Solving Pooling Problems by LP and SOCP Relaxations and Rescheduling Methods

Masaki Kimizuka[*] Sunyoung Kim[†] Makoto Yamashita[‡]

February, 2018

Abstract

The pooling problem is an important industrial problem in the class of network flow problems for allocating gas flow in pipeline transportation networks. For P-formulation of the pooling problem with time discretization, we propose second order cone programming (SOCP) and linear programming (LP) relaxations and prove that they obtain the same optimal value as the semidefinite programming relaxation. The equivalence among the optimal values of the three relaxations is also computationally shown. Moreover, a rescheduling method is proposed to efficiently refine the solution obtained by the SOCP or LP relaxation. The efficiency of the SOCP and the LP relaxation and the proposed rescheduling method is illustrated with numerical results on the test instances from the work of Nishi in 2010, some large instances, and Foulds 3, 4, 5 test problems.

Key words. Pooling problem, Semidefinite relaxation, Second order cone relaxation, Linear programming relaxation, Rescheduling method, Computational efficiency.

AMS Classification. 90C20, 90C22, 90C25, 90C26.

1 Introduction

The pooling problem is a network flow problem for allocating gas flow in pipeline transportation networks with minimum cost. It arises from applications in the petroleum industry. Networks of the pooling problem have three types of nodes: sources, blending tanks called pools and plants. Gas flows from the sources are blended in the blending tanks and plants to produce the desired final products. To model such blending, the pooling problem

[*]Department of Mathematical and Computing Science, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552, Japan (kimi3masa0@gmail.com).

[†]Department of Mathematics, Ewha W. University, 52 Ewhayeodae-gil, Sudaemoon-gu, Seoul 03760, Korea (skim@ewha.ac.kr). The research was supported by NRF 2017-R1A2B2005119.

[‡]Department of Mathematical and Computing Science, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguro-ku, Tokyo 152-8552, Japan (makoto.yamashita@is.titech.ac.jp). This research was partially supported by JSPS KAKENHI (Grant number: 15k00032).

is formulated as bilinear nonconvex optimization problems, thus nonconvex quadratically constrained quadratic problems (QCQPs) [11].

The pooling problem has been studied in two main formulations, the P-formulation [11] and Q-formulation [9]. Difficulties of solving the pooling problem arise from the existence of pipeline constraints formulated with binary variables and the blending process represented as nonlinear constraints. The resulting problem is a mixed-integer nonlinear program known as NP-hard [6]. Various solution methods including approximating heuristics, linear programming relaxations, decomposition techniques have been proposed for these formulations. In particular, successive linear relaxations [10, 14] approximate the pooling problem based on a first-order Taylor expansion. Another popular approach is branch-and-bound algorithms which have been implemented to solve large-scale problems [6].

While conic relaxations methods including semidefinite programming (SDP), second order cone (SOCP) and linear programming (LP) relaxations of general nonconvex QCQPs have been widely used to approximate the optimal values of the problems, they have not studied extensively for the pooling problem. In fact, no literature on SOCP and LP relaxations of the pooling problem could be found to the authors' best knowledge. SDP relaxations of the pooling problem were studied in [16, 22]. SDP relaxations of nonconvex QCQPs are known to provide tighter bounds for the optimal value than SOCP and LP relaxations, however, solving SDP relaxations by the primal-dual interior-point methods [17, 18, 20, 21] is computationally expensive. Thus, the size of the problems that can be solved by SDP relaxations remains very limited. From a computational perspective, SOCP and LP relaxations are more efficient than SDP relaxations, as a result, large-sized problems can be solved by SOCP and LP relaxations [12].

The main purpose of this paper is to propose an efficient computational method that employs LP and SOCP relaxations and a rescheduling method for the P-formulation of the pooling problem with time discretization. The LP relaxation of nonconvex QCQPs in this paper is different from the linear programming based on a first-order Taylor expansion [10, 14]. Let ζ^* be the optimal value of a general nonconvex QCQP that minimizes the objective function. Among ζ^* and the optimal values of SDP, SOCP and LP relaxations of the QCQP, we have the following relationship:

$$\zeta^*_{LP} \leq \zeta^*_{SOCP} \leq \zeta^*_{SDP} \leq \zeta^*.$$

LP relaxations are known to be most efficient and SDP relaxations most time-consuming among the three relaxations for solving general QCQPs. For our formulation of the pooling problem, we prove that the SDP, SOCP and LP relaxations provide the equivalent optimal value:

$$\zeta^*_{LP} = \zeta^*_{SOCP} = \zeta^*_{SDP} \leq \zeta^*. \tag{1}$$

More precisely, the LP relaxation can be used to obtain the same quality of the optimal value as that of the SDP relaxation with much less computational efforts. Thus, larger pooling problems can be handled with the LP relaxation. We theoretically prove the equivalence (1) and present the computational results that support (1). Moreover, we demonstrate that (1) holds for other formulations of the pooling problem where bilinear terms appear with no squared terms of the variables. The LP relaxation presented in this paper can be used to efficiently solve different formulations of the pooling problem.

The solution obtained by the LP, SOCP and SDP relaxations of the pooling problem should be refined to satisfy all the constraints of the original problem. For this issue, Nishi [16] proposed a method that solves a mixed-integer linear program using the solution obtained by the SDP relaxation, then applies an iterative procedure for the nonlinear terms, and finally uses a nonlinear program solver to attain a local optimal solution. In the three steps of his method, the nonlinear program solver particularly takes long computational time, making the entire method very time-consuming. To reduce the computational burden caused by applying a nonlinear program solver, we propose a rescheduling method which successively updates a local optimal solution by applying the SOCP or LP relaxation to partial time steps of the entire time discretization. The proposed technique significantly increases the computational efficiency of the entire method, which enables us to solve large pooling problems. For one test instance with 1228 variables, the rescheduling method reduced the computational time for a general nonlinear solver by 1/60.

This paper is organized as follows: Section 2 briefly describes the pooling problem. In Section 3, we illustrate SDP, SOCP, and LP relaxations of the general nonconvex QCQPs. Section 4 includes the proof of the optimal values on the three relaxations of our formulation of the pooling problem and discusses how the result can be applied to other formulations of the pooling problem. In Section 5, we describe the proposed rescheduling methods in detail. Section 6 presents numerical results on the test problems in [16], some large instances and Foulds 3, 4, 5. We conclude in Section 7.

2 The pooling problem

We first describe the formulation of the pooling problem with time discretization in [16], then present our formulation.

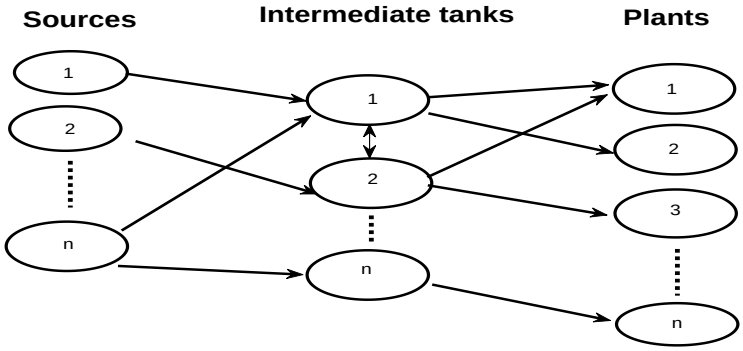

Figure 1: Overview

2.1 Notation

Let M_S, M_I and M_P denote the numbers of sources, intermediate tanks, and plants, respectively. We also let V be the set of all nodes. The sets of sources, intermediate tanks and

plants are denoted by V_S, V_I and V_P, respectively, as follows:

$$\begin{aligned} V_S &= \{1, 2, \ldots, M_S\}, \quad V_I = \{M_S + 1, \ldots, M_S + M_I\}, \\ V_P &= \{M_S + M_I + 1, \ldots, M_S + M_I + M_P\}, \; V = V_S \cup V_I \cup V_P. \end{aligned}$$

The arrows in Figure 2 mean pipelines. The pipeline between i and $j \in V$ is denoted as (i, j), and the set of pipelines is denoted as A. Furthermore, for the ith node, the set of entering nodes and that of leaving nodes are denoted, respectively, as

$$I(i) = \{j \in V | (j, i) \in A\}, \; E(i) = \{k \in V | (i, k) \in A\}.$$

We use M_T to mean the number of time discretization and each time slot can be identified by $t \in T = \{1, \ldots, M_T\}$.

As our formulation is based on time discretization, p_i^t denotes the quantity stored in $i \in V$, and q_i^t the quality of the ith node at time t. The flow in the pipeline (i, j) is denoted as a_{ij}^t, and binary variable u_{ij}^t means whether the pipeline (i, j) is used at time t or not. We also introduce v_i^t $(i \in V_P)$ to evaluate the quality shortage for the requirement at the plant $i \in V_P$, at time t. The variables, constants and sets are summarized in Table 1.

2.2 Problem formulation with time discretization

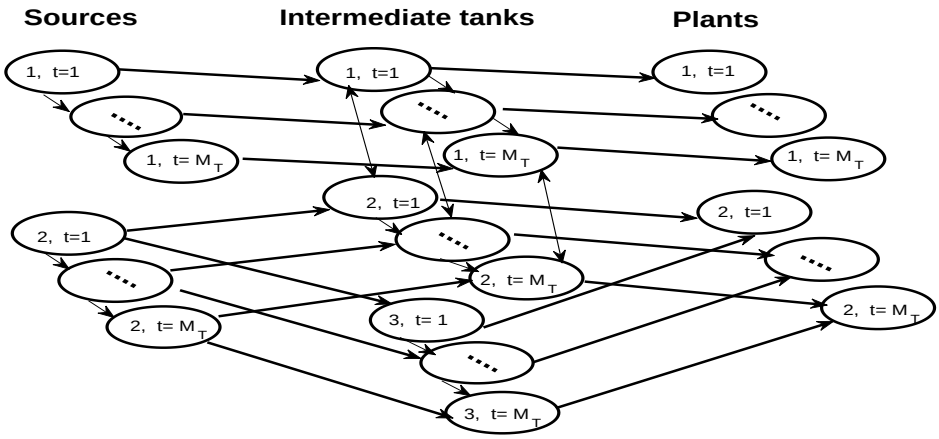

Figure 2: The pooling problem with time discretization

The pooling problem has been represented with various formulations. We formulate P-formulation with time discretization as a nonconvex mixed-integer QCQP. As shown in Figure 2, the pooling problem with time discretization can be viewed as a network with the arcs connecting sources, intermediate tanks, and plants at the same time step.

The objective function of the pooling problem can be modeled as follows:

$$\min_{a,p,q,v} \sum_{t \in T} \sum_{(i,j) \in A} CA_{ij} a_{ij}^t + \sum_{t \in T} \sum_{i \in V_P} CQ_i RC_i^t v_i^t.$$

Here CA_{ij} is the transportation cost for the pipeline (i, j), CQ_i the penalty cost for the shortage at the ith node, and RC_i^t the required quantity at the ith node. The first and

Table 1: The sets, constants and variables of P formulation

Sets			
V_S	the set of sources	V_I	the set of intermediate tanks
V_P	the set of plants	(i,j)	the pipeline between i and j
$I(i)$	the set of entering nodes to the ith nodes		
$E(i)$	the set of leaving nodes from the ith nodes		
Constants			
M_S	the number of sources	M_I	the number of intermediate tanks
M_P	the number of plants	M_T	the number of time discretization
$p_i^{\min}$	the minimum quantity	$p_i^{\max}$	the maximum quantity
SA_i^t	the supply quantity	SQ_i^t	the supply quality
U_{ij}	the maximum flow	L_{ij}	the minimun flow
RC_i^t	the required quantity	RQ_i^t	the required quality
CA_{ij}	the transportation cost for (i,j)	CQ_i	the penalty cost
Variables for the ith node at time t			
a_{ij}^t	flow in the pipeline (i,j)	p_i^t	the quantity
q_i^t	the quality	u_{ij}^t	binary variables
v_i^t	the quality shortage		

second terms of the objective function represent the transportation cost and the penalty cost. For constraints, v_i^t is introduced to denote the shortage in quality at the ith node. If RQ_i^t is used to denote the required quality at the ith node, then

$$v_i^t = \max\{0, RQ_i^t - q_i^t\} \quad (i \in V_P \ , \ t \in T).$$

At each time t, each node can be connected to at most one pipeline, therefore we must have

$$u_{ij}^t \in \{0,1\}, \quad \sum_{j \in I(i)} u_{ji}^t + \sum_{k \in E(i)} u_{ik}^t \leq 1 \ \ (i \in V, \ (i,j) \in A, \ t \in T = \{1, \ldots, M_T\}).$$

The flow of each pipeline has a lower and upper bound,

$$u_{ij}^t L_{ij} \leq a_{ij}^t \leq u_{ij}^t U_{ij} \ ((i,j) \in A, t \in T),$$

where L_{ij} and U_{ij} are the lower bound and upper bound for the flow in the pipeline (i,j).

