A polynomially solvable case of the pooling problem

Natashia Boland
Georgia Institute of Technology, Atlanta, U.S.A.

Thomas Kalinowski Fabian Rigterink
The University of Newcastle, Australia

August 13, 2015

Abstract

Answering a question of Haugland, we show that the pooling problem with one pool and a bounded number of inputs can be solved in polynomial time by solving a polynomial number of linear programs of polynomial size. We also give an overview of known complexity results and remaining open problems to further characterize the border between (strongly) NP-hard and polynomially solvable cases of the pooling problem.

Keywords Pooling problem · Computational complexity

1 Introduction, motivation and problem definition

The pooling problem is a nonconvex nonlinear programming problem with applications in the refining and petrochemical industries [9, 14], mining [5, 7], agriculture, food manufacturing, and pulp and paper production [16]. Informally, the problem can be stated as follows: given a set of raw material suppliers (inputs) and qualities of the material, find a cost-minimizing way of blending these raw materials in intermediate pools and outputs so as to satisfy requirements on the final output qualities. The blending in pools and outputs introduces bilinear constraints and makes the problem hard.

While the pooling problem has been known to be hard in practice ever since its proposal by Haverly in 1978 [13], it was only formally proven to be strongly NP-hard by Alfaki and Haugland in 2013 [1]. Their proof of strong NP-hardness, however, considered a very general case of the problem, with arbitrary parameters and an arbitrary network structure. Once the parameters and the network structure are more specific, e.g., by bounding the number of vertices, their in- and out-degrees, or the number of qualities, the complexity of the problem needs to be re-examined. This way, several polynomially solvable cases of the pooling problem were proven [2, 11, 12]. However, the border between (strongly) NP-hard and polynomially solvable cases of the pooling problem is still only partially characterized. This is mainly due to the combinatorial explosion of parameter choices for the problem. In this paper, we solve an open problem that has been pointed out in [11, 12]: the pooling problem with one pool and a bounded number of inputs is in fact polynomially solvable.

We consider a directed graph $G = (V, A)$ where V is the set of vertices and A is the set of arcs. V is partitioned into three subsets $I, L, J \subset V$: I is the set of inputs, L is the set of pools and J is the set of outputs. Flows are blended in pools and outputs. The pooling problem literature addresses a variety of problem instances with $A \subseteq (I \times L) \cup (L \times L) \cup (L \times J) \cup (I \times J)$. Instances with $A \cap (L \times L) = \emptyset$

Table 1: Notation for the pooling problem

Sets	
V	Set of vertices
I	Set of inputs
L	Set of pools
J	Set of outputs
K	Set of qualities
A	Set of arcs
A_I	Set of input-to-pool arcs: $A_I := A \cap (I \times L)$
A_J	Set of pool-to-output arcs: $A_J := A \cap (L \times J)$
A_v^{out}	Set of outgoing arcs of $v \in I \cup L$
A_v^{in}	Set of incoming arcs of $v \in L \cup J$

Parameters	
c_a	Cost of flow on arc $a \in A$
λ_{ik}	Quality value of input $i \in I$ for quality $k \in K$
λ_{ak}	$\lambda_{ak} \equiv \lambda_{ik},\ a \in A_i^{\text{out}},\ i \in I,\ k \in K$
μ_{jk}	Upper bound on quality value of output $j \in J$ for quality $k \in K$
C_v	Upper bound on total flow through vertex $v \in V$
u_a	Upper bound on flow on arc $a \in A$

Variables	
x_a	Flow on arc $a \in A_I$
y_a	Flow on arc $a \in A_J$
$p_{\ell k}$	Quality value of pool $\ell \in L$ for quality $k \in K$
p_{ak}	$p_{ak} \equiv p_{\ell k},\ a \in A_\ell^{\text{out}},\ \ell \in L,\ k \in K$

are referred to as *standard pooling problems* (SPPs), and instances with $A \cap (L \times L) \neq \emptyset$ are referred to as *generalized pooling problems* (GPPs). Both SPPs and GPPs can be modelled as bilinear programs, which are special cases of nonlinear programs. Instances with $L = \emptyset$ are referred to as *blending problems* and can be modelled as linear programs.

