Pooling Problem: Alternate Formulations and Solution Methods

Charles Audet

GERAD and École Polytechnique de Montréal, C.P. 6079, Succursalle Centre-ville, Montréal, Québec, Canada H3C 3A7,
charles.audet@gerad.ca

Jack Brimberg

School of Business Administration, University of Prince Edward Island,
Charlottetown, Prince Edward Island, Canada C1A 4P3, jbrimberg@upei.ca

Pierre Hansen, Sébastien Le Digabel, Nenad Mladenović

GERAD and École des Hautes Études Commerciales, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Québec, Canada H3T 2A7
{pierreh@crt.umontreal.ca, sebastien.le.digabel@gerad.ca, nenad@crt.umontreal.ca}

T he pooling problem, which is fundamental to the petroleum industry, describes a situation in which products possessing different attribute qualities are mixed in a series of pools in such a way that the attribute qualities of the blended products of the end pools must satisfy given requirements. It is well known that the pooling problem can be modeled through bilinear and nonconvex quadratic programming. In this paper, we investigate how best to apply a new branch-and-cut quadratic programming algorithm to solve the pooling problem. To this effect, we consider two standard models: One is based primarily on flow variables, and the other relies on the proportion of flows entering pools. A hybrid of these two models is proposed for general pooling problems. Comparison of the computational properties of flow and proportion models is made on several problem instances taken from the literature. Moreover, a simple alternating procedure and a variable neighborhood search heuristic are developed to solve large instances and compared with the well-known method of successive linear programming. Solution of difficult test problems from the literature is substantially accelerated, and larger ones are solved exactly or approximately.

1. Introduction

The classical blending problem arises in refinery processes when feeds possessing different attribute qualities, such as sulfur composition, density, or octane number, are mixed directly together into final products. A generalization known as the pooling problem is used to model many actual systems that have intermediate mixing (or pooling) tanks in the blending process. The latter problem may be stated in a general way as follows: Given the availabilities of a number of feeds, what quantities should be mixed in intermediate pools to meet the demands of various final blends whose attribute qualities must meet known requirements? There are usually several ways of satisfying the requirements, each way having its cost. The question to be answered consists of identifying the one that maximizes the difference between the revenue generated by selling the final blends and the cost of purchasing the feeds. The need for blending occurs, for example, when there are fewer pooling tanks available than feeds or final products, or simply when the requirements of a demand product are not

met by any single feed. The classical blending problem may be formulated as a linear program, whereas the pooling problem has nonlinear terms and may be formulated as a bilinear program (BLP), which is a particular case of a nonconvex quadratic program with nonconvex constraints (QP).

In this paper, we investigate how best to solve the pooling problem with a recent algorithm for QP (Audet et al. 2000). The results clearly show that the type of model formulation chosen and initial heuristic used may significantly affect the performance of the exact method.

The general BLP problem is usually formulated by dividing a set of variables into two subsets: linear and nonlinear variables (see, e.g., Baker and Lasdon 1985). The nonlinear variables may be further divided into two disjoint subsets, called noncomplicating and complicating, respectively. This partition is based on the fact that it is always possible to get a linear program when the variables from either subset are fixed. It may be found by solving the minimal transversal of the corresponding (unweighted) co-occurrence graph

(Hansen and Jaumard 1992), where vertices represent variables and edges represent bilinear terms either in the objective function or in the constraint set. Reformulation is a powerful tool of mathematical programming. In Audet et al. (1997) it was used to study relationships between structured global optimization problems and algorithms, revealing embeddings of algorithms one into the other and unifying them.

To formulate the BLP problem, which will be referred to in later sections, we introduce the following notation.

- n: number of variables
- m: number of constraints
- n_1: number of linear variables
- n_2: number of nonlinear noncomplicating variables
- n_3: number of nonlinear complicating variables ($n = n_1 + n_2 + n_3$)
- m_1: number of linear constraints
- m_2: number of nonlinear constraints ($m = m_1 + m_2$)
- $x = (x_1, \ldots, x_{n_1})^T$: linear variables
- $y = (y_1, \ldots, y_{n_2})^T$: nonlinear noncomplicating variables
- $z = (z_1, \ldots, z_{n_3})^T$: nonlinear complicating variables
- c_x, c_y, c_z: constant vectors from R^{n_1}, R^{n_2}, and R^{n_3}, respectively
- C: $n_2 \times n_3$ matrix (if $c_{ij} \neq 0$, then bilinear term $y_i z_j$ exists in the objective function)
- $b = (b_1, \ldots, b_m)^T$: constants on the right-hand side of constraints
- $g_{i,x}, g_{i,y}, g_{i,z}$: constant vectors from R^{n_1}, R^{n_2}, and R^{n_3}, respectively, $i = 1, \ldots, m$
- G_i: $n_2 \times n_3$ matrix, ($i = m_1 + 1, \ldots, m$)

The general BLP problem may then be represented as

$$\max f(x, y, z) = c_x^T x + c_y^T y + c_z^T z + y^T C z$$

s.t.

$$g_i(x, y, z) = g_{i,x}^T x + g_{i,y}^T y + g_{i,z}^T z \leq b_i, \quad i = 1, \ldots, m_1$$

$$g_i(x, y, z) = g_{i,x}^T x + g_{i,y}^T y + g_{i,z}^T z + y^T G_i z \leq b_i,$$
$$i = m_1 + 1, \ldots, m.$$

BLP is a strongly NP-hard problem since it subsumes the strongly NP-hard linear maxmin problem (Hansen et al. 1992). Moreover, simply finding a feasible solution is NP-hard as the constraint set generalizes the NP-hard linear complementarity problem (Chung 1989). The objective function is neither convex nor concave, and the feasible region is not convex and may even be disconnected.

The nonlinear structure of the pooling problem was first pointed out in the famous example by Haverly (1978). To illustrate the potential difficulty of the problem, a two-step iterative algorithm was presented. It consists of estimating and fixing the attribute qualities of the intermediate pools, then solving the resulting linear program. If the resulting qualities coincide with the estimated ones, stop; otherwise update the values and reiterate the steps. It is shown by Haverly (1978) that this process may not lead to a global optimum.

Refinery modeling produces large, sparse linear programs containing small bilinear subproblems. Successive linear programming (SLP) algorithms were designed for this type of problem. Such algorithms solve nonlinear optimization problems through a sequence of linear programs. The idea of the method consists of replacing bilinear terms by first-order Taylor expansions to obtain a direction in which to move. A step (of bounded length for convergence reasons) is taken in that direction, and the process is reiterated. The first paper on SLP was that of Griffith and Stewart (1961) of Shell Oil, who referred to the method as mathematical approximation programming (later usage replaced that name with SLP). Other SLP algorithms are detailed in Palacios-Gomez et al. (1982) and Zhang et al. (1985), and applications at Exxon are discussed in Baker and Lasdon (1985). Lasdon and Joffe (1990) show that the method implemented in commercial packages called distributive recursion is equivalent to SLP through a change of variables. A method using Benders' decomposition is detailed in Floudas and Aggarwal (1990). As in the previous methods, this procedure does not guarantee identification of the global optimum. Floudas and Visweswaran (1993a) propose a decomposition-based global optimization algorithm (GOP), improved in Floudas and Visweswaran (1993b, 1996) and proven to achieve a global ϵ-optimum. Androulakis et al. (1996) discuss a distributed implementation of the algorithm and present computational results for randomly generated pooling problems with up to 5 pools, 4 blends, 12 feeds, and 30 attributes. Analyzing continuous branch-and-bound algorithms, Dür and Horst (1997) show that the duality gap goes to zero for some general nonconvex optimization problems that include the pooling problem.

Lodwick (1992) discusses preoptimization and postoptimization analyses of the BLP model. This work allows identification of some constraints that will be tight and others that will be redundant at optimality. Foulds et al. (1992) apply Al-Khayyal and Falk's (1983) branch-and-bound algorithm for BLP to the pooling problem. This method finds in finite time a solution as close as desired to a globally optimal one. The general idea consists of replacing each bilinear term with a linear variable and adding linear constraints to force the linear variable to be equal to the bilinear term. This is done by taking the convex and concave envelopes of the bilinear function $g: \mathbb{R}^2 \to \mathbb{R}$,

$g(x, y) \mapsto xy$ over a hyper-rectangle. The branching rule of the algorithm then splits the hyper-rectangle in its middle and recursively explores the two parts. The method is extremely sensitive to the bounds on the variables. Computational time increases rapidly with the number of variables that are not at one of their bounds at optimality. Audet et al. (2000) strengthen several aspects of Al-Khayyal and Falk's (1983) algorithm (the improvements will be discussed later, when computational results are presented). This algorithm is also based on the reformulation-linearization techniques of Sherali and Tuncbilek (1992; 1997a, b).

Ben-Tal et al. (1994) use the proportion model (detailed in §2.2 below) to solve the pooling problem. The variables are partitioned into two groups: q and (x, y). The bilinear program can then be written

$$\max_{q \in Q} \max_{(x, y) \in P(q)} f(q, x, y),$$

where Q is a simplex, $P(q)$ is a set that depends on q, and f is a bilinear function. By taking the dual of the second parameterized program, the bilinear problem can be rewritten into an equivalent semi-infinite linear program, in which the constraints must be satisfied for all the proportion vectors q of the simplex Q. This problem is relaxed by taking only the constraints corresponding to the vertices of Q. These steps are integrated into a branch-and-bound algorithm that partitions Q into smaller sets until the duality gap is null.

Grossmann and Quesada (1995) study a more sophisticated modelization for general process networks (such as splitters, mixers, and linear process units that involve multicomponent streams), with two different formulations based on components' compositions and total flows. They use a reformulation-linearization technique (Alameddine and Sherali 1992) to obtain a valid lower bound given by an LP-relaxation. This reformulation is then used within a spatial branch-and-bound algorithm.