Two constraints can be derived from mixing two kinds of oil with different quantity p_i^t and quality q_i^t. For instance, we consider mixing oil 1 and 2 to produce new oil 3 in Figure 3. Assume that each node has the quantity, p_1, p_2 and p_3 and the quality, q_1, q_2 and q_3. The

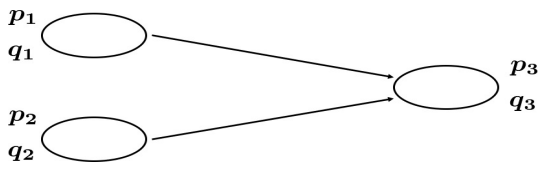

Figure 3: Flow of mixing oil

constraint for the amount of new oil p_3 is that it should be equivalent to the sum of oil 1 and 2, *i.e.,* $p_3 = p_1 + p_2$. Second, the quality of new oil q_3 should be computed by the weighted average of oil 1 and 2, *i.e.,* $p_3 q_3 = p_1 q_1 + p_2 q_2$ holds. Thus, necessary constraints for the pooling problem are

$$
\begin{aligned}
p_i^{t+1} = p_i^t + SA_i^t - \sum_{k \in E(i)} a_{ik}^t, \quad p_i^t \geq 0, \quad p_i^{M_T+1} = 0 \quad & (i \in V_S, t \in T), \\
p_i^{t+1} q_i^{t+1} = p_i^t q_i^t + SA_i^t SQ_i^t - \sum_{k \in E(i)} a_{ik}^t q_i^t \quad & (i \in V_S, t \in T), \\
p_i^{t+1} = p_i^t + \sum_{k \in I(i)} a_{ki}^t - \sum_{k \in E(i)} a_{ik}^t, \quad p_i^{\min} \leq p_i^t \leq p_i^{\max} \quad & (i \in V_I, t \in T), \\
p_i^{t+1} q_i^{t+1} = p_i^t q_i^t + \sum_{k \in I(i)} a_{ki}^t q_k^t - \sum_{k \in E(i)} a_{ik}^t q_i^t \quad & (i \in V_I, t \in T), \\
q_i^t = \frac{1}{RC_i^t} \sum_{j \in I(i)} a_{ji}^t q_j^t \quad & (i \in V_P, t \in T).
\end{aligned}
$$

Here, SA_i^t and SQ_i^t are the supplied quantity and quality at the source $i \in V_S$. The constraint $p_i^{M_T+1} = 0$ for $i \in V_S$ requires the quantity at the sources should be empty at time M_{T+1}.

We describe the formulation of the pooling problem in [16] as follows:

$$
\begin{aligned}
(PP) \qquad \min_{a,p,q,u,v} \quad & \sum_{t \in T} \sum_{(i,j) \in A} CA_{ij} a_{ij}^t + \sum_{t \in T} \sum_{i \in V_P} CQ_i RC_i^t v_i^t \\
\text{subject to} \quad & u_{ij}^t L_{ij} \leq a_{ij}^t \leq u_{ij}^t U_{ij} \quad ((i,j) \in A, t \in T), \\
& u_{ij}^t \in \{0,1\}, \quad \sum_{j \in I(i)} u_{ji}^t + \sum_{k \in E(i)} u_{ik}^t \leq 1 \quad (i \in V,\ (i,j) \in A,\ t \in T), \\
& p_i^{t+1} = p_i^t + SA_i^t - \sum_{k \in E(i)} a_{ik}^t, \quad p_i^t \geq 0\ ,\ p^{M_T+1} = 0 \quad (i \in V_S, t \in T), \\
& p_i^{t+1} q_i^{t+1} = p_i^t q_i^t + SA_i^t SQ_i^t - \sum_{k \in E(i)} a_{ik}^t q_i^t \quad (i \in V_S, t \in T), \\
& p_i^{t+1} = p_i^t + \sum_{k \in I(i)} a_{ki}^t - \sum_{k \in E(i)} a_{ik}^t, \quad p_i^{\min} \leq p_i^t \leq p_i^{\max} \quad (i \in V_I, t \in T), \\
& p_i^{t+1} q_i^{t+1} = p_i^t q_i^t + \sum_{k \in I(i)} a_{ki}^t q_k^t - \sum_{k \in E(i)} a_{ik}^t q_i^t \quad (i \in V_I, t \in T), \\
& q_i^t = \frac{1}{RC_i^t} \sum_{j \in I(i)} a_{ji}^t q_j^t \quad (i \in V_P, t \in T), \\
& v_i^t \geq \max\{0, RQ_i^t - q_i^t\} \quad (i \in V_P, t \in T).
\end{aligned}
$$

Note that some of the above constraints are quadratic and nonconvex. With nonconvex constraints and the binary variables $u_{ij}^t \in \{0,1\}$, the formulation (PP) of the pooling problem is a nonconvex mixed-integer nonlinear programming problem. Each quadratic term of the formulation of the pooling problem is always bilinear, and no squared terms of

variables appear in the constraints. Even if u can be removed, the problem is nonconvex, as a result, it is difficult to apply an existing mixed-integer nonlinear programming method. It is known that global optimum solutions cannot be obtained within reasonable time, since the pooling problem has been shown to be NP-hard [5].

Eliminating binary variables

In [16], the pipeline constraints were modified to remove the binary variables u before applying the SDP relaxation problem. We briefly describe the elimination of the binary variables. The constraints involving the binary variables were rewritten with a_{ij}^t using the relation between u_{ij}^t and a_{ij}^t. More precisely, the constraints given by

$$\sum_{j\in I(i)} u_{ji}^t + \sum_{k\in E(i)} u_{ik}^t \le 1 \quad (i\in V, \quad t\in T), \quad u_{ij}^t \in \{0,1\} \quad ((i,j)\in A, \quad t\in T).$$

require that at most one pipeline for all $i \in V$ and $t \in T$ should be used. Thus, an equivalent constraint can be described in terms of $\boldsymbol{a}$ as follows:

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

To remove the binary variables from $u_{ij}^t L_{ij} \le a_{ij}^t \le u_{ij}^t U_{ij}$ $((i,j)\in A, \quad t\in T)$, the lower bound on a_{ij}^t was modified to the following nonnegativity,

$$0 \le a_{ij}^t \le U_{ij} \quad ((i,j)\in A, \quad t\in T). \tag{2}$$

As a result, the following problem is derived:

$$\begin{aligned}
\min_{a,p,q,v} \quad & \sum_{t\in T}\sum_{(i,j)\in A} CA_{ij} a_{ij}^t + \sum_{t\in T}\sum_{i\in V_P} CQ_i RC_i^t v_i^t \\
\text{subject to} \quad & \sum_{j,k\in I(i), j\neq k} a_{ji}^t a_{ki}^t + \sum_{j,k\in E(i), j\neq k} a_{ij}^t a_{ik}^t + \sum_{j,k\in I(i), j\neq k} a_{ji}^t a_{ki}^t = 0 \quad (i\in V, \quad t\in T), \\
& 0 \le a_{ij}^t \le U_{ij}, \quad ((i,j)\in A, \quad t\in T) \\
& p_i^{t+1} = p_i^t + SA_i^t - \sum_{k\in E(i)} a_{ik}^t \ , \ p_i^t \ge 0 \ , \ p^{M_T+1} = 0 \quad (i\in V_S, t\in T), \\
& p_i^{t+1} q_i^{t+1} = p_i^t q_i^t + SA_i^t SQ_i^t - \sum_{k\in E(i)} a_{ik}^t q_i^t \quad (i\in V_S, t\in T), \\
& p_i^{t+1} = p_i^t + \sum_{k\in I(i)} a_{ki}^t - \sum_{k\in E(i)} a_{ik}^t \quad p_i^{\min} \le p_i^t \le p_i^{\max} \quad (i\in V_I, t\in T), \\
& p_i^{t+1} q_i^{t+1} = p_i^t q_i^t + \sum_{k\in I(i)} a_{ki}^t q_k^t, - \sum_{k\in E(i)} a_{ik}^t q_i^t \quad (i\in V_I, t\in T), \\
& q_i^t = \frac{1}{RC_i^t} \sum_{j\in I(i)} a_{ji}^t q_j^t \quad (i\in V_P, t\in T), \\
& v_i^t \ge \max\{0, RQ_i^t - q_i^t\} \quad (i\in V_P, t\in T).
\end{aligned} \tag{3}$$

While the number of constraints in (3) is the same as the number of constraints of the original pooling problem, the number of variables in (3) is small compared to the original pooling problem. Thus, (3) can be solved more efficiently by conic relaxation methods than the original pooling problem.

2.3 The proposed formulation

Although the modified problem (3) in [16] has reduced the number of variables and no binary variables, (3) may not have an interior point. If SDP relaxations are used to solve problems with no interior point, as in [16], SDP solvers based on primal-dual interior-pont methods [17, 18, 20, 21] frequently fail due to numerical instability. To avoid such numerical difficulty, we relax the equality to inequalities. For instance, we first transform equality constraints of the form $a^T x = b$ into $-\lambda \le a^T x - b \le \lambda$ introducing by a new variable λ. Then, we add λ to the objective function as a penalty function.

Our formulation of the pooling problem is:

$$
\begin{aligned}
\min_{a,p,q,v} \quad & \sum_{t\in T}\sum_{(i,j)\in A} CA_{ij}a_{ij}^t + \sum_{t\in T}\sum_{i\in V_P} CQ_i RC_i^t v_i^t + \delta \sum_{t\in T}\sum_{(i,j)\in A} \lambda_i^t \\
\text{subject to} \quad & -\lambda_i^t \le \sum_{j,k\in I(i),j\ne k} a_{ji}^t a_{ki}^t + \sum_{j,k\in E(i),j\ne k} a_{ij}^t a_{ik}^t + \sum_{j,k\in I(i),j\ne k} a_{ji}^t a_{ki}^t \le \lambda_i^t \quad (i\in V,\ \ t\in T), \\
& 0 \le a_{ij}^t \le U_{ij} \quad ((i,j)\in A,\ \ t\in T), \\
& -\lambda_i^t \le -p_i^{t+1} + p_i^t + SA_i^t - \sum_{k\in E(i)} a_{ik}^t \le \lambda_i^t, \ \ p_i^t \ge 0, \ \ p^{M_T+1} = 0 \quad (i\in V_S, t\in T), \\
& -\lambda_i^t \le -p_i^{t+1}q_i^{t+1} + p_i^t q_i^t + SA_i^t SQ_i^t - \sum_{k\in E(i)} a_{ik}^t q_i^t \le \lambda_i^t \quad (i\in V_S,\ t\in T), \\
& -\lambda_i^t \le -p_i^{t+1} + p_i^t + \sum_{k\in I(i)} a_{ki}^t - \sum_{k\in E(i)} a_{ik}^t \le \lambda_i^t, \quad p_i^{\min} \le p_i^t \le p_i^{\max} \quad (i\in V_I,\ t\in T), \\
& -\lambda_i^t \le -p_i^{t+1}q_i^{t+1} + p_i^t q_i^t + \sum_{k\in I(i)} a_{ki}^t q_k^t - \sum_{k\in E(i)} a_{ik}^t q_i^t \le \lambda_i^t \quad (i\in V_I,\ t\in T), \\
& -\lambda_i^t \le -q_i^t + \frac{1}{RC_i^t}\sum_{j\in I(i)} a_{ji}^t q_j^t \le \lambda_i^t \quad (i\in V_P,\ t\in T), \\
& v_i^t \ge \max\{0, RQ_i^t - q_i^t\} \quad (i\in V_P,\ t\in T), \quad \lambda_i^t \ge 0\ (i\in V,\ t\in T),
\end{aligned} \tag{4}
$$

where δ is a penalty parameter.