In this paper (as in [2, 11, 12]), we study the complexity of SPPs where $A \subseteq (I \times L) \cup (L \times J)$, i.e., all arcs are either input-to-pool or pool-to-output arcs. For notational simplicity, we denote the set of the former by $A_I := A \cap (I \times L)$ and the set of the latter by $A_J := A \cap (L \times J)$. We do not consider input-to-output arcs since for every such arc (i, j), we can add an auxiliary pool ℓ and replace (i, j) by an input-to-pool arc (i, ℓ) and a pool-to-output arc (ℓ, j). Throughout this paper, we use the term *pooling problem* to refer to a SPP without input-to-output arcs. We consider a set of qualities K whose quality values are tracked across the network. We assume linear blending, i.e., the quality value of a pool or output for a quality is the linear combination of the incoming quality values weighted by the incoming flows as a fraction of the total incoming flow.

For inputs and pools $v \in I \cup L$, we denote the set of outgoing arcs of v by A_v^{out}, and for pools and outputs $v \in L \cup J$, we denote the set of incoming arcs of v by A_v^{in}. Let x_a be the flow on input-to-pool arc $a \in A_I$, and let y_a be the flow on pool-to-output arc $a \in A_J$. The cost of flow on arc $a \in A$ (which may be negative) is given by c_a. The total flow through vertex $v \in V$ (resp. the flow on arc $a \in A$) is bounded above by C_v (resp. u_a). For every input $i \in I$ and quality $k \in K$, the quality value of the incoming raw material is given by λ_{ik}. Let $p_{\ell k}$ denote the quality value of the blended raw materials in pool $\ell \in L$ for quality $k \in K$. For every output $j \in J$ and quality $k \in K$, the upper bound on the quality value of the outgoing blend is given by μ_{jk}. In addition to λ_{ik} and $p_{\ell k}$, it is sometimes more convenient to have arc-based rather than node based quality parameters and variables. Since the quality of flow on arc (v, w) is equal to the blended quality of the total flow through vertex v, we have $\lambda_{ik} \equiv \lambda_{ak}$ for all inputs $i \in I$, their outgoing arcs $a \in A_i^{\text{out}}$ and qualities $k \in K$. Analogously, we have $p_{\ell k} \equiv p_{ak}$ for all pools $\ell \in L$, their outgoing arcs $a \in A_\ell^{\text{out}}$ and qualities $k \in K$. Table 1 summarises the notation for the pooling problem.

We now present the classical formulation of the pooling problem, commonly referred to as the P-formulation [13]. There are numerous alternative formulations of the pooling problem, including the Q- [4], PQ- [15] and HYB-formulations [3], and most recently multi-commodity flow formulations [1, 2, 6]. All formulations are equivalent in the sense that there is a one-to-one correspondence between a feasible

solution of one formulation and another, and they all have the same optimal objective value. However, the alternative formulations often show a better computational performance than the P-formulation, as studied e.g. in [6]. A recent paper by Gupte et al. [10] gives an excellent overview of topics that have been studied in the context of the pooling problem. Within the scope of this paper, however, we chose to prove complexity results using the classical P-formulation.

