A GLOBAL OPTIMIZATION APPROACH TO POOLING PROBLEMS IN REFINERIES

A Thesis

by

VIET PHAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2007

Major Subject: Chemical Engineering

A GLOBAL OPTIMIZATION APPROACH TO POOLING PROBLEMS IN REFINERIES

A Thesis

by

VIET PHAM

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2007

Major Subject: Chemical Engineering

ABSTRACT

A Global Optimization Approach
to Pooling Problems in Refineries. (August 2007)
Viet Pham, B.S., Ho Chi Minh City University of Technology
Chair of Advisory Committee: Dr. Mahmoud M. El-Halwagi

The pooling problem is an important optimization problem that is encountered in operation and scheduling of important industrial processes within petroleum refineries. The key objective of pooling is to mix various intermediate products to achieve desired properties and quantities of products. First, intermediate streams from various processing units are mixed and stored in intermediate tanks referred to as *pools*. The stored streams in pools are subsequently allowed to mix to meet varying market demands. While these pools enhance the operational flexibility of the process, they complicate the decision-making process needed for optimization. The problem to find the least costly mixing recipe from intermediate streams to pools and then from pools to sale products is referred to as the *pooling* problem. The research objective is to contribute an approach to solve this problem.

The pooling problem can be formulated as an optimization program whose objective is to minimize cost or maximize profit while determining the optimal allocation of intermediate streams to pools and the blending of pools to final products. Because of the presence of bilinear terms, the resulting formulation is nonconvex which makes it very difficult to attain the global solution. Consequently, there is a need to develop computationally-efficient and easy-to-implement global-optimization techniques to solve the pooling problem. In this work, a new approach is introduced for the global optimization of pooling problems. The approach is based on three concepts: linearization by discretizing nonlinear variables, pre-processing using implicit enumeration of the discretization to form a convex-hull which limits the size of the search space, and application of integer cuts to ensure compatibility between the original problem and the discretized formulation. The continuous quality variables contributing to bilinear terms are first discretized. The discretized problem is a mixed integer linear program (MILP) and can be globally solved in a computationally effective manner using branch and bound method. The merits of the proposed approach are illustrated by solving test case studies from literature and comparison with published results.

ACKNOWLEDGMENTS

I am grateful to my remarkable academic advisor, Dr. Mahmoud El-Halwagi, for his great support, encouragement and introduction to the fantastic world of process integration and optimization. I have received his warm help not only in this thesis work but also in the whole program of my master's degree.

I am thankful to Dr. Juergen Hahn and Dr. Guy L. Curry, the research committee members. The knowledge that I have learned from them is so valuable. Also, Dr. Carl Laird's help is priceless. I am indebted to him for a lot of time and insightful discussions in optimization.

I would like to thank the students in the Process Integration and Systems Optimization group as well as some other fellow students in the Department of Chemical Engineering for a friendly academic environment.

The financial support from the sponsor – the Ministry of Education and Training, Vietnam – is greatly appreciated.

Finally, this thesis could not have been accomplished without the unconditional love and care from my parents, sister and friend.

NOMENCLATURE

Symbols	**Definition**
Subscripts	
i	sources
j	pools
k	products
q	quality
r	integer index
u	integer index
v	integer index
Parameters	
a_i	source i quality
c_k	product k quality
C_i	cost of source i
D_k	product k demand
l	number of sources
L_j	lower bound of pool j capacity
m	maximum number of used pools
n	number of products
N_q	number of investigated qualities
N_p	number of discretized pools
P_k	product k price
S_i	available capacity of source i
t	number of intervals for a discretized range
U_j	upper bound of pool j capacity
Z_j	total flow through pool j

Variables

b_{jq}	quality q of pool j
f_j	binary variable associate to pool j
x_{ij}	fractional flow rate (or fractional amount) from source i to pool j
X_{ij}	flow rate (or amount) from source i to pool j
Y_{jk}	flow rate (or amount) from pool j to product k

Abbreviations

BARON	Branch-And-Reduce Optimization Navigator (software)
GBD	Generalized Bender's Decomposition
GOP	Global Optimization (algorithm)
GRG	Generalized Reduced Gradient
LP	Linear Program
MILP	Mixed Integer Linear Program
NLP	Non-Linear Program
RLT	Reformulation-Linearization Technique
SLP	Successive Linear Program