For the subsequent discussion, we express (4) using variable $\boldsymbol{x}$ defined as $\boldsymbol{x} = \{\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{q}, \boldsymbol{v}\}$.

More precisely, each set of variables are ordered in the following order.

$$
\begin{aligned}
\boldsymbol{a} &= \{a^1, \ldots, a^{M_T}\},\\
\boldsymbol{a}^t &= \{a^t_{ij} \mid (i,j) \in A\} \quad (t \in T),\\
\boldsymbol{p} &= \{p^2, \ldots, p^{M_T}, p^{M_T+1}\},\\
\boldsymbol{p}^t &= \{p^t_i,\ p^t_j \mid i \in V_S,\ j \in V_I\} \quad (t \in T \setminus \{1\}),\\
\boldsymbol{p}^{M_T+1} &= \{p^{M_T+1}_i \mid i \in V_I\},\\
\boldsymbol{q} &= \{q^1, \ldots, q^{M_T+1}\},\\
\boldsymbol{q}^1 &= \{q^1_i \mid i \in V_P\},\\
\boldsymbol{q}^t &= \{q^t_i,\ q^t_j,\ q^t_k \mid i \in V_S,\ j \in V_I,\ k \in V_P\} \quad (t \in T \setminus \{1\}),\\
\boldsymbol{q}^{M_T+1} &= \{q^{M_T+1}_i \mid i \in V_I\},\\
\boldsymbol{v} &= \{v^1, \ldots, v^{M_T}\},\\
\boldsymbol{v}^t &= \{v^t_i \mid i \in V_P\} \quad (t \in T).
\end{aligned}
$$

Let n be the length of $\boldsymbol{x}$, that is, $\boldsymbol{x} \in \mathbb{R}^n_+$ where $\mathbb{R}^n_+$ denotes the space of n-dimensional column vectors of nonnegative numbers. We also let $\mathbb{R}^n$ be the space of n-dimensional column vectors $\mathbb{R}^{d\times n}$ the space of $d \times n$ real matrices, and $\mathbb{S}^n$ the space of $n \times n$ symmetric matrices.

Let the number of the quadratic equalities of (4) is m, the number of linear equality constraints d, the number of linear inequalities e. Then, with appropriately chosen matrices $\boldsymbol{Q}_k \in \mathbb{S}^n$ $(k = 1, \ldots, m)$, $\boldsymbol{L}_{m+1} \in \mathbb{R}^{d\times n}, \boldsymbol{L}_{m+2} \in \mathbb{R}^{e\times n}$, and $\boldsymbol{q}_0 \in \mathbb{R}^n$, we assume that (3) can be written in the following general form:

$$
\begin{aligned}
\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad & \boldsymbol{q}_0^T\boldsymbol{x}\\
\text{subject to} \quad & \boldsymbol{x}^T\boldsymbol{Q}_k\boldsymbol{x} + \boldsymbol{q}_k^T\boldsymbol{x} + \gamma_k = 0 \quad (k = 1, \ldots, m),\\
& \boldsymbol{L}_{m+1}\boldsymbol{x} = \boldsymbol{b}_{m+1}, \quad \boldsymbol{L}_{m+2}\boldsymbol{x} \le \boldsymbol{b}_{m+2},\\
& \boldsymbol{\ell} \le \boldsymbol{x} \le \boldsymbol{u},
\end{aligned}
$$

where $\boldsymbol{\ell}, \boldsymbol{u} \in \mathbb{R}^n$ mean, respectively, the lower and upper bounds for $\boldsymbol{x}$, and $\boldsymbol{b}_{m+1} \in \mathbb{R}^d, \boldsymbol{b}_{m+2} \in \mathbb{R}^e$.

Consequently, (4) can be expressed as the following general form:

$$
\begin{aligned}
\min_{\boldsymbol{x}\in\mathbb{R}^n,\ \boldsymbol{\lambda}\in\mathbb{R}^{m+d}} \quad & \boldsymbol{q}_0^T\boldsymbol{x} + \delta\sum_{i=1}^{m+d}\lambda_i\\
\text{subject to} \quad & -\lambda^1_k \le \boldsymbol{x}^T\boldsymbol{Q}_k\boldsymbol{x} + \boldsymbol{q}_k^T\boldsymbol{x} + \gamma_k \le \lambda^1_k \quad (k = 1, \ldots, m),\\
& -\boldsymbol{\lambda}^2_{m+1} \le \boldsymbol{L}_{m+1}\boldsymbol{x} - \boldsymbol{b}_{m+1} \le \boldsymbol{\lambda}^2_{m+1}, \quad \boldsymbol{L}_{m+2}\boldsymbol{x} \le \boldsymbol{b}_{m+2},\\
& \boldsymbol{\lambda} \ge 0\ ,\ \boldsymbol{\ell} \le \boldsymbol{x} \le \boldsymbol{u},
\end{aligned}
\tag{5}
$$

where $\boldsymbol{\lambda} = [\boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2_{m+1}]^T \in \mathbb{R}^{m+d}$ and $\boldsymbol{\lambda}^2_{m+1} \in \mathbb{R}^d$.

If we let $\boldsymbol{q}_{m+r}$ be the rth row of $\boldsymbol{L}_{m+1}$ $(r = 1, \ldots, d)$ and $\boldsymbol{q}_{m+d+\rho}$ be the ρth row of

$\boldsymbol{L}_{m+2}$ $(\rho = 1, \ldots, e)$, then (5) is written as follows:

$$\begin{array}{ll} \min\limits_{\boldsymbol{x}\in\mathbb{R}^n,\ \boldsymbol{\lambda}\in\mathbb{R}^{m+d}} & \boldsymbol{q}_0^T\boldsymbol{x} + \delta\sum\limits_{i=1}^{m+d}\lambda_i \\ \text{subject to} & \boldsymbol{x}^T\boldsymbol{Q}_k\boldsymbol{x} + \boldsymbol{q}_k^T\boldsymbol{x} - \lambda_k^1 + \gamma_k \le 0, \\ & -\boldsymbol{x}^T\boldsymbol{Q}_k\boldsymbol{x} - \boldsymbol{q}_k^T\boldsymbol{x} - \lambda_k^1 - \gamma_k \le 0 \ (k = 1, \ldots, m), \\ & \boldsymbol{q}_{m+r}\boldsymbol{x} - (\boldsymbol{\lambda}_{m+1}^2)_r - (\boldsymbol{b}_{m+1})_r \le 0, \\ & -\boldsymbol{q}_{m+r}\boldsymbol{x} - (\boldsymbol{\lambda}_{m+1}^2)_r + (\boldsymbol{b}_{m+1})_r \le 0 \ (r = 1, \ldots, d), \\ & \boldsymbol{q}_{m+r+\rho}\boldsymbol{x} - (\boldsymbol{b}_{m+2})_\rho \le 0 \ (\rho = 1, \ldots, e), \\ & \boldsymbol{\lambda} \ge 0\ ,\ \boldsymbol{\ell} \le \boldsymbol{x} \le \boldsymbol{u}. \end{array} \tag{6}$$

Notice that all the diagonal elements of $\boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_m$ in the quadratic constraints of (6) are zeros. This will be exploited in Section 4.

3 SDP, SOCP and LP relaxations

We give a brief description of an SDP relaxation of general QCQPs which include (4). Then, SOCP [13] and LP relaxations of general QCQPs are described using the scaled diagonally dominant (SDD) matrices and diagonally dominant (DD) matrices, respectively.

3.1 SDP relaxations

Let $\boldsymbol{w} \in \mathbb{R}^n$. Consider a general form of QCQP:

$$\begin{array}{ll} \zeta^* := \min & \boldsymbol{w}^T\boldsymbol{Q}_0\boldsymbol{w} + \boldsymbol{\beta}_0^T\boldsymbol{w} + \gamma_0 \\ \text{subject to} & \boldsymbol{w}^T\boldsymbol{Q}_k\boldsymbol{w} + \boldsymbol{\beta}_k^T\boldsymbol{w} + \gamma_k \le 0 \quad (k = 1, \ldots, m), \end{array} \tag{7}$$

where $\boldsymbol{Q}_k \in \mathbb{S}^n$, $\boldsymbol{\beta}_k \in \mathbb{R}^n$ $(k = 0, \ldots, m)$ and $\gamma_k \in \mathbb{R}$ $(k = 1, \ldots, m)$. Since $\boldsymbol{Q}_k \in \mathbb{S}^n$ $(k = 0, \ldots, m)$ is not necessarily positive semidefinite, (7) is a nonconvex problem.

Introducing a new variable matrix $\boldsymbol{W} \in \mathbb{S}^n$, we let

$$\bar{\boldsymbol{Q}}_k := \begin{pmatrix} \gamma_k & \boldsymbol{\beta}_k^T/2 \\ \boldsymbol{\beta}_k/2 & \boldsymbol{Q}_k \end{pmatrix},\ \bar{\boldsymbol{W}} := \begin{pmatrix} w_{00} & \boldsymbol{w}^T \\ \boldsymbol{w} & \boldsymbol{W} \end{pmatrix},\ \text{and}\ \bar{\boldsymbol{H}}_0 := \begin{pmatrix} 1 & \boldsymbol{0}^T \\ \boldsymbol{0} & \boldsymbol{O} \end{pmatrix}.$$

Then, an SDP relaxation of (7) is given by

$$\begin{array}{ll} \zeta_{SDP}^* := \min & \bar{\boldsymbol{Q}}_0 \bullet \bar{\boldsymbol{W}} \\ \text{subject to} & \bar{\boldsymbol{Q}}_k \bullet \bar{\boldsymbol{W}} \le 0 \quad (k = 1, \ldots, m), \\ & \bar{\boldsymbol{H}}_0 \bullet \bar{\boldsymbol{W}} = 1, \\ & \bar{\boldsymbol{W}} \in \mathbb{S}_+^{n+1}, \end{array} \tag{8}$$

where the inner product $\bar{\boldsymbol{Q}} \bullet \bar{\boldsymbol{W}}$ means the standard inner product between two symmetric matrices, *i.e.,* $\bar{\boldsymbol{Q}} \bullet \bar{\boldsymbol{W}} = \sum_i \sum_k \bar{Q}_{ik}\bar{W}_{ik}$.

