Les Cahiers du GERAD ISSN: 0711–2440

Pooling Problem: Alternate Formulations and Solution Methods

C. Audet, J. Brimberg
P. Hansen, S. Le Digabel
N. Mladenović

G-2000-23

May 2000
Revised: July 2002

Les textes publiés dans la série des rapports de recherche HEC n'engagent que la responsabilité de leurs auteurs. La publication de ces rapports de recherche bénéficie d'une subvention du Fonds F.C.A.R.

Pooling Problem: Alternate Formulations and Solution Methods

Charles Audet
École Polytechnique de Montréal
C.P. 6079, Succ. Centre-ville
Montréal (Québec) Canada H3C 3A7
Charles.Audet@gerad.ca

Jack Brimberg
School of Business Administration
University of Prince Edward Island
Charlottetown (PEI) Canada C1A 4P3
jbrimberg@upei.ca

Pierre Hansen
Pierre.Hansen@gerad.ca

Sébastien Le Digabel
Sebastien.Le.Digabel@gerad.ca

Nenad Mladenović
nenad@crt.umontreal.ca

GERAD and École des Hautes Études Commerciales
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

May, 2000
Revised: July, 2002

Les Cahiers du GERAD
G–2000–23

Copyright © 2002 GERAD

Abstract

The pooling problem, which is fundamental to the petroleum industry, describes a situation where 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.

Key words: Pooling Problem, Bilinear Programming, Branch-and-cut, Heuristics, Variable Neighborhood Search.

Résumé

Le problème de mélange multi-échelons (pooling problem) qui est fondamental pour l'industrie pétrolière, décrit une situation dans laquelle des produits ayant des qualités différentes pour divers attributs sont mélangés dans une série de réservoirs de sorte que les qualités des attributs des produits mélangés dans les réservoirs finaux satisfassent des conditions données. Il est bien connu que le problème de mélange avec étapes intermédiaires peut être modélisé sous la forme d'un programme bilinéaire. Dans cet article nous étudions deux modèles standards : le premier est basé principalement sur des flots et le second sur la proportion des flots entrants dans les réservoirs. Un hybride de ces deux modèles est proposé pour le problème de mélange multi-échelons généralisé. Une comparaison des propriétés de calcul des modèles de flot et de proportion est faite sur la base de plusieurs exemples de la littérature, résolus par un algorithme de branchement et coupes. Une procédure alternante simple et une heuristique de recherche à voisinage variable sont développés pour la résolution d'exemples de grande taille et comparées avec la méthode bien connue d'approximations linéaires récursives. La résolution de problèmes tests difficiles de la littérature est substantiellement accélérée, et de plus grands problèmes sont résolus exactement ou approximativement.

Mots-clés : mélange multi-échelons, programme bilinéaire, branchement et coupes, heuristique, recherche à voisinage variable.

Acknowledgments: The authors would like to thank Ultramar Canada and Luc Massé for funding this project. Work of the first author was supported by NSERC (Natural Sciences and Engineering Research Council) fellowship PDF-207432-1998 and by CRPC (Center for Research on Parallel Computation). Work of the second author was supported by NSERC grant #OGP205041. Work of the third author was supported by FCAR (Fonds pour la Formation des Chercheurs et l'Aide à la Recherche) grant #95ER1048, and NSERC grant #GP0105574.

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 which 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 in order 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 in identifying that one which 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.* 2000a). 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 bilinear programming (BLP) problem is usually formulated by dividing a set of variables into two subsets: linear and nonlinear variables (see for example Baker and Lasdon, 1985). The nonlinear variables may be further divided into two disjoint subsets, called non complicating and complicating, respectively. This partition is based on the simple 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.

In order 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 non-complicating 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 non-complicating 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, $i = 1, .., m$
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.

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

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 small example by Haverly (1978). In order to illustrate the potential difficulty of the problem, a two step iterative algorithm was presented. It consists in 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 in replacing bilinear terms by first-order Taylor expansions in order 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 by SLP). Other SLP algorithms are detailed in Palacios-Gomez, Lasdon and Engquist (1982) and Zhang, Kim and Lasdon (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 and 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 five pools, four blends, twelve feeds and thirty attributes. Analysing continuous branch-and-bound algorithms, Dür and Horst (1997) show that the duality gap goes to zero for some general non-convex optimization problems that include the pooling problem.

Lodwick (1992) discusses pre-optimization and post-optimization analyses of the bilinear programming model. This work allows identification of some constraints that will be tight and others that will be redundant at optimality. Foulds, Haugland and Jörnsten (1992) apply Al-Khayyal and Falk's (1983) branch and bound algorithm for bilinear programming to the pooling problem. This method finds in finite time a solution as close as desired to a globally optimal solution. The general idea consists in replacing each bilinear term by a linear variable, and to add linear constraints in order 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 an 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.* (2000a) strengthen several aspects of Al-Khayyal and Falk's (1983) algorithm (the improvements will be discussed later in the paper when computational results are presented). This algorithm is also based on the Reformulation-Linearization Techniques of Sherali and Tuncbilek (1992), (1997a), (1997b).

Ben-Tal *et al.* (1994) use the proportion model (detailed in subsection 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 a LP-relaxation. This reformulation is then used within a spatial branch-and-bound algorithm.