LINGO code

@FOR	A command is executed for a range of indicated index
@SUM	Summation over a set
#LT#	Less than
#LE#	Less than or equal
#GT#	Greater than
#GE#	Greater than or equal
MAX	Maximized objective function

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

CHAPTER I

INTRODUCTION

Market demands for qualities and quantities of products may require the blending of several process streams to meet the desired requirements. For instance, intermediate streams from various processing units of a petroleum refinery are typically *blended* to produce value-added products satisfying quality specifications and demands. As an example, intermediate streams from reforming, cracking, and naphtha treatment units are typically mixed to yield gasoline. Quality specifications such as octane number, vapor pressure, and sulfur and aromatic concentrations are among the decisive stipulated quality constraints for gasoline. Prior to being blended and sent to final storages, intermediate streams are mixed and stored in intermediate tanks, called *pools*. Such pools enhance the operational flexibility of the process but complicate the decision-making process needed for optimization. The problem to find the least costly mixing recipe from intermediate streams to pools and then from pools to sale products is a pooling problem. The research objective is to contribute an approach to solve this problem.

The following general pooling problem (Figure 1) is investigated. Given is a set SOURCES = $\{i|i = 1,\ldots,l\}$ of intermediate streams. Each source has a given available capacity, S_i, a unit cost, C_i, and known values of N_q characterizing qualities, a_{iq}, where q is an index for qualities (e.g. octane number, Reid vapor pressure, and sulfur concentration). The amount (or flow rate) from source i to pool j is denoted by a variable X_{ij}. The sources have to be blended because there is not enough pools ($m < n$) and/or there are insufficient pool capacities to store sources and/or products separately. Sources can be sent to some or all of pools.

This thesis follows the style of *AIChE Journal.*

As a result of blending the sources, each pool j has unknown values of qualities b_{jq}. The amount from pool j to sale product is Y_{jk} and is to be determined. The n products each with a price P_k have constraints on known demand D_k and bounds on the values of desired quality specifications c_{kq}.

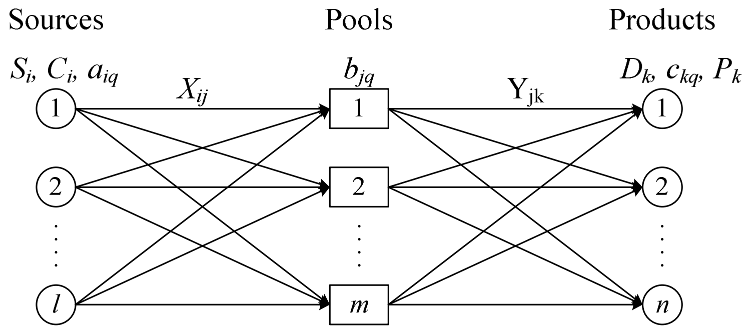


Figure 1 General pooling problem

With these notations, the problem is formulated as follows:

Objective function: margin return is maximized = $\sum_{k=1}^{n} P_k \cdot \sum_{j=1}^{m} Y_{jk} - \sum_{i=1}^{l} C_i \sum_{j=1}^{m} X_{ij}$

Subject to the following constraints:

Available supply: $\sum_{j=1}^{m} X_{ij} \le S_i$ for $i = 1,\dots,l$

Mass balance on pools: $\sum_{i=1}^{l} X_{ij} = \sum_{k=1}^{n} Y_{jk}$ for $j = 1,\dots,m$ (P)

Product demand: $\sum_{j=1}^{m} Y_{jk} \le D_k$ for $k = 1,\dots,n$

Mixing rule for pools: $b_{jq} \cdot \sum_{i=1}^{l} X_{ij} = \sum_{i=1}^{l} X_{ij} \cdot a_{iq}$ for $j = 1,\dots,m$ and $q=1,\dots,N_q$

Mixing rule for products (assuming that the desired quality is an upper bound. Similar constraints may be written for lower bounds):

$$c_{kq} \cdot \sum_{j=1}^{m} Y_{jk} \ge \sum_{j=1}^{m} Y_{jk} \cdot b_{jq} \text{ for } k = 1,\dots,n \text{ and } q=1,\dots,N_q$$

Non-negativity constraints: $X_{ij} \geq 0$, $Y_{jk} \geq 0$ for $i = 1,\ldots,l$

$j = 1,\ldots,m$

$k = 1,\ldots,n.$

The foregoing formulation is nonconvex because of the bilinear terms appearing in the mixing rule constraints $b_{jq} \cdot \sum_{i=1}^{l} X_{ij}$ and $\sum_{j=1}^{m} Y_{jk} \cdot b_{jq}$. Therefore, it is desired to develop a global solution procedure for this formulation.

