A Semidefinite Programming Relaxation Approach for the Pooling Problem

Guidance

Associate Professor Nobuo YAMASHITA

Takahiro NISHI

Department of Applied Mathematics and Physics

Graduate School of Informatics

Kyoto University

February 2010

A Semidefinite Programming Relaxation Approach for the Pooling Problem

Takahiro NISHI

Abstract

The pooling problem is to determine a scheduling of flows among several tanks in refinery processes of raw materials, such as crude oil and fuel gas. The refinery company first imports raw materials, and then produces final products by refining or mixing the materials in intermediate tanks, and finally send them to their consumers. The consumers require a certain level of quality of the final product. The pooling problem is regarded as a kind of the network flow problem. However, it has two difficulties as compared to the usual linear network flow problem. One is an existence of pipeline constraints, which is formulated by using binary variables. The other is that the process of mixing materials are formulated as nonlinear equations. Thus, the pooling problem is a mixed-integer nonlinear programming. Moreover, even if the binary variables are fixed, the problem is nonconvex, and hence still very difficult. Therefore, we cannot apply the general-purpose solvers for the mixed-integer linear programming or the mixed-integer nonlinear programming.

In this paper, we apply semidefinite programming (SDP) relaxations for the polynomial optimization problem (POP) equivalent to the pooling problem. A solution of the SDP relaxation to the POP is known to be a reasonable solution. However, the size of the SDP tends to be very large and its solution is not necessarily feasible for the original POP. Therefore, we first propose a formulation of the pooling problem without the binary variables in order to reduce the problem size. Then, to tighten the relaxation, we consider to add some valid inequalities. The valid inequality is a redundant constraint of the original problem. However, it may reduce the volume of the feasible region of the SDP, and hence we can expect to get a better solution. Finally, in order to get a feasible solution of the original pooling problem, we formulate a mixed-integer linear problem whose solution is feasible. Since the formulated problem includes information of the solution of the SDP relaxation, its solution is expected to be a reasonable feasible solution. We present some results of numerical experiments to show the validity of the proposed approach.

Contents

1 Introduction

The scheduling problem is one of the main problems in operations research, and arises in various fields, for example, resource allocation, job-shop scheduling, and process scheduling which includes batch processing, oil refinery and chemical processing. For some scheduling problems, efficient algorithms have been developed [3, 14]. In particular, problems formulated as the mixed integer problem (MIP) can be solved by the state-of-the-art MIP solvers. In this paper, we consider the pooling problem, which is one of the process scheduling problems formulated as the mixed-integer nonlinear problem.

The pooling problem is to determine a scheduling of flows among several tanks in refinery processes, such as crude oil and fuel gas [1, 2, 9]. A gas corporation or an oil company imports raw materials, produces final products by refining or mixing the materials, and send them to their consumers, such as industrial plants and households. A large amount of materials is imported at one time, and they usually have different qualities depending on their location of origin. The quality represents the value per unit amount, for example, a sulfur composition and an octane number for crude oil and a heating (calorific) value for fuel gas. The customers buy a small amount of the final product in a long period, and require a certain level of its quality. If the requirement is not satisfied, then the company has to charge off the loss, such as costs for adding some expensive high quality materials in order to fill a deficit. Thus, the company puts the intermediate tanks where he stores materials temporarily and mixes materials to generate the final products. In this paper, we assume that there exist three types of components: the places where raw materials are imported (**Source**), the intermediate tanks which store materials temporarily and mix them (**BlendTank**), and customers who consume the final products are (**Plant**). In addition to these components, the problem has pipelines (**Pipeline**) which connect the components.

Since Sources, BlendTanks and Plants are regarded as nodes and Pipelines are regarded as edges, the pooling problem is to determine flows of materials in the network. Thus, the pooling problem can be regarded as a kind of network flow problem. However, it has two difficulties as compared to the usual linear network flow problems. The first difficulty appears in the constraints associated with Pipelines. When we use a pipeline, the amount of the material a that flows in the pipeline must be more than the lower capacity L and be less than the upper capacity U, i.e., $L \leq a \leq U$. On the other hand, when we do not use the pipeline, the amount of the material must be zero, which is inconsistent with $L \leq a$. To overcome the inconsistency, we may employ the following binary variable which represents the use of the pipeline, i.e.,

$$u = \begin{cases} 1 & \text{(the pipeline is used)} \\ 0 & \text{(otherwise).} \end{cases}$$

By using the binary variable, we can represent the constraints of flow a in the pipeline as follows:

$$uL \leq a \leq uU.$$

Each component, especially BlendTank, is connected with several pipelines. Those pipelines cannot be used at the same time in order to prevent a back-flow and so on for safety. Thus we use at most **one** pipeline in each time period. The requirement is represented by the constraints that the sum of binary variables corresponding to the pipelines connected with the component should not exceed **one** at each time. Second, the process of mixing materials are formulated

as nonlinear equations, and hence the pooling problem has nonlinear equality constraints. We explain the constraints. Consider two materials 1 and 2 to be mixed into the new material 3. Suppose that the materials 1, 2 and 3 have the amounts p_1, p_2 and p_3 and qualities (which are values per unit amount) q_1, q_2 and q_3, respectively. Then, the following two equalities express proportional blending:

$$p_3 = p_1 + p_2$$
$$p_3 q_3 = p_1 q_1 + p_2 q_2.$$

The first equation represents the mass balance and it is linear. The second equation means that the quality of the new material is calculated by the weighted average, and it is nonlinear. Thus, the pooling problem can be written as a mixed-integer nonlinear programming problem. Moreover, even if the binary variables are fixed, the problem is a nonconvex problem. Therefore, we cannot expect to compute a global optimal solution for practical scale pooling problems, and hence we consider to obtain a reasonable solution within practical time.

A lot of methods have been proposed to solve general mixed-integer nonlinear programming problems. For instance, the outer-approximation method [5] and the generalized Benders decomposition method [8] are well studied. Both methods can obtain a local optimal solution by solving continuous linear subproblems repeatedly if the problem obtained by fixing the integer part is convex. However, since the pooling problem is nonconvex as mentioned above, these method are not guaranteed to obtain a global optimal solution. Several methods tailored to the pooling problem have been proposed, including the combined method of graph theory and linear programming, and the genetic algorithm [4]. The former method first calculates a lower bound of the pooling problem by solving a continuous linear subproblem without some difficult constraints, and then obtains a feasible solution by solving a minimum cost flow problem. The latter one solves a combinatorial problem of the binary variables as an upper level problem, and continuous subproblems with the binary variables fixed are solved for each gene. Since both methods ignore nonlinear constraints to generate a trial point, they cannot well treat the nonlinear functions.

In this paper, we propose to apply the semidefinite programming relaxation for the pooling problem. As mentioned above, the pooling problem is formulated as a mixed-integer nonlinear problem. As shown in Section 3, the pooling problem can be written as a continuous polynomial optimization problem (POP). The POP is a nonlinear programming problem whose objective function and constraint functions are polynomial. Finding a global solution of POP is also difficult in general because a polynomial function might be nonconvex. In recent years, as a promising approach to solve the POP, the semidefinite programming relaxation has been attracted much attention [6, 11, 10, 13, 15]. A problem in which all the monomials included in the POP are linearized is formulated as a semidefinite programming problem (SDP), and it is a relaxation of the original POP. The SDP can be solved by the primal-dual interior point method. We can get its solution efficiently. In addition, Kojima *et al.* [7, 17] proposed a method to reduce the size of the SDP by using the sparsity of polynomials. As a result, an approximate solution is expected to be obtained within a reasonable computational time.

The main problem of applying the SDP relaxation is that the size of the SDP becomes very large and the solution of the SDP is not necessarily feasible for the original problem. For example, when we consider scheduling with 10 discrete time horizon, 10 components and 50 pipelines, the pooling problem is formulated as a POP with 1200 variables, and its SDP relaxation problem

has at least a 1200×1200 matrix. In order to reduce the problem size, we propose a formulation that does not include the binary variables of the pooling problem. However, the removal causes excessive relaxation of the feasible region of the original problem. To tighten the region, we use valid inequalities. The valid inequality is known to be effective for the SDP relaxation approach. It is a redundant constraint of the original POP, but it reduces the feasible region of the SDP, and hence we can expect to get a better solution. Moreover, we also consider to fix some of the variables after we solve the SDP relaxation problem. Then, we repeatedly solve the SDP relaxation problem with some variables fixed until we need not fix the variables. Finally, in order to get a feasible solution of the pooling problem, we propose to solve a mixed-integer linear programming problem by exploiting the information of the solution obtained by solving the final relaxation problem. The problem can be solved more easily than the original pooling problem.

The remainder of the present paper is organized as follows. Section 2 describes the formulation of the pooling problem. The semidefinite programming relaxation approach for the pooling problem is proposed in Section 3. Numerical results are presented and discussed in Section 4. Finally, we make some concluding remarks in Section 5.

Throughout the paper, we use the following notations. For matrix $X \in \mathbb{R}^{n\times n}$, $X \succeq 0$ means X is positive semidefinite, i.e., $\boldsymbol{v}^T X \boldsymbol{v} \geq 0$ for all $\boldsymbol{v} \in \mathbb{R}^n$. For $\boldsymbol{x} \in \mathbb{R}^n$, $\|\boldsymbol{x}\|_1 = \sum_{i=1}^n |x_i|$. In addition, for a set A, $|A|$ represents the number of elements in A.

2 The Pooling Problem

In this section, we give a concrete formulation of the pooling problem, and discuss its difficulties.

The purpose of the pooling problem is to determine schedules of flows of materials, from importing raw materials (e.g., crude oil and fuel gas) to sending final products to consumers. We consider a system that has three types of components, Source, BlendTank and Plant, and pipelines that connect the components. Source imports the raw materials at specific times. BlendTank stores and blends (mixes) the materials, and generates the final products. Plant consumes (buys) the final products. The situation is shown in Fig.1.