Amos, Rönnqvist and Gill (1997) use cumulative functions describing the distillation yield to model the pooling problem through a nonlinear constrained least-square problem. Promising results are shown on examples derived 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 in order 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 bilinear programming problems. The variables are either partitioned into *flow* and *attribute* variables (as suggested in Haverly (1978), and made more formal in Foulds *et al.* (1992)) or into *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. The 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.*, 2000a) 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/or to 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 $k = 1, 2, \ldots n^B$
$\ell_{ik}^{FB}, u_{ik}^{FB}$	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 $j = 1, 2, \ldots n^P$
$\ell_{ij}^{FP}, u_{ij}^{FP}$	lower and upper bounds on the capacity of arc $(i,j) \in W$
$\ell_{jk}^{PB}, u_{jk}^{PB}$	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 for $a = 1, 2, \ldots n^A$ and $k = 1, 2, \ldots n^B$

Throughout the paper, the following notation concerning sets of pairs of indices such as X is used. 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 reduction of the number of variables. In order 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, due to their complexity (they contain logarithms or fourth order terms) these models are not considered in this paper.

The flow variables are given by:

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. To our best knowledge, these are the first complete formulation of the pooling problem which provide for automated construction of the model and comparison of the computational aspects of the two forms.

2.1 Flow Model of the Pooling Problem

In this subsection, we develop a bilinear programming 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 $i(j) \in W^{-1}_{(j)}$ is defined to be 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:

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

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^a_j is introduced to represent the attribute quality of the intermediate pool P_j. Assuming that the attribute qualities blend linearly, we obtain that

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

This equation simplifies to the bilinear constraint

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

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

$$\frac{\sum_{i \in X^{-1}_{(k)}} s^a_i x_{ik} + \sum_{j \in Y^{-1}_{(k)}} t^a_j y_{jk}}{\sum_{i \in X^{-1}_{(k)}} x_{ik} + \sum_{j \in Y^{-1}_{(k)}} y_{jk}}.$$

The flow bilinear programming formulation which 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: $\{t^a_j\}$ 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.

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

subject to :

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

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

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

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

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 $k = 1, 2, \ldots, n^B$:

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

capacity of the arcs:

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

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

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

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

non-negativity: $w \geq \mathbf{0}, x \geq \mathbf{0}, y \geq \mathbf{0}$

Figure 1: Flow model

Note also that if the t^a_j 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^a_j 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 upon the proportion of flow entering at the intermediate pools. The parameters are the same as 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:

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

The proportion bilinear programming formulation is given in Figure 2.

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 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.

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

subject to :

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

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

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

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

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

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

capacity of the arcs: $\ell^{FP}_{i(j)j} \leq \left(1 - \sum_{\substack{h\in W^{-1}_{(j)} \\ h\neq i(j)}} q_{hj}\right) \sum_{k\in Y_{(j)}} y_{jk} \leq u^{FP}_{i(j)j}$, $\left(i(j), j\right) \in W$

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

$\ell^{FB}_{ik} \leq x_{ik} \leq u^{FB}_{ik}$, $(i,k) \in X$

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

non-negativity and proportion $j = 1, 2, \ldots, n^P$: $x \geq \mathbf{0}, y \geq \mathbf{0}, q \geq \mathbf{0}, \sum_{\substack{h\in W^{-1}_{(j)} \\ h\neq i(j)}} q_{hj} \leq 1$

Figure 2: Proportion model

2.3 Comparison of the Flow and Proportion Models

For comparison purposes, the following ten examples are taken from the literature: Haverly's (1978) pooling problem (referred to as H1), Ben Tal *et al.*'s (1994) fourth and fifth pooling problems (BT4, BT5), Rehfeldt and Tisljar's (1997) first and second problems (RT1, RT2), Foulds *et al.*'s (1992) second problem (F2) 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 ten examples appears in Table 1. The table is divided into four groups of columns. The left one (Lin Var)

Example	Lin	Bilinear		Bilinear		Constraints		
	Var	Var	(Pool+Flow)	Trm	$\sum$(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

Table 1: Summary of the number of variables and constraints

shows the number of variables that are not involved in bilinear terms (the linear variables). The second group details the number of bilinear variables. They are partitioned in two subsets (each bilinear term consists of the product of one variable from each subset), and the number of variables in each subset of the partition is between 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 next 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 $\{t_1^1, t_1^2, t_1^3, t_1^4\} \times \{y_{11}, y_{12}, y_{13}\}$ and $\{t_2^1, t_2^2, t_2^3, t_2^4\} \times \{y_{21}, y_{22}, y_{23}\}$; 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 $\mathrm{L}_{\leq}$, quadratic inequalities $\mathrm{Q}_{\leq}$, and quadratic equalities $\mathrm{Q}_{=}$ in the constraint set of each problem.

The difficulty of the bilinear programming 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_j^a) will then be small. When the number of attributes increases, the advantage of the proportion model becomes apparent since 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 when these problems are solved exactly that this is not the case. 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.

