects the vertex of a batch b to the vertex of a pair (i,c), assign $f(b) = i$.
Step 2:	For every edge in the matching that connects the vertex of a batch b to an additional vertex, assign $f(b) = \phi$.

Observe that, by construction of H, the previous destination assignment does not violate the conditions of Theorem 1, which completes our reduction.

3.4 One-pipe Networks

Here, describe our procedure to *handle* a bridge $a = (i,j)$ of G, assuming that a is the only arc of G. Let us refer to this procedure as the One-pipe Procedure. Later, in Subsection 3.6, we show how to generalize our solution to any network graph. As we previously define, to handle a bridge a, we must move the pipeline network to a state where every non-further batch b such that $f(b)$ is a node of $S(a,G)$ ($S'(a,G)$) is already stored in this subgraph. In our case, the two subgraphs are single nodes.

We divide in to cases:

(i) $v(a) \geq X$;

(ii) $v(a) < X$.

In the case (i), a can be handled in four steps:

One-pipe Procedure – Case (i)	
Step 1:	Pump all batches from j into a.
Step 2:	Pump all further batches from i into a.
Step 3:	Pump all non-further batches from i into a.
Step 4:	Pump all further batches from j into a.

Figure 3 illustrates the execution of these steps. Let b be a batch such that $w_0(a) < y_0(b,a) < w_0'(a)$ (if it exists). By Lemma 1, b cannot leave the pipe a. Hence, if we assume that the instance is feasible, then b must be a further batch. Figure 3.(a) shows a general representation for the initial content of a. Figures 3.(b),3.(c),3.(d) and 3.(e) respectively represent this content after the executions of Steps 1, 2, 3 and 4. In these figures, the white color represent batches that must be further, the light gray color represent batches that may be further or not, and the dark gray color represent non-further batches. Moreover, the rectangle that corresponds to the batches that cannot leave the pipe has a bold printed border. Finally, each arrow between two content representations indicates the flow direction of the EPO's executed between the two corresponding states.

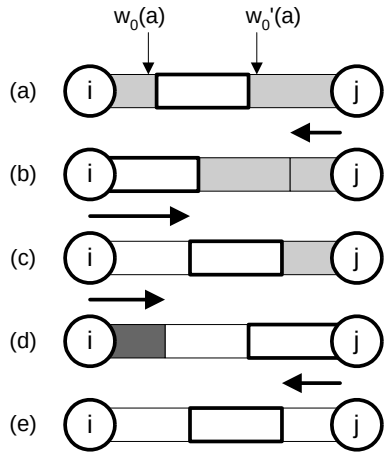

Figure 3: A feasible solution to a network with only one pipe a such that $v(a) \geq X$.

Observe that, if there are enough further batches to fill a, then a is filled with only further batches after the Step 4. Since, by Lemma 1, no batch can cross a, we conclude

also that all non-further batches are properly delivered after this step (if the instance is feasible).

Now, let us consider the case (ii). The crucial observation is that, if all excess of G is stored at the node i, then the first batch b pumped from i into a can reach the node j. This is true because the excess of G minus one is sufficient to fill a. On the other hand, we may have to pump all the batches of i into a to get b stored at j. Analogously, if all excess of G is stored at the node j, then the first batch pumped from j into a can reach the node i. This observations suggest an yo-yo like procedure to handle bridges. This procedure performs the following steps:

One-pipe Procedure – Case (ii)	
Step 1:	Pump all batches from j into a.
Step 2:	Pump into a every non-further batch b from i such that $f(b) = j$.
Step 3:	Pump all further batches from i into a.
Step 4:	Pump all remaining non-further batches from i into a.
Step 5:	Pump into a every non-further batch b from j such that $f(b) = i$.
Step 6:	Pump all further batches from j into a.
Step 7:	If there is a non-further batch b not stored at $f(b)$ then go to Step 1.

Observe that, the non-further batches stored at the corresponding destination nodes are the last pumped into a. As a result, since we must have enough further batches to fill a, every non-further batch b that is stored at $f(b)$ when executing the Step 7 returns to $f(b)$ before the next execution of this step. Moreover, due to the previous observations, we have that the number of non-further batches stored at the corresponding destination nodes increases on each execution of the Step 7. Hence, the condition of Step 7 is satisfied after at most r iterations.

3.5 Selecting Batches from Subgraphs

Let $a = (i, j)$ be a bridge of G. If a is the only pipe of G, then we can select any batch from i or j and pump it into a. Here, we describe a method to select a batch b either from $S(a, G)$ or from $S'(a, G)$ and pump it into a, where G represents any pipeline network. Let us refer to this method as the Selection Procedure.