In the P-formulation, a *feasible flow* $(\boldsymbol{x}, \boldsymbol{y})$ satisfies the following constraints:

$$\sum_{a \in A_\ell^{\text{in}}} x_a = \sum_{a \in A_\ell^{\text{out}}} y_a, \qquad \ell \in L, \tag{1}$$

$$\sum_{a \in A_i^{\text{out}}} x_a \leqslant C_i, \qquad i \in I, \tag{2}$$

$$\sum_{a \in A_\ell^{\text{in}}} x_a \leqslant C_\ell, \qquad \ell \in L, \tag{3}$$

$$\sum_{a \in A_j^{\text{in}}} y_a \leqslant C_j, \qquad j \in J, \tag{4}$$

$$x_a, y_a \leqslant u_a, \qquad a \in A_I,\ A_J, \text{ resp.} \tag{5}$$

Constraint (1) is flow conservation which ensures that at every pool, the total incoming flow equals the total outgoing flow. (2)–(4) are vertex capacity constraints and (5) is an arc capacity constraint. For notational simplicity, we denote the *set of feasible flows* by $\mathcal{F} := \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}_{\geqslant 0}^{|A_I|} \times \mathbb{R}_{\geqslant 0}^{|A_J|} : (1)\text{–}(5) \text{ are satisfied}\}$. The P-formulation can now be stated as follows:

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{p}} \quad & \sum_{a \in A_I} c_a x_a + \sum_{a \in A_J} c_a y_a \\
\text{s.t.} \quad & (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{F}, \\
& \sum_{a \in A_\ell^{\text{in}}} \lambda_{ak} x_a = p_{\ell k} \sum_{a \in A_\ell^{\text{out}}} y_a, \qquad \ell \in L,\ k \in K, && (6) \\
& \sum_{a \in A_j^{\text{in}}} p_{ak} y_a \leqslant \mu_{jk} \sum_{a \in A_j^{\text{in}}} y_a, \qquad j \in J,\ k \in K. && (7)
\end{aligned}$$

Equality (6) is the pool blending constraint which ensures that the p variables track the quality values across the network. Inequality (7) is the output blending constraint. We take the requirements that $\lambda_{ak} \equiv \lambda_{ik}$ for all $a \in A_i^{\text{out}}$, $i \in I$ and $k \in K$, and that $p_{ak} \equiv p_{\ell k}$ for all $a \in A_\ell^{\text{out}}$, $\ell \in L$ and $k \in K$, to be implicit in the model.

2 Known complexity results

Table 2 provides an overview of known complexity results, and Figure 1 shows most of these complexity results in a tree structure. All of these results were formally proven in [2, 11, 12]. When bounding the number of vertices, the cases of one input or output are polynomially solvable. Furthermore, the cases of one pool and a bounded number of outputs or qualities are polynomially solvable. If we only have one pool (and no other restrictions), then the problem remains strongly NP-hard. The same holds if we have only one quality. The problem also remains strongly NP-hard if we have one quality and two inputs or two outputs. Only if we have one quality, two inputs and two outputs, then the problem becomes NP-hard. Finally, the problem also remains strongly NP-hard if the out-degrees of inputs and outputs are bounded above by two, or if the in-degrees of pools and outputs are bounded above by two.

Table 2: Overview of known complexity results

#	bounded #vertices $\lvert I\rvert$	$\lvert L\rvert$	$\lvert J\rvert$	$\lvert K\rvert$	bounded in-/out-degrees $\forall i \in I$	$\forall \ell \in L$	$\forall j \in J$	Complexity	Reduction	Reference(s)
1	1							P		trivial
2		1						sNP	MIS	[2], Corollary 1; [11], Proposition 1; [12], Theorems 1–2
3			1					P		trivial
4				1				sNP	X3C	see #8, #9 and #11
5	$[1, i_{\max}]$	1						P		this paper
6		1	$[1, j_{\max}]$					P		[11], Proposition 2
7		1		$[1, k_{\max}]$				P		[2], Proposition 2; [11], Proposition 3
8	2			1				sNP	X3C	[12], Theorem 4
9			2	1				sNP	X3C	[12], Theorem 5
10	2		2	1				NP	BP2	[11], Proposition 5; [12], Theorem 3
11	$\min\{\lvert I\rvert, \lvert J\rvert\} = 2$			1	$\max\{\lvert A_\ell^{\text{in}}\rvert, \lvert A_\ell^{\text{out}}\rvert\} \leqslant 6$			sNP	X3C	[12], Corollary 1
12					$\lvert A_i^{\text{out}}\rvert \leqslant 2$	$\lvert A_\ell^{\text{out}}\rvert \leqslant 2$		sNP	MAX 2-SAT	[11], Proposition 7; [12], Theorem 6
13						$\lvert A_\ell^{\text{in}}\rvert \leqslant 2$	$\lvert A_j^{\text{in}}\rvert \leqslant 2$	sNP	MIN 2-SAT	[11], Proposition 6; [12], Theorem 7
14					$\min\{\lvert A_\ell^{\text{in}}\rvert, \lvert A_\ell^{\text{out}}\rvert\} = 1$			P		[11], Proposition 4; [12], Proposition 3