Each material has its own quality. The quality is the value per unit amount, for example, an octane number of gasoline and a heating (calorific) value of fuel gas. Here, high quality implies good material. The raw materials imported to Source have a variety of quantity and quality. However, Plant requires a certain amount of quantity and a certain level of quality of the final products at each time. If that requirement is not satisfied, then the insufficiency of the requirement cause some additional cost. Therefore, intermediate tanks (BlendTank) are placed in the system for the reason not only to store the materials but also to mix them in order to satisfy the requirement of the quality. There are types of two costs in the pooling problem. First, we transport the materials from one tank to another tank by electric power. This cost is called the "transportation cost". Second, when the requirement of the quality is not satisfied, the company must pay extra money as mentioned above. We call this cost the "compensation cost". We aim to minimize the sum of the total transportation costs and the total compensation costs.

The system in the pooling problem has a graph geometry where each component is regarded as a node, and each pipeline as an edge. Here, we denote the numbers of Sources, BlendTanks, Plants as M_S, M_B and M_P, respectively. The number of all nodes M_V is defined to be $M_V =$

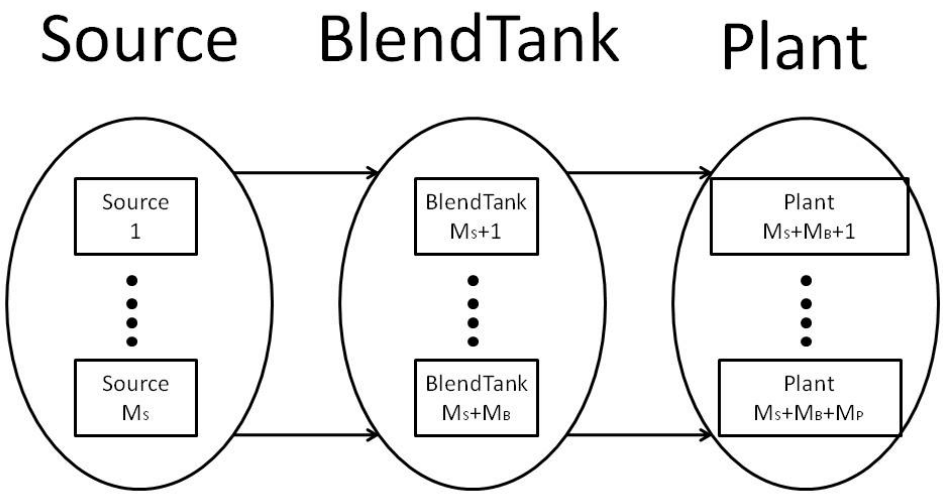

Figure 1: Overview of the pooling problem

$M_S + M_B + M_P$. Additionally, we denote the set of Sources by V_S, the set of BlendTanks by V_B and the set of Plants by V_P.

$$V_S = \{1, \ldots, M_S\},$$
$$V_B = \{M_S + 1, \ldots, M_S + M_B\},$$
$$V_P = \{M_S + M_B + 1, \ldots, M_S + M_B + M_P\}.$$

Let the set of all components be V. If there exists a pipeline that originates from node i and is destined for node j, we denote this pipeline as edge (i, j). The set of all edges is denoted as A. Here, we note that a pipeline has a direction, that is, we cannot transport from node j to node i by pipeline (i, j). For each node $i \in V$, we denote the set of nodes that have the edge destined for node i and the set of nodes that have the edge originating from node i by $I(i)$ and $E(i)$, respectively. Namely $I(i)$ and $E(i)$ are given as

$$I(i) \equiv \{j \in V \mid (j, i) \in A\},$$
$$E(i) \equiv \{k \in V \mid (i, k) \in A\}.$$

We formulate the problem with a uniform discretization of time in the given scheduling horizon. Let the final time of the horizon be M_T, and let the set of discretized times be $T = \{t \mid 1, \ldots, M_T\}$. In the following, we use indices i, j, and k for nodes and t for time period. The notations for the graph and indexes are summarized in Table 1.

Next, we list given constants and decision variables of the pooling problem.

(a) Initial conditions for Source (constants)

Table 1: Notation of parameter of graph and indices

Parameter of graph	
M_S:	Cardinality of Source.
M_B:	Cardinality of BlendTank.
M_P:	Cardinality of Plant.
M_V:	The number of all nodes.
M_T:	The number of discretization of time horizon.
V_S:	Set of Source.
V_B:	Set of BlendTank.
V_P:	Set of Plant.
V:	Set of all nodes.
A:	Set of all edges that represent pipeline .
$I(i)$:	Set of node which has an edge into node i.
$E(i)$:	Set of node which has an edge from node i.
T:	Set of discretized time.
Indices	
i, j, k:	Indices for each node.
t:	Index for time period.
(i, j):	Index for each edge.

p_i^0 : Initial amount of materials in source $i \in V_S$.
q_i^0 : Initial quality of materials in source $i \in V_S$.
SA_i^t: The amount of raw materials filled in source $i \in V_S$ at time $t \in T$.
SQ_i^t: The quality of raw materials filled in source $i \in V_S$ at time $t \in T$.

(b) Initial conditions for BlendTank (constants)

p_i^0 : Initial amount of materials in BlendTank $i \in V_B$.
q_i^0 : Initial quality of materials in BlendTank $i \in V_B$.
$p_i^{\min}$: Minimum capacity of BlendTank $i \in V_B$.
$p_i^{\max}$: Maximum capacity of BlendTank $i \in V_B$.

(c) Constants for Plant

DC_i^t: The requirement of quantity in Plant $i \in V_P$ at time $t \in T$.
DQ_i^t: The requirement of quality in Plant $i \in V_P$ at time $t \in T$.
CQ_i: Compensation cost per unit insufficiency of quality in Plant $i \in V_P$.

(d) Initial conditions for pipeline

L_{ij}: Lower bound on flow in edge (i, j) .
U_{ij}: Upper bound on flow in edge (i, j) .
CA_{ij}: Cost per unit material flow of edge (i, j).

(e) Decision variables for Source, BlendTank and Plant

p_i^t: The amount of materials in node $i \in V$ at time $t \in T$.
q_i^t: The quality of materials in node $i \in V$ at time $t \in T$.

(f) Decision variables for pipeline

a_{ij}^t: The flow of edge $(i,j) \in A$ at time $t \in T$.
u_{ij}^t: binary variable indicating the usage of edge (i,j).

We denote the vector $(p_i^1, \ldots, p_i^{M_T})^\top$ as $\boldsymbol{p}_i$ and the vector $(\boldsymbol{p_1}, \ldots, \boldsymbol{p_{M_V}})^\top$ as $\boldsymbol{p}$. Similarly, we denote other vector variables as $\boldsymbol{q}, \boldsymbol{a}, \boldsymbol{u}$.

We formulate the pooling problem as a mixed-integer programming problem. The objective of this problem is to minimize the sum of transportation costs and compensation costs. The compensation cost is proportional to the product of the quality insufficiency and the amount of the final product. The concrete objective function of the problem can therefore be written as

$$\sum_{t\in T}\sum_{(i,j)\in A} CA_{ij}a_{ij} + \sum_{t\in T}\sum_{i\in V_P} CQ_i DC_i^t \max\{0, DQ_i^t - q_i^t\}. \tag{1}$$

We note that the compensation cost is higher than the transportation cost in general. If we need to satisfy the requirement of the quality, we set $CQ := \infty$, which means that we add $q_i^t \geq DQ_i^t$ as constraints.

Next we describe the constraints. First, we note that when two materials are mixed, the newly generated material must have the same total amount and the same total value of the materials. As mentioned in Introduction, when we mix two materials 1 and 2 with quantity p_1, p_2 and quality q_1, q_2 into the new material 3 with quantity p_3 and quality q_3, the following two equalities express proportional blending.

$$p_3 = p_1 + p_2 \tag{2}$$
$$p_3 q_3 = p_1 q_1 + p_2 q_2 \tag{3}$$

The first equation (2) represents the material quantity balance, and it is linear. We call this equation the "quantity conservation law". The second equation (3) represents the material quality balance, and it is nonlinear. This equation is called the "value conservation law".

Now we give the whole constraints of the pooling problem.

Source constraints

The quantity conservation law (2) and the value conservation law (3) must be satisfied at all times, and the new raw material with the amount SA_i^t and quality SQ_i^t is added in time $t \in T$.

$$\begin{aligned} p_i^{t+1} &= p_i^t + SA_i^t - \sum_{k\in E(i)} a_{ik}^t, \\ p_i^{t+1} q_i^{t+1} &= p_i^t q_i^t + SA_i^t SQ_i^t - \sum_{k\in E(i)} a_{ik}^t q_i^t. \end{aligned} \tag{4}$$

Additionally, the amount of material of each Source $i \in V_S$ must be nonnegative at all times $t \in T$.

$$p_i^t \geq 0. \tag{5}$$

BlendTank constraints

The quantity conservation law (2) and the value conservation law (3) must be satisfied at

all times.

$$\begin{aligned} p_i^{t+1} &= p_i^t + \sum_{j\in I(i)} a_{ji}^t - \sum_{k\in E(i)} a_{ik}^t, \\ p_i^{t+1}q_i^{t+1} &= p_i^t q_i^t + \sum_{j\in I(i)} a_{ji}^t q_j^t - \sum_{k\in E(i)} a_{ik}^t q_i^t. \end{aligned} \tag{6}$$

A BlendTank $i \in V_B$ cannot be filled to more than its capacity and must keep the amount no less than its minimum volume.

$$p_i^{\min} \le p_i^t \le p_i^{\max}. \tag{7}$$

Plant constraints

In plant $i \in V_P$, the required amount of the final products is given by DC_i^t in each time $t \in T$, and the total value is $\sum_{j\in I(i)} a_{ji}^t q_j^t$. Thus the quality q_i^{t+1} is written as

$$q_i^{t+1} = \frac{\sum_{j\in I(i)} a_{ji}^t q_j^t}{DC_i^t}. \tag{8}$$

Pipeline constraints

For each pipeline $(i,j) \in A$, if there exists flow in time $t \in T$, we set binary variable $u_{ij}^t = 1$, otherwise we set $u_{ij}^t = 0$. Thus flow a_{ij}^t must satisfy

$$u_{ij}^t L_{ij} \le a_{ij}^t \le u_{ij}^t U_{ij}. \tag{9}$$

In each node $i \in V$, we can use at most **one** pipeline connected to node i. This requirement can be formulated as

$$\sum_{j\in I(i)} u_{ji}^t + \sum_{k\in E(i)} u_{ik}^t \le 1. \tag{10}$$

Summarizing the above arguments from (1) to (10), we give the concrete pooling problem as follows.