Other than the foregoing formulation, there have been some formulations with different variable definitions. But they all are also nonconvex optimization problems.

In addition to this structure of pooling problems, there are some implementations that make pooling problems more complicated to be solved. Allowing pools to interconnect with other pools is one of the implementations. A simple example is the problem proposed by Audet. [1] The structure is shown in Figure 2 where there may be a flow from the first pool to the second pool. A more generalized pooling problem is the superstructure introduced in Meyer and Floudas [2] in which each of pools is free to connect to others as well as all products, and each of sources may feed all pools and directly all products. This superstructure also represents a network of waste water treatment minimizing the total cost.

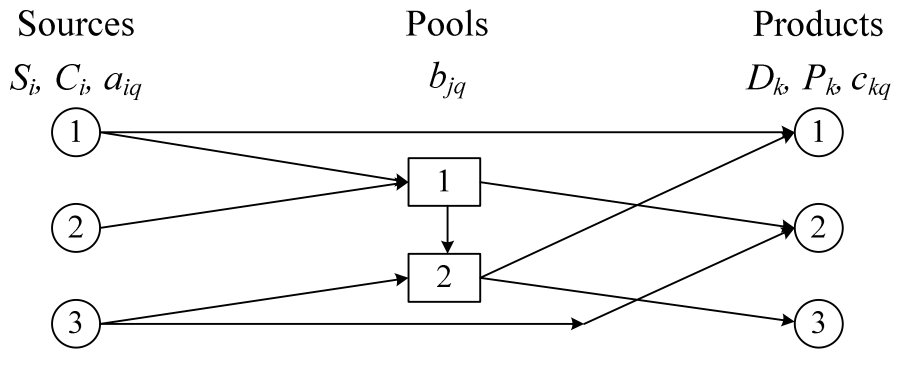


Figure 2 A variation of the pooling problem (Audet [1])

This work will focus on the pooling problem represented by Figure 1. The layout of this thesis is as follows. Chapter II is a survey on the previous literature attempts to solve the pooling problems. Chapter III describes the proposed approach to discretize and reformulate pooling problems, compares the advantages and disadvantages, and distinguishes this work from a recent discretization idea. Chapter IV presents the proposed methodology and discusses its conceptual and mathematical aspects. Chapter V discusses pooling problems previously published in literature. The results and discussion of how the proposed approach performs in solving these problems are presented in Chapter VI. Finally, the last chapter provides conclusions and recommendations for future work.

CHAPTER II

LITERATURE REVIEW

Much interest has been given to the solution of pooling problems because of its attractive pay-off in operating refineries and other process industries. The following is a survey of key publications on the statement, formulation, and solution of the pooling problem.

2.1 The initial approaches

One of the earliest solution procedures is the recursion approach to the simple pooling problems introduced by Haverly.[3] This approach reformulates the problem by introducing two additional sets of variables, say *over* and *under*, for pool quality ranges in each recursion iteration and another set of variables, say y_{jk}, representing flow rate fraction from pools to sale products. These combination variables, $y_{jk}.(over - under)$, are added to the mixing rule constraints. Then, the recursion begins with assigned or guessed values of quality variables b_{jq} and y_{jk} in the bilinear term to have a linear formulation. Solving this linear program, the author obtained the flow rate optimum X_{ij}, Y_{jk} and the calculated actual b_{jq}. The values of calculated *over* and *under* give a direction for the next iteration. The recursion is repeated until the assigned and calculated values of b_{jq} converge within acceptable tolerance. Because of depending on these starting points, the recursion procedure may not converge or may converge to a local optimum. When the problem is large, this method is likely unstable and takes much computational time.