Before describing the Selection Procedure, let us introduce the Cycling Procedure, proposed in [10]. Given an arc $a' = (i', j')$ of a Strongly Connected Component (SCC) of G and a batch b stored at i', this procedure transports b to the node j'. We point out that this

procedure does not generate reverse EPO's since this is a restriction of the model used in [10]. To make this paper self contained, let us briefly describe this procedure. It performs the following steps:

Cycling Procedure	
Step 1:	Find a directed path $\boldsymbol{p}$ from j' to i' (it always exists in a SCC).
Step 2:	Pump b into a' (this moves an excess of size 1 to j').
Step 3:	Move an excess of size 1 from j' to i' through the path $\boldsymbol{p}$.
Step 4:	Pump a batch from i' into a' (this also moves an excess of size 1 to j').
Step 5:	If b is not yet stored at j', then go back to Step 3.

We shall describe the Selection Procedure only for the case where b is selected from $S(a, G)$, since the other case is analogous. This procedure uses the following three assumptions:

1. $v(a)$ is maximum over all bridges of G;
2. $x_t(S(a, G)) > v(a)$, where t is the state at which b is selected;
3. all the excess of $S(a, G)$ is stored at i;

Later, we show how to satisfy these assumptions when solving general pipeline networks.

Next, we describe the Selection Procedure. If b is stored at i, then just pump b into a and we are done. Since, by the assumption 3, no batch in $S(a, G)$ is stored at a node other than i, in the only remaining case, b is contained in a pipe $a' = (i', j')$. In this case, we first show how to move b to a node. If a' is a bridge, then, by the assumption 1, $v(a') \leq v(a)$. In this case, it is enough to move an excess of size $v(a)$ through a'. Otherwise, if a' is not a bridge, then we start by moving an excess of size 1 to i'. In this case, we observe that the Cycling Procedure can be modified to transport b to j'. For that, we only need to remove the Step 2 (since b is already contained in a').

Now, b is already stored at a node. If this node is i, then just pump b into a and we are done. Hence, let us assume that b is stored at a node $k \neq i$. In this case, find a path $\boldsymbol{p}$ in G from k to i, where the bridges (and only them) may be reverse arcs. Then, move an excess from i to k so that k has an excess of size $v(a) + 1$ (including the batch b). After that, transport b together with an excess of size $v(a)$ through each arc a' of $\boldsymbol{p}$ as follows. If a' is a bridge, then, by the assumption 1, $v(a') \leq v(a)$. In this case, it is enough to pump into a' the batch b followed by $v(a)$ batches. Otherwise, if $a' = (i', j')$ is not a bridge, then $v(a')$ may be greater than $v(a)$. Hence, we use the Cycling Procedure to transport b from

i' to j'. After that, we also move an excess of size $v(a)$ from i' to j'. After transporting b through all arcs of $\boldsymbol{p}$, we have b stored at the node i. Then, we just pump b into a.

3.6 Handling Bridges

Finally, we describe the Bridge Procedure. Given an input graph G, the Bridge Procedure selects a bridge $a = (i, j)$ that maximizes $v(a)$ in G, handle it and recursively solve the two subproblems that correspond to $S(a, G)$ and $S'(a, G)$. If G has no bridge, then, by the choice of the arc directions, G is strongly connected. In this case, the SCC Procedure, that we introduce in the next subsection, is called to solve the current problem with no recursion.

Before describing the Bridge Procedure, we introduce the concept of *rolling back* EPO's. Let (b, a) be a direct EPO executed when the content of a is given by $[b_1, b_2, \ldots, b_{v(a)}]$, for $a = (i, j)$. After this EPO, the content of a changes to $[b, b_1, b_2, \ldots, b_{v(a)-1}]$, and $b_{v(a)}$ is stored at the node j. In this case, we may *roll back* the previous EPO by executing the reverse EPO $(b_{v(a)}, a)$, which changes the state of the network back to the state that precedes the execution of (b, a). Observe that reverse EPO's may also be rolled back by executing the corresponding direct EPO's. We also observe that any sequence of EPO's may be rolled back. For that, we must roll back each EPO of the sequence in the reverse order of their execution.

The Bridge Procedure is a generalization of the One-pipe Procedure that uses the Selection Procedure whenever it is necessary to pump into a a batch that is not stored at a node adjacent to a. We point out that, in the generalized method, the recursive call to solve the two generated subproblems occurs while handling a, not after that. This is necessary because the Selection Procedure can only be used if the corresponding subgraph has an excess of size greater than $v(a)$.