$$\begin{aligned} \text{(P)} \quad \underset{\mathbf{a},\mathbf{u},\mathbf{p},\mathbf{q}}{\text{minimize}} \quad & \sum_{t\in T}\sum_{(i,j)\in A} CA_{ij}a_{ij} + \sum_{t\in T}\sum_{i\in V_P} CQ_i DC_i^t \max\{0,\ DQ_i^t - q_i^t\} \\ \text{subject to} \quad & u_{ij}^t L_{ij} \le a_{ij}^t \le u_{ij}^t U_{ij} \qquad (\forall (i,j)\in A,\ \forall t \in T) \\ & \sum_{j\in I(i)} u_{ji}^t + \sum_{k\in E(i)} u_{ik}^t \le 1 \quad (\forall i \in V,\ \forall t \in T) \\ & u_{ij}^t \in \{0,1\} \qquad (\forall (i,j)\in A,\ \forall t \in T) \\ & (\boldsymbol{a},\boldsymbol{p},\boldsymbol{q}) \in \bar{\Omega}, \end{aligned} \tag{11}$$

where

$$\bar{\Omega} = \{(\boldsymbol{a},\ \boldsymbol{p},\ \boldsymbol{q}) \mid (4),\ (5),\ (6),\ (7), \text{and } (8)\}.$$

Since the pooling problem has binary variables $\boldsymbol{u} = (u_{ij}^t)$, it is very difficult to solve. Another difficult aspect is that it includes nonlinear functions in the constraints, i.e., the value conservation law (3). Since the value conservation law (3) is a nonlinear equation (not inequality), the pooling problem cannot be a convex problem even if we fix the binary variables $\boldsymbol{u}$. A mixed-integer nonlinear programming problem can be solved by some algorithms, such as generalized Benders decomposition method [8] and outer approximation method [5]. However, as mentioned

in Introduction, because they need the convexity of the subproblems obtained by fixing the integer part, they cannot be applied to the pooling problem. Moreover, even if we fix the binary variables, we cannot easily obtain a global solution, because many nonconvex constraints still remain. In addition to this difficulty, the number of binary variables is $M_T \times |A|$. When the scale of the problem becomes large, the combination of binary variables $\boldsymbol{u}$ grows exponentially. Thus, it is difficult to apply the state-of-the-art method for combinational problems, such as Branch and Bound algorithms, to the pooling problem. Therefore, we must develop methods that are tailored to the pooling problem.

3 Semidefinite Programming Relaxation Approach

In this section, we propose a framework for solving the pooling problem (11) using the semidefinite programming (SDP) relaxation approach. For applying the approach, we first reformulate the pooling problem as a polynomial optimization problem (POP). The POP is a nonlinear programming problem whose objective and constraint functions are polynomial. The standard form of POP is

$$\begin{array}{lll} \text{(POP)} & \text{minimize} & f_0(\boldsymbol{x}) \\ & \text{subject to} & f_i(\boldsymbol{x}) \geq 0 \quad (i = 1, \ldots, m), \end{array}$$

where all functions $f_i : \mathbb{R}^n \to \mathbb{R}$ $(i = 0, \ldots, m)$ are polynomial. Note that equality constraints $g(\boldsymbol{x}) = 0$ can be formulated as two inequalities of $g(\boldsymbol{x}) \geq 0$ and $-g(\boldsymbol{x}) \geq 0$.

By linearizing all monomials included in the POP, the POP is transfered into the linear SDP [11]. The SDP relaxation problem SDR_r(POP) of the POP can be written as

$$\begin{array}{lll} \mathrm{SDR}_r\text{(POP)} & \text{minimize} & F_0(\boldsymbol{y}) \\ & \text{subject to} & F_i(\boldsymbol{y}) \succeq O \quad (i = 1, \ldots, m) \\ & & M_r(\boldsymbol{y}) \succeq O, \end{array}$$

where r is the relaxation order, each F_i is a linearized function of f_i, and M_r is a symmetric matrix called a moment matrix (see [11] for details). The moment matrix M_r plays an important role in the tightness of the relaxation, because it guarantees that a solution of the problem SDR_r(POP) coincides with a solution of the problem POP when relaxation order r is sufficiently large [11]. However, since the size of M_r is ${}_{n+r}C_r \times {}_{n+r}C_r$, the size of the problem SDR_r(POP) increases exponentially as r and n, and hence it is very difficult to solve. Since the pooling problem (11) has at most quadratic functions, we can choose any relaxation order $r \geq 1$. While the SDR_1(POP) has an $(n+1) \times (n+1)$ matrix as its decision variables, and the SDR_2(POP) has an $\frac{n(n+1)}{2} \times \frac{n(n+1)}{2}$ matrix. Thus when $n = 100$, the largest matrix size of SDR_1(POP) is $10,201$, while SDR_2(POP) is $25,502,500$. Since the pooling problem usually has hundreds of variables, we have to choose $r = 1$. However, note that a solution of the problem SDR_r(POP) with lower r is not always feasible for the POP.

3.1 Reformulation of the pooling problem to POP

Here we give a POP formulation equivalent to the pooling problem (11). The pooling problem (11) includes max functions and binary valuables, and thus it is not a POP. First note that $u_{ij}^t \in \{0, 1\}$ can be written as the polynomial constraint $u_{ij}^t(u_{ij}^t - 1) = 0$, $(\forall\, (i,j) \in A, \forall\, t \in T)$.

Thus, using the auxiliary variables v_i^t $(i \in V_P, t \in T)$ for the max functions, the pooling problem (11) is written as

$$\begin{aligned}
\text{(P-POP)} \quad \underset{\mathbf{a},\mathbf{u},\mathbf{p},\mathbf{q},\mathbf{v}}{\text{minimize}} \quad & \sum_{t\in T}\sum_{(i,j)\in A} CA_{ij}a_{ij}^t + \sum_{t\in T}\sum_{i\in V_P} CQ_i DC_i^t v_i^t \\
\text{subject to} \quad & u_{ij}^t L_{ij} \le a_{ij}^t \le u_{ij}^t U_{ij} && (\forall (i,j)\in A, \forall t \in T) \\
& \sum_{j\in I(i)} u_{ji}^t + \sum_{k\in E(i)} u_{ik}^t \le 1 && (\forall i \in V, \forall t \in T) \\
& u_{ij}^t(u_{ij}^t - 1) = 0 && (\forall (i,j)\in A, \forall t \in T) \\
& u_{ij}^t \in \mathbb{R} && (\forall (i,j)\in A, \forall t \in T) \\
& (\boldsymbol{a},\boldsymbol{p},\boldsymbol{q},\boldsymbol{v}) \in \Omega,
\end{aligned} \tag{12}$$

where

$$\Omega = \{(\boldsymbol{a},\boldsymbol{p},\boldsymbol{q},\boldsymbol{v}) \mid (\boldsymbol{a},\boldsymbol{p},\boldsymbol{q}) \in \bar{\Omega},\ v_i^t \ge DQ_i^t - q_i^t,\ v_i^t \ge 0\}.$$

We try to get an approximate solution of the problem (12) by the SDP relaxation approach. We denote the SDP relaxation problem of the problem (12) with relaxation order r as SDR $_r$(P-POP). By solving the problem SDR$_r$(P-POP), we can obtain an approximate solution $(\bar{\boldsymbol{a}},\bar{\boldsymbol{u}},\bar{\boldsymbol{p}},\bar{\boldsymbol{q}},\bar{\boldsymbol{v}})$ of the original problem (12). If the approximate solution $(\bar{\boldsymbol{a}},\bar{\boldsymbol{u}},\bar{\boldsymbol{p}},\bar{\boldsymbol{q}},\bar{\boldsymbol{v}})$ is feasible for the original problem (12), then $(\bar{\boldsymbol{a}},\bar{\boldsymbol{u}},\bar{\boldsymbol{p}},\bar{\boldsymbol{q}},\bar{\boldsymbol{v}})$ is an optimal solution of the original problem (12). However, the approximate solution $(\bar{\boldsymbol{a}},\bar{\boldsymbol{u}},\bar{\boldsymbol{p}},\bar{\boldsymbol{q}},\bar{\boldsymbol{v}})$ is not necessarily feasible for the original problem (12). To obtain an optimal solution of (12), we must set the relaxation order r to be large enough. However, SDR$_r$(P-POP) becomes huge for a large r. For example, consider a scheduling problem with 10 discrete time horizon, 10 components and 30 pipelines. Then the total number of variables is about 800, the size of the largest matrix in SDR$_1$(P-POP) is about 800×800, and the size of the largest matrix in SDR$_2$(P-POP) is about $320,000 \times 320,000$. Therefore, it is practically impossible to solve the SDP relaxation with relaxation order $r \ge 2$. We consider to obtain a reasonable approximate solution from SDR$_1$(P-POP).

Even if $r = 1$, the problem SDR$_1$(P-POP) is still large. In Subsection 3.2, we first propose a formulation without u_{ij}^t in order to reduce the problem size. The relaxation order $r = 1$ may imply that the relaxation is very loose. Thus, in Subsection 3.3, we consider to add appropriate valid inequalities to the problem in order to obtain a better approximate solution. In Subsection 3.4, we propose a procedure to fix some of flow variables a_{ij}^t at zero after we solve the SDP relaxation problem, which reduces the size of the problem. The reduced SDP relaxation may provide a tight relaxation. Finally, in Subsection 3.5, in order to get a feasible solution of the original pooling problem, we formulate a mixed-integer linear programming problem whose solution is feasible for the pooling problem. Since the formulated problem includes information of the solution of the SDP relaxation problem, its solution is expected to be a reasonable feasible solution.

3.2 Removal of $\boldsymbol{u}$ from (12)

When we reduce the number of variables in (12), we can reduce the size of the SDP relaxation problem drastically, because the size of the SDP relaxation problem is usually the square of that of the original problem. Thus, we consider to remove $\boldsymbol{u}$ since they are artificial variables. However, the deletion causes the following problem. We cannot represent the constraint (10)

that means we can use at most **one** pipeline connected to the same node at the same time, and the constraint (9) that represents for each pipeline $(i,j) \in A$, if flow exists in time $t \in T$, flow a_{ij}^t satisfies $L_{ij} \le a_{ij}^t \le U_{ij}$, otherwise $a_{ij}^t = 0$.