Lasdon et al. [4] utilized generalized reduced gradient (GRG) and successive linear programming (SLP) algorithms to solve pooling problems. The SLP algorithm eliminates the bilinear relations by first order Taylor series expansion. The iteration starts with assigned flow rate values and then followed by a sequence of linear programs. The steps are summarized in Figure 3.

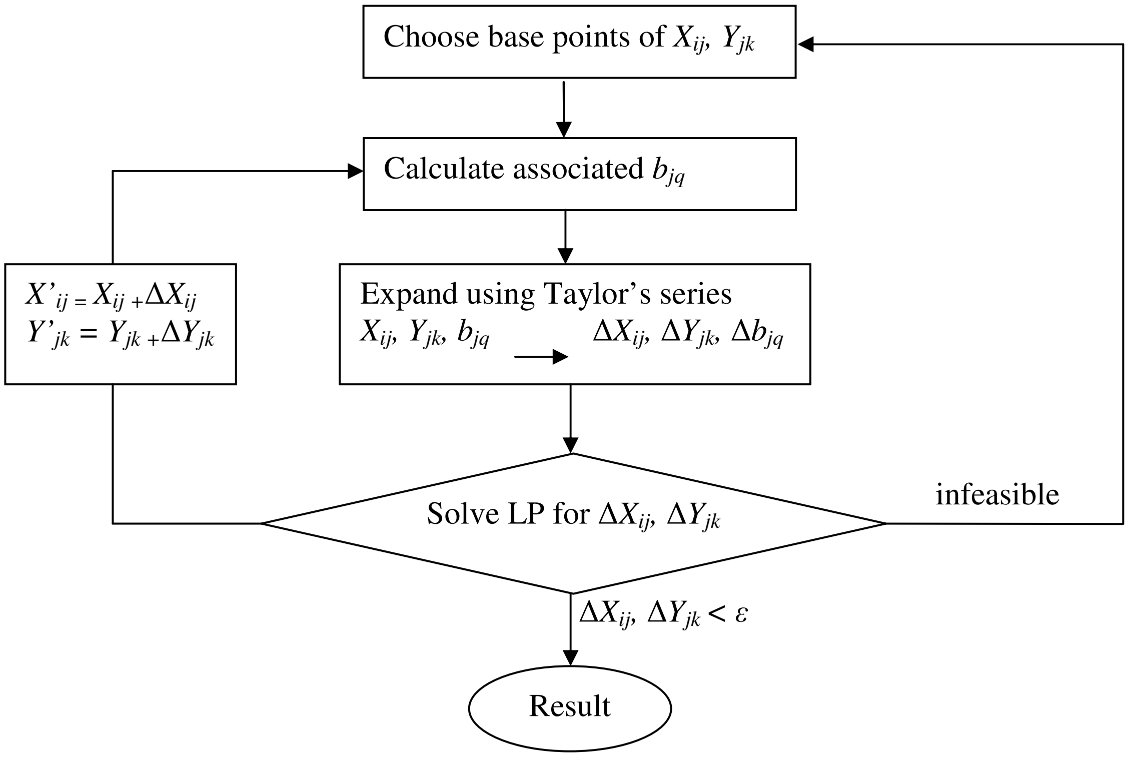

Choose base points of X_{ij}, Y_{jk}
Calculate associated b_{jq}
$X'_{ij} = X_{ij} + \Delta X_{ij}$
$Y'_{jk} = Y_{jk} + \Delta Y_{jk}$
Expand using Taylor's series
$X_{ij}, Y_{jk}, b_{jq} \longrightarrow \Delta X_{ij}, \Delta Y_{jk}, \Delta b_{jq}$
Solve LP for ΔX_{ij}, ΔY_{jk}
infeasible
$\Delta X_{ij}, \Delta Y_{jk} < \varepsilon$
Result

Figure 3 Successive linear programming algorithm (Lasdon et al. [4])

First of all, a feasible flow rate set of X_{ij}*'s* and Y_{jk}'s is chosen no matter what values of $b_{j.q}$'s are. From these initial flow rates, $b_{j.q}$'s are calculated. By using first order Taylor's series for bilinear terms, the formulation is transformed into a linear program with sets of variables ΔX_{ij}, ΔY_{jk}, Δb_{jq}. The solved optimum ΔX_{ij}, ΔY_{jk} will decide a termination of the iterations. If none of the optima is found, the iteration starts over with a new chosen set of base points. If an optimum ΔX_{ij}, ΔY_{jk} is found, the iteration repeats with an updated set of base points where $X'_{ij} = X_{ij} + \Delta X_{ij}$ and $Y'_{jk} = Y_{jk} + \Delta Y_{jk}$. The calculation is terminated when the solved optimum ΔX_{ij}, ΔY_{jk} is within an acceptable tolerance ε.