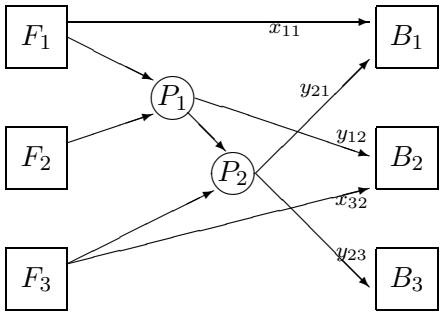

Figure 3: A Generalized Pooling Problem (GP1)

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

Table 2: Characteristics of GP1

The proportion model of the generalized pooling problem is not a bilinear program, since 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).

$$
\begin{aligned}
\max_{q,t,v,w,x,y} \quad & -6(x_{11}+y_{12}+v_{12}-q_{21}(y_{12}+v_{12}))-16q_{21}(y_{12}+v_{12})-10(x_{32}+y_{21}+y_{23}-v_{12}) \\
& +9(x_{11}+y_{21})+13(y_{12}+x_{32})+14y_{23} \\
\text{s.t.} & \\
\text{supply:} \quad & x_{11}+y_{12}+v_{12}-q_{21}(y_{12}+v_{12}) \le 18 \\
& q_{21}(y_{12}+v_{12}) \le 18 \\
& x_{32}+y_{21}+y_{23}-v_{12} \le 18 \\
\text{demand:} \quad & x_{11}+y_{21} \le 10 \\
& y_{12}+x_{32} \le 15 \\
& y_{23} \le 20 \\
\text{capacity:} \quad & y_{12}+v_{12} \le 20 \\
& y_{21}+y_{23} \le 20 \\
\text{attribute:} \quad & (3-2q_{21})v_{12}+2(y_{21}+y_{23}-v_{12})-t_2^1(y_{21}+y_{23})=0 \\
& 3x_{11}+t_2^1 y_{21} \le 2.5(x_{11}+y_{21}) \\
& 2x_{32}+(3-2q_{21})y_{12} \le 1.75(x_{32}+y_{12}) \\
& t_2^1 \le 1.5 \\
\text{pos. flow:} \quad & y_{21}+y_{23}-v_{12} \ge 0 \\
& q \ge 0, t \ge 0, v \ge \mathbf{0}, w \ge \mathbf{0}, x \ge \mathbf{0}, y \ge \mathbf{0}, q_{21} \le 1.
\end{aligned}
$$

2.5 Simplification of some pooling problem instances

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

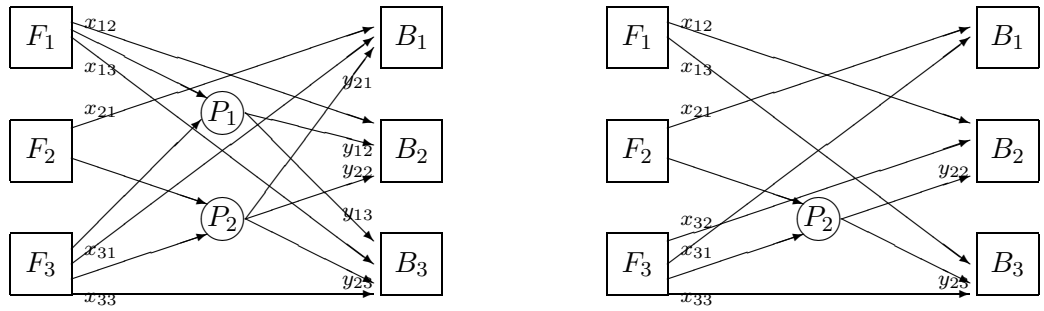

Figure 4: Simplification of Rehfeldt and Tisljar's first pooling problem (RT1)

A closer look at the definition of the problem allows considerable simplifications. 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.

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 fixing it 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 since all intermediate pools may be eliminated. Thus, Rehfeldt and Tilsjar'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 $s_i^1 = \frac{9+i}{10}$ and $p_i^F = 21 - i$ for $i = 1, 2, ..., 11$. 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 $p_k^B = \frac{41-k}{2}$ and $u_k^1 = 1 + .05k$ for $k = 1, 2, ..., 16$.

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 get 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 $k = 1, 2, ..., 16$. This optimal solution is displayed in Figure 5.

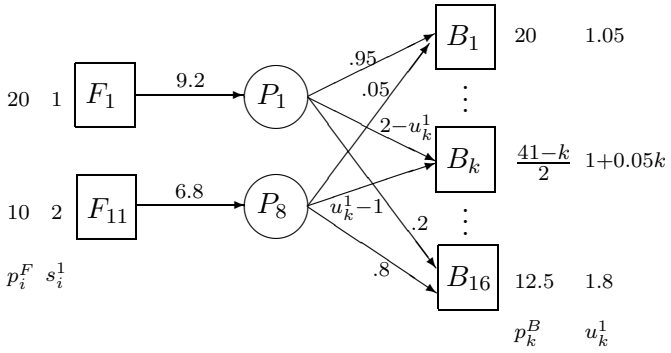

Figure 5: Simplification of F3, F4 and F5

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 successive linear programming (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.* (2000a) 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), (1997b). The improvement to these methods are: (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 cuts: cuts derived from under-approximation 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.