To represent the constraint (10), we propose to add the following complementarity conditions.

$$a_{ji}^t a_{ki}^t = 0 \ (j,k \in I(i), j \neq k),\ a_{ij}^t a_{ik}^t = 0 \ (j,k \in E(i), j \neq k),\ a_{ji}^t a_{ik}^t = 0 \ (j \in I(i), k \in E(i)).$$

Moreover, since $a_{ij}^t \ge 0 \ (\forall (i,j) \in A, \forall t \in T)$, these constraints are equal to following constraint.

$$\sum_{j,k \in I(i), j \neq k} a_{ji}^t a_{ki}^t + \sum_{j,k \in E(i), j \neq k} a_{ij}^t a_{ik}^t + \sum_{j \in I(i), k \in E(i)} a_{ji}^t a_{ik}^t = 0.$$

Next, to represent (9), we relax the constraint $L_{ij} \le a_{ij}^t$ when flow exists. That is, we replace the constraint (9) by the following constraint.

$$0 \le a_{ij}^t \le U_{ij}.$$

Now we get the following relaxed problem of (12).

$$\begin{aligned}
(\text{P-}\overline{\text{POP}}) \quad \underset{\mathbf{a},\mathbf{u},\mathbf{p},\mathbf{q},\mathbf{v}}{\text{minimize}} \quad & \sum_{t \in T} \sum_{(i,j) \in A} CA_{ij} a_{ij}^t + \sum_{t \in T} \sum_{i \in V_P} CQ_i DC_i^t v_i^t \\
\text{subject to} \quad & \sum_{j,k \in I(i), j \neq k} a_{ji}^t a_{ki}^t + \sum_{j,k \in E(i), j \neq k} a_{ij}^t a_{ik}^t + \sum_{j \in I(i), k \in E(i)} a_{ji}^t a_{ik}^t = 0 \quad (\forall i \in V, \forall t \in T) \\
& 0 \le a_{ij}^t \le U_{ij} \qquad (\forall (i,j) \in A, \forall t \in T) \\
& (\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}) \in \Omega.
\end{aligned} \tag{13}$$

The number of variables of the problem (13) is $M_T \times |A|$ fewer than those of the original problem (12). On the other hand, the number of the constraints of the problem (13) is the same as that of the original problem (12).

3.3 Valid inequalities of (13)

A **valid inequality** is known to be effective for the SDP relaxation approach. Although the valid inequality is a redundant constraint for the original POP, it tightens the feasible region of SDP, and hence we can expect to get a better solution. Especially, since the difficulty of the pooling problem comes from the value conservation law (3), it is effective to add valid inequalities related to the quality and pipeline constraints. We denote the problem (P-$\overline{\text{POP}}$) with valid inequalities as (P-$\overline{\text{POP}}_v$).

Trivial upper and lower bounds of each variable: Each variable has trivial upper and lower bounds. For example, the quality variable q_i^t has an upper (lower) bound which is given by the largest (smallest) quality value of the given constants. Specifically, we set $q^{\max} = \max_{i,t}\{q_i^0,\ SQ_i^t\}$ and $q^{\min} = \min_{i,t}\{q_i^0,\ SQ_i^t\}$. In this case, q_i^t has the following valid inequalities.

$$q^{\min} \le q_i^t \le q^{\max}. \tag{14}$$

Note that it is time invariant. Since the quality of the final product must be greater than the worst case quality $q^{\min}$, the compensation variable v_i^t must satisfy

$$v_i^t \le DQ_i^t - q^{\min}. \tag{15}$$

Upper and lower bounds of monomials with degree 2: For monomial xy with $0 \le L_x \le x \le U_x$ and $0 \le L_y \le y \le U_y$, the following inequalities hold [12].

$$xy \le U_y x + L_x y - L_x U_y,\ xy \le L_y x + U_x y - U_x L_y,$$
$$xy \ge U_y x + U_x y - U_x U_y,\ xy \ge L_y x + L_x y - L_x L_y.$$

Using these relations, $p_i^{\min} \le p_i^t \le p_i^{\max}$ and $q^{\min} \le q_i^t \le q^{\max}$ give the following valid inequalities.

$$p_i^t q_i^t \le q^{\max} p_i^t + p_i^{\min} q_i^t - p_i^{\min} q^{\max},\ p_i^t q_i^t \le q^{\min} p_i^t + p_i^{\max} q_i^t - p_i^{\max} q^{\min}, \tag{16}$$

$$p_i^t q_i^t \ge q^{\max} p_i^t + p_i^{\max} q_i^t - p_i^{\max} q^{\max},\ p_i^t q_i^t \ge q^{\min} p_i^t + p_i^{\min} q_i^t - p_i^{\min} q^{\min}, \tag{17}$$

$$(q_i^t)^2 \le (q^{\min} + q^{\max}) q_i^t - q^{\min} q^{\max}, \tag{18}$$

$$(q_i^t)^2 \ge 2q^{\max} q_i^t - (q^{\max})^2,\ (q_i^t)^2 \ge 2q^{\min} q_i^t - (q^{\min})^2. \tag{19}$$

The monomials $a_{ij}^t q_i^t$, $(p_i^t)^2$, $(a_i^t)^2$ have similar valid inequalities. From the property of the pooling problem, the counterparts of the quality q_i^t and its square $(q_i^t)^2$ in the SDP relaxation problem tend to become as large as possible. Thus we use (17) and (18) in the numerical experiments.

Implicit upper and lower bounds of variables: As mentioned before, p_i^t and q_i^t have the trivial time invariant upper and lower bounds. However, there exist implicit time dependent bounds due to the other constraints. For example, p_i^1 must satisfy $p_i^1 \le p_i^0 + \max_{j \in I(i)}\{U_{ij}\}$. Though estimating these bounds requires some additional calculations, the time variant implicit bounds may provide tight valid inequalities of (16)-(19), in particular in earlier stages of the time period.

Now we give a naive algorithm to estimate the implicit bounds for p_i^t and q_i^t.

Bounds of p_i^t: We denote the estimated upper and lower bounds of p_i^t at time t as $\overline{p}_i^t$ and $\underline{p}_i^t$, respectively. We can easily get an upper bound $\overline{p}_i^t$ by solving the following problem.

$$\begin{aligned}
\underset{\mathbf{a},\mathbf{u},\mathbf{p}}{\text{minimize}} \quad & -p_i^t \\
\text{subject to} \quad & \begin{cases} p_i^{t+1} = p_i^t + SA_i^t - \sum_{k \in E(i)} a_{ik}^t \\ p_i^t \ge 0 \end{cases} && (\forall i \in V_S, \forall t \in T) \\
& \begin{cases} p_i^{t+1} = p_i^t + \displaystyle\sum_{j \in I(i)} a_{ji}^t - \sum_{k \in E(i)} a_{ik}^t \\ p_i^{\min} \le p_i^t \le p_i^{\max} \end{cases} && (\forall i \in V_B, \forall t \in T) \\
& \begin{cases} DC_i^t = \displaystyle\sum_{j \in I(i)} a_{ji}^t \end{cases} && (\forall i \in V_P, \forall t \in T) \\
& u_{ij}^t L_{ij} \le a_{ij}^t \le u_{ij}^t U_{ij} && (\forall (i,j) \in A, \forall t \in T) \\
& \displaystyle\sum_{j \in I(i)} u_{ji}^t + \sum_{k \in E(i)} u_{ik}^t \le 1 && (\forall i \in V, \forall t \in T) \\
& a_{ij}^t \in \mathbb{R},\ u_{ij}^t \in \{0,1\} && (\forall (i,j) \in A, \forall t \in T) \\
& p_i^t \in \mathbb{R} && (\forall i \in V, \forall t \in T),
\end{aligned} \tag{20}$$

which is to maximize p_i^t subject to the same constraints as those of the pooling problem (11) expect for the nonlinear constraints (3). Similarly, the estimated lower bound $\underline{p}_i^t$ is given by an optimal value of problem (20) with the objective function p_i^t instead of $-p_i^t$. Because the problem (20) is a mixed-integer linear programming problem, we get the optimal value more easily than the original problem (12). Note that, if we need to obtain all bounds of quantity variables p_i^t, we have to solve MILPs $2 \times M_V \times M_T$ times.

Bounds of q_i^t: In the following, we assume that $\overline{p}_i^t$ and $\underline{p}_i^t$ have already been obtained. Let $\overline{q}_i^t$ and $\underline{q}_i^t$ be the estimated time variant upper and lower bounds. For estimating $\overline{q}_i^{t+1}$, we first check the quality $\overline{q}_j^t$ of $j \in I(i)$ at t. If $\overline{q}_i^t \geq \overline{q}_j^t$ holds for all $j \in I(i)$, then we set $\overline{q}_i^{t+1} = \overline{q}_i^t$. Otherwise, we set the quality $\overline{q}_i^{t+1}$ to the maximum quality calculated as follows. We assume that the quantity p_i^t of component i is the minimum quantity $\underline{p}_i^t$, and the quantity p_j^t of component $j \in I(i)$ is its maximum $\overline{p}_j^t$. First, we calculate the maximum flow e_{ij} from component j into component i. Note that component i can receive at most $p_i^{\max} - \underline{p}_i^t$, the flow upper bound of pipeline (i,j) is U_{ij} and component j can send at most $\overline{p}_j^t - p_j^{\min}$. Thus the maximum flow e_{ij} is given by

$$e_{ij} = \min\{p_i^{\max} - \underline{p}_i^t,\ U_{ij},\ \overline{p}_j^t - p_j^{\min}\}.$$

Using e_{ij}, we calculate the maximum quality value $\overline{q}_i^{t+1}$ as follows:

$$\overline{q}_i^{t+1} = \max_j \left\{ \frac{\underline{p}_i^t \overline{q}_i^t + e_{ij}\overline{q}_j^t}{\underline{p}_i^t + e_{ij}} \;\middle|\; j \in I(i),\ \overline{q}_i^t < \overline{q}_j^t \right\}.$$

We can calculate $\underline{q}_i^{t+1}$ similarly.