Lasdon et al. [4] performed the application of these two nonlinear programming algorithms to Haverly's pooling problems. The problems were solved with various initial points. The results showed some advantages from these algorithms over recursion. The convergence speeded up and leaded to an optimum when the starting points made the algorithms move out of trivial solutions. The formulation needs not much effort to suit the algorithm as recursion approach does.

More information on these algorithms can be found in Griffith and Stewart, [5] Palacios Gomez et al., [6] Baker and Lasdon, [7] and Greenberg. [8]

Griffith and Stewart [5] are the first to introduce SLP under the name Mathematical Approximation Program and apply in Shell Oil.

Palacios Gomez et al. [6] proposed an efficient SLP algorithm for linearly constrained formulation and showed more successful computational results than the generalized reduced gradient algorithm did, especially in large problems with low degrees of freedom.

Baker and Lasdon [7] suggested a multiplicative formulation for the linearized subproblems to be solved by SLP, with nonnegative deviation variables to prevent the occurrence of infeasibility, and applied this idea for nonlinear optimization problems in Exxon. The multiplicative form of formulation usually leads to a fewer number of nonlinear variables than the additive form does. It also derives a linearized problem compatible with existing LP formulations.

Greenberg [8] used quality diagram to analyze sensitivity and diagnose infeasibility of pooling problems. This geometry approach visualizes the range of percentage flow rates, pool qualities and the range of source qualities for the problem to be feasible. The cost parameter was analyzed as another attribute of streams. From this viewpoint, the author discovered that the calculated costs of pools and products were the Lagrangian multipliers which were associated with the constraints on product demands and mass balances for pools.

2.2 Decomposition approaches

Decomposition is another approach to solve pooling problems. The idea is to decompose the problem into two linear subproblems by fixing a variable in the bilinear terms. At each iteration, these subproblems are solved for their respective global optimums and the iterations continue until stopping conditions are satisfied. The idea is clarified in the following mathematical formulations (see Floudas and Aggarwal [9] for more details.)

Consider the optimization problem with two sets of decision variables x and y:

$$\begin{aligned} &\min_{x,y} f(x,y) \\ &\text{subject to} \quad g(x,y) \le 0 \\ &\qquad\qquad\quad\ h(x,y) = 0 \end{aligned}$$

Between these sets of variables, the one causing nonlinearity (let say y) is referred as complicating variables and the other (x) is called the set of non-complicating variables. This discrimination must be performed before the optimization problem is decomposed into two linear subproblems. The first subproblem or the primal problem is derived from fixing values of complicating variables and mathematically stated as follows.

$$\begin{aligned} &\min_{x} f(x,\underline{y}) \\ &\text{subject to} \quad g(x,\underline{y}) \le 0 \\ &\qquad\qquad\quad\ h(x,\underline{y}) = 0 \end{aligned}$$

where $\underline{y}$ is considered as parameters.

By fixing some variables, more constraints have been added to the original optimization problem. Thus, the primal problem gives an upper bound for the minimization problem. Besides, it also gives the associated optimum values of x (say x^*) and the Lagrangian multipliers u^* and λ^* for the constraints $g(x,\underline{y}) \le 0$ and $h(x,\underline{y}) = 0$ respectively.

The second subproblem is the dual problem, referred as a relaxed master problem:

$$\min_{y} \mu$$

subject to: $f(x^*,y) + u^*.g(x^*,y) + \lambda^*.h(x^*,y) \leq \mu$

where the left hand side of the constraint is the Lagrangian function of the primal problem, denoted as $L(x^*,y,u^*,\lambda^*)$. The objective solution of the relaxed master problem is the lower bound on the global optimum of the original problem. The optimum set of values y is used as the fixed values for the primal problem in the next iteration. Generalized Bender's Decomposition (GBD) is one of the methods to interact these subproblems.