Example	**[ref]**	$\mathbf{n^F}$	$\mathbf{n^P}$	$\mathbf{n^B}$	$\mathbf{n^A}$	**solution**
AST1	[1]	5	2	4	4	549.803
AST2	[1]	5	2	4	6	549.803
AST3	[1]	8	3	4	6	561.048
AST4	[1]	8	2	5	4	877.649
BT4	[11]	4	1	2	1	45
BT5	[11]	5	3	5	2	350
F2	[20]	6	2	4	1	110
H1	[27]	3	1	2	1	40
H2	[27]	3	1	2	1	60
H3	[27]	3	1	2	1	75
RT1	[32]	3	2	3	4	4136.22
RT2	[32]	3	2	3	4	4391.83
GP1		3	2	3	1	60.5

Table 3: Instances characteristics

Computational experiments were completed on a Sun Ultra-60 (UltraSPARC II 360 MHz processor), with a C++ implementation of the algorithm. The tested instances have been reviewed in Section 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 pre-processing 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 used to prove the optimality of the solution.

Table 4-(i) displays the computational results on the flow model of the pooling problems. The pre-processing 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

Example	Time (sec)			Nodes	Additional	
	P.P.	Tree	Total		Var	Cstr
AST1	7.39	7778.65	7786.04	4145	108	6991
AST2	-	-	-	-	-	-
AST3	-	-	-	-	-	-
AST4	13.87	940.03	953.9	723	50	3414
BT4	.17	.05	.22	9	4	41
BT5	5	660.63	665.63	97	30	5776
F2	.51	.16	.67	15	10	128
H1	.2	.06	.26	9	4	41
H2	.13	.03	.16	3	2	20
H3	.14	.03	.17	3	1	15
RT1	2.12	29.69	31.81	179	38	1546
RT2	3.97	201.02	204.99	489	63	2581

(i) Flow model, complete solution

Example	Time (sec)			Nodes	Additional	
	P.P.	Tree	Total		Var	Cstr
AST1	10.23	4258.53	4268.76	3177	80	5801
AST2	-	-	-	-	-	-
AST3	-	-	-	-	-	-
AST4	55.43	4587.01	4642.44	2355	43	3656
BT4	.19	.03	.22	1	0	7
BT5	1.28	.16	1.44	1	0	90
F2	3.66	6.34	10	155	33	687
H1	.21	.02	.23	1	0	7
H2	.4	0	.4	0	0	0
H3	.17	.02	.19	1	0	6
RT1	13.69	.48	14.17	43	10	225
RT2	8.37	366.28	374.65	1381	62	2626

(ii) Flow model, optimality proof

Example	Time (sec)			Nodes	Additional	
	P.P.	Tree	Total		Var	Cstr
AST1	.47	8.59	9.06	245	40	838
AST2	.39	9.28	9.67	267	35	777
AST3	2.45	66.06	68.5	537	38	1065
AST4	2.83	175.15	177.98	693	48	205
BT4	.15	.88	1.03	43	14	197
BT5	1.81	29.29	31.1	39	20	1274
F2	.35	.05	.4	1	0	24
H1	.17	.05	.22	9	6	57
H2	.13	.04	.17	13	6	41
H3	.13	.04	.17	7	4	35
RT1	.55	.05	.6	7	4	37
RT2	.85	1.11	1.96	59	19	347

(iii) Proportion model, complete solution

Example	Time (sec)			Nodes	Additional	
	P.P.	Tree	Total		Var	Cstr
AST1	1.26	14.47	15.73	391	48	874
AST2	1.91	2.55	4.46	139	18	408
AST3	2.19	173.67	175.86	911	44	1416
AST4	5.91	681.14	687.05	2127	47	2324
BT4	.75	.14	.89	27	8	112
BT5	.52	0	.52	0	0	0
F2	.32	.02	.34	1	0	24
H1	.18	.02	.2	1	0	7
H2	.14	.01	.15	0	0	0
H3	.23	.03	.26	11	5	37
RT1	.4	.05	.45	1	0	15
RT2	6.89	.31	7.2	47	11	175

(iv) Proportion model, optimality proof

Model	Time (sec)			Nodes	Additional	
	P.P.	Tree	Total		Var	Cstr
Flow	.19	.59	.78	47	18	238
Prop.	.22	2.5	2.72	35	18	457
Hybrid	.68	.53	1.21	27	12	225

(v) Generalized pooling problem (GP1), complete solution

Model	Time (sec)			Nodes	Additional	
	P.P.	Tree	Total		Var	Cstr
Flow	.62	.07	.69	13	6	78
Prop.	1.23	.17	1.4	17	8	132
Hybrid	.62	0	.62	0	0	0

(vi) GP1, optimality proof

Table 4: Computational results

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 problems AST and for Example RT2 where the computational time increased. As for the remaining examples, the pre-processing time increased, but the tree exploration phase decreased. This is explained by the addition of the non-linear 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. Contrary to the standard pooling problem, the proportion model of the generalized problem appears to be harder to solve than the flow model. The addition of 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, i.e. 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.* (2000a) ; 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 Audet *et al.* (2000a) algorithm appears to notably advance the state-of-the-art.