3.4 Fixing variables after solving the SDP relaxation problem

Some flows a_{ij}^t in the solution of the SDP relaxation problem may become zero. Then we may construct a SDP relaxation problem with such a_{ij}^t fixed to zero. The resulting SDP becomes smaller, and may provide a tight relaxation. Now we explain how to fix such variables. Let $\bar{\boldsymbol{a}}$ be the flows in the solution of the SDP relaxation problem, and let ϵ be a small positive scaler. Then, the set J_0^t of the expected no flow pipelines is defined by $J_0^t = \{(i,j) \in A \mid a_{ij}^t \leq \epsilon\}$. Let J_f^t be $J_f^t := A \setminus J_0^t$. Note that J_f^t is the set of pipelines that are supposed to be used. We set $\boldsymbol{J}_0 := \{(i,j,t) | (i,j) \in J_0^t,\ t \in T\}$, $\boldsymbol{J}_f := \{(i,j,t) | (i,j) \in J_f^t,\ t \in T\}$. The relaxed pooling problem (P-$\overline{\text{POP}}$) with $\boldsymbol{J}_f$ and $\boldsymbol{J}_0$ are stated as follows.

$$\begin{aligned}
(\text{NPP}(\boldsymbol{J}_f, \boldsymbol{J}_0))\ \underset{\mathbf{a},\mathbf{u},\mathbf{p},\mathbf{q},\mathbf{v}}{\text{minimize}}\quad & \sum_{t\in T}\sum_{(i,j)\in I_f^t} CA_{ij}a_{ij}^t + \sum_{t\in T}\sum_{i\in V_P} CQ_i DC_i^t v_i^t & \\
\text{subject to}\quad & \sum_{j,k\in \tilde{I}^t(i), j\neq k} a_{ji}^t a_{ki}^t + \sum_{j,k\in \tilde{E}^t(i), j\neq k} a_{ij}^t a_{ik}^t + \sum_{j\in \tilde{I}^t(i), k\in \tilde{E}^t(i)} a_{ji}^t a_{ik}^t = 0 & (\forall i \in V, \forall t \in T) \\
& 0 \leq a_{ij}^t \leq U_{ij} & (\forall (i,j) \in I_f^t, \forall t \in T) \\
& (\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}) \in \hat{\Omega}, &
\end{aligned} \tag{21}$$

where $\tilde{I}^t(i)$ and $\tilde{E}^t(i)$ are given by

$$\begin{aligned}
\tilde{I}^t(i) &\equiv \{j \in I(i) \mid (j,i) \in J_f^t\}, \\
\tilde{E}^t(i) &\equiv \{k \in E(i) \mid (i,k) \in J_f^t\},
\end{aligned}$$

and, $\hat{\Omega}$ is the set Ω with $\tilde{I}^t(i)$ and $\tilde{E}^t(i)$ instead of $I(i)$ and $E(i)$, respectively. Comparing the problem (21) with the original problem (13), the number of flow variables $\boldsymbol{a}$ decreases by $|I_0|$. Note that the problem without fixed variables (NPP$(A, \emptyset)$) is equal to the original problem (13). We denote the semidefinite relaxation problem of the problem (21) with relaxation order r as SDR$_r$(NPP$(\boldsymbol{J}_f, \boldsymbol{J}_0)$).

3.5 Finding a feasible solution

When the solution $(\bar{\boldsymbol{a}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}, \bar{\boldsymbol{v}})$ of the SDP relaxation problem is not feasible for the original problem (12), we should find a reasonable feasible solution of (12). We can get a feasible solution by solving the pooling problem without nonlinear equality constraints (3), which is a mixed-integer linear programming problem. However, such a problem ignores the most important cost, that is, the compensation cost. Thus, we consider to exploit information of the solution $(\bar{\boldsymbol{a}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}, \bar{\boldsymbol{v}})$ by two ways. First, we search a feasible solution near the approximate solution $(\bar{\boldsymbol{a}}, \bar{\boldsymbol{u}}, \bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}, \bar{\boldsymbol{v}})$. That is, we regard a feasible solution that minimizes $\|\boldsymbol{p}-\bar{\boldsymbol{p}}\|_1$ as a better solution. Second, we calculate a compensation $\boldsymbol{v}$ by using the quality $\bar{\boldsymbol{q}}$ of the solution. Thus, using the auxiliary variable $\boldsymbol{s}$ for $\|\boldsymbol{p}-\bar{\boldsymbol{p}}\|_1$, our proposal is formulated as follows:

$$
\begin{aligned}
\mathrm{FFS}(\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}) \quad \underset{\mathbf{a},\mathbf{u},\mathbf{p},\mathbf{q},\mathbf{v},\mathbf{s}}{\text{minimize}} \quad & w \sum_{t\in T}\sum_{i\in V} s_i^t + \sum_{t\in T}\sum_{(i,j)\in A} CA_{ij}a_{ij}^t + \sum_{t\in T}\sum_{i\in V_P} v_i^t \\
\text{subject to} \quad & -\boldsymbol{s} \le \boldsymbol{p} - \bar{\boldsymbol{p}} \le \boldsymbol{s} \\
& \begin{cases} p_i^{t+1} = p_i^t + SA_i^t - \displaystyle\sum_{k\in E(i)} a_{ik}^t \\ p_i^t \ge 0 \end{cases} && (\forall i \in V_S, \forall t \in T) \\
& \begin{cases} p_i^{t+1} = p_i^t + \displaystyle\sum_{j\in I(i)} a_{ji}^t - \sum_{k\in E(i)} a_{ik}^t \\ p_i^{\min} \le p_i^t \le p_i^{\max} \end{cases} && (\forall i \in V_B, \forall t \in T) \\
& \begin{cases} DC_i^t = \displaystyle\sum_{j\in I(i)} a_{ji}^t \\ DC_i^t q_i^{t+1} = \displaystyle\sum_{j\in I(i)} a_{ji}^t \bar{q}_j^t \\ q_i^t \ge DQ_i^t - v_i^t \\ v_i^t \ge 0 \end{cases} && (\forall i \in V_P, \forall t \in T) \\
& u_{ij}^t L_{ij} \le a_{ij}^t \le u_{ij}^t U_{ij} && (\forall (i,j) \in A, \forall t \in T) \\
& \sum_{j\in I(i)} u_{ji}^t + \sum_{k\in E(i)} u_{ik}^t \le 1 && (\forall i \in V, \forall t \in T) \\
& u_{ij}^t \in \{0,1\} && (\forall (i,j) \in A, \forall t \in T),
\end{aligned}
\tag{22}
$$

where w is a weight coefficient for the term $\|\boldsymbol{p}-\bar{\boldsymbol{p}}\|$ in the objective function. Since the problem FFS$(\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}})$ is MILP, we get the optimal value by an existing an MILP solver. After we obtain the optimal solution $(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{u}}, \tilde{\boldsymbol{p}})$ of the problem FFS$(\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}})$, we can calculate the quality $\tilde{\boldsymbol{q}}$ and the

compensations $\boldsymbol{v}$ as follows,

$$
\begin{cases} \tilde{q}_i^1 &= q_i^{\text{start}} \\ \tilde{q}_i^{t+1} &= \frac{\tilde{p}_i^t \tilde{q}_i^t + SA_i^t SQ_i^t - \sum_{k \in E(i)} \tilde{a}_{ik}^t \tilde{q}_i^t}{\tilde{p}_i^{t+1}} \end{cases} \qquad (\forall i \in V_S, \forall t \in T),
$$

$$
\begin{cases} \tilde{q}_i^1 &= \tilde{q}_i^{\text{start}} \\ \tilde{q}_i^{t+1} &= \frac{p_i^t \tilde{q}_i^t + \sum_{j \in I(i)} a_{ji}^t \tilde{q}_j^t - \sum_{k \in E(i)} a_{ik}^t \tilde{q}_i^t}{p_i^{t+1}} \end{cases} \qquad (\forall i \in V_B, \forall t \in T),
$$

$$
\begin{cases} \tilde{q}_i^{t+1} &= \frac{\sum_{j \in I(i)} a_{ji}^t \tilde{q}_j^t}{DC_i^t} \\ \tilde{v}_i^t &= \max(0, DQ_i^t - \tilde{q}_i^t) \end{cases} \qquad (\forall i \in V_P, \forall t \in T).
$$

3.6 Summary of the proposed approach

We summarize the relations among the problems defined in Subsections 3.1 to 3.4 in Fig.2.

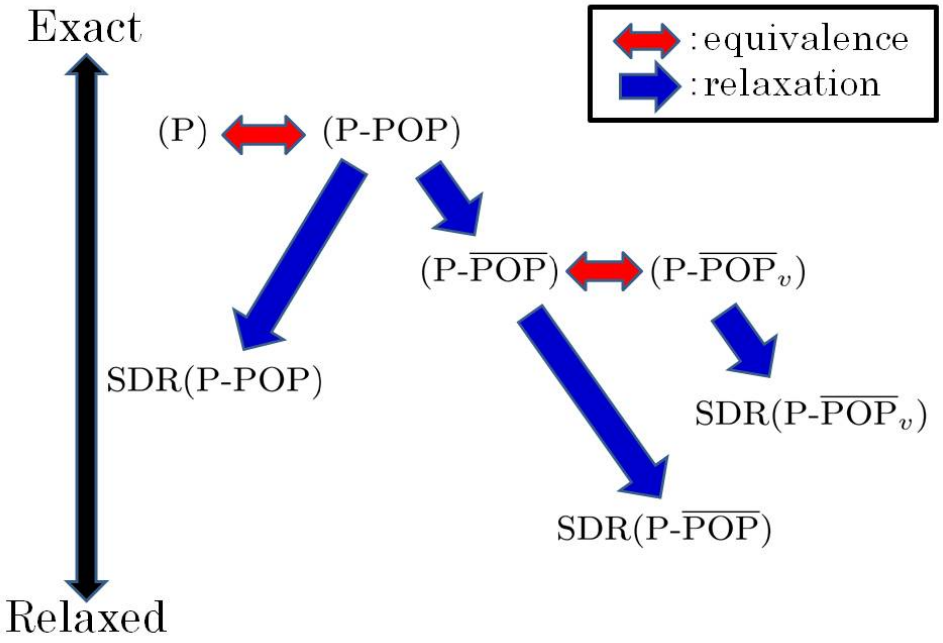

Figure 2: Relations among the problems

Moreover, the following inequalities hold for the optimum values of the problems.

$$\min(\text{P}) = \min(\text{P-POP}) \geq \min(\text{P-}\overline{\text{POP}}) = \min(\text{P-}\overline{\text{POP}}_v) \geq \min(\text{SDR}(\text{P-}\overline{\text{POP}}_v)) \geq \min(\text{SDR}(\text{P-}\overline{\text{POP}})),$$

where $\min(\text{P})$ denotes the optimum value of the problem (P). Note that the inequality $\min(\text{SDR}(\text{P-POP})) \geq \min(\text{SDR}(\text{P-}\overline{\text{POP}}))$ does not always hold.