Amos et al. (1997) use cumulative functions describing the distillation yield to model the pooling problem through a nonlinear constrained least-square problem. Promising results are shown in examples from the New Zealand Refining Company.

Adhya et al. (1999), after an exhaustive literature review, obtain tighter lower bounds with their Lagrangian approach because of the original choice of the relaxed constraints, which is not made to obtain easier-to-solve subproblems (in fact, after reformulation, they have a mixed-integer problem). They test their algorithm with the global optimization software BARON (Sahinidis 1996) on several problems from the literature and on four new difficult instances.

This paper is divided into three parts. In the next section, we analyze two different ways of modeling the pooling problem into BLP problems. The variables are partitioned into either *flow* and *attribute* variables (as suggested in Haverly 1978 and made more formal in Foulds et al. 1992) or *flow* and *proportion* variables (as in Ben-Tal et al. 1994). A combination of these two, called the hybrid model, is also presented for the generalized pooling problem. This type of formulation is seen to affect the number of nonlinear variables obtained in the model. The second part of the paper applies a recent branch-and-cut algorithm (Audet et al. 2000) to the flow and proportion models of several problem instances taken from the literature. The results suggest that the proportion formulation is preferable for this algorithm. The computational results also demonstrate that good heuristics are needed to obtain starting solutions for exact methods and solve larger problem instances approximately. The last part of the paper introduces a new variable neighborhood search heuristic (VNS) and compares this method to the successive linear programming method and the alternate procedure. VNS is seen to outperform the existing heuristics on the same set of problem instances from the literature as well as on large randomly generated problem sets.

2. Model Formulation

The classical blending problem determines the optimal way of mixing feeds directly into blends. The basic structure of the pooling problem is similar except for one set of additional intermediate pools, where the feeds are mixed prior to being directed to the final blends. Therefore, there are three types of pools: source pools, having a single purchased feed as input; intermediate pools with multiple inputs and outputs; and final pools, having a single final blend as output. The objective function is derived through the input of the source pools and the output of the final pools.

It is assumed that the intermediate pools receive flow from at least two feeds and are connected to at least two blends. The motivation behind these conditions is that if either is not satisfied, the intermediate pool can be eliminated by merging it to the feed or the blend, thus trivially simplifying the model. Let F_i, P_j, and B_k denote feed i, pool j, and blend k, respectively. The following parameters are introduced.

n^F, n^P, n^B, n^A: number of feeds, intermediate pools, blends and attribute qualities

X: the set of indices $\{(i, k)$: there is an arc from F_i to $B_k\}$

W: the set of indices $\{(i, j)$: there is an arc from F_i to $P_j\}$

Y: the set of indices $\{(j, k)$: there is an arc from P_j to $B_k\}$

p_i^F, p_k^B: prices of feed i and blend k for $i = 1, 2, \ldots, n^F$ and $k = 1, 2, \ldots, n^B$

l_i^F, u_i^F: lower and upper bounds on the availability of feed i for $i = 1, 2, \ldots, n^F$

l_k^B, u_k^B: lower and upper bounds on the demand of blend k for $k = 1, 2, \ldots, n^B$

l_{ik}^{FB}, u_{ik}^{FB}: lower and upper bounds on the capacity of arc $(i, k) \in X$

l_j^P, u_j^P: lower and upper bounds on the capacity of pool j for $j = 1, 2, \ldots, n^P$

l_{ij}^{FP}, u_{ij}^{FP}: lower and upper bounds on the capacity of arc $(i, j) \in W$

l_{jk}^{PB}, u_{jk}^{PB}: lower and upper bounds on the capacity of arc $(j, k) \in Y$

s_i^a: attribute quality a of feed i for $a = 1, 2, \ldots, n^A$ and $i = 1, 2, \ldots, n^F$

l_k^a, u_k^a: lower and upper bounds on the requirements of attribute quality a of blend k for $a = 1, 2, \ldots, n^A$ and $k = 1, 2, \ldots, n^B$

Throughout the paper, the following notation is used concerning sets of pairs of indices such as X. For a given first element i of a pair of indices, $X_{(i)}$ is defined to be the set of forward indices $\{k: (i, k) \in X\}$, and for a given second element k, $X_{(k)}^{-1}$ is the set of backward indices $\{i: (i, k) \in X\}$. The set of forward and backward indices of elements of W and Y are defined in a similar fashion. The flow conservation property allows the number of variables to be reduced. To do so, at each intermediate pool, the flow of one of the entering feeds is deduced by the difference of the total exiting blend and the sum of the other entering feeds. An additional index parameter that identifies which entering feed is deduced from the others is required; we denote by

$i(j)$: the smallest index of $W_{(j)}^{-1}$ for $j = 1, 2, \ldots, n^P$.

When different products are mixed together, it is assumed that the attribute qualities blend linearly: The attribute quality of the pool or blend is the weighted sum of the entering streams, where each weight is the volume proportion of the corresponding entering stream over the total volume. In what follows, we assume that all attribute qualities blend in this manner. DeWitt et al. (1989) present more precise models for octane and distillation blending. However, because of their complexity (they contain logarithms or fourth-order terms), these models are not considered in this paper.

The flow variables follow.

x_{ik}: flow from F_i to B_k on the arc $(i, k) \in X$

w_{ij}: flow from F_i to P_j on the arc $(i, j) \in W$

y_{jk}: flow from P_j to B_k on the arc $(j, k) \in Y$

We present two bilinear formulations of the pooling problem that differ in the representation of the flow from the feeds to the intermediate pools.

2.1. Flow Model of the Pooling Problem

In this subsection, we develop a BLP model of the pooling problem based on the primary flow variables. For each arc (i, j) in W, the flow originating from feed i to pool j is denoted by the variable w_{ij}, except when i is the index $i(j)$. Recall that $i(j) \in W_{(j)}^{-1}$ is defined as the smallest index of the input feed connected to the intermediate pool P_j. The flow conservation property ensures that the flow on the arc $(i(j), j)$ of W is the difference between the total flow exiting P_j and the flows on the other arcs entering P_j:

$$w_{i(j)j} = \sum_{k \in Y_{(j)}} y_{jk} - \sum_{\substack{i \in W_{(j)}^{-1} \\ i \neq i(j)}} w_{ij}.$$

Thus, the variable $w_{i(j)j}$ is not required in the model.

For each attribute quality $a \in \{1, 2, \ldots, n^A\}$, a variable t_j^a is introduced to represent the attribute quality of the intermediate pool P_j. Assuming that the attribute qualities blend linearly, we obtain that

$$t_j^a = \frac{s_{i(j)}^a \left(\sum_{k \in Y_{(j)}} y_{jk} - \sum_{i \in W_{(j)}^{-1}: i \neq i(j)} w_{ij} \right) + \sum_{i \in W_{(j)}^{-1}: i \neq i(j)} s_i^a w_{ij}}{\sum_{k \in Y_{(j)}} y_{jk}}.$$

This equation simplifies to the bilinear constraint

$$\sum_{k \in Y_{(j)}} (s_{i(j)}^a - t_j^a) y_{jk} - \sum_{\substack{i \in W_{(j)}^{-1} \\ i \neq i(j)}} (s_{i(j)}^a - s_i^a) w_{ij} = 0.$$

The attribute quality $a = 1, 2, \ldots, n^A$ of blend $k = 1, 2, \ldots, n^B$ may be calculated as the ratio

$$\frac{\sum_{i \in X_{(k)}^{-1}} s_i^a x_{ik} + \sum_{j \in Y_{(k)}^{-1}} t_j^a y_{jk}}{\sum_{i \in X_{(k)}^{-1}} x_{ik} + \sum_{j \in Y_{(k)}^{-1}} y_{jk}}.$$

The flow BLP formulation that maximizes the net profit of the pooling is shown in Figure 1.

Observe in the flow formulation that the objective function and all constraints are linear except for those constraints dealing with the attribute qualities of pools and final blends. The bilinear variables are divided into two sets: $\{t_j^a\}$ and $\{y_{jk}\}$, giving a total of $n^A n^P + |Y|$ nonlinear variables. Furthermore, there are as many as $n^A(n^P + 2n^B)$ bilinear constraints.

Note also that if the t_j^a variables are fixed at a feasible point in the solution subspace, a feasible solution of the pooling problem is obtained by solving the resulting LP on the flow variables. Given a feasible set of flow values, the t_j^a are the unique solution obtained directly from the attribute constraints for the pools.