4 Heuristic Methods

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

Example	Foulds 92[20] CDC 4340 25 Mhz	Floudas 93[18] HP9000/730 67 Mhz	Floudas 96[19] HP9000/730 67 Mhz	Adhya *et al.* 99[1] RS6000/43P 100-133 Mhz	Audet *et al.* 02 Ultra-60 360 Mhz
AST1	-	-	-	425	9.06
AST2	-	-	-	1115	9.67
AST3	-	-	-	19314	68.5
AST4	-	-	-	182	177.98
BT4	-	44.54	.95	.11	.22
BT5	-	40.31	5.8	1.12	31.1
F2	3	-	-	.1	.4
H1	.7	.95	.22	.09	.22
H2	-	3.19	.21	.09	.16
H3	-	-	.26	.13	.17
RT1	-	-	-	-	.6
RT2	-	-	-	-	1.96
GP1	-	-	-	-	.78

Table 5: Comparative CPU times for exact solution

solutions for use in exact algorithms. In this section, we propose a local search procedure which alternately fixes the non-complicating 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 variable neighborhood search procedure. The alternating and variable neighborhood search methods are applied to the problem cases of Section 2 and to a large set of randomly-generated pooling problems. The results are compared with the commonly-used successive linear programming (SLP) method.

4.1 Alternate heuristic (ALT)

The principle of the ALT heuristic consists, given two subsets of variables, in 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 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

[1]	give some feasible initial values for the complicating variables z (or non-complicating 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)

Figure 6: Alternate heuristic (ALT)

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 alternate heuristic 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.*

Proof: Fixing feasible values in z (or y) leads to an LP with a non-empty feasible region in (x, y) (or (x, z)). Thus, the sequence of LP's 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 (VNS)

We assume a knowledge of the basic rules of variable neighborhood search (VNS). For a review of VNS and its applications to a range of problems, see Hansen and Mladenović

(1997 and 1999). In a nutshell, the variable neighborhood search consists in 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; else the original current point is kept and the k parameter is increased, for a bigger perturbation in step (i).

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 $\mathcal{P}_1(s)$ and $\mathcal{P}_2(s)$. The first neighborhood $\mathcal{N}_1(s)$ is represented by the union of all feasible extreme points adjacent to either (x', y') (in $\mathcal{P}_1(s)$) or (x', z') (in $\mathcal{P}_2(s)$). Thus, the cardinality of $\mathcal{N}_1(s)$ is less than or equal to $2n_1 + n_2 + n_3$, since 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). $\mathcal{N}_2(s)$ would then be the set of adjacent extreme points on $\mathcal{P}_1(s)$ or $\mathcal{P}_2(s)$ obtained by changing exactly two elements in the respective bases; $\mathcal{N}_3(s)$ exactly three elements; and so on. It is easy to see that the cardinality of $\mathcal{N}_k(s)$ increases exponentially with k.

For solving BLP by VNS, we define the *shaking* operator with respect to the neighborhoods $\mathcal{N}_k$. That is, a point s' from $\mathcal{N}_k(s)$ $(k = 1, \ldots, k_{max})$, 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 $\mathcal{P}_1$ or $\mathcal{P}_2$, respectively. If the shaking is done in $\mathcal{P}_1$, the search by ALT will start in $\mathcal{P}_2$. The next shaking operation will also be carried out in $\mathcal{P}_2$. That is, we alternate between subspaces by setting δ to its complement $\overline{\delta}$. 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 section 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 $\mathcal{P}_1$ and $\mathcal{P}_2$ does not mean that the global optimum necessarily belongs to the set $\cup_k \mathcal{N}_k(s)$, 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 $\mathcal{S}_s$ that consists of all feasible extreme points of $\mathcal{P}_1(s)$ and $\mathcal{P}_2(s)$. A distance function ρ_s is then specified for the set $\mathcal{S}_s$ (subscript s indicates that the discretization depends on the current solution). Let s_1 and s_2 be any two solutions that belong to $\mathcal{S}_s$, and let $\mathcal{B}_1$ and $\mathcal{B}_2$ be the corresponding sets of basic variables. We say that

$$\rho_s(s_1, s_2) = k \Longleftrightarrow |\mathcal{B}_1 \setminus \mathcal{B}_2| = |\mathcal{B}_2 \setminus \mathcal{B}_1| = k,$$

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

[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 $\mathcal{N}_k(s)$ at random using current value of δ
[ii] **Local search**
$\delta \leftarrow \overline{\delta}$
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 $\mathcal{N}_k(s)$ for the new current solution (i.e., update the simplex table, or polyhedron, where s" is found)
else $k \leftarrow k + 1$
$it \leftarrow it + 1$

Figure 7: VNS algorithm

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) since the solutions of the LPs typically have many zero-valued variables, set every variable in z (or y) to zero with probability .5; if the variable is decided to be non-zero, 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.

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.

In order 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 Section 2 and 3, and 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 3 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 on 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 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.