3.2 SOCP relaxations

In [13], an SOCP relaxation was proposed using the 2×2 principle submatrices of the variable matrix $\bar{\boldsymbol{W}}$ of (8). They showed that the SOCP relaxation provides the exact optimal solution for QCQP if the off-diagonal elements of $\bar{\boldsymbol{Q}}_k$ $(k = 0, \ldots, m)$ are nopositive. The SOCP relaxation in [13] is closely related to the dual of the first level relaxation of the hierarchy of the scaled diagonally dominant sum-of-squares (SDSOS) relaxations proposed in [3]. By applying the approach in [13], we obtain the following SOCP relaxation:

$$\left.\begin{array}{ll} \min & \bar{\boldsymbol{Q}}_0 \bullet \bar{\boldsymbol{W}} \\ \text{subject to} & \bar{\boldsymbol{Q}}_k \bullet \bar{\boldsymbol{W}} \leq 0 \ (1 \leq k \leq m), \ \bar{\boldsymbol{H}}_0 \bullet \bar{\boldsymbol{W}} = 1, \\ & \bar{W}_{jj} \geq 0 \ (1 \leq j \leq n+1), \\ & (\bar{W}_{ij})^2 \leq \bar{W}_{ii}\bar{W}_{jj} \ (1 \leq i < j \leq n+1). \end{array}\right\} \tag{9}$$

Using

$$w^2 \leq \xi\eta, \ \xi \geq 0 \ \text{ and } \eta \geq 0 \text{ if and only if } \left\| \begin{pmatrix} \xi - \eta \\ 2w \end{pmatrix} \right\| \leq \xi + \eta, \tag{10}$$

(9) is converted to an SOCP. Thus, the following SOCP is equivalent to the problem (9).

$$\left.\begin{array}{ll} \zeta^*_{SOCP} := \min & \bar{\boldsymbol{Q}}_0 \bullet \bar{\boldsymbol{W}} \\ \text{subject to} & \bar{\boldsymbol{Q}}_p \bullet \bar{\boldsymbol{W}} \leq 0 \ (1 \leq p \leq m), \ \bar{\boldsymbol{H}}_0 \bullet \bar{\boldsymbol{W}} = 1, \\ & \left\| \begin{pmatrix} \bar{W}_{ii} - \bar{W}_{jj} \\ 2\bar{W}_{ij} \end{pmatrix} \right\| \leq \bar{W}_{ii} + \bar{W}_{jj} \ (1 \leq i < j \leq n+1). \end{array}\right\} \tag{11}$$

Since $\bar{\boldsymbol{W}} \in \mathbb{S}^n_+$ implies $(\bar{W}_{ij})^2 \leq \bar{W}_{ii}\bar{W}_{jj}$ $(1 \leq i < j \leq n+1)$, the optimal value ζ^*_{SOCP} of (11) is weaker than ζ^*_{SDP} of (8):

$$\zeta^*_{SOCP} \leq \zeta^*_{SDP} \leq \zeta^*.$$

3.3 LP relaxations

We derive LP relaxations of (7) using the diagonally dominant sum-of-squares relaxation (DSOS) in [3].

Consider the cone of diagonally dominant matrices of dimension $n+1$ defined by

$$\mathcal{D}^{n+1} := \left\{ \boldsymbol{W} \in \mathbb{S}^{n+1} : W_{ii} \geq \sum_{j \neq i} |W_{ij}| \quad (1 \leq i \leq n+1) \right\}.$$

In [8], the dual of $\mathcal{D}^{n+1}$ is given by

$$\begin{array}{rcl} (\mathcal{D}^{n+1})^* & := & \left\{ \boldsymbol{W} \in \mathbb{S}^{n+1} : \boldsymbol{w}^T \boldsymbol{W} \boldsymbol{w} \geq 0 \text{ for } \forall \boldsymbol{w} \text{ with at most 2 nonzero elements 1 or -1} \right\} \\ & = & \left\{ \boldsymbol{W} \in \mathbb{S}^{n+1} : W_{ii} \geq 0 \ (1 \leq i \leq n+1), \ W_{ii} + W_{jj} - 2|W_{ij}| \geq 0 \quad (1 \leq i < j \leq n+1) \right\} \end{array}$$

Using $(\mathcal{D}^{n+1})^*$, an LP relaxation of (7) can be derived as

$$\left.\begin{array}{ll} \zeta^*_{LP} := \min & \bar{\boldsymbol{Q}}_0 \bullet \bar{\boldsymbol{W}} \\ \text{subject to} & \bar{\boldsymbol{Q}}_k \bullet \bar{\boldsymbol{W}} \leq 0 \ (1 \leq k \leq m), \ \bar{\boldsymbol{H}}_0 \bullet \bar{\boldsymbol{W}} = 1, \\ & \bar{W}_{ii} \geq 0 \ (1 \leq i \leq n+1), \\ & \bar{W}_{ii} + \bar{W}_{jj} - 2|\bar{W}_{ij}| \geq 0 \quad (1 \leq i < j \leq n+1). \end{array}\right\} \tag{12}$$

Let $\bar{\boldsymbol{W}}$ be a feasible solution of (11). Then, $|\bar{W}_{ij}| \leq \sqrt{\bar{W}_{ii}\bar{W}_{jj}}$ $(1 \leq i < j \leq n+1)$. Since $\sqrt{\bar{W}_{ii}\bar{W}_{jj}} \leq (\bar{W}_{ii} + \bar{W}_{jj})/2$ always holds for all nonnegative $\bar{W}_{ii}$ and $\bar{W}_{jj}$, $\bar{\boldsymbol{W}}$ is a feasible solution of (12). Thus, the LP relaxation (12) is an weaker relaxation than the SOCP relaxation (11) and the following relation holds for the optimal values of the three relaxations:

$$\zeta^*_{LP} \leq \zeta^*_{SOCP} \leq \zeta^*_{SDP} \leq \zeta^*. \tag{13}$$

4 The equivalence of the optimal values of SDP, SOCP and LP relaxations

Now, we show the equivalence among the optimal values of (8), (11) and (12) under the following assumptions. As mentioned at the end of Section 2, the pooling problem satisfies the following assumption.

Assumption 4.1. *All the diagonal elements in $\boldsymbol{Q}_0, \boldsymbol{Q}_1, \ldots, \boldsymbol{Q}_m$ of (7) are zeros.*

Theorem 4.1. *Suppose that Assumption 4.1 holds. Then, $\zeta^*_{SDP} = \zeta^*_{SOCP} = \zeta^*_{LP}$.*

Proof. Let $\bar{\boldsymbol{W}} = \begin{pmatrix} w_{00} & \boldsymbol{w}^T \\ \boldsymbol{w} & \boldsymbol{W} \end{pmatrix}$ be a feasible solution of (12). It always holds that $w_{00} = 1$ by the constraint $\bar{\boldsymbol{H}}_0 \bullet \bar{\boldsymbol{W}} = 1$. If we add a sufficiently large number $\alpha \geq \lambda_{\max}\left(\frac{\boldsymbol{w}\boldsymbol{w}^T}{w_{00}} - \boldsymbol{W}\right)$ to the diagonal of $\bar{\boldsymbol{W}}$ except the first diagonal element w_{00} of $\bar{\boldsymbol{W}}$, the resulting matrix $\begin{pmatrix} w_{00} & \boldsymbol{w}^T \\ \boldsymbol{w} & \boldsymbol{W} + \alpha \boldsymbol{I} \end{pmatrix}$ becomes positive semidefinite by the Schur complement. Here $\lambda_{\max}$ means the largest eigenvalue. The inequality constraints, however, still hold and the objective value remains same, since the diagonal elements in $\boldsymbol{Q}_0, \ldots, \boldsymbol{Q}_m$ are zeros by Assumption 4.1. Thus, $\begin{pmatrix} w_{00} & \boldsymbol{w}^T \\ \boldsymbol{w} & \boldsymbol{W} + \alpha \boldsymbol{I} \end{pmatrix}$ is a feasible solution of the SDP relaxation (8). Therefore, we can construct a feasible solution in the SDP relaxation whose objective value is same as $\bar{\boldsymbol{W}}$, and this leads to $\zeta^*_{SDP} \leq \zeta^*_{LP}$. In view of this with (13), the desired result $\zeta^*_{SDP} = \zeta^*_{SOCP} = \zeta^*_{LP}$ follows. □

From Theorem 4.1, we show the relationship among the optimal values of the primal and dual problems in the subsequent discussion. The dual of (8) can be written as

$$\begin{array}{ll} \mu^*_{SDP} := \max & \mu \\ \text{subject to} & \bar{\boldsymbol{Q}}_0 + \sum_{k=1}^m \eta_k \bar{\boldsymbol{Q}}_k - \mu \bar{\boldsymbol{H}}_0 - \bar{\boldsymbol{S}} = \boldsymbol{O}, \\ & \eta_1, \ldots, \eta_m \geq 0,\ \mu \in \mathbb{R}, \\ & \bar{\boldsymbol{S}} \in \mathbb{S}^{n+1}_+. \end{array} \tag{14}$$

By Assumption 4.1, this problem has no interior point. Thus, the positive duality gap between ζ^*_{SDP} and μ^*_{SDP} might exist as the Slater condition does not hold. In the following Corollary 4.1, we show that there is no duality gap, that is, $\zeta^*_{SDP} = \mu^*_{SDP}$, using the dual of the SOCP relaxation (11) and the LP relaxation (12). These dual problems are closely related to scaled diagonally dominant sum of squares (SDSOS) and diagonally dominant sum of squares (DSOS) in [3].

In [2], SDSOS relaxations were proposed using SDD matrices. A matrix $\boldsymbol{B} \in \mathbb{S}^n$ is SDD if and only if it can be expressed as

$$\boldsymbol{B} = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \boldsymbol{B}^{ij},$$

where the nonzero elements of $\boldsymbol{B}^{ij} \in \mathbb{S}^n$ are from the 2×2 principal submatrix of a positive semidefinite matrix $\boldsymbol{C} \in \boldsymbol{S}^n_+$ with ith and jth rows and columns of $\boldsymbol{C}$ and all the other elements of $\boldsymbol{B}^{ij}$ are zero. More precisely, $\boldsymbol{B}^{ij}$ is a symmetric matrix with nonzero elements only in (i,i)th, (i,j)th, (j,i)th and (j,j)th positions such that $\begin{bmatrix} (\boldsymbol{B}^{ij})_{ii} & (\boldsymbol{B}^{ij})_{ij} \\ (\boldsymbol{B}^{ij})_{ij} & (\boldsymbol{B}^{ij})_{jj} \end{bmatrix} \in \mathbb{S}^2_+$. Thus, each $\boldsymbol{B}^{ij} \in \mathbb{S}^n$ is positive semidefinite. Let $\mathcal{SD}^n$ be the cone of SDD matrices. It is well-known that any DD matrix is SDD, therefore, $\mathcal{D}^n \subset \mathcal{SD}^n \subset \mathbb{S}^n_+$ holds.

Replacing $\bar{\boldsymbol{S}} \in \mathbb{S}^{n+1}_+$ in (14) by $\bar{\boldsymbol{S}} \in \mathcal{SD}^{n+1}$ corresponds to the first level of the hierarchy of SDSOS relaxation for QCQPs in [2], and it is the dual of (11):

$$\begin{array}{rl} \mu^*_{SOCP} := \max & \mu \\ \text{subject to} & \bar{\boldsymbol{Q}}_0 + \sum_{k=1}^m \eta_k \bar{\boldsymbol{Q}}_k - \mu \bar{\boldsymbol{H}}_0 - \bar{\boldsymbol{S}} = \boldsymbol{O}, \\ & \eta_1, \ldots, \eta_m \geq 0,\ \mu \in \mathbb{R}, \\ & \bar{\boldsymbol{S}} \in \mathcal{SD}^{n+1}. \end{array} \tag{15}$$

The dual of (12) is given as

$$\begin{array}{rl} \mu^*_{LP} := \max & \mu \\ \text{subject to} & \bar{\boldsymbol{Q}}_0 + \sum_{k=1}^m \eta_k \bar{\boldsymbol{Q}}_k - \mu \bar{\boldsymbol{H}}_0 - \bar{\boldsymbol{S}} = \boldsymbol{O}, \\ & \eta_1, \ldots, \eta_m \geq 0,\ \mu \in \mathbb{R}, \\ & \bar{\boldsymbol{S}} \in \mathcal{D}^{n+1}. \end{array} \tag{16}$$

In general,

$$\mu^*_{SDP} \geq \mu^*_{SOCP} \geq \mu^*_{LP} \tag{17}$$

holds from $\mathcal{D}^{n+1} \subset \mathcal{SD}^{n+1} \subset \mathbb{S}^{n+1}_+$.

We will show that the primal problems and the dual problems attain the same optimal values, and this indicates that there is no duality gap between the SDP relaxation (8) and (14).