The most important observation to develop the Bridge Procedure is the following. For a given iteration of the One-pipe Procedure – Case (ii), if this iteration is not the last one, then the last $v(a)$ batches pumped from i during the Steps 2, 3, and 4 will return to the node i. This is true because, since $X > v(a)$, more than $v(a)$ batches will be pumped from j before the procedure halts. The same observation is also valid for the last $v(a)$ batches pumped from j during the Steps 5 and 6, and the Step 1 of the next iteration. As a result, these batches may be replaced by any available batches, that is, we do not need to call the Selection Procedure before these EPO's. This observation assures that we satisfy the assumption 2 of the Selection Procedure when generalizing the One-pipe Procedure, except

for both Case (i) and the last iteration of Case (ii). To avoid calling the Selection Procedure in these exceptional cases, we recursively call the Bridge Procedure just before they occur. In this case, the subproblem that correspond to $S(a, G)$ ($S'(a, G)$) is slightly modified so that the batches that shall be pumped into a are destined to the node $i(j)$. Consequently, when the procedure returns from this recursive call, these batches are already stored at a node adjacent to a, and the Selection Procedure no longer needs to be called. Then, the original destination nodes of the batches are restored. For simplicity, we omit these details in the following pseudocodes.

The Bridge Procedure performs the following steps for Case (i):

Bridge Procedure – Case (i)	
Step 1:	Pump all excess from $S'(a, G)$ into a.
Step 2:	Recursively call the Bridge Procedure to solve the subproblem that corresponds to $S(a, G)$.
Step 3:	Pump all further batches from i into a.
Step 4:	Pump all remaining excess from $S(a, G)$ into a.
Step 5:	Recursively call the Bridge Procedure to solve the subproblem that corresponds to $S'(a, G)$.
Step 6:	Pump all further batches from j into a.
Step 7:	Roll back the EPO's of Step 4, except the pumpings into a.

The steps performed for Case (ii) are described by the pseudocode of Figure 4.

Observe that we have a correspondence between the steps of the Bridge Procedure – Case(i) (except for Steps 2, 5 and 7) and that of the One-pipe Procedure – Case (i). Moreover, we also have a correspondence between the steps of Figure 4 (except for Steps 2, 6 and 10) and that of the One-pipe Procedure – Case (ii). In Figure 4, the condition "if this can be the last iteration" is a simplified statement for a more complex condition. This condition is satisfied when, assuming that any batch can be pumped into a (which is not actually true due to the limitations of the Selection Procedure), the condition of Step 9 will be satisfied in the current iteration. The aim of this condition is to assure that only one recursive call is issued for each generated subgraph of G.

Next, we briefly discuss the correctness of the bridge procedure. Assuming that the main problem is feasible, we show that the generated subproblems are also feasible. First, we assure that the condition (i) of Theorem 1 holds for the subproblems whenever it holds for the main problem. Observe that in both cases (i) and (ii) of the Bridge Procedure, the recursive call to solve each subproblem occurs at a moment where all the excess of G is

Bridge Procedure – Case (ii)	
Step 1:	Pump all the excess from $S'(a, G)$ into a.
Step 2:	If this can be the last iteration, then recursively call the Bridge Procedure to solve the subproblem that corresponds to $S(a, G)$.
Step 3:	While there is a non-further batch b from $S(a, G)$ such that $f(b)$ is a node in $S'(a, G)$, call the Selection Procedure to transport b to i (if necessary) and pump it into a.
Step 4:	While the number of further batches in both a and $S'(a, G)$ is not sufficient to fill both a and the pipes of $S'(a, G)$, and a further batch b either is stored at i or can be transported to it through the Selection Procedure, then call the Selection Procedure to transport b to i (if necessary) and pump it into a.
Step 5:	Pump all the remaining excess from $S(a, G)$ into a.
Step 6:	If this can be the last iteration, then recursively call the Bridge Procedure to solve the subproblem that corresponds to $S'(a, G)$.
Step 7:	While there is a non-further batch b from $S'(a, G)$ such that $f(b)$ is a node in $S(a, G)$, call the Selection Procedure to transport b to j (if necessary) and pump it into a.
Step 8:	While the number of further batches in both a and $S(a, G)$ is not sufficient to fill both a and the pipes of $S(a, G)$, and a further batch b either is stored at j or can be transported to it through the Selection Procedure, then call the Selection Procedure to transport b to j (if necessary) and pump it into a.
Step 9:	If there is a non-further batch b that either is contained in a or must cross a to reach its destination then go to Step 1.
Step 10:	Roll back the EPO's of Step 5, except the pumpings into a.

Figure 4: A pseudocode for the Bridge Procedure – Case (ii)

stored at the corresponding subgraph. Observe also that, if the condition (i) of Theorem 1 is satisfied, then no EPO can change this status (the argument for this claim is analogous to the proof of Lemma 1). Since the network excesses in the subproblems are equal to the excess of G, we obtain that every bridge that satisfies this condition in the main problem also satisfies it in the subproblems. In the case of condition (ii), we point out that it is satisfied due to Steps 4 and 8 of Figure 4.