2.2. Proportion Model of the Pooling Problem

A bilinear formulation is presented in this subsection, based on the proportion of flow entering at the intermediate pools. The parameters are the same as

Figure 1   Flow Model

$$\max_{t, w, x, y} \sum_{\substack{(i(j), j) \in W \\ i \neq i(j)}} p^F_{i(j)} \left(\sum_{i \in W^{-1}_{(j)}} w_{ij} - \sum_{k \in Y_{(j)}} y_{jk} \right) - \sum_{\substack{(i, j) \in W \\ i \neq i(j)}} p^F_i w_{ij}$$

$$+ \sum_{(i, k) \in X} \left(p^B_k - p^F_i \right) x_{ik} + \sum_{(j, k) \in Y} p^B_k y_{jk}$$

subject to:

supply $i = 1, 2, \ldots, n^F$:

$$l^F_i \leq \sum_{\substack{(i(j), j) \in W \\ i = i(j)}} \left(\sum_{k \in Y_{(j)}} y_{jk} - \sum_{\substack{i \in W^{-1}_{(j)} \\ i \neq i(j)}} w_{ij} \right) + \sum_{\substack{j \in W_{(i)} \\ i \neq i(j)}} w_{ij} + \sum_{k \in X_{(i)}} x_{ik} \leq u^F_i$$

demand $k = 1, 2, \ldots, n^B$: $l^B_k \leq \sum_{i \in X^{-1}_{(k)}} x_{ik} + \sum_{j \in Y^{-1}_{(k)}} y_{jk} \leq u^B_k$

pool capacity $j = 1, 2, \ldots, n^P$: $l^P_j \leq \sum_{k \in Y_{(j)}} y_{jk} \leq u^P_j$

attribute $a = 1, 2, \ldots, n^A$ of pool $j = 1, 2, \ldots, n^P$:

$$\sum_{k \in Y_{(j)}} \left(s^a_{i(j)} - t^a_j \right) y_{jk} - \sum_{\substack{i \in W^{-1}_{(j)} \\ i \neq i(j)}} \left(s^a_{i(j)} - s^a_i \right) w_{ij} = 0$$

requirement of attribute $a = 1, 2, \ldots, n^A$ of blend $k = 1, 2, \ldots, n^B$:

$$l^a_k \left(\sum_{i \in X^{-1}_{(k)}} x_{ik} + \sum_{j \in Y^{-1}_{(k)}} y_{jk} \right) \leq \sum_{i \in X^{-1}_{(k)}} s^a_i x_{ik} + \sum_{j \in Y^{-1}_{(k)}} t^a_j y_{jk}$$

$$\leq u^a_k \left(\sum_{i \in X^{-1}_{(k)}} x_{ik} + \sum_{j \in Y^{-1}_{(k)}} y_{jk} \right)$$

capacity of the arcs:

$$l^{FP}_{i(j)j} \leq \sum_{k \in Y_{(j)}} y_{jk} - \sum_{\substack{i \in W^{-1}_{(j)} \\ i \neq i(j)}} w_{ij} \leq u^{FP}_{i(j)j}, \quad (i(j), j) \in W$$

$$l^{FP}_{ij} \leq \qquad w_{ij} \qquad \leq u^{FP}_{ij}, \quad (i, j) \in W: i \neq i(j)$$

$$l^{FB}_{ik} \leq \qquad x_{ik} \qquad \leq u^{FB}_{ik}, \quad 4(i, k) \in X$$

$$l^{PB}_{jk} \leq \qquad y_{jk} \qquad \leq u^{PB}_{jk}, \quad (j, k) \in Y$$

nonnegativity: $w \geq 0, \ x \geq 0, \ y \geq 0$

for the flow model, including the index $i(j)$. However, instead of incorporating explicitly the flow variables from the feeds to the intermediate pools, and the attribute variables, only proportion variables are introduced as follows; we denote by

q_{ij}: the proportion of the total flow into P_j from F_i along the arc $(i, j) \in W$, for all $i \neq i(j)$.

The proportion variables allow computation of the flow on the arc $(i, j) \in W$, $i \neq i(j)$, as

$$w_{ij} = q_{ij} \sum_{k \in Y_{(j)}} y_{jk}.$$

The flow on the arc $(i(j), j) \in W$ is

$$w_{i(j)j} = \left(1 - \sum_{\substack{h \in W^{-1}_{(j)} \\ h \neq i(j)}} q_{hj} \right) \sum_{k \in Y_{(j)}} y_{jk}.$$

The total flow leaving F_i, $i = 1, 2, \ldots, n^F$, is given by

$$\sum_{k \in X_{(i)}} x_{ik} + \sum_{\substack{j \in W_{(i)} \\ i \neq i(j)}} q_{ij} \sum_{k \in Y_{(j)}} y_{jk} + \sum_{\substack{j \in W_{(i)} \\ i = i(j)}} \left(1 - \sum_{\substack{h \in W^{-1}_{(j)} \\ h \neq i(j)}} q_{hj} \right) \sum_{k \in Y_{(j)}} y_{jk}.$$

The attribute $a = 1, 2, \ldots, n^A$ of intermediate pool $j = 1, 2, \ldots, n^P$ is

$$\sum_{\substack{i \in W^{-1}_{(j)} \\ i \neq i(j)}} s^a_i q_{ij} + s^a_{i(j)} \left(1 - \sum_{\substack{i \in W^{-1}_{(j)} \\ i \neq i(j)}} q_{ij} \right) = s^a_{i(j)} + \sum_{\substack{i \in W^{-1}_{(j)} \\ i \neq i(j)}} \left(s^a_i - s^a_{i(j)} \right) q_{ij}.$$

The attribute quality $a = 1, 2, \ldots, n^A$ of blend $k = 1, 2, \ldots, n^B$ is

$$\frac{\sum_{i \in X^{-1}_{(k)}} s^a_i x_{ik} + \sum_{j \in Y^{-1}_{(k)}} \left(s^a_{i(j)} + \sum_{\substack{i \in W^{-1}_{(j)} \\ i \neq i(j)}} (s^a_i - s^a_{i(j)}) q_{ij} \right) y_{jk}}{\sum_{i \in X^{-1}_{(k)}} x_{ik} + \sum_{j \in Y^{-1}_{(k)}} y_{jk}}.$$

Note that the objective function is no longer linear. The bilinear variables are now given by the two sets, $\{q_{ij}\}$ and $\{y_{jk}\}$. Hence, the total number of nonlinear variables equals $|W| - n^P + |Y|$, which is independent of the number of attribute qualities. Including arc capacities, there are as many as $2(n^F + n^A n^B) + |W|$ bilinear constraints. Also note that given any feasible point in the solution subspace (q_{ij}), the feasible region of the (y_{jk}) is defined by a polyhedron, and vice versa. The proportion BLP formulation is given in Figure 2.

2.3.   Comparison of the Flow and Proportion Models

For comparison purposes, the following 10 examples are taken from the literature: Haverly's (1978) pooling problem (referred to as H1, H2, and H3), the fourth and fifth pooling problems (BT4, BT5) in Ben-Tal et al. (1994), Rehfeldt and Tisljar's (1997) first and second problems (RT1, RT2), the second problem (F2) in Foulds et al. (1992), and the four examples proposed in Adhya et al. (1999), referred to as AST1, 2, 3, and 4. For brevity, only RT2 is shown in detail in the appendix. The data describing all examples considered in the paper can be found at www.gerad.ca/Charles.Audet.

A complete description of the number of variables and constraints for these 10 examples appears in Table 1. The table is divided into five groups of columns. The left column (Example) displays the name of the problem. The second column (Lin var) shows the number of variables that are not involved in bilinear terms (the linear variables). The next group details the number of bilinear variables. These variables are partitioned in two subsets (each bilinear

Figure 2   Proportion Model

$$\max_{x,y,q} \sum_{(i(j),j)\in W} p^F_{i(j)}\left(\sum_{\substack{h\in W^{-1}_{(j)}\\h\neq i(j)}} q_{hj} - 1\right)\sum_{k\in Y_{(j)}} y_{jk} - \sum_{\substack{(i,j)\in W\\i\neq i(j)}} p^F_i q_{ij}\sum_{k\in Y_{(j)}} y_{jk}$$

$$+ \sum_{(i,k)\in X}(p^B_k - p^F_i)x_{ik} + \sum_{(j,k)\in Y} p^B_k y_{jk}$$

subject to:

supply $i = 1, 2, \ldots, n^F$:

$$l^F_i \le \sum_{\substack{(i(j),j)\in W\\i=i(j)}}\left(1 - \sum_{\substack{h\in W^{-1}_{(j)}\\h\neq i(j)}} q_{hj}\right)\sum_{k\in Y_{(j)}} y_{jk} + \sum_{\substack{j\in W_{(i)}\\i\neq i(j)}} q_{ij}\sum_{k\in Y_{(j)}} y_{jk} + \sum_{k\in X_{(i)}} x_{ik} \le u^F_i$$

demand $k = 1, 2, \ldots, n^B$: $\quad l^B_k \le \sum_{i\in X^{-1}_{(k)}} x_{ik} + \sum_{j\in Y^{-1}_{(k)}} y_{jk} \le u^B_k$

pool capacity $j = 1, 2, \ldots, n^P$: $\quad l^P_j \le \sum_{k\in Y_{(j)}} y_{jk} \le u^P_j$

requirement of attribute $a = 1, 2, \ldots, n^A$ of blend $k = 1, 2, \ldots, n^B$:

$$l^a_k\left(\sum_{i\in X^{-1}_{(k)}} x_{ik} + \sum_{j\in Y^{-1}_{(k)}} y_{jk}\right) \le \sum_{i\in X^{-1}_{(k)}} s^a_i x_{ik} + \sum_{j\in Y^{-1}_{(k)}}\left(s^a_{i(j)} + \sum_{\substack{i\in W^{-1}_{(j)}\\i\neq i(j)}}(s^a_i - s^a_{i(j)})q_{ij}\right)y_{jk}$$

$$\le u^a_k\left(\sum_{i\in X^{-1}_{(k)}} x_{ik} + \sum_{j\in Y^{-1}_{(k)}} y_{jk}\right)$$

capacity of the arcs:

$$l^{FP}_{i(j)j} \le \left(1 - \sum_{\substack{h\in W^{-1}_{(j)}\\h\neq i(j)}} q_{hj}\right)\sum_{k\in Y_{(j)}} y_{jk} \le u^{FP}_{i(j)j}, \quad (i(j),j)\in W$$

$$l^{FP}_{ij} \le \qquad q_{ij}\sum_{k\in Y_{(j)}} y_{jk} \qquad \le u^{FP}_{ij}, \quad (i,j)\in W: i\neq i(j)$$

$$l^{FB}_{ik} \le \qquad x_{ik} \qquad \le u^{FB}_{ik}, \quad (i,k)\in X$$

$$l^{PB}_{jk} \le \qquad y_{jk} \qquad \le u^{PB}_{jk}, \quad (j,k)\in Y$$

nonnegativity and proportion $j = 1, 2, \ldots, n^P$:

$$x \ge 0, \ y \ge 0, \ q \ge 0, \quad \sum_{\substack{h\in W^{-1}_{(j)}\\h\neq i(j)}} q_{hj} \le 1$$

term consists of the product of one variable from each subset), and the number of variables in each subset of the partition is set in parentheses (*Pool+Flow*). *Flow* is the number of flow variables, and *Pool* is the number of attribute or proportion variables (depending on the model used). These numbers are decomposed in the fourth group to explain the total number of bilinear terms (Trm) introduced by the intermediate pools. For example, the flow model of RT2 has four attributes and three exiting feeds at each pool, P_1 and P_2. The number of bilinear terms is the number of cross-product elements of $\{t^1_1, t^2_1, t^3_1, t^4_1\} \times \{y_{11}, y_{12}, y_{13}\}$

and $\{t^1_2, t^2_2, t^3_2, t^4_2\} \times \{y_{21}, y_{22}, y_{23}\}$; therefore, there are a total of $4 \times 3 + 4 \times 3 = 24$ bilinear terms. The last group of columns gives the number of linear inequalities $L_\le$, quadratic inequalities $Q_\le$, and quadratic equalities $Q_=$ in the constraint set of each problem.