Corollary 4.1. *Under Assumption 4.1, it holds that*

$$\zeta^*_{SDP} = \zeta^*_{SOCP} = \zeta^*_{LP} = \mu^*_{LP} = \mu^*_{SOCP} = \mu^*_{SDP}.$$

Proof. For (14), we let $\bar{\boldsymbol{S}} = \begin{pmatrix} s_{00} & \boldsymbol{s}^T \\ \boldsymbol{s} & \boldsymbol{S} \end{pmatrix}$. From Assumption 4.1, the diagonal of $\boldsymbol{S}$ is zero. Since $\boldsymbol{S} \in \mathbb{S}^n_+$, we have $\boldsymbol{S} = \boldsymbol{O}$, thus, $\bar{\boldsymbol{S}} \in \mathbb{S}^{n+1}_+$ leads to $\boldsymbol{s} = \boldsymbol{0}$. Hence, (14) is equivalent to the following problem:

$$\begin{array}{rl} \max & \mu \\ \text{subject to} & \gamma_0 + \sum_{k=1}^m \eta_k \gamma_k - \mu - s_{00} \geq 0, \\ & \boldsymbol{q}_0 + \sum_{k=1}^m \eta_k \boldsymbol{q}_k = \boldsymbol{0}, \\ & \boldsymbol{Q}_0 + \sum_{k=1}^m \eta_k \boldsymbol{Q}_k = \boldsymbol{O}, \\ & \eta_1, \ldots, \eta_m \geq 0,\ \mu \in \mathbb{R},\ s_{00} \geq 0. \end{array} \tag{18}$$

Similarly, for (16), we can show that $\boldsymbol{S} = \boldsymbol{O}$ and $\boldsymbol{s} = \boldsymbol{0}$ using the zero diagonal of $\bar{\boldsymbol{S}}$. As a result, (16) is equivalent to (18), and the optimal values of (16) and (18) coincide, i.e., $\mu^*_{SDP} = \mu^*_{LP}$. Since the duality theorem holds on linear programming problems regardless of the existence of interior points, the optimal values of (12) and (16) are equivalent, *i.e.,* $\zeta^*_{LP} = \mu^*_{LP}$. By $\mu^*_{SDP} = \mu^*_{LP} = \zeta^*_{LP}$, (13), (17) and Theorem 4.1, the desired result follows.

□

5 Computational methods

In this section, we discuss two computational methods for adjusting and refining an approximate solution obtained by the SOCP or LP relaxation of the pooling problem. As the pooling problem is NP-hard, only approximate solutions can be obtained by the relaxation methods. In addition, the bounds for the variables of the pooling problem have been modified when binary variables have been removed in (2). As a result, an approximate solution by SOCP or LP relaxation may not be a solution to the original problem.

In Nishi's method [16], a mixed-integer linear program was first solved for finding a feasible solution of the original pooling problem. Then, `fmincon` in Matlab, a nonlinear programming solver, was applied to find a local optimum solution. This step turned out to be very time-consuming. To improve the computational efficiency for finding a solution that satisfies the plant requirements, we propose a rescheduling method based on successive refining the solution obtained by solving the SOCP or LP problem.

Nishi [16]'s method can be described as follows:

Algorithm 5.1: **Nishi's method**

Step 1. Solve an SDP relaxation of the pooling problem (3).

Step 2. Apply a procedure called FFS (finding feasible solution) to find a feasible solution $\boldsymbol{x}$.

Step 3. Use a general nonlinear programming solver (`fmincon`) starting from $\boldsymbol{x}$ for a local optimal solution.

We note that the feasible solution $\boldsymbol{x}$ obtained in Step 2 is not necessarily a local minimum of the original problem. Step 3 is very time-consuming, as shown in numerical results in Section 6.

In our method, the SOCP or LP relaxation is used instead of the SDP relaxation. We also propose a rescheduling method for Step 3 of Nishi's method. More precisely, after applying applying the SOCP or LP relaxation and FFS, which is described in Section 5.1 in detail, an approximate solution is further refined by the rescheduling method. The main steps of our method is described as follows:

Algorithm 5.2: **The proposed method**

Step 1. Solve the SOCP or LP relaxation (12) of the pooling problem.

Step 2. Apply FFS to obtain a feasible solution $\boldsymbol{x}$.

Step 3. Perform the proposed rescheduling method.

We briefly review FFS [16] in Section 5.1 and describe our proposed rescheduling method in Section 5.2.

5.1 A method for finding a feasible solution

As a solution attained by the SDP, SOCP or LP relaxation is not necessarily feasible for the original problem, the following mixed-integer linear problem was introduced to find a feasible solution in [16] as the first step of the procedure FFS. More precisely, the solution $(\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}})$ obtained by the relaxation methods is used for the following problem called FFS1:

$$
\begin{aligned}
\text{FFS1 } (\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}) := \min_{\boldsymbol{a},\boldsymbol{p},\boldsymbol{q},\boldsymbol{v},\boldsymbol{u},\boldsymbol{s}} \quad & \alpha \sum_{t\in T}\sum_{i\in V} s_i^t + \sum_{t\in T}\sum_{(i,j)\in A} CA_{ij} a_{ij}^t + \sum_{t\in T}\sum_{i\in V_P} v_i^t \\
\text{s.t.} \quad & -\boldsymbol{s} \le \boldsymbol{p} - \bar{\boldsymbol{p}} \le \boldsymbol{s}, \\
& p_i^{t+1} = p_i^t + SA_i^t - \sum_{k\in E(i)} a_{ik}^t, \quad p_i^t \ge 0 \ (i \in V_S, t \in T), \\
& p_i^{t+1} = p_i^t + \sum_{j\in I(i)} a_{ji}^t - \sum_{k\in E(i)} a_{ik}^t \ (i \in V_I, t \in T), \\
& p_i^{\min} \le p_i^t \le p_i^{\max} \ (i \in V_I, t \in T), \\
& RC_i^t = \sum_{j\in I(i)} a_{ji}^t \ , \ RC_i^t q_i^t = \sum_{j\in I(i)} a_{ji}^t \bar{q}_j^t (i \in V_P, t \in T), \\
& q_i^t \ge RQ_i^t - v_i^t \ , \ v_i^t \ge 0 \ (i \in V_P, t \in T), \\
& u_{ij}^t L_{ij} \le a_{ij}^t \le u_{ij}^t U_{ij} \ , \ u_{ij}^t \in \{0,1\} \ ((i,j) \in A, t \in T), \\
& \sum_{j\in I(i)} u_{ji}^t + \sum_{k\in E(i)} u_{ik}^t \le 1 \ (i \in V, t \in T),
\end{aligned} \tag{19}
$$

where α denotes a weight coefficient for $\| \boldsymbol{p} - \bar{\boldsymbol{p}} \|_1$ in the objective function. We note that (19) is a mixed-integer linear programming problem, thus it is computationally efficient to solve (19).

After solving (19), a procedure FFS2 to further refine a feasible solution of the pooling problem is employed using the solution of (19) in [16]. More precisely, using the output $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{u}}, \hat{\boldsymbol{p}}, \hat{\boldsymbol{q}}, \hat{\boldsymbol{v}})$ of (19), FFS2 produces $\tilde{\boldsymbol{q}}$ and $\tilde{\boldsymbol{v}}$ by

$$
\begin{cases}
\tilde{q}_i^1 = q_i^1 \\
\tilde{q}_i^{t+1} = (\hat{p}_i^t \hat{q}_i^t + SA_i^t SQ_i^t - \sum_{k\in E(i)} \hat{a}_{ik}^t \tilde{q}_i^t)/\hat{p}_i^{t+1} \quad (i \in V_S, t \in T)
\end{cases}
$$

$$
\begin{cases}
\tilde{q}_i^1 = q_i^1 \\
\tilde{q}_i^{t+1} = (\hat{p}_i^t \hat{q}_i^t + \sum_{j\in E(i)} \hat{a}_{ji}^t \tilde{q}_j^t - \sum_{k\in E(i)} \hat{a}_{ik}^t \tilde{q}_i^t)/\hat{p}_i^{t+1} \quad (i \in V_I, t \in T)
\end{cases}
$$

$$
\begin{cases}
\tilde{q}_i^t = (\sum_{j\in I(i)} \hat{a}_{ji}^t \tilde{q}_j^t)/RC_i^t \quad (i \in V_P, t \in T) \\
\tilde{v}_i^t = \max\{0, RQ_i^t - \tilde{q}_i^t\}.
\end{cases} \tag{20}
$$

Consequently, a solution $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{p}}, \tilde{\boldsymbol{q}}, \hat{\boldsymbol{u}}, \tilde{\boldsymbol{v}})$ of the pooling problem is attained. Notice that solving (19) determines $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{u}}, \hat{\boldsymbol{p}})$ using $\bar{\boldsymbol{p}}, \bar{\boldsymbol{q}}$, and a feasible solution $(\tilde{\boldsymbol{q}}, \tilde{\boldsymbol{v}})$ is obtained by (20) using $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{u}}, \hat{\boldsymbol{p}})$. We also note that the equations in (20) involves nonlinear terms. It should be mentioned that a feasible solution $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{p}}, \tilde{\boldsymbol{q}}, \hat{\boldsymbol{u}}, \tilde{\boldsymbol{v}})$ for the pooling problem does not necessarily satisfy the quality requirement $\tilde{v}_i^t = 0$. For this requirement, we refine the solution by a rescheduling method.

5.2 A rescheduling method

We propose a rescheduling method to refine the obtained solution $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{p}}, \tilde{\boldsymbol{q}}, \hat{\boldsymbol{u}}, \tilde{\boldsymbol{v}})$ from the procedure in Section 5.1. The rescheduling method is based on successive refinement of the solution. The algorithm continues until all requirements are satisfied, *i.e.,* $v_i^t = 0$ $(i \in V_P, t \in T)$, or it is determined that the successive refinement cannot satisfy the plant requirements in Step 3.6.

We denote the starting time step for the rescheduling method as $\hat{t}$. The rescheduling algorithm for Step 3 of Algorithm 5.2 is described as follows:

Algorithm 5.3: **The proposed rescheduling method**

Step 3.1. Initialize $\hat{t} = 0$. Set $\boldsymbol{x}^*$ as the zero vector of dimension n, the length of $(\boldsymbol{a}, \boldsymbol{p}, \boldsymbol{q}, \boldsymbol{u}, \boldsymbol{v})$.

Step 3.2. Formulate the pooling problem with discretized time step $\hat{T} = \{\hat{t}+1, \ldots, M_T\}$.

Step 3.3. Solve SOCP (11) (or LP (12)) relaxation formulated for $\hat{T}$, apply FFS (19) and (20) to obtain a feasible solution $\boldsymbol{x}^+ = (\hat{\boldsymbol{a}}, \hat{\boldsymbol{p}}, \tilde{\boldsymbol{q}}, \hat{\boldsymbol{u}}, \tilde{\boldsymbol{v}})$.

Step 3.4. If $\boldsymbol{x}^+$ satisfies the requirement $\tilde{v}_i^t = 0$ for all $(i,t) \in V_P \times \hat{T}$, then replace $\boldsymbol{x}^*$ with $\boldsymbol{x}^+$ for $t \in \hat{T}$, output $\boldsymbol{x}^*$ and terminate.

Step 3.5. Find the smallest time step t^+ such that $\tilde{v}_i^{t^+} > 0$ for some $i \in V_P$.

Step 3.6. Modify $\boldsymbol{x}^+$ as $\breve{\boldsymbol{x}} = (\breve{\boldsymbol{a}}, \breve{\boldsymbol{p}}, \breve{\boldsymbol{q}}, \hat{\boldsymbol{u}}, \breve{\boldsymbol{v}})$ and replace $\boldsymbol{x}^*$ with $\breve{\boldsymbol{x}}$ for the time steps $\{\hat{t}+1, \ldots, t^+\}$.

Step 3.7. Let $\hat{t} = t^+$. Return to Step 3.2. (If $\hat{t} = M_T$, output $\boldsymbol{x}^*$ and stop.)

Note that the size of the relaxation problem solved in Step 3.3 will become smaller as $\hat{t}$ approaches to M_T. Steps 3.2 and 3.3 can be skipped at $\hat{t} = 0$ as Steps 1 and 2 of Algorithm 5.2 have been performed.