P = polynomial, NP = NP-hard, sNP = strongly NP-hard,

BP2 = bin packing with 2 bins, MAX 2-SAT = maximum 2-satisfiability, MIN 2-SAT = minimum 2-satisfiability, MIS = maximal independent set, X3C = exact cover by 3-sets

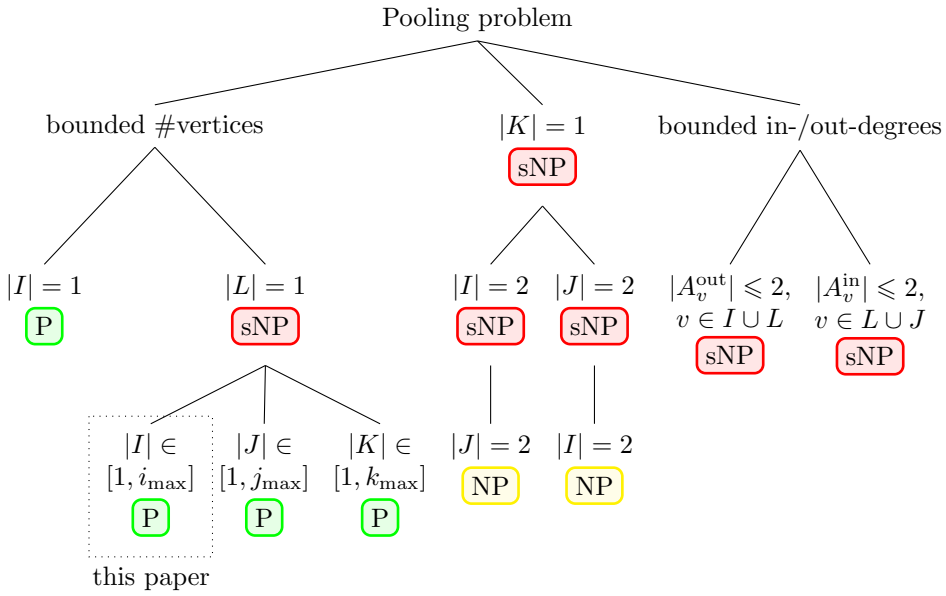

Figure 1: Overview of known complexity results in a tree structure. For simplicity, we omit #11 and #14 from Table 2.

3 The pooling problem with one pool and a bounded number of inputs

In this section, we consider the pooling problem with

- m inputs (let $I = \{v_1, \dots, v_m\}$),
- one pool (let $L = \{\ell\}$),
- n outputs (let $J = \{w_1, \dots, w_n\}$),
- q qualities (let $K = \{1, \dots, q\}$),
- the set of input-to-pool arcs $A_I = \{a_1, \dots, a_m\} = \{(v_1, \ell), \dots, (v_m, \ell)\}$, and
- the set of pool-to-output arcs $A_J = \{a_{m+1}, \dots, a_{m+n}\} = \{(\ell, w_1), \dots, (\ell, w_n)\}$.