The difficulty of the BLP problem can be roughly estimated by the number of bilinear variables, terms, and constraints. The advantage of the flow model occurs when there are few attributes. The number of complicating variables (the t^a_j) will then be small. When the number of attributes increases, the advantage of the proportion model becomes apparent because the number of bilinear variables and terms stays the same. These numbers are determined by the number of entering and exiting flows at the intermediate pools. (For total number of bilinear variables, the turnover between the flow and proportional models occurs when $n^A = |W|/n^P - 1$; that is, the average number of entering arcs at the intermediate pools less one.) Referring to Table 1, we see that the proportion models of Examples RT1 and RT2 are considerably smaller than the corresponding flow models. At first glance, the flow model appears to be simpler than the proportion in BT5. We will see in the next section that this is not the case when these problems are solved exactly. The number of bilinear constraints is also an important factor.

2.4. Generalized Pooling Problem
The complexity of the model increases when several pools are linked together in parallel or in series. We present in Figure 3 (and Table 2) an example in which the intermediate pools are allowed to be interconnected. The pooling problem is thus extended to the case where exiting blends of some intermediate pools are entering feeds of others.

The proportion model of the generalized pooling problem is not a bilinear program, because the variables are not partitioned into two sets. Some bilinear terms will be of the form $q_{ij}q_{lm}$. Therefore, this formulation belongs to the class of quadratically constrained quadratic programs.

A hybrid formulation may be used for the generalized problem. The flow model is applied to intermediate pools that receive flow from at least one other intermediate pool, and the proportion model is used otherwise. In both cases, the exiting flow of an intermediate pool is modeled through flow variables.

A hybrid model for GP1 is given below, where v_{12} denotes the flow from P_1 to P_2 (all other variables are defined as before).

$$\max_{q,t,v,w,x,y} \quad -6(x_{11} + y_{12} + v_{12} - q_{21}(y_{12} + v_{12}))$$

$$-16q_{21}(y_{12} + v_{12}) - 10(x_{32} + y_{21} + y_{23} - v_{12})$$

$$+9(x_{11} + y_{21}) + 13(y_{12} + x_{32}) + 14y_{23}$$

Table 1   Summary of the Number of Variables and Constraints

Example	Lin Var	Bilinear Var	(Pool + flow)	Trm	Bilinear $\sum$(Pool × flow)	$L_\leq$	$Q_\leq$	$Q_=$
Flow model								
AST1	3	16	(8 + 8)	32	(4 × 4) + (4 × 4)	10	32	8
AST2	3	20	(12 + 8)	48	(6 × 4) + (6 × 4)	10	48	12
AST3	5	30	(18 + 12)	72	(6 × 4) + (6 × 4) + (6 × 4)	13	48	18
AST4	6	18	(8 + 10)	40	(4 × 5) + (4 × 5)	11	40	8
BT4	4	3	(1 + 2)	2	(1 × 2)	6	4	1
BT5	14	21	(6 + 15)	30	(2 × 5) + (2 × 5) + (2 × 5)	18	20	6
F2	10	10	(2 + 8)	8	(1 × 4) + (1 × 4)	12	8	2
H1	3	3	(1 + 2)	2	(1 × 2)	6	4	1
RT1	7	12	(7 + 5)	17	(4 × 2) + (3 × 3)	16	20	7
RT2	8	14	(8 + 6)	24	(4 × 3) + (4 × 3)	14	21	8
Proportion model								
AST1	0	11	(3 + 8)	12	(1 × 4) + (2 × 4)	7	37	0
AST2	0	11	(3 + 8)	12	(1 × 4) + (2 × 4)	7	53	0
AST3	0	17	(5 + 12)	20	(1 × 4) + (2 × 4) + (2 × 4)	9	56	0
AST4	0	14	(6 + 8)	24	(3 × 4) + (3 × 4)	9	48	0
BT4	2	4	(2 + 2)	4	(2 × 2)	5	7	0
BT5	5	24	(9 + 15)	45	(3 × 5) + (3 × 5) + (3 × 5)	14	24	0
F2	8	10	(2 + 8)	8	(1 × 4) + (1 × 4)	8	12	0
H1	2	3	(1 + 2)	2	(1 × 2)	4	6	0
RT1	5	7	(2 + 5)	5	(1 × 2) + (1 × 3)	9	23	0
RT2	4	10	(4 + 6)	12	(2 × 3) + (2 × 3)	11	24	0

s.t. supply: $x_{11} + y_{12} + v_{12} - q_{21}(y_{12} + v_{12}) \leq 18$

$q_{21}(y_{12} + v_{12}) \leq 18$

$x_{32} + y_{21} + y_{23} - v_{12} \leq 18$

demand: $x_{11} + y_{21} \leq 10$

$y_{12} + x_{32} \leq 15$

$y_{23} \leq 20$

capacity: $y_{12} + v_{12} \leq 20$

$y_{21} + y_{23} \leq 20$

attribute: $(3 - 2q_{21})v_{12} + 2(y_{21} + y_{23} - v_{12})$

$\quad - t_2^1(y_{21} + y_{23}) = 0$

$3x_{11} + t_2^1 y_{21} \leq 2.5(x_{11} + y_{21})$

$2x_{32} + (3 - 2q_{21})y_{12} \leq 1.75(x_{32} + y_{12})$

$t_2^1 \leq 1.5$

pos. flow: $y_{21} + y_{23} - v_{12} \geq 0$

$q \geq 0, t \geq 0, v \geq \mathbf{0}, w \geq \mathbf{0},$

$x \geq \mathbf{0}, y \geq \mathbf{0}, q_{21} \leq 1.$

Figure 3   Generalized Pooling Problem (GP1)

2.5. Simplification of Some Pooling Problem Instances

It is sometimes useful to analyze the basic structure of a given pooling problem to attempt to simplify it. Consider, for example, the instance detailed in RT1 and reproduced in the left part of Figure 4.

A closer look at the definition of the problem allows considerable simplification. Observe that the flows x_{21}, x_{31}, and y_{21} entering blend B_1 originate solely from the two feeds F_2 and F_3. A positive flow y_{21} of any feasible solution could be transferred to the unconstrained flows x_{21} and x_{31} without affecting feasibility or objective function value. Therefore, we can assume without any loss of generality that the flow variable y_{21} is fixed to zero.
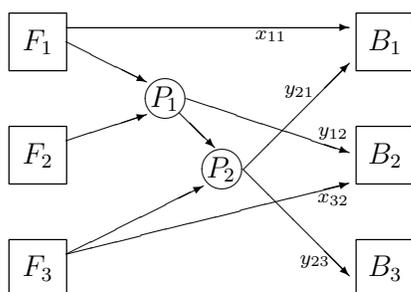
Table 2   Characteristics of GP1

Feed	Price $/bbl	Attribute quality	Max supply	Pool	Max capacity	Blend	Price $/bbl	Max demand	Attribute max
F_1	6	3	18	P_1	20	B_1	9	10	2.5
F_2	16	1	18	P_2	20	B_2	13	15	1.75
F_3	10	2	18			B_3	14	20	1.5

Similarly, by considering the flows y_{13}, x_{13}, and x_{33}, we can deduce that y_{13} can be fixed to zero. This observation has important consequences: The pool P_1 has therefore a unique exiting flow y_{12}, and thus that pool can be combined with the final pool B_2. The flow from F_3 to B_2 is bounded above by the capacity of pool P_1. This constraint is, however, redundant, since the maximum demand of B_2 is less than that capacity. These simplifications are illustrated on the right part of Figure 4, and they lead to new ones. Indeed, consider the flow from the feed F_3 to the intermediate pool P_2: It can be transferred to x_{32} and x_{33} without altering feasibility or objective function value, thus allowing it to be fixed to zero. It follows that the intermediate pool P_2 can be combined with F_2, as the capacity of pool P_2 is greater than the availability of feed F_2. Therefore, this example can be reduced to an equivalent blending problem because all intermediate pools may be eliminated. Thus, Rehfeldt and Tisljar's (1997) first pooling problem can be solved by linear programming.

The instances F3, F4, and F5 of Foulds et al. (1992) can also be simplified. In fact, the linear structure describing these instances allows an analytical solution. We will show how to solve F3; F4 and F5 can be treated similarly. In that instance, the single-attribute value and cost of the 11 feeds are $s_i^1 = (9+i)/10$ and $p_i^F = 21 - i$ for $i = 1, 2, \ldots, 11$. The capacities of the arcs and pools are unlimited. Each of the 16 blends has a maximal demand of 1. Their price and maximal attribute values are $p_k^B = (41 - k)/2$ and $u_k^1 = 1 + 0.05k$ for $k = 1, 2, \ldots, 16$.