Step 3.6 plays an important role for the overall performance of the rescheduling method, in particular, to successfully find a solution to the pooling problem. At Step 3.6, $\boldsymbol{x}^+ = (\hat{\boldsymbol{a}}, \hat{\boldsymbol{p}}, \tilde{\boldsymbol{q}}, \hat{\boldsymbol{u}}, \tilde{\boldsymbol{v}})$ is modified for the time steps $\{\hat{t}+1, \ldots, t^+\}$ to obtain a solution $\breve{\boldsymbol{x}} = (\breve{\boldsymbol{a}}, \breve{\boldsymbol{p}}, \breve{\boldsymbol{q}}, \breve{\boldsymbol{u}}, \breve{\boldsymbol{v}})$ which satisfies the plant demand in the time steps $\{\hat{t}+1, \ldots, t^+\}$.

More precisely, in Step 3.6, for each $t \in \{\hat{t}+1, \ldots, t^+\}$, we first check whether $\tilde{q}_i^t$ $(i \in V_P)$ satisfies the plant requirements, *i.e.,* $\tilde{v}_i^t = 0$ or not. Depending on the computed values of $\tilde{q}_i^t$ $(i \in V_P)$, we consider two cases: (Case I) if $\tilde{q}_i^t \geq RQ_i^t$ holds for all $i \in V_P$, it means that we have excessive supplies, (Case II) if there exists some $i \in V_P$ that satisfies $\tilde{q}_i^t < RQ_i^t$,

then it means shortage in supplies.

For (Case I), we modify the requirement $\breve{q}_i^t$ by RQ_i^t at plants $i \in V_P$ and time t, to reduce excessive supplies so that more quantities can be available at the intermediate tanks for the subsequent modifications in $\hat{t}+1, \ldots, t^+$. This modification on the requirement at plant $i \in V_P$ in turn affects the intermediate tanks $j \in V_I$ along with the network arcs determined by $\tilde{u}_{ij}^t = 1$ in FFS1, and to the source $k \in V_S$. The algorithm for (Case I) is described as Algorithm 5.4(I), which is employed as Step 3.6 in Algorithm 5.3.

Algorithm 5.4(I) [The case of excessive supplies]:
For the time step $\hat{t}$, let $\breve{p}_i^{\hat{t}} = \hat{p}_i^{\hat{t}}$ and $\breve{q}_i^{\hat{t}} = \tilde{q}_i^{\hat{t}}$ $(i \in V_S \cup V_I)$.
For each $t = \hat{t}+1, \ldots, t^+$, apply the following steps:

Step 3.6.(I)1. For $i \in V_P$, set $\breve{q}_i^t = RQ_i^t$ and $\breve{v}_i^t = 0$.

Step 3.6.(I)2. For $j \in V_I$ and $i \in V_P$ such that $(j,i) \in A$, if $\tilde{u}_{ji}^t = 1$, compute $\breve{a}_{ji}^t = \hat{a}_{ji}^t - RC_i^t(\tilde{q}_i^t - RQ_i^t)/\breve{q}_j^t$, otherwise set $\breve{a}_{ji}^t = \hat{a}_{ji}^t$.

Step 3.6.(I)3. For $i \in V_P$, compute $\breve{p}_i^{t+1} = \hat{p}_i^{t+1}$ and $\breve{q}_i^{t+1} = \frac{1}{RC_i^t} \sum_{j \in I(i)} \breve{a}_{ji}^t \breve{q}_j^t$.

Step 3.6.(I)4. For $j \in V_I$, compute $\breve{p}_j^{t+1} = \breve{p}_j^t - \breve{a}_{ji}^t$ and $\breve{q}_j^{t+1} = \breve{q}_j^t$.

Step 3.6.(I)5. For $k \in V_S$, set $\breve{p}_k^{t+1} = \breve{p}_k^t$ and $\breve{q}_k^{t+1} = \breve{q}_k^t$.

Step 3.6.(I)6. For $k \in V_S$ and $j \in V_I$ such that $(k,j) \in A$, if $\tilde{u}_{kj}^t = 1$, compute $\breve{a}_{kj}^t = \min\{U_{kj},\ \breve{p}_k^t,\ p_j^{\max} - \breve{p}_j^t\}$, and adjust $\breve{p}_k^{t+1}$, $\breve{q}_k^{t+1}$, $\breve{p}_j^{t+1}$ and $\breve{q}_j^{t+1}$ by

$$\breve{p}_k^{t+1} = \breve{p}_k^t - \breve{a}_{kj}^t,\ \breve{q}_k^{t+1} = \breve{q}_k^t \text{ (if } \breve{p}_k^{t+1} = 0, \text{ then set } \breve{q}_k^{t+1} = 0),$$
$$\breve{p}_j^{t+1} = \breve{p}_j^t + \breve{a}_{kj}^t,\ \breve{q}_j^{t+1} = (\breve{p}_j^t \breve{q}_j^t + \breve{a}_{kj}^t \breve{q}_k^t)/\breve{p}_j^{t+1}.$$

Step 3.6.(I)7. For $(i,j) \in A$ such that $\tilde{u}_{ij}^t = 0$, set $\breve{a}_{ij}^t = 0$.

Step 3.6.(I)8. For $i \in V_P$, recalculate $\breve{q}_i^{t+1} = \frac{1}{RC_i^t} \sum_{j \in I(i)} \breve{a}_{ji}^t \breve{q}_j^t$.

For (Case II), Algorithm 5.4(II) is applied. The output of the FFS procedure is $\hat{u}_{ij}^t$. If $\hat{u}_{ij}^t = 1$, then it means that the arc $(i,j) \in A$ should be connected at time t. With the connected arcs constructed from the output $\hat{u}_{ij}^t$, it cannot be guaranteed that the quality requirements at the plants are satisfied. Thus, we perform the following steps to meet the quality requirements: Between the intermediate tanks V_I and the plants V_P, we first consider the arcs (denoted in A_{IP} in Algorithm 5.4(II)) that need the greatest requirements at the plants. Then, between the sources V_S and the intermediate tanks V_I, the arcs (denoted in A_{SI} in Algorithm 5.4(II)) that provide more quantities to the intermediate tanks from the sources are used.

Algorithm 5.4(II): [**The case of insufficient supplies**]
For the time step $\hat{t}$, let $\breve{p}_i^{\hat{t}} = \hat{p}_i^{\hat{t}}$ and $\breve{q}_i^{\hat{t}} = \tilde{q}_i^{\hat{t}}$ $(i \in V)$.
For each $t = \hat{t}+1, \ldots, t^+$, apply the following steps:

Step 3.6.(II)1. For each plant node $i \in V_P$, compute the requirements $D_i^t := RC_i^t \times RQ_i^t$. For each intermediate tank $j \in V_I$, calculate the maximum supply value defined by $S_j := \min\{\breve{p}_j^t - p_j^{\min},\ U_{ji}\} \times \breve{q}_j^t$. Sort the plants and intermediate tanks in the descending order such that $D_{M_S+M_I+1}^t \geq \ldots \geq D_{M_S+M_I+M_P}^t$ and $S_{M_S+1}^t \geq \ldots \geq S_{M_S+M_I}^t$.

Step 3.6.(II)2. For each $i \in V_P$, find $j_i \in V_I$ such that $(j_i, i) \in A$, $S_{j_i}^t \geq D_i^t$ and $j_i > j_{i-1} > \cdots > j_{M_S+M_I+1}$. We denote the set of such matching arcs by $A_{IP} := \{(j_i, i) \in A : i = M_S + M_I + 1, \ldots, M_S + M_I + M_P\}$. If such matching arcs cannot be found, return $\boldsymbol{x}^+$.

Step 3.6.(II)3. For $i \in V_P$, set $\breve{v}_i^t = 0$. For $(j, i) \in A_{IP}$, calculate $\breve{q}_i^t,\ \breve{a}_{ji}^t,\ \breve{p}_j^{t+1}$ and $\breve{q}_j^{t+1}$ by $\breve{q}_i^t = RQ_i^t,\ \breve{a}_{ji}^t = \frac{1}{\breve{q}_j^t} RC_i^t RQ_i^t,\ \breve{p}_j^{t+1} = \breve{p}_j^t - \breve{a}_{ji}^t$, and $\breve{q}_j^{t+1} = \breve{q}_j^t$.

Step 3.6.(II)4. For each source $k \in V_S$, compute the maximum supply value of $\bar{S}_k^t := \min\{\breve{p}_k^t,\ U_{kj}\} \times \breve{q}_k^t$. Sort the sources in the descending order such that $\bar{S}_1^t \geq \ldots \geq \bar{S}_{V_S}^t$. Let $J := V_I \backslash \{j \in V_I : (j, i) \in A_{IP} \text{ for some } i \in V_P\}$ and $\bar{M} := \min\{M_S, (M_I - M_P)\}$. Find an arc set $A_{SI} := \{(k_\alpha, j_\alpha) \in (V_S \times J) \cap A | \alpha = 1, \ldots, \bar{M}\}$ such that $\bar{S}_{k_\alpha} \geq \bar{S}_{j_\beta}$ and $\bar{S}_{j_\alpha} \leq \bar{S}_{j_\beta}$ for $\alpha < \beta$.

Step 3.6.(II)5. For $(k, j) \in A_{SI}$, compute $\breve{a}_{kj}^t, \breve{p}_k^{t+1}, \breve{q}_k^{t+1}, \breve{p}_j^{t+1}$ and $\breve{q}_j^{t+1}$ by

$$\breve{a}_{kj}^t = \min\{\breve{p}_k^t,\ U_{kj},\ p_j^{\max} - \breve{p}_j^t\},\ \breve{p}_k^{t+1} = \breve{p}_k^t - \breve{a}_{kj}^t,\ \breve{q}_k^{t+1} = \breve{q}_k^t,$$
$$\breve{p}_j^{t+1} = \breve{p}_j^t + \breve{a}_{kj}^t,\ \breve{q}_j^{t+1} = (\breve{p}_j^t \breve{q}_j^t + \breve{a}_{kj}^t \breve{q}_k^t)/\breve{p}_j^{t+1}.$$

Step 3.6.(II)6. For each arc $(i, j) \in A$ whose two nodes (i and j) have not been updated in the previous steps, set $\breve{a}_{ij}^t = 0$ to indicate that the arc (i, j) is unused at time t. Compute $\breve{p}_i^{t+1} = \breve{p}_i^t,\ \breve{q}_i^{t+1} = \breve{q}_i^t,\ \breve{p}_j^{t+1} = \breve{p}_j^t$ and $\breve{q}_j^{t+1} = \breve{q}_j^t$.

Step 3.6.(II)7. For $i \in V_P$, recalculate $\breve{q}_i^{t+1} = \frac{1}{RC_i^t} \sum_{j \in I(i)} \breve{a}_{ji}^t \breve{q}_j^t$.

Note that the proposed rescheduling method is a heuristic method, therefore, there still remains possibility that the rescheduling method cannot meet all quality requirements. In that case, the rescheduling method terminates with $\boldsymbol{x}^+$ at Step 3.6.(II).2.

6 Numerical results

The main purposes of our numerical experiments are to see whether the optimal values of the SDP, SOCP and LP relaxation coincide as shown in Theorem 4.1, and to demonstrate the numerical efficiency of the rescheduling method over Matlab function `fmincon` used in Nishi's method [16]. We also illustrate the computational efficiency of the proposed method by solving large-sized problems from the standard pooling test problems.