Now, we give the concrete algorithm as follows.

Proposed Algorithm

Step 0: Set $\boldsymbol{J}_0 = \emptyset$, $\boldsymbol{J}_f = \{(i,j,t) \mid (i,j) \in A, t \in T\}$, and choose a small $\epsilon > 0$ such that $\epsilon < \min_{(i,j)\in A} L_{ij}$.

Step 1: Solve the SDP relaxation problem for NPP($\boldsymbol{J}_f, \boldsymbol{J}_0$) with some valid inequalities mentioned in Subsection 3.4, and obtain its solution $(\bar{\boldsymbol{a}}, \bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}, \bar{\boldsymbol{v}})$.

Step 2: Let $\bar{\boldsymbol{J}}_0 = \{(i,j,t) \in \boldsymbol{J}_f \mid a^t_{ij} \leq \epsilon\}$. If $\bar{\boldsymbol{J}}_0$ is not empty, then set $\boldsymbol{J}_0 = \boldsymbol{J}_0 \cup \bar{\boldsymbol{J}}_0$ and $\boldsymbol{J}_f = \boldsymbol{J}_f \setminus \bar{\boldsymbol{J}}_0$, and go to Step 1.

Step 3: Solve FFS($\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}$) defined in Subsection 3.5, and obtain a feasible solution $(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{p}}, \tilde{\boldsymbol{q}}, \tilde{\boldsymbol{v}})$ of the original problem.

Step 4: Obtain a local optimal solution $(\boldsymbol{a}^*, \boldsymbol{p}^*, \boldsymbol{q}^*, \boldsymbol{v}^*)$ by a general nonlinear programming solver starting from the feasible solution $(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{p}}, \tilde{\boldsymbol{q}}, \tilde{\boldsymbol{v}})$.

The feasible solution $(\tilde{\boldsymbol{a}}, \tilde{\boldsymbol{p}}, \tilde{\boldsymbol{q}}, \tilde{\boldsymbol{v}})$ obtained in Step 3 is not necessarily a local optimum for the pooling problem with the binary variables fixed. Thus, in Step 4, we solve the problem by some local search method.

4 Numerical Experiments

In this section, we present some numerical experiments with the proposed method. The experiments were carried out for eight problems. The network systems of the problems are presented in Figs. 3-6 and the given constants are shown in Tables 9-16. The numbers of their components, discretized time horizons and pipelines are shown in Table 2.

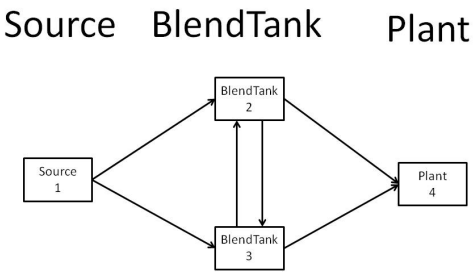

Figure 3: Problems 1, 2

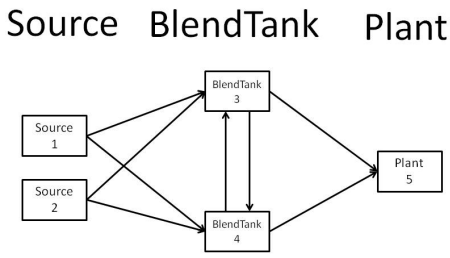

Figure 4: Problems 3, 4

Problems 1 and 2 have the same network system, while they have different time horizons and constants. Similarly, Problems 3 and 4 have different time horizons and the same network. Note that their network is different from that of Problems 1 and 2. Problems 1-4 have only two BlendTanks. Since the requirement of Plant is always positive, one of the tanks should send the final products to the Plant. Thus the pipelines between the tanks are not used due to the constraints (10). On the other hand, Problems 5-8 have more than three BlendTanks. Thus, they may have flows between each pair of BlendTanks. Problems 7 and 8 are relatively large

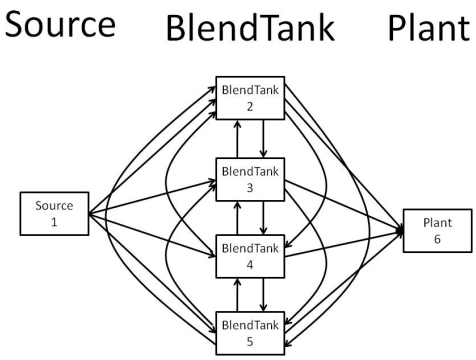

Figure 5: Problems 5, 6

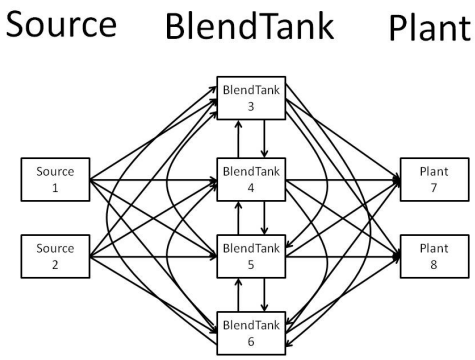

Figure 6: Problems 7, 8

Table 2: The sizes of sets and the numbers of variables for each problem

	M_S	M_B	M_P	M_T	$\|A\|$	$\#\boldsymbol{a}$	$\#\boldsymbol{u}$	$\#\boldsymbol{p}$	$\#\boldsymbol{q}$	$\#\boldsymbol{v}$
Problem 1	1	2	1	10	6	60	60	30	30	10
Problem 2	1	2	1	20	6	120	120	60	60	20
Problem 3	2	2	1	10	8	80	80	40	30	10
Problem 4	2	2	1	20	8	160	160	80	60	20
Problem 5	1	4	1	7	12	84	84	35	35	7
Problem 6	1	4	1	14	20	280	280	70	70	14
Problem 7	2	4	2	28	28	784	784	168	168	56
Problem 8	2	4	2	28	28	784	784	168	168	56

problems. They have 28 discretized time horizons, which correspond to one week with four divisions of one day or about four weeks. Thus they have about 2,000 decision variables. Note that, in all problems, the total of the initial values is more than the value required by Plants in the time horizon, which means that it may be possible to satisfy all requirements.

In numerical experiments, we use a simple algorithm which does not include Step 2, that is, we solved FFS($\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}$) in Step 3 soon after the first SDP relaxation problem was solved.

All algorithms were coded in MATLAB 7.4, and run on a machine with 3.2GHz Pentium 4 CPU and 3.2 GB memory. We generate sparse SDP relaxation problems by SparsePOP [18], and solve the SDP relaxation problems by SeDuMi [16]. Moreover, we solve the MILPs by the ILOG CPLEX® 11.0 solver and local search is carried out by the function **fmincon** in MATLAB which is a solver for nonlinear programming problem.

We investigate effects of the formulation of (P-$\overline{\text{POP}}$) with the valid inequalities proposed in Section 3. We compare the proposed formulation with the naive formulation (P-POP). However, the SDP relaxation problem derived from (P-POP) may become unbounded. Thus the trivial upper and lower bounds (14) and (15) are added to all formulations in order to avoid the unboundedness. Moreover, since some large problems may not be solvable because of numerical instability in the SDP, we scaled all problems in an appropriate manner. Moreover, in each problem, we added the constraints

$$p_i^{M_T} = 0 \ (\forall i \in V_S), \tag{23}$$

which mean that, in the last time period, all the sources must be empty.

First, we compare the naive formulation (P-POP) with the proposed formulation (P-$\overline{\text{POP}}$). Here, we did not add the valid inequalities to (P-$\overline{\text{POP}}$). Table 3 shows the size of the SDP solved in Step 1. For all problems, due to the removal of $\boldsymbol{u}$, the size of the primal SDP is reduced about a half, and that of the dual SDP is drastically reduced.

Table 3: The number of variables of the SDP relaxation problem generated by SparsePOP

	SDR$_1$(P-POP)	SDR$_1$(P-$\overline{\text{POP}}$)
problem 1	(457,2469)	(327,769)
problem 2	(907,4919)	(647,1519)
problem 3	(638,4013)	(448,1023)
problem 4	(1268,8003)	(888,2023)
problem 5	(1805,21637)	(1196,2520)
problem 6	(3597,43239)	(2379,5055)
problem 7	(12054,184568)	(7406,15439)
problem 8	(12054,184569)	(7406,15439)

Two numbers in parentheses represent the number of variables of the primal problem (left) and that of the dual problem (right).

Next, we show the computation times and the objective values in Table 4. The objective value means the final optimum value obtained by the proposed algorithm, that is, the objective function value at the local solution $(\boldsymbol{a}^*, \boldsymbol{u}^*, \boldsymbol{p}^*, \boldsymbol{q}^*, \boldsymbol{v}^*)$ obtained in Step 4. The computational times of the proposed formulation are less than those of the naive formulation due to the smaller SDP sizes. On the other hand, from the viewpoint of the objective value, the proposed formulation

Table 4: Computational times [s] and objective values

	the naive formulation (P-POP)		the proposed formulation (P-$\overline{\text{POP}}$)	
	time	objective value	time	objective value
Problem 1	5.57	16800	3.65	16800
Problem 2	10.3	13100	5.52	13100
Problem 3	5.95	9550	3.44	9550
Problem 4	22.7	17400	14.9	17400
Problem 5	21.4	36300	8.66	29700
Problem 6	60.4	62600	15.6	60200
Problem 7	430	44800	138	70200
Problem 8	449	59900	137	98400

is comparable to the naive formulation. Moreover, it is interesting to note that the objective value of (P-$\overline{\text{POP}}$) for Problems 1, 5 and 6 are much better than those by (P-POP).

Next, we investigate the effects of the valid inequalities proposed in Subsection 3.3. We examined the following four types of valid inequalities.