Floudas et al. [10] proposed a decomposition-based global optimization approach for nonlinear programs (NLPs) and mix integer nonlinear programs (MINLPs); and then (Floudas and Agrawal [9]) applied it and GBD method to the pooling problems with 3 pools, 5 products and 2 qualities. In their proposed decomposition algorithm, a constraint consisting of an updated Lagrangian function was added at each iteration. If the solution of a relaxed master problem resulted in infeasibility in the next primal problem, a positive slack variable was added to each of the constraints in that primal problem. The primal problem was then relaxed and transformed to a minimization problem of the slack variable which represents the infeasibility as follows:

$$\min_{x} \alpha$$

$$\begin{aligned}\text{subject to}\quad & g(x,\underline{y}) - \alpha \leq 0\\ & h(x,\underline{y}) - \alpha \leq 0\\ & -h(x,\underline{y}) - \alpha \leq 0\\ & \alpha \geq 0\end{aligned}$$

The objective of the proposed algorithm is to structure the subproblems in such a manner that they are solved for respective global optima at every iteration. However, there is no guarantee that a found solution at the end of the algorithm is the global optimum of the original optimization problem.

Androulakis et al. [11] proposed a distributed implementation of global optimization algorithm (GOP). The GOP approach bases on decomposition and duality theory. The algorithm is almost the same to above, except for some implementations in relaxed dual problem solving steps. At each iteration, a series of relaxed dual problems are to be solved. These problems are in forms of:

$$
\begin{aligned}
&\min_{y,\mu_B} \ \mu_B \\
&\text{subject to:} \quad L(x^{B_j}, y, u^*, \lambda^*) \le \mu_B \\
&\qquad \nabla_{x_i} L(x^{B_j}, y, u^*, \lambda^*) \le 0, x_i^{B_j} = x_i^U \\
&\qquad \nabla_{x_i} L(x^{B_j}, y, u^*, \lambda^*) \ge 0, x_i^{B_j} = x_i^L
\end{aligned}
$$

where the two additional families of constraints (the last two ones) are the gradients of Lagrangian function and referred as qualifying constraints. Each of relaxed dual problems associates with a combination B_j of the bounds of variable *x,* instead of x^*, from previous primal problem's solution. Minimum μ_B is chosen from all solutions of these serial dual problems and compared to the lower bound in previous iteration for an updated lower bound. The GOP algorithm converges to an ε-global solution but requires a large number of calculations for solving dual problem series.

Androulakis et al. [11] discussed some implementations to tackle this computational bottleneck. Following the primal problem, variable bound problems were formulated and simultaneously solved for tighter bounds on variables. As a result, the number of relaxed dual problems was decreased; then, the overall convergence was more rapidly. Solving relax dual problems were also performed in parallel manner. The procedure to find the minimum among dual problems' solutions compared as many pairs as possible at the same time to reduce computational time. In the broader viewpoint, the idea of these distributed implementations is to do many tasks simultaneously by taking advantage of the multiprocessor computer Intel-Paragon. Several randomly generated large scale pooling problems were solved without published input data to demonstrate the improvements.

2.3 Branch and bound framework

Branch-and-bound method has also been a base method for many proposed approaches to solve pooling problem. For more details on branch and bound approach, see section 4 in Chapter IV.

Foulds et al. [12] partitioned the feasible regions into two equal parts to produce two branching sub-problems. These sub-problems were then bounded by convex and concave envelops, using the linear relaxation technique of McCormick [13] and Al-Khayyal and Falk. [14] From the found optimum solution in previous step, the algorithm partitioned the rectangle of feasible region into four sub-rectangles, linearized these subproblems by convex underestimation and solved for global optima. The sub-rectangle associated with the best solution among them (the highest in maximum problem) was chosen to be the base point for next partition step. The algorithm continued with smaller and smaller sub-rectangles and was proven to converge to optimal solution by Al-Khayyal and Falk. [14] The solution may be arbitrarily close to the global optimum. The proposed algorithm was demonstrated on some generated pooling problems with one quality (see page 44 for problems' descriptions).