The graph of this special case of the pooling problem is shown in Figure 2. We write

- x_i for the flow on input-to-pool arc a_i $(i = 1, \dots, m)$,
- y_j for the flow on pool-to-output arc a_{m+j} $(j = 1, \dots, n)$,
- c_i for the cost of flow on arc a_i $(i = 1, \dots, m+n)$,
- λ_{ik} for the k-th quality value at the tail node of input-to-pool arc a_i $(i = 1, \dots, m)$, and
- μ_{jk} for the bound on the k-th quality value at the head node of arc a_{m+j} $(j = 1, \dots, n)$.

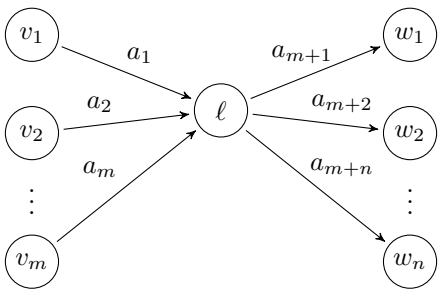

Figure 2: Graph of the pooling problem with one pool and a bounded number of inputs

For a positive integer N, we use $[N]$ to denote the set $\{1, 2, \ldots, N\}$. If for some $j \in [n]$, there exists a $k \in [q]$ such that $\min\{\lambda_{ik} : i \in [m]\} > \mu_{jk}$, then $y_j = 0$ in every feasible solution. Hence, without loss of generality, we assume for all $j \in [n]$ and $k \in [q]$, $\min\{\lambda_{ik} : i \in [m]\} \leqslant \mu_{jk}$. Note that $y_j > 0$ implies

$$\forall k \in [q] \qquad \sum_{i=1}^{m} \lambda_{ik} x_i \leqslant \mu_{jk} \sum_{i=1}^{m} x_i. \tag{8}$$

It has been observed, for instance in [12], that for a fixed set $J' \subseteq [n]$ of outputs, an optimal solution that satisfies the quality constraints for all $j \in J'$ and has $y_j = 0$ for all $j \in [n] \setminus J'$, can be found by solving the following linear program which we denote by $\mathrm{LP}(J')$:

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{y}} \quad & \sum_{i=1}^{m} c_i x_i + \sum_{j \in J'} c_j y_j \\
\text{s.t.} \quad & (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{F}, \\
& \sum_{i=1}^{m} x_i = \sum_{j \in J'} y_j, \\
\sum_{i=1}^{m-1} (\lambda_{ik} - \lambda_{mk}) x_i \leqslant (\mu_{jk} - \lambda_{mk})(x_1 + \cdots + x_m), & \qquad j \in J',\ k \in [q].
\end{aligned}$$

Let $\mathrm{val}(J')$ denote the optimal value of problem $\mathrm{LP}(J')$. An optimal solution for the pooling problem can be obtained by solving $\mathrm{LP}(J')$ for every $J' \subseteq [n]$, and choosing one with minimum $\mathrm{val}(J')$. Below we argue that if the number m of inputs is fixed, then it is sufficient to consider a polynomial number of subsets J', where the polynomial is of degree $m - 1$ in both n and q.

Introducing variables $z_i = x_i / \sum_{i'=1}^{m} x_{i'}$ for $i \in [m-1]$, condition (8) can be rewritten as

$$\forall k \in [q] \qquad \sum_{i=1}^{m-1} (\lambda_{ik} - \lambda_{mk}) z_i \leqslant \mu_{jk} - \lambda_{mk}. \tag{9}$$

The vector $\boldsymbol{z}$ is an element of the simplex $\Delta^{m-1} = \{\boldsymbol{z} \in [0,1]^{m-1} : z_1 + \cdots + z_{m-1} \leqslant 1\}$. For $\boldsymbol{z} \in \Delta^{m-1}$, we define the *reachable output set* $J(\boldsymbol{z})$ as

$$J(\boldsymbol{z}) = \left\{ j \in [n] : \sum_{i=1}^{m-1} (\lambda_{ik} - \lambda_{mk}) z_i \leqslant \mu_{jk} - \lambda_{mk} \text{ for all } k \in [q] \right\}. \tag{10}$$

Lemma 1. *The objective value for any feasible flow corresponding to $\boldsymbol{z} \in \Delta^{m-1}$ is at least* $\mathrm{val}(J(\boldsymbol{z}))$.