3.7 Strongly Connected Components

Now, we describe the SCC Procedure to solve instances of the ABPTP such that the graph G is strongly connected. As mentioned in the beginning of this section, if G has no bridge, than one can choose directions for the arcs such that G becomes strongly connected. Since these directions are merely a convention to represent the arc contents, they can be chosen without loss of generality. Recall that the SCC Procedure is called by the Bridge Procedure to solve the subproblems that correspond to the SCC's of G.

The SCC procedure is divided in two main phases:

Phase 1: Fill all arcs with further batches.

Phase 2: Transport all non-further batches to the corresponding destination nodes.

Next, we give a pseudocode for Phase 1.

```
While not all arcs in G are filled by further batches do
    Find a node i that stores a further batch;
    Find a directed path p from i such that
        all arcs but the last one are filled by further batches;
    For each arc a in p do
        Pump a further batch into a;
    End-for
End-While
```

In the previous pseudocode, we must observe the following:

1. the node i is always found (for feasible instances) because we must have enough batches to fill all pipes;

2. the path $\boldsymbol{p}$ is always found because G is strongly connected;

3. there is always a further batch to pump into a because, in the first iteration of the inner loop, the start node of a is the node i, and, in the following iterations, the start node of a has received a further batch as a result of the previous EPO.

4. only further batches are pumped into the arcs;

5. on each iteration of the outer loop, one further batch is pumped into an arc that is not filled by further batches;

Due to the observations 4 and 5, we can conclude that phase 1 terminates before the number of iterations is greater than the total number of pipe positions in G.

For Phase 2, we use the Cycling Procedure (see Subsection 3.5). In this case, given an arc $a = (i, j)$ and a batch b stored at i, we must observe that this procedure transports b to j moving only b and some additional further batches. Moreover, at the end of this procedure, we still have all arcs filled by further batches. Hence, in Phase 2, we call the Cycling procedure to transport each non-further batch b through each arc of a path from the node that stores b to $f(b)$. After that, we have all arcs filled by further batches and all non-further batches stored at the corresponding destination nodes.

4 Final Remarks

In this paper, we presented a polynomial time algorithm for planning the transportation of multiple commodities in bidirectional pipeline networks. Despite of the simplifying assumptions of this model, our algorithm gives a better understand of the problem's combinatorial structure. Hence, we believe that it can be used to develop more effective methods to the practical problem.

References

[1] Viviane Monteiro Braconi. Heurísticas multifluxo para roteamento de produtos em redes dutoviárias. Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Departamento de Informática, April 2002.

[2] Eduardo Camponogara. A-teams para um problema de transporte de derivados de petróleo. Master's thesis, Departamento de Ciência da Computação, IMECC - UNICAMP, December 1995.

[3] Adriano Cavalcanti da Silva. Otimização do transporte em oleodutos utilizando algoritmos genéticos e programação linear. Master's thesis, Departamento de Engenharia de Sistemas, UNICAMP, April 1999.

[4] Shimon Even. *Graph Algorithms*. Computer Science Press, 1979.

[5] Christopher A. Hane. *Scheduling Multi-Product Flows in Pipelines*. PhD thesis, Georgia Institute of Technology, School of Industrial and Systems Engineering, September 1991.

[6] Christopher A. Hane and H. Donald Ratliff. Sequencing inputs to multi-commodity pipelines. *Annals of Operations Research*, 57, 1995. Mathematics of Industrial Systems I.

[7] Ruy Luiz Milidiú, Frederico dos Santos Liporace, and Carlos José P. de Lucena. Pipesworld: Planning pipeline transportation of petroleum derivatives. In *Workshop on the Competition, Trento, Italy*, June 2003.

[8] Ruy L. Milidiú, Eduardo S. Laber, Artur A. Pessoa, and Pablo A. Rey. Petroleum products scheduling in pipelines. In *The International Workshop on Harbour, Maritime & Industrial Logistics Modeling and Simulation*, september 1999.

[9] Ruy L. Milidiú, Artur A. Pessoa, and Eduardo S. Laber. Transporting petroleum products in pipelines (abstract). In *ISMP 2000 – 17th International Symposium on Mathematical Programming*, pages 134–135, Atlanta, Georgia, USA, August 2000.

[10] Ruy L. Milidiú, Artur A. Pessoa, and Eduardo S. Laber. Pipeline transportation of petroleum products with no due dates. In *Proceedings of the LATIN'2002*, pages 248–262, Canún, Mexico, april 2002.

[11] Ruy L. Milidiú, Artur A. Pessoa, and Eduardo S. Laber. The complexity of makespan minimization for pipeline transportation. *Theorectical Computer Science*, 306(1-3):339–351, 2003. presented in the APPROX'2002, Rome, Italy.

[12] H.E. Robbins. A theorem on graphs, with an application to a problem of traffic control. *American Mathematical Monthly*, 46:281–283, 1939.