Sherali and Alameddine [15] proposed a reformulation – linearization technique to solve bilinear programming problems in which the bilinear terms only appeared in objective functions. This technique consists of two stages as it's named. In the first stage, the problem is reformulated by generating additional valid nonlinear constraints from pair-wise multiplications between the original problem constraints and nonnegative variable factors derived by rearranging variables' bounds. In the second stage, the resulting formulation is linearized by substituting the nonlinear terms with new defined variables. This relaxed formulation's optimum is an upper bound (in maximized optimization problems) of the original bilinear programming problems. This bound is proven to be tighter than that of convex hyper-rectangle envelop overestimation. The authors showed that the RLT procedure generates convex envelop representation of a bilinear function over special triangular and quadrilateral poly-topes in R^2.

Ben-Tal et al. [16] partitioned the fractional flow rate variables in the proportion formulation in order to reduce the duality gap between the primal and its Lagrangian dual problem until less than a predetermined small ε. The algorithm took advantages of branch and bound approach and duality properties. The algorithm starts with a feasible solution of primal problem (an upper bound); then, the dual problem is solved for a lower bound. If the difference between these two bound is still more than ε, the fractional flow rate region is partitioned into many sub-polytopes. Each dual problem associating with each sub-polytope is solved and compared to find the sub-polytope providing a minimum dual bound. The primal problem is resolved using a local search from that optimum sub-polytope. At the end of the iteration, these two updated bounds are checked with the stopping condition. Numerical examples on pooling problems with almost two qualities were presented

Quesada and Grossmann [17] also used branch and bound search to solve their reformulated models of general process networks which consists of splitters, mixers and linear process units. This problem structure may be simplified to represent simple pooling problems. When linearized using the reformulation and linearization technique (RLT) of Sherali and Alameddine, [15] some mass balance constraints in individual component flow formulation were transformed into those in composition model, and versa. The linearized model, which combines the variables from both formulations, was embedded in a branch and bound frame work to be solved for a global optimum which is a tighter lower bound than previous relaxation approaches. The approach's limitations are solving linear process unit, ignoring enthalpy effect and not allowing binary variables.

Branch and bound procedure using selective Lagrangian relaxation proposed by Adhya et al. [18] produces a tighter lower bound than McCormick estimator-based linearization relaxation. The term "selective" refers to a manner that the choice of relaxed constraints results in a Lagrangian subproblem which may not be easier to solve. Instead of dualizing only bilinear constraints (the "hard" constraints,) the authors dualized all of the constraints to obtain a Lagrangian subproblem with a bilinear objective function defined over a hypercube. Then, a reformulation procedure was proposed to solve this not-easy subproblem. This reformulated model is a mixed integer program and is embedded in a branch and bound algorithm to obtain a local optimum that is the lower bound of the original pooling problem.

Audet et al. [1] formulated pooling problems on three models which are based on flow variables, flow proportion variables and their hybrid ones. This proposed formulation was shown to be suitable for branch and cut quadratic algorithm introduced by Audet et al. [19] The branch and cut algorithm takes some advantages of both branch-and-bound algorithm and reformulation-linearization technique to have some improvements. Firstly, branching by partitioning hyper-rectangle is not necessarily at the middle lines to reduce potential errors. Secondly, instead of adding all bound factors and constraint factors, only those bilinear terms which are violated are linearized and contribute to relaxed problem's constraints. As a result, the problem size does not increase quickly. Thirdly, the linear variables are closer to their respective bilinear terms by introducing cuts on the convex paraboloid. Every cut is valid for the whole nodes of the branch tree.

Meyer and Floudas [2] proposed a piecewise linear reformulation based on the RLT technique to solve the superstructure model of generalized pooling problems. Before the RLT is applied, the continuous space of each quality is partitioned into many subintervals. Some binary variables are introduced to indicate which interval includes the optimum quality. Then RLT is used with a note that the bilinear terms stay in the mixing rule constraints for pooling problems instead of objective function in the problem investigated by Sherali and Alameddine. [15] These constraints are excluded from the reformulation step but included in the linearization step according to the approach of Meyer and Floudas. [2] The introduction of binary variables to the formulation augments the lower bound. Therefore, the calculation time is less despite the addition of variables. However, it does not produce an upper bound. The authors verified the ε-global optimum by doing a run series in which the quality partition scheme is restructured after each run. In a large scale industrial problem, the approach can reduce the gap between the lower and upper bound to 1.2%. The upper bound was found by using DICOPT. More discussion on this paper is presented in section 3.6.