Proof. For a fixed $\boldsymbol{z} \in \Delta^{m-1}$, we can find the optimal flow by solving the linear program

$$\begin{aligned}
\min_{\boldsymbol{x},\boldsymbol{y}} \quad & \sum_{i=1}^{m} c_i x_i + \sum_{j \in J(\boldsymbol{z})} c_j y_j \\
\text{s.t.} \quad & (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{F}, \\
& \sum_{i=1}^{m} x_i = \sum_{j \in J(\boldsymbol{z})} y_j, \\
& x_i = z_i (x_1 + \cdots + x_m), \qquad i \in [m].
\end{aligned}$$

Every feasible solution for this problem is also feasible for $\mathrm{LP}(J(\boldsymbol{z}))$ and the claim follows. □

The inequalities (9) define a partition of $\mathbb{R}^{m-1}$ (and therefore of Δ^{m-1}) into regions of constant $J(\boldsymbol{z})$. To be more precise, let $\mathcal{H}$ be the hyperplane arrangement $\mathcal{H} = \{H_{jk} : j \in [n],\, k \in [q]\}$, where

$$H_{jk} = \left\{ \boldsymbol{z} \in \mathbb{R}^{m-1} : \sum_{i=1}^{m-1} (\lambda_{ik} - \lambda_{mk}) z_i = \mu_{jk} - \lambda_{mk} \right\}.$$

The system $\mathcal{H}$ induces a partition of $\mathbb{R}^{m-1}$. Let H^0_{jk} and H^1_{jk} be defined by

$$H^0_{jk} = \left\{ \boldsymbol{z} \in \mathbb{R}^{m-1} : \sum_{i=1}^{m-1} (\lambda_{ik} - \lambda_{mk}) z_i \leqslant \mu_{jk} - \lambda_{mk} \right\},$$
$$H^1_{jk} = \left\{ \boldsymbol{z} \in \mathbb{R}^{m-1} : \sum_{i=1}^{m-1} (\lambda_{ik} - \lambda_{mk}) z_i > \mu_{jk} - \lambda_{mk} \right\}.$$

If, for every vector $\boldsymbol{\varepsilon} = (\varepsilon_{jk})_{j\in[n],\, k\in[q]} \in \{0,1\}^{nq}$, we define the set

$$P(\boldsymbol{\varepsilon}) = \bigcap_{j=1}^{n} \bigcap_{k=1}^{q} H^{\varepsilon_{jk}}_{jk},$$

then the space $\mathbb{R}^{m-1}$ is the disjoint union of the sets $P(\boldsymbol{\varepsilon})$, and for every $\boldsymbol{z} \in \Delta^{m-1}$ the set $J(\boldsymbol{z})$ is determined by the vector $\boldsymbol{\varepsilon}$ with $\boldsymbol{z} \in P(\boldsymbol{\varepsilon})$.

Lemma 2. *For $\boldsymbol{\varepsilon} \in \{0,1\}^{nq}$, let $J(\boldsymbol{\varepsilon}) = \{j \in [n] \ : \ \forall k \in [q]\ \varepsilon_{jk} = 0\}$. Then, for all $\boldsymbol{\varepsilon} \in \{0,1\}^{nq}$ and for all $\boldsymbol{z} \in P(\boldsymbol{\varepsilon}) \cap \Delta^{m-1}$, we have $J(\boldsymbol{z}) = J(\boldsymbol{\varepsilon})$.*

Proof. Let $\boldsymbol{\varepsilon} \in \{0,1\}^{nq}$ and $\boldsymbol{z} \in P(\boldsymbol{\varepsilon}) \cap \Delta^{m-1}$. Then

$$j \in J(\boldsymbol{z}) \quad \overset{(10)}{\iff} \quad \forall k \in [q] \quad \boldsymbol{z} \in H^0_{jk} \quad \overset{\boldsymbol{z}\in P(\boldsymbol{\varepsilon})}{\iff} \quad \forall k \in [q] \quad \varepsilon_{jk} = 0 \quad \iff \quad j \in J(\boldsymbol{\varepsilon}). \qquad \square$$

It is well known that the number of nonempty sets $P(\boldsymbol{\varepsilon})$ is bounded by a polynomial of degree m in nq (see for example [8]). In order to keep this paper self-contained, we include the short proof of the precise statement that is needed in our context.