(a) We add only the trivial upper and lower bounds (14) and (15), which we call the default bound constraints. We refer to the problem (P-$\overline{\text{POP}}$) with these valid inequalities as (P-$\overline{\text{POP}}_{v_a}$).

(b) We add the valid inequalities (17) and (18) for the monomials with degree 2 in addition to the default bound constraints. We refer to the problem (P-$\overline{\text{POP}}$) with these valid inequalities as (P-$\overline{\text{POP}}_{v_b}$).

(c) We add the time dependent upper and lower bounds on $\boldsymbol{p}$ and $\boldsymbol{q}$ by the method proposed in Subsection 3.3 in addition to the default bound constraints. We refer to the problem (P-$\overline{\text{POP}}$) with these valid inequalities as (P-$\overline{\text{POP}}_{v_c}$).

(d) We add the valid inequalities (17) and (18) with the time dependent upper and lower bounds. We refer to the problem (P-$\overline{\text{POP}}$) with these valid inequalities as (P-$\overline{\text{POP}}_{v_d}$).

Table 5 shows the size of the SDP which is solved in Step 1. Since the size of SDR$_1$(P-$\overline{\text{POP}}_{v_a}$) is the same as that of SDR$_1$(P-$\overline{\text{POP}}_{v_c}$), and since SDR$_1$(P-$\overline{\text{POP}}_{v_b}$) is the same as that of SDR$_1$(P-$\overline{\text{POP}}_{v_d}$), we omit the numbers for SDR$_1$(P-$\overline{\text{POP}}_{v_b}$) and SDR$_1$(P-$\overline{\text{POP}}_{v_d}$). Table 6 shows the computational times. We see that, when we added the upper and lower bound of monomials with degree 2, that is the type (b) or (d), the computational time for solving the SDP increases. Moreover, when we used the time dependent upper and lower bounds, that is type (c) or (d), the computational time for solving the MILPs increases. In relatively small problems (Problems 1-6), the computational time required to solve the MILPs is reasonable compared to those required in the other part of computations.

Table 7 shows the objective value obtained in each case. In Table 7, "ratio" represents the ratio of each objective value to the objective value of (P-$\overline{\text{POP}}_{v_a}$). For almost all problems, we could get better solutions by adding valid inequalities.

Table 5: The number of variables of the SDP relaxation problem generated by SparsePOP

	$\mathrm{SDR}_1(\text{P-}\overline{\mathrm{POP}}_{v_a})$	$\mathrm{SDR}_1(\text{P-}\overline{\mathrm{POP}}_{v_b})$
Problem 1	(327, 769)	(898, 5830)
Problem 2	(647, 1519)	(1764, 11403)
Problem 3	(448, 1023)	(1077, 11403)
Problem 4	(888, 2023)	(1077, 6565)
Problem 5	(1196, 2520)	(2021, 12021)
Problem 6	(2379, 5055)	(3537, 25664)
Problem 7	(7406, 15439)	(7106, 50954)
Problem 8	(7406, 15439)	(19164, 136434)

Two numbers in parentheses represent the number of variables of the primal problem (left) and that of the dual problem (right).

Table 6: Computational times [s]

	$\mathrm{SDR}_1(\text{P-}\overline{\mathrm{POP}}_{v_a})$	$\mathrm{SDR}_1(\text{P-}\overline{\mathrm{POP}}_{v_b})$	$\mathrm{SDR}_1(\text{P-}\overline{\mathrm{POP}}_{v_c})$	$\mathrm{SDR}_1(\text{P-}\overline{\mathrm{POP}}_{v_d})$
Problem 1	1.53 (1.53, 0.03)	6.90 (6.90, 0.04)	1.59 (1.59, 0.37)	5.65 (5.65, 0.33)
Problem 2	3.55 (3.55, 0.27)	11.7 (11.7, 0.27)	3.57 (3.57, 16.8)	12.3 (12.3, 15.9)
Problem 3	2.28 (2.28, 0.06)	8.07 (8.07, 0.06)	2.39 (2.39, 0.65)	6.55 (6.55, 0.61)
Problem 4	5.01 (5.01, 0.82)	10.8 (10.8, 0.83)	5.05 (5.05, 78.5)	10.6 (10.6, 73.3)
Problem 5	4.58 (4.58, 0.19)	57.9 (57.9, 0.22)	4.48 (4.48, 0.90)	42.4 (42.4, 0.90)
Problem 6	11.2 (11.2, 2.46)	292 (292, 3.64)	11.6 (11.6, 8.13)	478 (478, 9.82)
Problem 7	58.7 (58.7, 58.2)	229 (229, 58.9)	58.7 (58.7, 177)	228 (228, 178)
Problem 8	58.5 (58.5, 58.1)	229 (229, 58.0)	58.7 (58.7, 179)	228 (228, 180)

The number at the top of each entry represents the total computational time, and two numbers in parentheses represent the computational times of solving SDP (left) and MILP (right)

Table 7: Objective values

		SDR_1(P-$\overline{\mathrm{POP}}_{v_a}$)	SDR_1(P-$\overline{\mathrm{POP}}_{v_b}$)	SDR_1(P-$\overline{\mathrm{POP}}_{v_c}$)	SDR_1(P-$\overline{\mathrm{POP}}_{v_d}$)
Prob. 1	objective value	16800	17800	7940	14100
	ratio (%)	(100)	(106)	(47.2)	(83.7)
Prob. 2	objective value	13100	2250	4560	4560
	ratio (%)	(100)	(17.2)	(34.8)	(34.8)
Prob. 3	objective value	9550	9550	550	5800
	ratio (%)	(100)	(100)	(5.76)	(60.7)
Prob. 4	objective value	17500	14400	16900	14400
	ratio (%)	(100)	(82.1)	(96.7)	(82.1)
Prob. 5	objective value	29700	400	8380	400
	ratio (%)	(100)	(1.35)	(28.2)	(1.35)
Prob. 6	objective value	60300	20900	51500	5040
	ratio (%)	(100)	(34.7)	(85.5)	(8.36)
Prob. 7	objective value	70200	26800	26600	42500
	ratio (%)	(100)	(38.2)	(37.9)	(60.5)
Prob. 8	objective value	98400	57100	101000	43300
	ratio (%)	(100)	(58.0)	(103)	(44.0)

Finally, Table 8 shows a kind of customer satisfaction, which is defined by

$$(\text{satisfaction}) = 100 * \frac{(\text{the total required value}) - (\text{the total insufficiency of the value})}{(\text{the total required value})}.$$

In most cases, we were able to get better satisfactions by the proposed formulations.

Table 8: Customer satisfaction (%)

	SDR_1(PP')	SDR_1(P-$\overline{\mathrm{POP}}_{v_1}$)	SDR_1(P-$\overline{\mathrm{POP}}_{v_2}$)	SDR_1(P-$\overline{\mathrm{POP}}_{v_3}$)	SDR_1(P-$\overline{\mathrm{POP}}_{v_4}$)
problem 1	96.4	96.4	96.2	98.3	97.0
problem 2	98.7	98.7	99.9	99.6	99.6
problem 3	98.0	98.0	98.0	100	98.8
problem 4	98.2	98.2	98.5	98.2	98.5
problem 5	92.4	93.8	100	98.3	100
problem 6	93.5	93.8	97.9	94.7	99.6
problem 7	96.8	94.9	98.1	98.1	97.0
problem 8	95.7	92.8	95.9	92.6	96.9

5 Concluding Remarks

In this paper, we considered to apply the SDP relaxation approach to the pooling problem. We proposed a new formulation of the pooling problem in order to reduce the size of the SDP relaxation problem. Moreover, we proposed some valid inequalities that can tighten the feasible region of the relaxation problem. Finally, we carried out some numerical experiments, which show that the proposed formulations enable us to get better solutions compared to the naive formulation.

We hope that this work contributes to future development of efficient algorithms for the pooling problem. The proposed formulation (22) to get a feasible solution exploits a primal solution of the SDP relation problem, but does not use a dual solution. Thus, by using the dual solution, we may construct a problem that can find a better feasible solution. In addition, more numerical experiments should be conducted in order to examine more practical situations, for example, by using real data of a gas company.

Acknowledgements

The author would like to express his sincerest thanks and appreciation to Associate Professor Nobuo Yamashita for his kind guidance and direction in this study, invaluable discussions, constructive criticisms in the writing of the manuscripts, valuable comments for the presentation, extreme patience, and encouragement. The author wishes to tender his acknowledgments to Professor Masao Fukushima for his constructive comments and kind guidance. The author also wishes to express his thanks to Assistant Professor Shunsuke Hayashi for his invaluable advice. The author greatly appreciates the help of all members of Fukushima Laboratory. Last but not least, precious thanks are due to his family for their strong encouragement and financial support.

References

[1] C. Audet, J. Brimberg, P. Hansen, S. L. Digabel, and N. Mladenovic, *Pooling problem: Alternate formulations and solution methods*, Management Science, 50 (2004), pp. 761–776.

[2] J. Bisschop, *AIMMS - Optimization Modeling*, Lulu.com, 2006.

[3] P. Brucker, *Scheduling Algorithms*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.

[4] K. Dahal, G. Burt, J. NcDonald, and A. Moyes, *A case study of scheduling storage tanks using a hybrid genetic algorithm*, IEEE Transactions on Evolutionary Computation, 5 (2001), pp. 283–294.

[5] M. A. Duran and I. E. Grossmann, *An outer-approximation algorithm for a class of mixed-integer nonlinear programs*, Mathematical Programming, 36 (1986), pp. 307–339.

[6] T. Fujie and M. Kojima, *Semidefinite programming relaxation for nonconvex quadratic programs*, Journal of Global Optimization, 10 (1997), pp. 367–380.

[7] M. Fukuda, M. Kojima, K. Murota, and K. Nakata, *Exploiting sparsity in semidefinite programming via matrix completion, I: General framework*, SIAM Journal on Optimization, 11 (2001), pp. 647–674.

[8] A. M. Geoffrion, *A generalized Benders decomposition*, Journal of Optimization Theory and Applications, 10 (1972), pp. 237–260.

[9] C. A. Haverly, *Studies of the behaviour of recursion for the pooling problem*, ACM SIGMAP Bulletin, 25 (1978), pp. 19–28.

[10] J. B. Lasserre, *An explicit exact sdp relaxation for nonlinear 0-1 programs*, in Proceedings of the 8th International IPCO Conference on Integer Programming and Combinatorial Optimization, London, UK, 2001, Springer-Verlag, pp. 293–303.