For numerical experiments, we first test the eight instances in Nishi [16] to compare our results with those in [16]. The eight test instances in [16] which have a solution satisfying the plant requirements are illustrated in Figures 4 and 5, and their sizes are shown in Table 2. For example, instance 8 has 2 sources, 4 intermediate tanks and 2 plants with time discretization 28, which has the same number of nodes as an instance with 8×28 nodes without time discretization. The intermediate tanks in the test instances are connected to each other as the complete graph. Second, we generated two larger test instances, instance 9 and 10, whose sizes are shown in Table 2. More precisely, the numbers of sources, intermediate tanks and plants of the test instances are increased up to 20, 10, respectively, with time discretization $M_T = 2$, to see the computational efficiency of the SOCP and LP relaxation. The pipelines between the intermediate tanks of the instances 9 and 10 are also connected as the complete graph. The number of variables of the two instances are 1344 and 1616, respectively. Third, we tested on Foulds 3, Foulds 4 and Foulds 5 whose sizes are larger than the other standard pooling test problems [6].

Numerical experiments were conducted on a Mac with OS X EI Captin version 10.11, processor 3.2GHz Intel Core i5, memory 8GB, 1867MHz DDR3, MATLAB_R2016a.

To implement Nishi's method [16] based on Algorithm 5.1, SparsePOP [19], a general polynomial optimization problem solver that includes `fmincon` after solving the SDP relaxation, was applied to the test instances. When `fmincon` was used as a nonlinear programming solver for Nishi's method, parameters were changed with TolFun=10^{-1} and TolCon=10^{-3}. For our proposed method described in Algorithm 5.2, we used SPOTless [4] and MOSEK [7] to solve the SOCP (11) and LP relaxation (12) and CPLEX [1] to solve FFS (19), and applied our rescheduling method Algorithm 5.3.

We used 10^{-4} for the penalty weight δ in our proposed formulation (5).

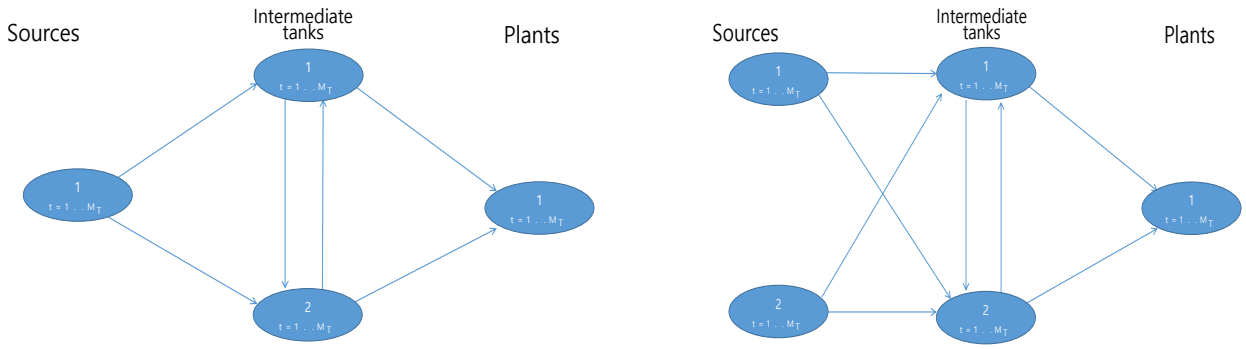

Figure 4: Instances 1, 2, 3, and 4

The experimental results of each instance are summarized in Tables 3, 4 and 5. We use the following ratio to measure how much the obtained solution successfully satisfies the requirements of all nodes:

$$\begin{aligned} \text{sucs.ratio} &= \left(\sum_{t \in T} \sum_{i \in V_P} RQ_i^t - \sum_{t \in T} \sum_{i \in V_P} v_i^t \right) \Big/ \sum_{t \in T} \sum_{i \in V_P} RQ_i^t \\ &= \frac{(\text{the sum of the requirements}) - (\text{the sum of insuffcienct values})}{(\text{the sum of requirements})}. \end{aligned}$$

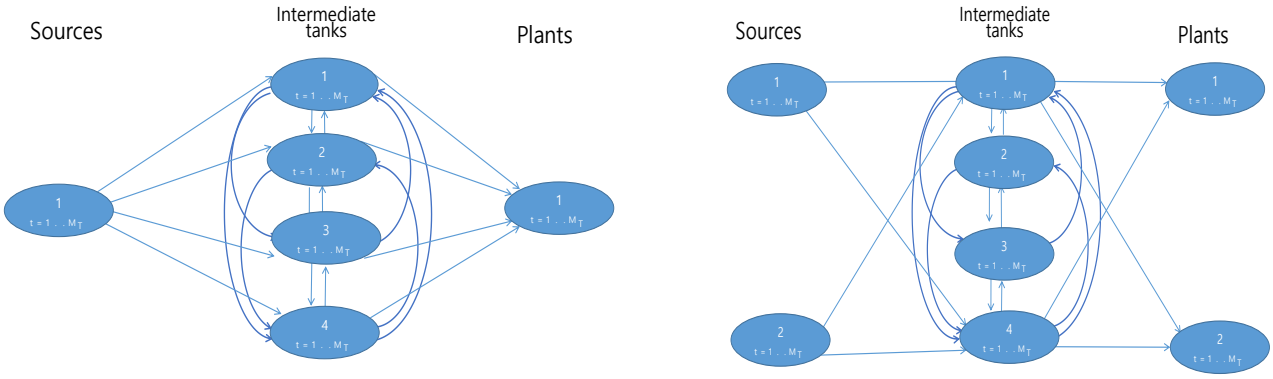

Figure 5: Instances 5, 6, 7, and 8

Table 2: M_S: the number of sources, M_I: the number of intermediate tanks, M_P: the number of plants, $(\#a, \#p, \#q, \#v)$: the numbers of variables, $|A|$: the number of pipelines, n: sum of all variables.

Instance	M_S	M_I	M_P	M_T	$\|A\|$	$\#a$	$\#p$	$\#q$	$\#v$	n
1	1	2	1	10	6	60	29	39	10	138
2	1	2	1	20	6	120	59	79	20	298
3	2	2	1	10	8	80	38	48	10	176
4	2	2	1	20	8	160	78	98	10	356
5	1	4	1	7	20	140	34	41	7	222
6	1	4	1	14	20	280	69	83	14	446
7	2	4	2	28	28	784	166	222	56	1228
8	2	4	2	28	28	784	166	222	56	1228
9	10	18	7	2	612	1224	46	60	14	1344
10	8	20	10	2	740	1480	48	68	20	1616
Foulds 3	11	8	16	1	160	160	0	8	0	168
Foulds 4	11	8	16	1	160	160	0	8	0	168
Foulds 5	11	4	16	1	96	96	0	4	0	100

Note that sucs.ratio in the rescheduling method is less than 100% only when the rescheduling method terminates at Step 3.6.(II)2., as it cannot meet the quality requirements. In the tables, we show sucs.ratio, the optimal values of the relaxation methods, the computed objectives values of the test instances, execution time for each relaxation, and `fmincon` or the rescheduling method in Section 5. Sdp.ffs.nls means applying three procedures: (i) the SDP relaxation, (ii) FFS and (iii) `fmincon`. Socp and Lp mean the SOCP and LP relaxation, respectively. Socp.ffs.reschd and Lp.ffs.reschd denote that the instances were solved by applying the rescheduling method including the SOCP or LP relaxation, and FFS as described in Section 5. We note that Nishi's method employs the SDP relaxation of (3). The SDP relaxation in Sdp.ffs.nls is obtained from (4). As the number of constraints in (4) is larger than that of (3), it takes longer to solve the SDP relaxation of (4) than that of Nishi's method. In the column "Ffs.fmincon or Ffs.reschd", the computation time by

FFS and fmincon for the methods that employ fmincon is shown. FFS took very short time and most of CPU time was consumed by fmincon in the experiments. For the rescheduling method, the computational time for the entire rescheduling method excluding the conic relaxation is shown.

Table 3 displays the results on the test instances 1-4 in [16] whose number of variables varies from $n = 138$ to $n = 356$. For all instances, we see in the column Total that the SOCP and LP relaxation consumed much shorter CPU time than Nishi's method and the SDP relaxation of (4). The rescheduling method took much shorter CPU time than the nonlinear program solver `fmincon`. As a result, Socp.ffs.reschd and Lp.ffs.reschd show shorter total CPU time than the other methods, except for the instance 2. For the instance 2, the optimal value could be found by the SDP relaxation of (3), thus, `fmincon` did not take long to converge with the stopping criteria. In the column Relax, the CPU time by Socp.ffs.reschd and Lp.ffs.reschd is longer than that of Socp.ffs.nls and Lp.ffs.nls as the SOCP and LP relaxations are repeatedly solved in the rescheduling method as described in Algorithm 5.3.

For the objective values obtained by the methods in Table 3, we confirm that the SDP, SOCP and LP relaxations of (4) compute the equivalent objective values, as described in Section 4. For Socp.ffs.reschd and Lp.ffs.reschd, two values are shown for the column of Relax.obj.val to denote the objective value at the starting time and the final time, respectively, as the SOCP and LP relaxations are repeatedly solved. Socp.ffs.reschd and Lp.ffs.reschd frequently provide 100% sucs.ratio with the smallest objective values in the column of Obj.val. We observe that Socp.ffs.reschd and Lp.ffs.reschd are computationally efficient and effective to obtain smaller objective values than the other methods.

Table 4 shows the results for the instances 5, 6, 7, and 8 in [16] where n ranges from 222 to 1228. We also see in the column Total that the CPU time spend by Socp.ffs.reschd and Lp.ffs.reschd is much shorter than the other methods. In the column of Obj.val, the smallest objective values were obtained by Lp.ffs.reschd except for the instance 5. We notice that the highest sucs.ratio leads to the smallest objective value for all instances. For the instances 7 and 8 where n is large, Socp.ffs.reschd and Lp.ffs.resched achieve 100% sucs.ratio, and Lp.ffs.reschd consumed the shortest CPU time.

Next, we tested the proposed method on the problems with the minimum number of time discretization to see how large number of variables can be solved with the proposed method. The number of sources, intermediate tanks, and plants are shown in Table 2 and the intermediate tanks in the instances 9 and 10 are connected as a complete graph shown in Figure 5. The SDP relaxations for Nishi's method and (8) could not be solved since the sizes of the SDP relaxations were too large to handle on our computer. In Table 5, we observe that `fmincon` used in the methods Socp.ffs.nls and Lp.ffs.nls took long time, and could not provide a solution within 24 hours for the instances 9 and 10. On the other hand, the rescheduling method, Socp.ffs.reschd and Lp.ffs.reschd, successfully solve the problems in much shorter time. The objective values obtained by Socp.ffs.reschd and Lp.ffs.reschd are smaller than those of Socp.ffs.nls and Lp.ffs.nls, providing 100% sucs.ratio.

We tested the proposed method on Foulds 3, 4 and 5 which have the largest number of variables among the standard test problems for the pooling problem in P-formulation such as Haverly, Ben-Tal, Foulds, Adhya, and RT2. As time discretization is not used in the problems, the SDP, SOCP and LP relaxations and FFS are applied to Foulds 3, 4 and 5.

Table 3: The results for the instances 1, 2, 3, and 4. "more than 24 hours" means that the method could not provide a solution within 24 hours.