Example	Parameters			Solution				CPU time (s)			Error (%)		
	nsp	k_{max}	*nt*	Exact	MSLP	MALT	VNS	MSLP	MALT	VNS	MSLP	MALT	VNS
Flow Model													
AST1	1000	10	1	549.803	276.661	532.901	545.27	2.20	2.45	2.81	49.68	3.07	.82
AST2	1500	10	1	549.803	284.186	535.617	543.909	9.18	5.21	5.68	48.31	2.58	1.07
AST3	1000	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	.82	.77	1.01	-	.16	.16
BT4	5	0	0	45	39.6970	45	45	.01	.01	.01	11.78	0	0
BT5	10	15	2	350	327.016	324.077	350	.03	.09	1.11	6.57	7.41	0
F2	120	10	1	110	100	107.869	110	.07	.44	.57	9.09	1.94	0
H1	5	0	0	40	40	40	40	.02	.01	.01	0	0	0
H2	5	0	0	60	60	60	60	.02	.01	.01	0	0	0
H3	5	3	1	75	60.7332	70	75	.02	.01	.03	19.02	6.67	0
RT1	5	0	0	4136.22	126.913	4136.22	4136.22	1.34	.04	.04	96.93	0	0
RT2	5	5	1	4391.83	-	4330.78	4391.83	.04	.47	.60	-	1.39	0
GP1	50	5	1	60.5	28.732	35	46	.01	.04	.08	52.51	42.15	23.97
Proportion Model													
AST1	1000	10	1	549.803	544.307	532.901	533.783	1.14	2.38	2.61	1	3.07	2.91
AST2	1500	10	1	549.803	548.407	535.617	542.54	3.04	4.97.	5.37	.25	2.58	1.32
AST3	1000	10	1	561.048	551.081	397.441	558.835	4.98	4.98	5.93	1.68	29.09	.3
AST4	230	0	0	877.649	-	876.206	876.206	1.19	1.21	1.55	-	.16	.16
BT4	5	0	0	45	39.7019	45	45	.01	.02	.02	11.77	0	0
BT5	10	15	2	350	292.532	323.12	350	.12	.16	1.53	16.42	7.68	0
F2	120	0	0	110	110	110	110	.15	.49	.49	0	0	0
H1	5	0	0	40	40	40	40	.02	.01	.01	0	0	0
H2	5	0	0	60	60	60	60	.02	.01	.01	0	0	0
H3	5	3	1	75	69.9934	70	75	.02	.01	.02	6.68	6.67	0
RT1	5	0	0	4136.22	3061.03	4136.22	4136.22	.07	.03	.03	25.99	0	0
RT2	5	5	1	4391.83	4391.02	4330.77	4391.82	.04	.58	.72	.02	1.39	0

Table 6: Pooling test problems from the literature

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. For comparison purposes, the data describing these examples can be found at www.gerad.ca/Charles.Audet.

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 $nt = 1$.

Ex	Parameters				Dimensions						Solution				CPU time (s)				Error (%)		
	n^F	n^P	n^B	n^A	Lin Var	Bilinear Var	Bilinear Trm	Constraints $L_{\leq}$	Constraints $Q_{\leq}$	Constraints $Q_{=}$	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	.39	.51	.51	0	0	0
R2	6	3	5	3	13	12	9	10	18	9	572	394	572	572	1.33	.55	.6	.6	31.12	0	0
R3	6	3	5	3	14	12	18	13	24	6	1626.41	1542.41	1626.41	1626.41	5.03	.56	.94	.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	.69	.69	26.9	0	0
R5	6	3	5	3	20	26	42	17	24	12	1553	1553	1553	1553	7.87	3.32	.46	.46	0	0	0
R6	6	3	5	3	20	19	30	17	30	9	257	257	257	257	8.29	1.42	.63	.63	0	0	0
R7	6	4	5	3	34	29	51	19	30	12	302	302	302	302	16.48	6.37	.55	.55	0	0	0
R8	6	5	5	3	27	31	48	20	30	15	904	904	904	904	17.05	7.63	.6	.6	0	0	0
R9	7	5	6	3	42	35	60	23	36	15	2129.85	2071	2129.85	2129.85	17.82	15.35	.66	.66	2.76	0	0
R10	8	5	7	3	47	40	75	24	42	15	2457.89	2073.01	2457.89	2457.89	26.18	15.83	.83	.83	15.66	0	0
R11	8	6	7	3	46	38	60	26	42	18	3291.11	2938.69	3291.11	3291.11	28.05	16.96	.99	.99	10.71	0	0
R12	9	6	8	3	56	33	45	29	48	18	2332.52	2311.52	2332.52	2332.52	32.98	14.39	1.08	1.08	.9	0	0
R13	9	7	8	3	58	59	114	30	48	21	-	2171.16	2736.75	2736.75	-	37.42	1.29	1.29	-	-	-
R14	10	7	9	3	76	56	105	33	54	21	-	3545	4967.46	4967.46	-	47.55	2.51	2.51	-	-	-
R15	10	8	9	3	90	66	126	34	54	24	-	2711.49	2887.32	2887.32	-	68.24	25.66	25.66	-	-	-
R16	10	9	9	3	96	75	144	36	54	27	-	2169.34	2276.74	2410.69	-	123.58	25.76	48.83	-	-	-
R17	12	5	4	4	44	31	44	24	32	20	-	1262	1623.69	1626	-	16.3	.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	-	3717.17	3928.26	3928.37	-	726.57	703.53	763.15	-	-	-

Table 7: Randomly generated pooling test problems

Observe that (i) VNS always gives the best results, (ii) MALT and VNS outperform MSLP in all cases, (iii) the improvements made by VNS are obtained in small amounts of additional computing time over that of MALT, (iv) the exact algorithm solves larger instances than done previously, in very moderate time, and (v) the largest instances (R13 and above) could not be solved by the current version of the exact algorithm, due to memory limitation.

5 Discussion

Three general bilinear programming formulations of the pooling problem are developed: these represent the flow and proportion models, and a new hybrid model which may be used when intermediate pools are configured in series as well as in parallel. Using sample problems from the literature, a key observation is made that the type of formulation chosen can significantly impact the complexity of the model, and as a result, the computational effort required to solve it.