A consequence of this linear structure is that the cost of producing a unit amount of a blend with fixed attribute $\alpha \in [1, 2]$ comprises a constant purchase cost,

$30 - 10\alpha$, independent of which feeds are blended together. Therefore, a strategy to achieve the optimal solution is to simply direct 9.2 units of feed F_1 into pool P_1 and 6.8 units of F_{11} into P_8, then blend $2 - u_k^1$ units of P_1 together with $u_k^1 - 1$ units of P_8 into blend B_k, for $k = 1, 2, \ldots, 16$. This optimal solution is displayed in Figure 5.

3. Computational Results for Exact Solution

Since the objective function and feasible region are nonconvex, the pooling problem requires a global optimization approach. Methods using local searches along descent directions, such as SLP, are only guaranteed to find a local optimum, the quality of which is unknown.

The pooling examples described in the preceding section are solved to global optimality using a recent branch-and-cut algorithm by Audet et al. (2000) for the general class of nonconvex quadratically constrained quadratic programs. Their method is inspired by Al-Khayyal and Falk's (1983) branch-and-bound algorithm for bilinear programming and the reformulation-linearization techniques of Sherali and Tuncbilek (1992; 1997a, b). The improvements to these methods are as follows: (i) selection of the branching value—splitting of the hyper-rectangle is done in a way that minimizes the resulting potential error, and thus not necessarily in its middle; (ii) approximation of bilinear terms—instead of systematically adding all linear inequalities defining the convex and concave envelopes, only those violated are added to the model, thus keeping the linear program size from growing too fast; (iii) introduction of a new class of

Figure 4   Simplification of Rehfeldt and Tisljar's (1997) First Pooling Problem (RT1)
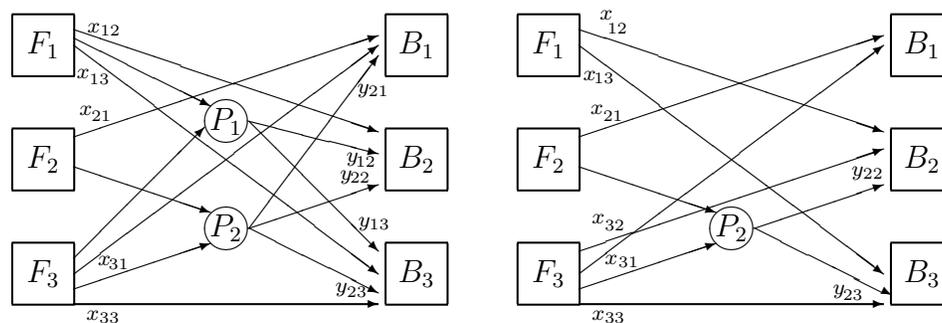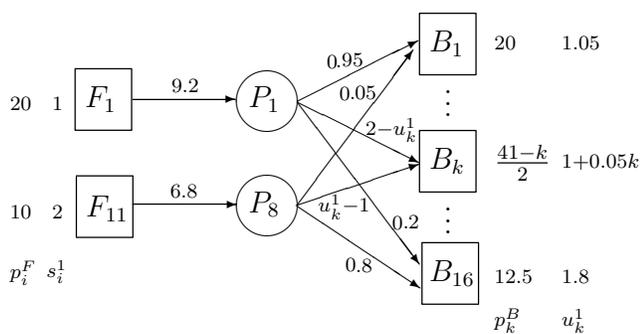
Figure 5   Simplification of F3, F4, and F5

cuts—cuts derived from underapproximation of the convex paraboloid are used to force linear variables to approach the corresponding bilinear term; (iv) the proposed algorithm is of the branch-and-cut type—cuts introduced at any node of the exploration tree are valid at all other nodes.

Computational experiments were completed on a Sun Ultra-60 (UltraSPARC II 360 MHz processor, and LINPACK score of 10 Mflops/s) with a C++ implementation of the algorithm. The stopping criterion is an absolute tolerance of $\epsilon_r = \epsilon_z = 10^{-6}$. The tested instances have been reviewed in §2, and their characteristics are summarized in Table 3. We also include two variants of H1 given by Haverly (1978), which are denoted by H2 and H3, respectively, and the generalized pooling problem GP1.

The following tables display the computing time in seconds of the preprocessing phase and the exploration of the search tree phase, as well as the total time. The total number of nodes in the search tree, as well as the number of additional variables (Var) and constraints (Cstr) generated by the algorithm are also presented.

For all instances, the algorithm was executed twice. A first execution was done to solve the instance

without any additional information. A second execution was done to evaluate the performance of the algorithm when jointly used with a good heuristic method. The optimal solution (from Execution 1) was fed to the algorithm, and the second execution was used to prove the optimality of the solution.

Table 4(i) displays the computational results on the flow model of the pooling problems. The preprocessing time is listed under column P.P., and the *tree exploration* phase under Tree. The "—" for the AST2 and AST3 instances indicate that the memory limit imposed by the current implementation of the algorithm was reached.

Table 4(ii) shows the performance of the optimality proof on the flow model of the pooling problems. An additional constraint that bounds the objective value is added to shrink the feasible region. At the early nodes of the enumeration algorithm, branching is done on the incumbent variables involved in quadratic terms that are not at either their lower or upper bound. This allows a more precise linearization near the incumbent solution. Computational times remain comparable for the small instances but drop significantly for the larger problems, except for AST4, where proving optimality is more expensive than searching from scratch. This seems to justify the joint use of a heuristic to obtain a good incumbent solution.

Results of the algorithm on the proportion model of the pooling problem are displayed in Table 4(iii). For the larger instances, the proportion model is significantly easier to solve than the flow model, even for BT5, where this model has a greater number of bilinear variables and terms than the former one. Note, however, that the proportion model of BT5 has fewer bilinear constraints than the flow model (11 versus 16). The significant difference in computation times in Tables 4(i) and 4(iii) suggests that care should be taken initially to choose the right model.

Optimality proof performance on the proportion model of the pooling problems appears in Table 4(iv). Again, the proportion model seems easier to solve than the flow model. Moreover, the time required for the optimality proof is less or comparable to the time of solving the original problem, except for the problem AST and for Example RT2, where the computational time increased. As for the remaining examples, the preprocessing time increased, but the tree exploration phase decreased. This is explained by the addition of the nonlinear constraint to bound the feasible region. The feasible region becomes small and hard to approximate by outer approximations.

Tables 4(v) and 4(vi) display computing times for solving the three equivalent formulations of the generalized pooling problem presented in Example GP1.

Table 3   Instances Characteristics

Example	Ref	n^F	n^P	n^B	n^A	Solution
AST1	Adhya et al. (1999)	5	2	4	4	549.803
AST2	Adhya et al. (1999)	5	2	4	6	549.803
AST3	Adhya et al. (1999)	8	3	4	6	561.048
AST4	Adhya et al. (1999)	8	2	5	4	877.649
BT4	Ben-Tal et al. (1994)	4	1	2	1	45
BT5	Ben-Tal et al. (1994)	5	3	5	2	350
F2	Foulds et al. (1992)	6	2	4	1	110
H1	Haverly (1978)	3	1	2	1	40
H2	Haverly (1978)	3	1	2	1	60
H3	Haverly (1978)	3	1	2	1	75
RT1	Rehfeldt and Tisljar (1997)	3	2	3	4	4,136.22
RT2	Rehfeldt and Tisljar (1997)	3	2	3	4	4,391.83
GP1		3	2	3	1	60.5

Table 4   Computational Results

(i) Flow Model, Complete Solution

Example	Time (sec)			Nodes	Additional	
	P.P.	Tree	Total		Var	Cstr
AST1	7.39	7,778.65	7,786.04	4,145	108	6,991
AST2	—	—	—	—	—	—
AST3	—	—	—	—	—	—
AST4	13.87	940.03	953.9	723	50	3,414
BT4	0.17	0.05	0.22	9	4	41
BT5	5	660.63	665.63	97	30	5,776
F2	0.51	0.16	0.67	15	10	128
H1	0.2	0.06	0.26	9	4	41
H2	0.13	0.03	0.16	3	2	20
H3	0.14	0.03	0.17	3	1	15
RT1	2.12	29.69	31.81	179	38	1,546
RT2	3.97	201.02	204.99	489	63	2,581

(ii) Flow Model, Optimality Proof

Time (sec)			Nodes	Additional	
P.P.	Tree	Total		Var	Cstr
10.23	4,258.53	4,268.76	3,177	80	5,801
—	—	—	—	—	—
—	—	—	—	—	—
55.43	4,587.01	4,642.44	2,355	43	3,656
0.19	0.03	0.22	1	0	7
1.28	0.16	1.44	1	0	90
3.66	6.34	10	155	33	687
0.21	0.02	0.23	1	0	7
0.4	0	0.4	0	0	0
0.17	0.02	0.19	1	0	6
13.69	0.48	14.17	43	10	225
8.37	366.28	374.65	1,381	62	2,626

(iii) Proportion Model, Complete Solution

Example	Time (sec)			Nodes	Additional	
	P.P.	Tree	Total		Var	Cstr
AST1	0.47	8.59	9.06	245	40	838
AST2	0.39	9.28	9.67	267	35	777
AST3	2.45	66.06	68.5	537	38	1,065
AST4	2.83	175.15	177.98	693	48	205
BT4	0.15	0.88	1.03	43	14	197
BT5	1.81	29.29	31.1	39	20	1,274
F2	0.35	0.05	0.4	1	0	24
H1	0.17	0.05	0.22	9	6	57
H2	0.13	0.04	0.17	13	6	41
H3	0.13	0.04	0.17	7	4	35
RT1	0.55	0.05	0.6	7	4	37
RT2	0.85	1.11	1.96	59	19	347