Problem	Instance 1 ($n = 138$)					
				CPU time (seconds)		
Methods	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	97.61%	202.3	11243.44	6.95	1925.17	1932.12
Sdp.ffs.nls	95.70%	222.5	19827.98	142.67	340.28	482.95
Socp.ffs.nls	97.83%	222.5	10262.53	12.51	218.38	230.89
Lp.ffs.nls	95.27%	222.5	21792.29	11.76	5.96	17.72
Socp.ffs.reschd	100.00%	222.5, 122.5	470.00	30.46	2.05	32.51
Lp.ffs.reschd	98.34%	222.5, 162.5	7944.94	20.29	1.25	21.54
Problem	Instance 2 ($n = 298$)					
	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	99.78%	617.92	3058.10	8.83	16.30	25.13
Sdp.ffs.nls	-	-	-	more than 24 hours		
Socp.ffs.nls	99.74%	800.00	3407.69	40.60	15.74	56.34
Lp.ffs.nls	99.87%	800.00	2253.85	29.54	15.73	45.28
Socp.ffs.reschd	100.00%	800.00, 687.06	1092.94	75.15	1.99	77.14
Lp.ffs.reschd	100.00%	800.00, 687.06	1092.94	61.69	1.92	63.61
Problem	Instance 3 ($n = 176$)					
	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	98.66%	181.14	6594.55	7.82	2866.87	2874.69
Sdp.ffs.nls	99.99%	300.00	546.91	548.06	2856.68	3404.74
Socp.ffs.nls	80.11%	300.00	89974.73	16.21	2885.36	2901.57
Lp.ffs.nls	99.99%	300.00	510.81	14.17	2892.56	2906.73
Socp.ffs.reschd	100.00%	300.00, 91.57	503.97	36.49	1.62	38.11
Lp.ffs.reschd	99.70%	300.00, 55.57	1852.02	32.75	1.78	34.52
Problem	Instance 4 ($n = 356$)					
	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	96.06%	728.18	36635.41	13.20	7044.23	7057.45
Sdp.ffs.nls	-	-	-	more than 24 hours		
Socp.ffs.nls	97.03%	900.00	27842.37	77.87	4294.91	4372.78
Lp.ffs.nls	99.48%	900.00	5758.37	58.74	7334.06	7392.80
Socp.ffs.reschd	100.00%	900.00, 41.87	1076.34	186.49	4.28	190.77
Lp.ffs.reschd	100.00%	900.00, 170.71	1062.41	161.04	5.71	166.75

Table 4: The results for the instances 5, 6, 7, and 8. "more than 24 hours" means that the method could not provide a solution within 24 hours.

Problem	Instance 5 ($n = 222$)					
				CPU time (seconds)		
Methods	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	99.44%	17.87	3046.25	13.33	185.67	198.99
Sdp.ffs.nls	92.59%	134.00	35658.10	3239.96	14.54	3254.50
Socp.ffs.nls	91.60%	134.00	40381.10	19.27	153.36	172.63
Lp.ffs.nls	92.76%	134.00	34890.66	17.81	786.83	804.64
Socp.ffs.reschd	92.62%	134.00, 54.00	35483.11	29.10	1.73	30.84
Lp.ffs.reschd	94.34%	134.00, 54.00	27358.17	30.07	1.60	31.67

Problem	Instance 6 ($n = 446$)					
	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	95.74%	186.81	41470.49	33.55	18000.13	18033.68
Sdp.ffs.nls	-	-	-	more than 24 hours		
Socp.ffs.nls	93.14%	334.00	66194.84	102.33	1055.38	1157.71
Lp.ffs.nls	96.51%	334.00	34102.63	80.02	9738.29	9818.31
Socp.ffs.reschd	96.11%	334.00, 134.00	37889.83	189.82	8.90	198.72
Lp.ffs.reschd	97.85%	334.00, 38.37	21272.54	226.01	8.09	234.10

Problem	Instance 7 ($n = 1228$)					
	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	99.51%	475.33	7176.40	263.59	39716.96	39980.55
Sdp.ffs.nls	-	-	-	more than 24 hours		
Socp.ffs.nls	99.99%	824.00	1649.13	1754.10	32448.45	34202.55
Lp.ffs.nls	99.99%	824.00	1667.51	1368.04	9707.45	11075.49
Socp.ffs.reschd	100.00%	824.00, 1916.96	1605.70	11761.98	674.45	12436.43
Lp.ffs.reschd	100.00%	824.00, 1996.89	1599.44	9177.46	688.87	9866.32

Problem	Instance 8 ($n = 1228$)					
	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	98.27%	458.22	22214.26	256.89	39443.43	39700.32
Sdp.ffs.nls	-	-	-	more than 24 hours		
Socp.ffs.nls	99.99%	824.00	1714.70	1746.94	20929.01	22675.94
Lp.ffs.nls	99.33%	824.00	8957.09	1368.09	39583.53	40951.62
Socp.ffs.reschd	100.00%	824.00, 20.07	1636.82	13311.83	800.91	14112.74
Lp.ffs.reschd	100.00%	824.00, 20.07	1634.99	8835.86	660.60	9496.46

Table 5: The results for the instances 9 and 10. The instances 9 and 10 with $M_T = 2$. "> 24 hours" means that the method could not provide a solution within 24 hours.

Problem	Instance 9 ($n = 1344$)					
				CPU time (seconds)		
	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	Fail to solve the SDP relaxation due to out-of-memory					
Socp.ffs.nls	82.83%	490.05	81347.30	805.78	> 24 hours	–
Lp.ffs.nls	91.39%	490.05	41146.81	588.55	> 24 hours	–
Socp.ffs.reschd	100.00%	490.05, 95.12	657.35	919.18	112.92	1032.10
Lp.ffs.reschd	100.00%	490.05, 94.77	657.35	674.68	112.96	787.64
Problem	Instance 10 ($n = 1616$)					
				CPU time (seconds)		
	Sucs.ratio	Relax.obj.val	Obj.val	Relax	Ffs.fmincon or Ffs.reschd	Total
Nishi's method	Fail to solve the SDP relaxation due to out-of-memory					
Socp.ffs.nls	86.00%	511.00	105842.11	1236.56	> 24 hours	–
Lp.ffs.nls	89.53%	511.00	81377.36	911.34	> 24 hours	–
Socp.ffs.reschd	100.00%	511.00, 108.87	759.06	1425.09	28.29	1453.38
Lp.ffs.reschd	100.00%	511.00, 102.26	759.06	1050.38	30.07	1080.45

Table 6 displays the numerical results on the test problems. We see that the objective value obtained by the SDP, SOCP and LP relaxations are equivalent. FFS was applied to find a feasible solution of the original pooling problem. From the CPU time spent by the SDP, SOCP and LP relaxations, we observe that the LP relaxation is most efficient. The optimal values of Foulds 3, 4 are known as -8 [15]. The proposed method finds an approximate solution of Foulds 3, 4 with $n = 168$ with much shorter computational time than that in [15]. As the instances 9 and10 are much larger than Foulds 3, 4 and 5, the proposed method can be applied to larger pooling problems than the standard pooling test problems.

7 Concluding remarks

We have proposed an efficient computational method for the pooling problem with time discretization using the SOCP and LP relaxations and the rescheduling method. From the form of QCQPs for the pooling problem in [16], our formulation with time discretization has been obtained by relaxing the equality constraints into inequality constraints and introducing penalty terms in the objective function. We have shown theoretically that there exists no gap among the optimal values of the SDP, SOCP and LP relaxations of our formulation. This theoretical result can be used in other formulations of the pooling problem where only bilinear terms appear.

Computational results have been presented to show the efficiency of the proposed method over the SDP relaxations of the pooling problems and applying a nonlinear programming solver. From the numerical results, we have demonstrated that the SOCP and LP relaxations are more computationally efficient than the SDP relaxations while obtaining the

Table 6: Numerical results on Foulds problems

Problem	Foulds 3 ($n = 168$)				
Relaxation			CPU time (seconds)		
	Relax.obj.val	FFS.obj.val	Relax.	Ffs	Total
Sdp.ffs	−9.00	−3.82	317.63	0.81	318.44
Socp.ffs	−9.00	−3.82	9.81	0.45	10.26
Lp.ffs	−9.00	−3.82	8.83	0.43	9.26
Problem	Foulds 4 ($n = 168$)				
	Relax.obj.val	FFS.obj.val	Relax.	Ffs	Total
Sdp.ffs	−9.00	−2.64	300.51	0.44	300.95
Socp.ffs	−9.00	−2.64	10.02	0.44	10.46
Lp.ffs	−9.00	−2.64	9.24	0.44	9.68
Problem	Foulds 5 ($n = 100$)				
	Relax.obj.val	FFS.obj.val	Relax.	Ffs	Total
Sdp.ffs	−11.00	−0.83	23.47	0.45	23.92
Socp.ffs	−11.00	−0.83	6.56	45	7.01
Lp.ffs	−11.00	−0.83	5.97	0.47	6.42

same optimal value. Moreover, our proposed rescheduling method is much faster than the nonlinear programming solver `fmincon` and effective in obtaining a solution that satisfies all the requirements. As a result, large instances up to $n = 1616$ could be solved with the LP relaxation and the rescheduling method.

As the pooling problem is a bilinear problem, it may be possible to utilize the structure [15] of the problem to further improve the computational efficiency. We hope to investigate the underlying structure of the pooling problem for solving large-scale problems.

References

[1] IBM ILOG CPLEX user's manual. IBM, Tech. Rep., 2015.

[2] A. A. Ahmadi, S. Dashb, and G. Hal. Optimization over structured subsets of positive semidefinite matrices via column generation. *Discrete Optim.*, 24:129–151, 2017.

[3] A. A. Ahmadi and A. Majumbar. DSOS and SDSOS optimization: Lp and socp-based alternatives to sum of squares optimization. In *Proceedings of the 48th Annual Conference on Information Sciences and Systems*, pages 1–5, 2014.

[4] A. A. Ahmadi and A. Majumdar. Spotless: Software for DSOS and SDSOS optimization, 2014.

[5] M. Alfaki. *Models and Solution Methods for the Pooling Problem.* PhD thesis, University of Bergen, Department of Informatics, University of Bergen, March 2012.

[6] M. Alfaki and D. Haugland. Strong formulations for the pooling problem. *J. Global Optim.*, 56:897–916, 2013.

[7] MOSEK ApS. The mosek optimization toolbox for matlab manual. version 7.1 (revision 28.), 2015.

[8] G. P. Barker and D. Carlson. Cones of diagonally dominant matrices. *Pacific Journal of Mathematics*, 57(1):15–32, 1975.

[9] A. Ben-Tal, G. Eiger, and V. Gershovitz. Global minimization by reducing the duality gap. *Math. Program.*, 63(1-3):193–212, 1994.

[10] C.E. Gounaris, R. Misener, and C.A. Floudas. Computational comparison of piecewise-linear relaxations for pooling problems. *Ind. Eng. Chem. Res.*, 48(12):5742–5766, 2009.

[11] C.A. Haverly. Studies of the behavior of recursion for the pooling problem. *ACM SIGMAP Bull.*, 25:19–28, 1978.

[12] S. Kim and M. Kojima. Second order cone programming relaxation of nonconvex quadratic optimization problems. *Optim. Methods and Softw.*, 15(3-4):201–224, 2001.

[13] S. Kim and M. Kojima. Exact solutions of some nonconvex quadratic optimization problems via SDP and SOCP relaxations. *Comput. Optim. Appl.*, 26:143–154, 2003.

[14] L. Libertu and C.C. Pantelides. An exact reformulation algorithm for large nonconvex nlps involving bilinear terms. *J. Global Optim.*, 36(2):161–189, 2006.

[15] A. Marandia, E. de Klerk, and J. Dahlc. Solving sparse polynomial optimization problems with chordal structure using the sparse, bounded-degree sum-of-squares hierarchy. *Discrete Appl. Math.*, to appear, 2018.

[16] T. Nishi. A semidefinite programming relaxation approach for the pooling problem a semidefinite programming relaxation approach for the pooling problem. Master's thesis, Kyoto University, Department of Applied Mathematics and Physics, Kyoto University, February 2010.

[17] J. F. Sturm. SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods and Softw.*, 11&12:625–653, 1999.

[18] R. H. Tütüncü, K. C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Math. Program.*, 95:189–217, 2003.

[19] H. Waki, S. Kim, M. Kojima, M. Muramatsu, and H. Sugimoto. Algorithm 883: SparsePOP: a sparse semidefinite programming relaxation of polynomial optimization problems. *ACM Trans. Math. Softw.*, 35(15), 2008.

[20] M. Yamashita, K. Fujisawa, M. Fukuda, K. Kobayashi, K. Nakata, and M. Nakata. Latest developments in the SDPA family for solving large-scale SDPs. In *Handbook on semidefinite, conic and polynomial optimization*, pages 687–713. Springer, 2012.

[21] M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0). *Optim. Methods and Softw.*, 18(4):491–505, 2003.

[22] E. Zanni. Can semidefinite programming be a key approach to the pooling problem? University of Edinburgh, March 2013.