A recently developed branch and cut algorithm is tested on the sample problems. The results show the computational time increases rapidly with the number of bilinear variables and constraints. Furthermore, using a good heuristic in conjunction with the exact procedure is demonstrated to substantially reduce this time. Also observe that a combination of branch-and-cut and of best model formulation can yield much better computation times than those of state-of-the-art algorithms.

Two new heuristic procedures are proposed. The first, a simple iterative scheme applied to two LPs, differs from previous recursive methods by identifying a set of linear variables and including them in both LPs. As a local search, the new alternating procedure (ALT) is seen to perform as well or better than the well-studied method of successive linear programming (SLP) on a set of randomly-generated pooling problems. Furthermore, ALT automatically produces a sequence of feasible solutions once an initial feasible solution is found without any need for step-size adjustments. Some heuristic rules are also provided which appear to work well in generating initial feasible solutions. The second heuristic procedure performs a variable neighborhood search (VNS) on a set of extreme points identified at any current solution. The shaking operation chooses a random point at progressively further pivots from the current solution. A local search using ALT is then carried out at the random point. Our preliminary results suggest that VNS will improve the quality of solutions obtained by multistart versions of SLP and ALT, particularly for larger problem instances, within a modest increase in computing time.

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. Here, its effect on computational performance is investigated for the pooling problem. Other structured global optimization problems can be studied in the same way. For instance, this

approach led to determination of the octagon with unit diameter and largest area (Audet *et al.* 2002) and to the first exact algorithm for fractional goal programming (Audet *et al.* 2000b). Several other such problems will be investigated in future work.

Appendix

The following figure and table summarize the second pooling problem in Rehfeldt and Tisljar (1997).

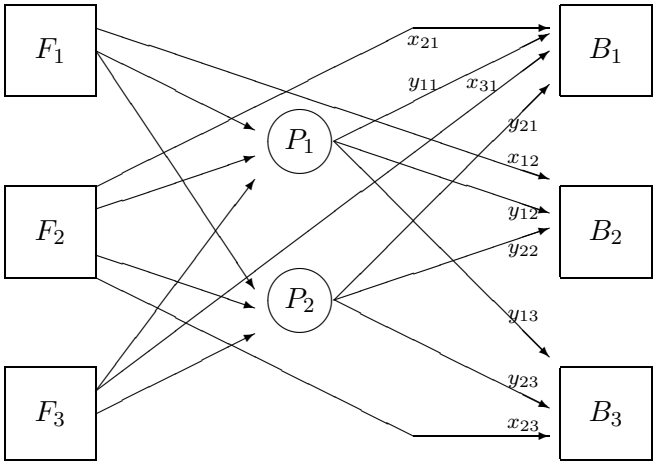

Figure 8: Rehfeldt and Tisljar's second pooling problem

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	$\times 10^2 bbl$ max
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	.82	3	99.2	90.5	B_1	.74	95	85	.79	-
F_2	.62	0	87.9	83.5	B_2	.74	96	88	.79	.9
F_3	.75	0	114	98.7	B_3	.74	91	-	.79	-

Table 8: Characteristics

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 reduces significantly the feasible region.

References

[1] ADHYA N., SAHINIDIS N.V. and TAWARMALANI M. (1999), "A Lagrangian Approach to the Pooling Problem," *Industrial & Engineering Chemistry Research* 38, 1956–1972.

[2] AL-KHAYYAL F.A. and FALK J.E.(1983), "Jointly Constrained Biconvex Programming," *Mathematics of Operations Research* 8 (2), 273–286.

[3] ALAMEDDINE A. and SHERALI H.D. (1992), "A new reformulation-linearization technique for bilinear programming problems," *Journal of Global optimization* 2, 379–410.

[4] AMOS F., RÖNNQVIST and GILL G.(1997), "Modeling the Pooling Problem at the New Zealand Refining Company," *Journal of the Operational Research Society* 48, 767–778.

[5] ANDROULAKIS I.P., FLOUDAS C.A. and VISWESWARAN V.(1996), "Distributed Decomposition-Based Approaches," in C.A. Floudas and P.M. Pardalos (eds.) *State of the Art in Global Optimization: Computational Methods and Applications,* Kluwer Academics Publishers, Dordrecht.

[6] AUDET C., HANSEN P., JAUMARD B. and SAVARD G.(1997), "Links between the Linear Bilevel and Mixed $0-1$ Programming Problem," *Journal of Optimization Theory and Applications* 93(2), 273–300.

[7] AUDET C., HANSEN P., JAUMARD B. and SAVARD G. (2000a), "A Branch and Cut Algorithm for Nonconvex Quadratically Constrained Quadratic Programming," *Mathematical Programming* 87(1), 131–152.

[8] AUDET C., CARRIZOSA E. and HANSEN P. (2000b), "An Exact Method for Fractional Goal Programming," *Les Cahiers du Gerad* G-2000-64.

[9] AUDET C., HANSEN P., MESSINE F. and XIONG J. (2002), "The Largest Small Octagon," *Journal of Combinatorial Theory, series A* 98(1),46–59.

[10] BAKER T.E. and LASDON L.S.(1985), "Successive Linear Programming at Exxon," *Management Science* 31 (3), 264–274.