(iv) Proportion Model, Optimality Proof

Time (sec)			Nodes	Additional	
P.P.	Tree	Total		Var	Cstr
1.26	14.47	15.73	391	48	874
1.91	2.55	4.46	139	18	408
2.19	173.67	175.86	911	44	1,416
5.91	681.14	687.05	2,127	47	2,324
0.75	0.14	0.89	27	8	112
0.52	0	0.52	0	0	0
0.32	0.02	0.34	1	0	24
0.18	0.02	0.2	1	0	7
0.14	0.01	0.15	0	0	0
0.23	0.03	0.26	11	5	37
0.4	0.05	0.45	1	0	15
6.89	0.31	7.2	47	11	175

(v) Generalized Pooling Problem (GP1), Complete Solution

Model	Time (sec)			Nodes	Additional	
	P.P.	Tree	Total		Var	Cstr
Flow	0.19	0.59	0.78	47	18	238
Prop.	0.22	2.5	2.72	35	18	457
Hybrid	0.68	0.53	1.21	27	12	225

(vi) GP1, Optimality Proof

Time (sec)			Nodes	Additional	
P.P.	Tree	Total		Var	Cstr
0.62	0.07	0.69	13	6	78
1.23	0.17	1.4	17	8	132
0.62	0	0.62	0	0	0

Contrary to the standard pooling problem, the proportion model of the generalized problem appears to be harder to solve than the flow model. Including the variable q_{22} that represents the product of two proportion variables adds a level of complexity to the outer approximation scheme. It seems that the hybrid model is the easiest one to solve. The proportion variables allow elimination of the attribute variable of pool P_1 without adding complexity.

Table 5 shows a computational results comparison, similar to that of Adhya et al. (1999). For the easier problems—that is, BT4, BT5, F2, H1, H2, and H3—the state-of-the-art algorithm of Adhya et al. (1999) takes less computer time than the algorithm of Audet et al. (2000); however, for the three difficult instances AST1, AST2, and AST3, the CPU times of this last algorithm are 46 to 282 times (or 17 to 104 times, taking into account the relative CPU speeds) less than those of Adhya et al. (1999). So use of the proportion model and the Audet et al. (2000) algorithm appears to be competitive, for many difficult instances, with the state of the art.

4. Heuristic Methods

As implied in the preceding section, heuristic approaches to the pooling problem are required to obtain (i) good solutions for larger problem instances

Table 5   Comparative CPU Times for Exact Solution

Example	Foulds et al. (1992) CDC 4340 25 MHz	Floudas and Visweswaran (1993b) HP9000/730 67 MHz	Floudas and Visweswaran (1996) HP9000/730 67 MHz	Adhya et al. (1999) RS6000/43P 100–133 MHz	Audet et al. (2000) Ultra-60 360 MHz
AST1	—	—	—	425	9.06
AST2	—	—	—	1,115	9.67
AST3	—	—	—	19,314	68.5
AST4	—	—	—	182	177.98
BT4	—	44.54	0.95	0.11	0.22
BT5	—	40.31	5.8	1.12	31.1
F2	3	—	—	0.1	0.4
H1	0.7	0.95	0.22	0.09	0.22
H2	—	3.19	0.21	0.09	0.16
H3	—	—	0.26	0.13	0.17
RT1	—	—	—	—	0.6
RT2	—	—	—	—	1.96
GP1	—	—	—	—	0.78

and (ii) good initial solutions for use in exact algorithms. In this section, we propose a local search procedure that alternately fixes the noncomplicating and complicating variables (y and z, respectively) and solves the linear program in the resulting subspace. The polyhedron structure of both feasible sets leads to a natural neighborhood structure, defined by the number of pivots from a current extreme point. This will allow us to develop a new VNS procedure. The alternating and VNS methods are applied to the problem cases of §2 and to a large set of randomly generated pooling problems. The results are compared with the commonly used SLP method.

4.1. Alternate Heuristic

The principle of the alternate heuristic (ALT), given two subsets of variables, consists of alternately solving the problem with the variables of one of the subsets fixed. These two problems, by the choice of the subsets, must be linear. When one of these linear problems is solved, its solution becomes a set of parameters in the other one. For the general BLP given in the introduction, we proceed with the steps described in Figure 6.

In the proportion model, z denotes the set of q_{ij} variables, y the set of y_{jk}s, and x the x_{ik}s. For the

Figure 6   Alternate Heuristic (ALT)

[1]   Give some feasible initial values for the complicating variables z (or noncomplicating variables y).

[2]   Solve the resulting LP in (x, y) (or in (x, z)).

[3]   For y (or z), found in the previous step, solve the LP in (x, z) (or in (x, y)).

[4]   Repeat Steps 2 and 3 until stability is reached (with a given tolerance).

flow model, z becomes the attribute variables t_j^a, y remains the same, and x includes the x_{ik} and w_{ij} variables. Observe that in the flow formulation, if both sets of flow variables x and y found in the previous iteration are fixed (not only y, as in ALT), the t_j^a may be updated directly from the equality constraints for the attribute values at the pools. Repetitively solving for (x, y) and updating the t_j^a leads to the popular *recursive* technique first given by Haverly (1978). This recursive procedure (in its original form) does not extend to the proportion model. The ALT is a natural solution approach to the bilinear program. It has been suggested before in other contexts than the pooling problem (e.g., see Brimberg et al. 1997). However, to our best knowledge, this is the first time that computational experience is reported.

PROPOSITION 4.1. *Assuming a unique solution in each iteration of Steps 2 and 3, ALT converges to a local optimum (i.e., varying (x, y) or (x, z) does not improve the objective function value).*

PROOF. Fixing feasible values in z (or y) leads to an LP with a nonempty feasible region in (x, y) (or (x, z)). Thus, the sequence of LPs will maintain feasibility and provide a monotonic sequence of improving solutions. Since a well-formulated pooling problem must have a bounded feasible region, we conclude that the sequence of solutions will converge to a unique finite value of the objective function. Since there are no ties in the LP iterations, this must correspond to a unique attraction point. Furthermore, local ascent is not possible, since the (x, y) and (x, z) subspaces only permit local descent. □

4.2. Variable Neighborhood Search

We assume a knowledge of the basic rules of VNS. For a review of VNS and its applications to a range of

problems, see Hansen and Mladenović (1997, 1999). In a nutshell, the VNS consists of repeating the following two steps: (i) Perturb the current solution within a neighborhood of length k (initially set to 1); (ii) from this perturbed point, find a new point with a local search. If this new local optimum is better, it becomes the new current point, and the k parameter is set again to 1, or else the original current point is kept and the k parameter is increased, for a bigger perturbation in the first step.

Let $s = (x', y', z')$ be a feasible solution obtained by ALT. It follows that (x', y') and (x', z') are extreme points of a polyhedron in the respective subspaces. Let us denote these two polyhedrons by $\mathscr{P}_1(s)$ and $\mathscr{P}_2(s)$. The first neighborhood $\mathscr{N}_1(s)$ is represented by the union of all feasible extreme points adjacent to either (x', y') (in $\mathscr{P}_1(s)$) or (x', z') (in $\mathscr{P}_2(s)$). Thus, the cardinality of $\mathscr{N}_1(s)$ is less than or equal to $2n_1 + n_2 + n_3$, because the number of adjacent points in a polyhedron cannot be larger than the dimension of the space. The equality occurs when the LP problems are not degenerate (in both (x, y) and (x, z) subspaces). $\mathscr{N}_2(s)$ would then be the set of adjacent extreme points on $\mathscr{P}_1(s)$ or $\mathscr{P}_2(s)$ obtained by changing exactly two elements in the respective bases, $\mathscr{N}_3(s)$ exactly three elements, and so on. It is easy to see that the cardinality of $\mathscr{N}_k(s)$ increases exponentially with k.

For solving BLP by VNS, we define the *shaking* operator with respect to the neighborhoods $\mathscr{N}_k$. That is, a point s' from $\mathscr{N}_k(s)$ ($k = 1, \ldots, k_{\max}$), is taken at random. A local search is then carried out from s' using ALT. In the description of the VNS heuristic that follows, we use a Boolean variable δ that has values 1 or 0 if the search is performed in $\mathscr{P}_1$ or $\mathscr{P}_2$, respectively. If the shaking is done in $\mathscr{P}_1$, the search by ALT will start in $\mathscr{P}_2$. The next shaking operation will also be carried out in $\mathscr{P}_2$. That is, we alternate between subspaces by setting δ to its complement $\bar{\delta}$. The detailed VNS algorithm is shown on Figure 7. Note that the second step of the algorithm can be repeated several times, which is decided by the parameter nt (see §4.3.1).