Lemma 3. *There are at most $\displaystyle\sum_{i=0}^{m-1} \binom{q}{i} n^i$ vectors $\boldsymbol{\varepsilon} \in \{0,\, 1\}^{nq}$ such that $P(\boldsymbol{\varepsilon}) \neq \emptyset$.*

Proof. We proceed by induction on m. The case $m = 1$ is trivial as $\boldsymbol{\varepsilon} = \mathbf{0}$ is the only vector with $P(\boldsymbol{\varepsilon}) \neq \emptyset$, namely $P(\boldsymbol{\varepsilon}) = \{0\} = \mathbb{R}^0$. For $m = 2$, the nq inequalities partition $\mathbb{R}^1$ into at most $1 + nq$ intervals, so we have established the base for the induction. For $m \geqslant 3$, we proceed by induction on q. For $q = 1$, the n parallel hyperplanes $H_{11}, \ldots, H_{n1}$ partition $\mathbb{R}^{m-1}$ into at most $1 + n = \binom{1}{0} n^0 + \binom{1}{1} n^1$ parts. Now let $q > 1$. By induction on q, the system $\{H_{jk} : j \in [n],\ k \in [q-1]\}$ cuts $\mathbb{R}^{m-1}$ into at most $\sum_{i=0}^{m-1} \binom{q-1}{i} n^i$ parts. By induction on m, for every $j \in [n]$, the hyperplane H_{jq} is cut by the system $\{H_{j'k} \cap H_{jq} \ : \ j' \in [n],\ k \in [q-1]\}$ into at most

$$\sum_{i=0}^{m-2} \binom{q-1}{i} n^i$$

parts. Therefore, the number of parts into which $\mathbb{R}^{m-1}$ is cut by $\mathcal{H}$ is at most

$$\sum_{i=0}^{m-1}\binom{q-1}{i}n^i + n\sum_{i=0}^{m-2}\binom{q-1}{i}n^i = \sum_{i=0}^{m-1}\binom{q-1}{i}n^i + \sum_{i=1}^{m-1}\binom{q-1}{i-1}n^i$$
$$= \binom{q-1}{0}n^0 + \sum_{i=1}^{m-1}\left(\binom{q-1}{i} + \binom{q-1}{i-1}\right)n^i = \sum_{i=0}^{m-1}\binom{q}{i}n^i. \quad \square$$

Note that the proof of Lemma 3 also provides a recursive method to determine the vectors $\boldsymbol{\varepsilon}$ with $P(\boldsymbol{\varepsilon}) \neq \emptyset$ in polynomial time.

Theorem 1. *For every positive integer m, the pooling problem with one pool and m inputs can be solved in polynomial time. More precisely, it can be reduced to solving at most*

$$\sum_{i=0}^{m-1}\binom{q}{i}n^i$$

linear programs with $m+n$ variables and $m+n(q+1)+2$ constraints, where q is the number of qualities and n is the number of outputs.