[11] BEN-TAL A., EIGER G. and GERSHOVITZ V.(1994), "Global Minimization by Reducing the Duality Gap," *Mathematical Programming* 63, 193–212.

[12] BRIMBERG J., ORON G. and MEHREZ A.(1997). "An Operational Model for Utilizing Water Sources of Varying Qualities in an Agricultural Enterprise", *Geography Research Forum* 17, 67–77.

[13] CHUNG S.J.(1989), "NP-Completeness of the Linear Complementarity Problem," *Journal of Optimization Theory and Applications* 60 (3), 393–399.

[14] DEWITT C.W., LASDON L.S., BRENNER D.A. and MELHEM S.A.(1989), "OMEGA: An Improved Gasoline Blending System for Texaco," *Interfaces* 19, 85–101.

[15] DÜR M. and HORST R.(1997), "Lagrange Duality and Partitioning Techniques in Nonconvex Global Optimization," *Journal of Optimization Theory and Applications* 95 (2), 347–369.

[16] FLOUDAS C.A. and AGGARWAL A.(1990), "A Decomposition Strategy for Global Optimum Search in the Pooling Problem," *ORSA Journal on Computing* 2 (3), 225–235.

[17] FLOUDAS C.A. and VISWESWARAN V.(1993a), "A Primal-Relaxed Dual Global Optimization Approach," *Journal of Optimization Theory and Applications*, 78(2), 187–225.

[18] FLOUDAS C.A. and VISWESWARAN V.(1993b), "New Properties and Computational Improvement of the GOP Algorithm For Problems With Quadratic Objective Function and Constraints," *Journal of Global optimization* 3, 439–462.

[19] FLOUDAS C.A. and VISWESWARAN V.(1996), "New Formulations and Branching Strategies for the GOP Algorithm," in I.E. Grossmann (ed.) *Global Optimization in Engineering Design,* Kluwer Academics Publishers, Dordrecht.

[20] FOULDS L.R, HAUGLAND D. and JÖRNSTEN K.(1992), "A Bilinear Approach to the Pooling Problem," *Optimization* 24, 165–180.

[21] GRIFFITH R.E. and STEWART R.A.(1961), "A Nonlinear Programming Technique for the Optimization of Continuous Processing Systems," *Management Science* 7, 379–392.

[22] GROSSMANN I.E. and QUESADA I.(1995), "Global Optimization of Bilinear Process Networks with Multicomponents Flows," *Computers & Chemical Engineering* 19(12), 1219–1242.

[23] HANSEN P. and JAUMARD B.(1992), "Reduction of Indefinite Quadratic Programs to Bilinear Programs," *Journal of Global optimization* 2, 41–60.

[24] HANSEN P., JAUMARD B. and SAVARD G.(1992), "New Branch-and-Bound Rules for Linear Bilevel Programming," *SIAM Journal on Scientific and Statistical Computing* 13, 1194–1217.

[25] HANSEN P. and MLADENOVIĆ N. (1997). Variable Neighborhood Search. *Computers and Operations Research* 24, 1097-1100.

[26] HANSEN, P. and. MLADENOVIĆ, N.. (1999). An Introduction to Variable Neighborhood Search. In S. Voss *et al.* (eds.), *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, pp.433–458, Kluwer, Dordrecht.

[27] HAVERLY C.A.(1978), "Studies of the Behaviour of Recursion for the Pooling Problem," *ACM SIGMAP Bulletin* 25, 19–28.

[28] LASDON L. and JOFFE B.(1990), "The Relationship between Distributive Recursion and Successive Linear Programming in Refining Production Planning Models," *NPRA Computer Conference* Seattle, Washington.

[29] LASDON L., WAREN A., SARKAR S. and PALACIOS-GOMEZ F.(1979), "Solving the Pooling Problem Using Generalized Reduced Gradient and Successive Linear Programming Algorithms," *ACM Sigmap Bulletin* 27, 9–25.

[30] LODWICK W.A.(1992), "Preprocessing Nonlinear Functional Constraints with Applications to the Pooling Problem ," *ORSA Journal on Computing* 4 (2), 119-131.

[31] PALACIOS-GOMEZ F., LASDON L.S. and ENGQUIST M.(1982), "Nonlinear Optimization by Successive Linear Programming," *Management Science* 28 (10), 1106–1120.

[32] REHFELDT AND TISLJAR (1997). Private communication.

[33] SAHINIDIS N.V. (1996), "BARON: A General Purpose Global Optimization Software Package," *Journal of Global Optimization* 8, 201–205.

[34] SHERALI H.D. and TUNCBILEK C.H.(1992), "A Global Optimization Algorithm for Polynomial Programming Using a Reformulation-Linearization Technique," *Journal of Global Optimization* 2, 101–112.

[35] SHERALI H.D. and TUNCBILEK C.H.(1997a), "Comparison of Two Reformulation-Linearization Technique Based Linear Programming Relaxations for Polynomial Programming Problems," *Journal of Global Optimization* 10, 381–390.

[36] SHERALI H.D. and TUNCBILEK C.H.(1997b), "New Reformulation Linearization/Convexification Relaxations for Univariate and Multivariate Polynomial Programming Problems," *Operations Research Letters* 21, 1–9.

[37] ZHANG J, KIM N.H. and LASDON L.S.(1985), "An Improved Successive Linear Programming Algorithm," *Management Science* 31 (10), 1312–1331.