Implementation of the VNS rules to solve BLP is not hard. However, there are theoretical questions that need to be clarified. The current reduction (discretization) of the continuous solution space to the extreme points of $\mathscr{P}_1$ and $\mathscr{P}_2$ does not mean that the global optimum necessarily belongs to the set $\bigcup_k \mathscr{N}_k(s)$, as is the case in combinatorial optimization. This last set depends on the current solution $s = (x', y', z')$; i.e., for each $s = (x', y', z')$ we can construct a different exponential discretization of the continuous solution space R^n. In fact, in solving the continuous BLP problem, we first introduce the finite set $\mathscr{S}_s$ that consists of all feasible extreme points of $\mathscr{P}_1(s)$ and $\mathscr{P}_2(s)$. A distance function ρ_s is then specified for

Figure 7   VNS Algorithm

[1] **Initialization**
    find an initial feasible solution s
    choose a stopping condition nt
    $it \leftarrow 1$
[2] **While** $it \leq nt$
    $k \leftarrow 1$
    While $k \leq k_{\max}$
      [i] **Shaking**
        get s' from $\mathscr{N}_k(s)$ at random using current value of δ
      [ii] **Local search**
        $\delta \leftarrow \bar{\delta}$
        use ALT with s' as initial solution to obtain local optimum s''
      [iii] **Move or not**
        if s'' *better than* s
          move to s'' ($s \leftarrow s''$)
          $k \leftarrow 1$
          update the neighborhoods $\mathscr{N}_k(s)$ for the new current solution (i.e., update the simplex table, or polyhedron, where s'' is found)
        else $k \leftarrow k + 1$
    $it \leftarrow it + 1$

the set $\mathscr{S}_s$ (subscript s indicates that the discretization depends on the current solution). Let s_1 and s_2 be any two solutions that belong to $\mathscr{S}_s$, and let $\mathscr{B}_1$ and $\mathscr{B}_2$ be the corresponding sets of basic variables. We say that

$$\rho_s(s_1, s_2) = k \iff |\mathscr{B}_1 \backslash \mathscr{B}_2| = |\mathscr{B}_2 \backslash \mathscr{B}_1| = k,$$

that is, the distance metric is defined by the symmetric difference of two sets.

4.3.   Computer Results

Three heuristic methods were tested: an efficient version of SLP (noted as SLPR in Palacios-Gomez et al. 1982), and the alternate and VNS procedures given above. All three heuristics were coded in C++ and run on the Ultra-60 station as before. Recall that finding a feasible initial solution (a requirement for all three heuristics) is in itself a very difficult problem. For example, in RT2, 10,000 sets of proportion values were generated at random, and not one feasible solution was found by ALT. The following "tricks" were used to significantly improve the rate of success:

(i) Because the solutions of the LPs typically have many zero-valued variables, set every variable in z (or y) to zero with probability 0.5; if the variable is decided to be nonzero, set its value randomly,

(ii) For some problem instances, we observe that it is better to start with random values of z, while for others a feasible solution is easier to find if the variables from y are fixed at random. Therefore, we choose to start with y or z with probability 0.5.

Table 6   Pooling Test Problems from the Literature

Example	Parameters			Solution				CPU time (sec)			Error (%)		
	nsp	k_{max}	nt	Exact	MSLP	MALT	VNS	MSLP	MALT	VNS	MSLP	MALT	VNS
Flow model													
AST1	1,000	10	1	549.803	276.661	532.901	545.27	2.20	2.45	2.81	49.68	3.07	0.82
AST2	1,500	10	1	549.803	284.186	535.617	543.909	9.18	5.21	5.68	48.31	2.58	1.07
AST3	1,000	10	1	561.048	255.846	397.441	412.145	18.71	4.96	5.34	54.35	29.09	26.47
AST4	230	0	0	877.649	—	876.206	876.206	0.82	0.77	1.01	—	0.16	0.16
BT4	5	0	0	45	39.6970	45	45	0.01	0.01	0.01	11.78	0	0
BT5	10	15	2	350	327.016	324.077	350	0.03	0.09	1.11	6.57	7.41	0
F2	120	10	1	110	100	107.869	110	0.07	0.44	0.57	9.09	1.94	0
H1	5	0	0	40	40	40	40	0.02	0.01	0.01	0	0	0
H2	5	0	0	60	60	60	60	0.02	0.01	0.01	0	0	0
H3	5	3	1	75	60.7332	70	75	0.02	0.01	0.03	19.02	6.67	0
RT1	5	0	0	4,136.22	126.913	4,136.22	4,136.22	1.34	0.04	0.04	96.93	0	0
RT2	5	5	1	4,391.83	—	4,330.78	4,391.83	0.04	0.47	0.60	—	1.39	0
GP1	50	5	1	60.5	28.732	35	46	0.01	0.04	0.08	52.51	42.15	23.97
Proportion model													
AST1	1,000	10	1	549.803	544.307	532.901	533.783	1.14	2.38	2.61	1	3.07	2.91
AST2	1,500	10	1	549.803	548.407	535.617	542.54	3.04	4.97	5.37	0.25	2.58	1.32
AST3	1,000	10	1	561.048	551.081	397.441	558.835	4.98	4.98	5.93	1.68	29.09	0.3
AST4	230	0	0	877.649	—	876.206	876.206	1.19	1.21	1.55	—	0.16	0.16
BT4	5	0	0	45	39.7019	45	45	0.01	0.02	0.02	11.77	0	0
BT5	10	15	2	350	292.532	323.12	350	0.12	0.16	1.53	16.42	7.68	0
F2	120	0	0	110	110	110	110	0.15	0.49	0.49	0	0	0
H1	5	0	0	40	40	40	40	0.02	0.01	0.01	0	0	0
H2	5	0	0	60	60	60	60	0.02	0.01	0.01	0	0	0
H3	5	3	1	75	69.9934	70	75	0.02	0.01	0.02	6.68	6.67	0
RT1	5	0	0	4,136.22	3,061.03	4,136.22	4,136.22	0.07	0.03	0.03	25.99	0	0
RT2	5	5	1	4,391.83	4,391.02	4,330.77	4,391.82	0.04	0.58	0.72	0.02	1.39	0

The above procedure allowed us to obtain about 25% feasible solutions in RT2. The choice of the set of variables to be fixed has in general no important incidence on the efficiency of this method, except for the AST problems under the flow formulation: No feasible point was generated when the set of smallest cardinality was chosen.

To improve the quality of the solutions, multistart versions of SLP and ALT (referred to as MSLP and MALT, respectively) were used. The rule above was taken for the starting values of MALT. The starting procedure suggested in Palacios-Gomez et al. (1982) was applied in MSLP. If the starting solution was found to be infeasible, iterations of ALT were allowed to continue until a feasible point was found. The best solution from MALT was taken as the initial solution for VNS. In Tables 6 and 7, a "—" means that the MSLP method could not find a solution or that the memory limit was reached for the exact algorithm.

4.3.1.   Pooling Problems from the Literature. We first tested the heuristics on the problem instances discussed in §§2 and 3, for which exact solutions were already found. The results are summarized in Table 6

for flow and proportion models. In this table, we give the parameters used for the three methods MSLP, MALT, and VNS. They are—

nsp: number of starting points for the MSLP and MALT algorithms

k_{max}: VNS parameter, maximum length of the neighborhood

nt: VNS parameter, number of repetitions of VNS's Phase 2 (see the algorithm in Figure 7).

Because the proportion formulation of GP1 is not a bilinear problem, only the resolution with flow formulation is made.

Given that the initial solution of VNS is the best solution of MALT, the CPU times given for the VNS include the time of MALT. If no improvement can be made by the VNS, the k_{max} and nt parameters are set to zero, and the times for MALT and VNS are the same.

Table 6 shows that there is not a significant difference between the flow and proportion formulations (except for AST3 and F2). We note that MALT gives the optimal solution for five instances, MSLP in only three instances, and VNS obtained the exact solution

Table 7 Randomly Generated Pooling Test Problems

Example	Parameters				Dimensions						Solution				CPU time (sec)				Error (%)		
					Bilinear			Constraints													
	n^F	n^P	n^B	n^A	Lin var	Var	Trm	$L_\leq$	$Q_\leq$	$Q_=$	Exact	MSLP	MALT	VNS	Exact	MSLP	MALT	VNS	MSLP	MALT	VNS
R1	6	3	5	3	10	12	18	11	18	6	888	888	888	888	1.23	0.39	0.51	0.51	0	0	0
R2	6	3	5	3	13	12	9	10	18	9	572	394	572	572	1.33	0.55	0.6	0.6	31.12	0	0
R3	6	3	5	3	14	12	18	13	24	6	1,626.41	1,542.41	1,626.41	1,626.41	5.03	0.56	0.94	0.94	5.17	0	0
R4	6	3	5	3	19	17	24	14	24	9	634.71	464	634.71	634.71	4.29	1.89	0.69	0.69	26.9	0	0
R5	6	3	5	3	20	26	42	17	24	12	1,553	1,553	1,553	1,553	7.87	3.32	0.46	0.46	0	0	0
R6	6	3	5	3	20	19	30	17	30	9	257	257	257	257	8.29	1.42	0.63	0.63	0	0	0
R7	6	4	5	3	34	29	51	19	30	12	302	302	302	302	16.48	6.37	0.55	0.55	0	0	0
R8	6	5	5	3	27	31	48	20	30	15	904	904	904	904	17.05	7.63	0.6	0.6	0	0	0
R9	7	5	6	3	42	35	60	23	36	15	2,129.85	2,071	2,129.85	2,129.85	17.82	15.35	0.66	0.66	2.76	0	0
R10	8	5	7	3	47	40	75	24	42	15	2,457.89	2,073.01	2,457.89	2,457.89	26.18	15.83	0.83	0.83	15.66	0	0
R11	8	6	7	3	46	38	60	26	42	18	3,291.11	2,938.69	3,291.11	3,291.11	28.05	16.96	0.99	0.99	10.71	0	0
R12	8	6	8	3	56	33	45	29	48	18	2,332.52	2,311.52	2,332.52	2,332.52	32.98	14.39	1.08	1.08	0.9	0	0
R13	9	7	8	3	58	59	114	30	48	21	—	2,171.16	2,736.75	2,736.75	—	37.42	1.29	1.29	—	—	—
R14	10	7	9	3	76	56	105	33	54	21	—	3,545	4,967.46	4,967.46	—	47.55	2.51	2.51	—	—	—
R15	10	8	9	3	90	66	126	34	54	24	—	2,711.49	2,887.32	2,887.32	—	68.24	25.66	25.66	—	—	—
R16	10	9	9	3	96	75	144	36	54	27	—	2,169.34	2,276.74	2,410.69	—	123.58	25.76	48.83	—	—	—
R17	12	5	4	4	44	31	44	24	32	20	—	1,262	1,623.69	1,626	—	16.3	0.95	9.94	—	—	—
R18	10	5	4	30	46	162	360	22	240	150	—	—	689.161	689.161	—	—	11.61	11.61	—	—	—
R19	12	10	11	4	119	93	212	43	88	40	—	3,717.17	3,928.26	3,928.37	—	726.57	703.53	763.15	—	—	—