Proof. We claim that the pooling problem can be solved by choosing a minimum cost solution obtained from solving the problem $\mathrm{LP}(J(\boldsymbol{\varepsilon}))$ for every $\boldsymbol{\varepsilon}$ with $P(\varepsilon) \cap \Delta^{m-1} \neq \emptyset$, and by Lemma 3 the number of these linear programs is bounded as claimed. Clearly, $B = \min\{\mathrm{val}(J(\boldsymbol{\varepsilon})) \ : \ P(\boldsymbol{\varepsilon}) \cap \Delta^{m-1} \neq \emptyset\}$ is an upper bound because a solution for $\mathrm{LP}(J(\boldsymbol{\varepsilon}))$ is always feasible for the pooling problem. By Lemma 2, for every $\boldsymbol{z} \in \Delta^{m-1}$ there exists some $\boldsymbol{\varepsilon}$ with $J(\boldsymbol{z}) = J(\boldsymbol{\varepsilon})$, and using Lemma 1 it follows that B is also a lower bound. $\square$

4 Remaining open problems

To further characterize the complexity of the pooling problem, the following open problems could be addressed in the future [11, 12]:

1. For all the cases that can be solved in polynomial time by reduction to polynomially many linear programs of polynomial size, does there exist a *strongly* polynomial algorithm, i.e., an algorithm that is polynomial in the number of vertices and qualities?
2. Is the pooling problem with one quality and in-degrees at most two polynomially solvable?
3. Is the pooling problem with one quality and out-degrees at most two polynomially solvable?
4. Do polynomial algorithms exist for the pooling problem with two pools and some bounds on the number of inputs, outputs, and qualities?

Acknowledgements This research was supported by the ARC Linkage Grant no. LP110200524, Hunter Valley Coal Chain Coordinator (hvccc.com.au) and Triple Point Technology (tpt.com).

References

[1] M. Alfaki and D. Haugland. A multi-commodity flow formulation for the generalized pooling problem. *Journal of Global Optimization*, 56(3):917–937, 2013.

[2] M. Alfaki and D. Haugland. Strong formulations for the pooling problem. *Journal of Global Optimization*, 56(3):897–916, 2013.

[3] C. Audet, J. Brimberg, P. Hansen, S. Le Digabel, and N. Mladenović. Pooling problem: Alternate formulations and solution methods. *Management Science*, 50(6):761–776, 2004.

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

[5] N. Boland, T. Kalinowski, and F. Rigterink. Discrete flow pooling problems in coal supply chains. 2015. optimization-online:2015/07/5041.

[6] N. Boland, T. Kalinowski, and F. Rigterink. New multi-commodity flow formulations for the pooling problem. 2015. optimization-online:2015/06/4959.

[7] N. Boland, T. Kalinowski, F. Rigterink, and M. Savelsbergh. A special case of the generalized pooling problem arising in the mining industry. 2015. optimization-online:2015/07/5025.html.

[8] R. C. Buck. Partition of space. *The American Mathematical Monthly*, 50(9):541–544, 1943.

[9] C. W. DeWitt, L. S. Lasdon, A. D. Waren, D. A. Brenner, and S. A. Melhem. OMEGA: An improved gasoline blending system for Texaco. *Interfaces*, 19(1):85–101, 1989.

[10] A. Gupte, S. Ahmed, S. S. Dey, and M. S. Cheon. Relaxations and discretizations for the pooling problem. 2015. optimization-online:2015/04/4883.

[11] D. Haugland. The hardness of the pooling problem. In L. G. Casado, I. García, and E. M. T. Hendrix, editors, *Proceedings of the XII global optimization workshop, mathematical and applied global optimization, MAGO 2014*, pages 29–32, Málaga, September 2014.

[12] D. Haugland. The computational complexity of the pooling problem. *Journal of Global Optimization*, pages 1–17, 2015. DOI: 10.1007/s10898-015-0335-y.

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

[14] B. Rigby, L. S. Lasdon, and A. D. Waren. The evolution of Texaco's blending systems: From OMEGA to StarBlend. *Interfaces*, 25(5):64–83, 1995.

[15] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*, volume 65 of *Nonconvex Optimization and its Applications*. Springer US, 2002.

[16] V. Visweswaran. MINLP: Applications in blending and pooling problems. In *Encyclopedia of Optimization*, pages 2114–2121. Springer US, 2009.