[11] J. B. Lasserre, *Global optimization with polynomials and the problems of moments*, SIAM Journal on Optimization, 11 (2001), pp. 796–817.

[12] G. P. McCormick, *Nonlinear Programming: Theory, Algorithms, and Applications*, John Wiley & Sons, Inc., New York, NY, USA, 1983.

[13] Y. Nesterov, *Semidefinite relaxation and nonconvex quadratic optimization*, Optimization Methods and Software, 9 (1998), pp. 141–160.

[14] M. Pinedo, *Scheduling: Theory, Algorithms and Systems Second Ed*, Prentice Hall, 2001.

[15] S. Poljak, F. Rendl, and H. Wolkowicz, *A recipe for semidefinite relaxation for (0,1)-quadratic programming*, Journal of Global Optimization, 7 (1995), pp. 51–73.

[16] J. F. Sturm, *Using SeDuMi 1.02, A MATLAB toolbox for optimization over symmetric cones*, Optimization Methods and Software, 11-12 (1999), pp. 625–653.

[17] H. Waki, S. Kim, M. Kojima, and M. Muramatsu, *Sums of squares and semidefinite programming relaxations for polynomial optimization problems with structured sparsity*, SIAM Journal on Optimization, 17 (2006), pp. 218–242.

[18] H. Waki, S. Kim, M. Kojima, M. Muramatsu, and H. Sugimoto, *Algorithm 883: Sparsepop—a sparse semidefinite programming relaxation of polynomial optimization problems*, ACM Transactions on Mathematical Software, 35 (2008), pp. 1–13.

A Problem Data for Numerical Experiments

Table 9: The constants of Problem 1

constant		value
M_T		10
L_{ij}	$(\forall (i,j) \in A)$	10
U_{ij}	$(\forall (i,j) \in A)$	30
CA_{ij}	$(\forall (i,j) \in A)$	1
p_1^0		200
q_1^0		20
SA_1^t	$(\forall\ t\ \in\ T)$	0
SQ_1^t	$(\forall\ t\ \in\ T)$	0
(p_2^0, p_3^0)		(100,100)
(q_2^0, q_3^0)		(20,10)
$(p_2^{\max}, p_3^{\max})$		(200,200)
$(p_2^{\min}, p_3^{\min})$		(10,10)
CQ_4		100
DC_4^t	$(\forall\ t\ \in\ T)$	30
DQ_4^t	$(\forall\ t\ \in\ T)$	15

Table 10: The constants of Problem 2

constant		value
M_T		20
L_{ij}	$(\forall (i,j) \in A)$	10
U_{ij}	$(\forall (i,j) \in A)$	30
CA_{ij}	$(\forall (i,j) \in A)$	1
p_1^0		500
q_1^0		20
SA_1^t	$(\forall\ t\ \in\ T)$	0
SQ_1^t	$(\forall\ t\ \in\ T)$	0
(p_2^0, p_3^0)		(100,100)
(q_2^0, q_3^0)		(17,13)
$(p_2^{\max}, p_3^{\max})$		(200,200)
$(p_2^{\min}, p_3^{\min})$		(10,10)
CQ_4		100
DC_4^t	$(\forall\ t\ \in\ T)$	30
DQ_4^t	$(\forall\ t\ \in\ T)$	15

Table 11: The constants of Problem 3

constant		value
M_T		10
L_{ij}	$(\forall (i,j) \in A)$	10
U_{ij}	$(\forall (i,j) \in A)$	30
CA_{ij}	$(\forall (i,j) \in A)$	1
(p_1^0, p_2^0)		(150,0)
(q_1^0, q_2^0)		(25,0)
SA_1^t	$(\forall\ t\ \in\ T)$	0
SQ_1^t	$(\forall\ t\ \in\ T)$	0
SA_2^5		100
SA_2^t	$(\forall\ t\ \in\ T \setminus 5)$	0
SQ_2^5		10
SQ_2^t	$(\forall\ t\ \in\ T \setminus 5)$	0
(p_3^0, p_4^0)		(100,100)
(q_3^0, q_4^0)		(17,12)
$(p_3^{\max}, p_4^{\max})$		(200,200)
$(p_3^{\min}, p_4^{\min})$		(10,10)
CQ_5		100
DC_5^t	$(\forall\ t\ \in\ T)$	30
DQ_5^t	$(\forall\ t\ \in\ T)$	15

Table 12: The constants of Problem 4

constant		value
M_T		20
L_{ij}	$(\forall (i,j) \in A)$	10
U_{ij}	$(\forall (i,j) \in A)$	30
CA_{ij}	$(\forall (i,j) \in A)$	1
(p_1^0, p_2^0)		(300,0)
(q_1^0, q_2^0)		(25,0)
SA_1^t	$(\forall\ t\ \in\ T)$	0
SQ_1^t	$(\forall\ t\ \in\ T)$	0
SA_2^4		250
SA_2^t	$(\forall\ t\ \in\ T \setminus 10)$	0
SQ_2^4		10
SQ_2^t	$(\forall\ t\ \in\ T \setminus 10)$	0
(p_3^0, p_4^0)		(100,100)
(q_3^0, q_4^0)		(17,12)
$(p_3^{\max}, p_4^{\max})$		(200,200)
$(p_3^{\min}, p_4^{\min})$		(10,10)
CQ_5		100
DC_5^t	$(\forall\ t\ \in\ T)$	30
DQ_5^t	$(\forall\ t\ \in\ T)$	15

Table 13: The constants of Problem 5

constant		value
M_T		7
L_{ij}	$(\forall (i,j) \in A)$	10
U_{ij}	$(\forall (i,j) \in A)$	40
CA_{ij}	$(\forall (i,j) \in A)$	1
p_1^0		100
q_1^0		20
SA_1^t	$(\forall\ t\ \in\ T)$	0
SQ_1^t	$(\forall\ t\ \in\ T)$	0
$(p_2^0, p_3^0, p_4^0, p_5^0)$		(100,100,80,80)
$(q_2^0, q_3^0, q_4^0, q_5^0)$		(20,10,15,20)
$(p_2^{\max}, p_3^{\max}, p_4^{\max}, p_5^{\max})$		(200,200,200,200)
$(p_2^{\min}, p_3^{\min}, p_4^{\min}, p_5^{\min})$		(10,10,10,10)
CQ_6		100
DC_6^t	$(\forall\ t\ \in\ T)$	40
DQ_6^t	$(\forall\ t\ \in\ T)$	17

Table 14: The constants of Problem 6

constant		value
M_T		14
L_{ij}	$(\forall (i,j) \in A)$	10
U_{ij}	$(\forall (i,j) \in A)$	40
CA_{ij}	$(\forall (i,j) \in A)$	1
p_1^0		300
q_1^0		20
SA_1^t	$(\forall\ t\ \in\ T)$	0
SQ_1^t	$(\forall\ t\ \in\ T)$	0
$(p_2^0, p_3^0, p_4^0, p_5^0)$		(100,100,80,80)
$(q_2^0, q_3^0, q_4^0, q_5^0)$		(20,10,15,20)
$(p_2^{\max}, p_3^{\max}, p_4^{\max}, p_5^{\max})$		(200,200,200,200)
$(p_2^{\min}, p_3^{\min}, p_4^{\min}, p_5^{\min})$		(10,10,10,10)
CQ_6		100
DC_6^t	$(\forall\ t\ \in\ T)$	40
DQ_6^t	$(\forall\ t\ \in\ T)$	17

Table 15: The constants of Problem 7

constant		value
M_T		28
L_{ij}	$(\forall (i,j) \in A)$	10
U_{ij}	$(\forall (i,j) \in A)$	30
CA_{ij}	$(\forall (i,j) \in A)$	1
(p_1^0, p_2^0)		(400,0)
(q_1^0, q_2^0)		(18,0)
SA_1^t	$(\forall\ t\ \in\ T)$	0
SQ_1^t	$(\forall\ t\ \in\ T)$	0
SA_2^{14}		400
SA_2^t	$(\forall\ t\ \in\ T \setminus 14)$	0
SQ_2^{14}		13
SQ_2^t	$(\forall\ t\ \in\ T \setminus 14)$	0
$(p_3^0, p_4^0, p_5^0, p_6^0)$		(100,100,100,100)
$(q_3^0, q_4^0, q_5^0, q_6^0)$		(20,17,15,13)
$(p_3^{\max}, p_4^{\max}, p_5^{\max}, p_6^{\max})$		(300,300,300,300)
$(p_3^{\min}, p_4^{\min}, p_5^{\min}, p_6^{\min})$		(10,10,10,10)
(CQ_7, CQ_8)		(100,100)
(DC_7^t, DC_8^t)	$(\forall\ t\ \in\ T)$	(20,10)
(DQ_7^t, DQ_8^t)	$(\forall\ t\ \in\ T)$	(15,18)

Table 16: The constants of Problem 8

constant		value
M_T		28
L_{ij}	$(\forall (i,j) \in A)$	10
U_{ij}	$(\forall (i,j) \in A)$	30
CA_{ij}	$(\forall (i,j) \in A)$	1
(p_1^0, p_2^0)		(400,0)
(q_1^0, q_2^0)		(13,0)
SA_1^t	$(\forall\ t\ \in\ T)$	0
SQ_1^t	$(\forall\ t\ \in\ T)$	0
SA_2^{14}		400
SA_2^t	$(\forall\ t\ \in\ T \setminus 14)$	0
SQ_2^{14}		18
SQ_2^t	$(\forall\ t\ \in\ T \setminus 14)$	0
$(p_3^0, p_4^0, p_5^0, p_6^0)$		(100,100,100,100)
$(q_3^0, q_4^0, q_5^0, q_6^0)$		(20,17,15,13)
$(p_3^{\max}, p_4^{\max}, p_5^{\max}, p_6^{\max})$		(300,300,300,300)
$(p_3^{\min}, p_4^{\min}, p_5^{\min}, p_6^{\min})$		(10,10,10,10)
(CQ_7, CQ_8)		(100,100)
(DC_7^t, DC_8^t)	$(\forall\ t\ \in\ T)$	(20,10)
(DQ_7^t, DQ_8^t)	$(\forall\ t\ \in\ T)$	(15,18)