Table A1   Characteristics

Feed	Price DM/bbl	Supply $\times 10^2$ bbl	Pool	Capacity $\times 10^2$ bbl	Blend	Price DM/bbl	Demand min $\times 10^2$ bbl	Arc	Max $\times 10^2$ bbl
F_1	49.2	60.9756	P_1	12.5	B_1	190	5	x_{12}	7.5
F_2	62.0	161.29	P_2	17.5	B_2	230	5	x_{31}	7.5
F_3	300.0	5			B_3	150	5		

	Attribute					Minimum			Maximum	
Feed	DEN	BNZ	ROZ	MOZ	Blend	DEN	ROZ	MOZ	DEN	BNZ
F_1	0.82	3	99.2	90.5	B_1	0.74	95	85	0.79	—
F_2	0.62	0	87.9	83.5	B_2	0.74	96	88	0.79	0.9
F_3	0.75	0	114	98.7	B_3	0.74	91	—	0.79	—

in all cases except the AST problems. This is caused by the degeneracy in the AST problems, which does not allow efficient shaking. Note that any improvements by VNS to MALT were obtained very quickly.

4.3.2. Randomly Generated Pooling Problems. We generated 19 problems with the following predetermined characteristics.

- number of feeds n^F varies from 6 to 12
- number of pools n^P varies from 3 to 10
- number of blends n^B varies from 4 to 11
- number of attributes n^A varies from 3 to 30

All other input parameters of the model are generated at random within intervals derived from feasibility requirements.

Computer results for the random pooling problems are summarized in Table 7. The flow formulation is used here, and we use the same following parameters for all the instances: $nsp = 100$, $k_{max} = 100$, and $nt = 1$.

Observe several things:
- VNS always gives the best results,
- MALT and VNS outperform MSLP in all cases,
- the improvements made by VNS are obtained in small amounts of additional computing time over that of MALT,
- the exact algorithm solves larger instances than previously, in very moderate time,
- the largest instances (R13 and above) could not be solved by the current version of the exact algorithm because of memory limitation, and
- MALT and VNS are always optimal for the problems solved exactly and differ in three instances.

Figure A1   Rehfeldt and Tisljar's Second Pooling Problem
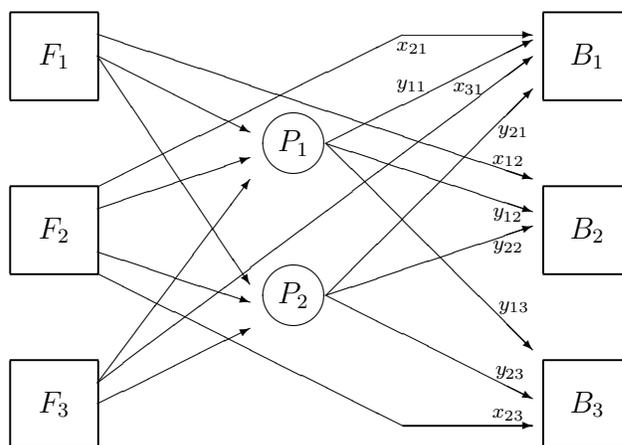
Appendix
Figure A1 and Table A1 summarize the second pooling problem in Rehfeldt and Tisljar (1997).

Since each feed may enter each intermediate pool and each intermediate pool is connected to each final blend, we may assume without any loss of generality that the flow in the largest pool is greater than or equal to that of the smallest. This additional constraint significantly reduces the feasible region.

References
Adhya, N., N. V. Sahinidis, M. Tawarmalani. 1999. A Lagrangian approach to the pooling problem. *Indust. Engrg. Chemistry Res.* **38** 1956–1972.

Al-Khayyal, F. A., J. E. Falk. 1983. Jointly constrained biconvex programming. *Math. Oper. Res.* **8**(2) 273–286.

Alameddine, A., H. D. Sherali. 1992. A new reformulation-linearization technique for bilinear programming problems. *J. Global Optim.* **2** 379–410.

Amos, F., M. Rönnqvist, G. Gill. 1997. Modeling the pooling problem at the New Zealand refining company. *J. Oper. Res. Soc.* **48** 767–778.

Androulakis, I. P., V. Visweswaran, C. A. Floudas. 1996. Distributed decomposition-based approaches. C. A. Floudas, P. M. Pardalos, eds. *State of the Art in Global Optimization: Computational Methods and Applications.* Kluwer Academics Publishers, Dordrecht, The Netherlands, 285–301.

Audet, C., P. Hansen, B. Jaumard, G. Savard. 1997. Links between the linear bilevel and mixed 0–1 programming problem. *J. Optim. Theory Appl.* **93**(2) 273–300.

Audet, C., P. Hansen, B. Jaumard, G. Savard. 2000. A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Math. Programming* **87**(1) 131–152.

Baker, T. E., L. S. Lasdon. 1985. Successive linear programming at Exxon. *Management Sci.* **31**(3) 264–274.

Ben-Tal, A., G. Eiger, V. Gershovitz. 1994. Global minimization by reducing the duality gap. *Math. Programming* **63** 193–212.

Brimberg, J., G. Oron, A. Mehrez. 1997. An operational model for utilizing water sources of varying qualities in an agricultural enterprise. *Geography Res. Forum* **17** 67–77.

Chung, S. J. 1989. NP-completeness of the linear complementarity problem. *J. Optim. Theory Appl.* **60**(3) 393–399.

Dewitt, C. W., L. S. Lasdon, D. A. Brenner, S. A. Melhem. 1989. OMEGA: An improved gasoline blending system for Texaco. *Interfaces* **19** 85–101.

Dür, M., R. Horst. 1997. Lagrange duality and partitioning techniques in nonconvex global optimization. *J. Optim. Theory Appl.* **95**(2) 347–369.

Floudas, C. A., A. Aggarwal. 1990. A decomposition strategy for global optimum search in the pooling problem. *ORSA J. Comput.* **2**(3) 225–235.

Floudas, C. A., V. Visweswaran. 1993a. A primal-relaxed dual global optimization approach. *J. Optim. Theory Appl.* **78**(2) 187–225.

Floudas, C. A., V. Visweswaran. 1993b. New properties and computational improvement of the GOP algorithm for problems with quadratic objective function and constraints. *J. Global Optim.* **3** 439–462.

Floudas, C. A., V. Visweswaran. 1996. New formulations and branching strategies for the GOP algorithm. I. E. Grossmann, ed. *Global Optimization in Engineering Design*. Kluwer Academics Publishers, Dordrecht, The Netherlands.

Foulds, L. R., D. Haugland, K. Jörnsten. 1992. A bilinear approach to the pooling problem. *Optimization* **24** 165–180.

Griffith, R. E., R. A. Stewart. 1961. A nonlinear programming technique for the optimization of continuous processing systems. *Management Sci.* **7** 379–392.

Grossmann, I. E., I. Quesada. 1995. Global optimization of bilinear process networks with multicomponents flows. *Comput. Chemical Engrg.* **19**(12) 1219–1242.

Hansen, P., B. Jaumard. 1992. Reduction of indefinite quadratic programs to bilinear programs. *J. Global Optim.* **2** 41–60.

Hansen, P., N. Mladenović. 1997. Variable neighborhood search. *Comput. Oper. Res.* **24** 1097–1100.

Hansen, P., N. Mladenović. 1999. An introduction to variable neighborhood search. S. Voss, S. Martello, I. Osman, R. Roucairol, eds. *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, Dordrecht, The Netherlands, 433–458.

Hansen, P., B. Jaumard, G. Savard. 1992. New branch-and-bound rules for linear bilevel programming. *SIAM J. Sci. Statist. Comput.* **13** 1194–1217.

Haverly, C. A. 1978. Studies of the behaviour of recursion for the pooling problem. *ACM SIGMAP Bull.* **25** 19–28.

Lasdon, L., B. Joffe. 1990. The relationship between distributive recursion and successive linear programming in refining production planning models. *NPRA Comput. Conf.* Seattle, WA.

Lasdon, L., A. Waren, S. Sarkar, F. Palacios-Gomez. 1979. Solving the pooling problem using generalized reduced gradient and successive linear programming algorithms. *ACM SIGMAP Bull.* **27** 9–25.

Lodwick, W. A. 1992. Preprocessing nonlinear functional constraints with applications to the pooling problem. *ORSA J. Comput.* **4**(2) 119–131.

Palacios-Gomez, F., L. S. Lasdon, M. Engquist. 1982. Nonlinear optimization by successive linear programming. *Management Sci.* **28**(10) 1106–1120.

Rehfeldt, M., R. Tisljar. 1997. Private communication, November 20.

Sahinidis, N. V. 1996. BARON: A general purpose global optimization software package. *J. Global Optim.* **8** 201–205.

Sherali, H. D., C. H. Tuncbilek. 1992. A global optimization algorithm for polynomial programming using a reformulation-linearization technique. *J. Global Optim.* **2** 101–112.

Sherali, H. D., C. H. Tuncbilek. 1997a. Comparison of two reformulation-linearization technique based linear programming relaxations for polynomial programming problems. *J. Global Optim.* **10** 381–390.

Sherali, H. D., C. H. Tuncbilek. 1997b. New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems. *Oper. Res. Lett.* **21** 1–9.

Zhang, J., N. H. Kim, L. S. Lasdon. 1985. An improved successive linear programming algorithm. *Management Sci.* **31**(10) 1312–1331.