Sahinidis [20] reviewed the theory and algorithms of the branch and reduce approach for the global optimization of NLPs and MINLPs, which has evolved from the traditional branch and bound approach. Two steps were implemented. The preprocessing step before relaxation is to reduce the range of all problem variables. After relaxed problem is solved, the post-processing step utilizes the solution to further reduce the problem variable ranges prior to next branching iteration. As the ranges are reduced, the variable bounds are tighter; therefore, this implemented branch and bound converges faster. The approach has been developed, integrated in the computational system BARON, and applied in various engineering problems

For an overview of published approaches, some of their characteristics are summarized in Table 1. Basically, the approaches are categorized into three groups: local optimum, lower bound and ε-global optimum.

This research proposes discretization approach which produces global optimums or near global optimum results (in practically acceptable manner) and in computationally realistic time aided by existing computer capacity. The procedure is to finitely discretize the quality variables in the bilinear terms (flow variables multiplied by quality variables) exhaustively or implicitly to obtain a mix integer linear programming (MILP) formulation. This formulation is easy to be programmed in commercial programming software, e.g. LINGO, for its global optimum. The remaining of this thesis describes discretization approach and its applications, which are the main research contributions.

Table 1 Summary of approaches on pooling problems

Publication	Base approach	Optimality	Alignment to the global minimum	Quality	Implementation
Haverly [3]	Recursion	Local	Local optimum	Single	
Griffith and Stewart [5]	SLP	Local	Local optimum	N/A	
Palacios Gomez et al. [6]	SLP	Local	Local optimum	N/A	
Baker and Lasdon [7]	SLP	Local	Local optimum	Single	Multiplicative formulation
Lasdon et al. [4]	GRG and SLP	Local	Local optimum	Single	
Greenberg [8]	Geometry	N/A	N/A	Multiple	Sensitivity analysis and infeasibility diagnosis
Floudas et al. [10]	Decomposition	Global	Local optimum	Single	Generalized Bender's Decomposition
Androulakis et al. [11]	Decomposition	Global	Lower bound	Multiple	Distributed implementations of GOP algorithm
Foulds et al. [12]	Branch and bound	Global	Lower bound	Single	Convex envelop relaxation
Sherali and Alameddine [15]	Branch and bound	Global	Lower bound	N/A	RLT
Ben-Tal et al. [16]	Branch and bound	Global	ε-global minimum	Multiple	Partition fractional flow variable
Quesada and Grossmann [17]	Branch and bound	Global	ε-global minimum	Single	RLT
Adhya et al. 18	Branch and bound	Local	Lower bound	Multiple	Selective Lagrangian relaxation
Sahinidis [20]	Branch and bound	Global	Lower bound	Multiple	Reduce variable range
Audet et al. [1]	Branch and cut	Global	Lower bound	Multiple	Formulation, branching and cut
Meyer and Floudas [2]	Branch and bound	Global	Lower bound	Multiple	Piecewise RLT

CHAPTER III

NEW CONCEPTS OF PROPOSED DISCRETIZATION APPROACH

This Chapter introduces the key concepts to be used in the new solution procedure of the pooling problems. The proposed global optimization approach is based on three concepts:

1.) Discretization of qualities for each pool: The characterizing qualities for each pool, b_{jq}, are unknown. Let us discretize the search space of the qualities of the pools into a set of N_p vectors of known values: $\{(b_{j,1}\ b_{j,2},\ \ldots,\ b_{j,Nq})|j=1,\ldots,N_p)$. The rationale for the selection of these values will be discussed later. It is also worth noting that if N_p is large enough, the discretized space can approximate the original continuous search space. The discretization of the unknown pool qualities into known values transforms the formulation into a linear program. However, there is a potential violation of the given number of pools. Since the actual number of pools must be limited to m, $N_p - m$ should not be selected in the final solution. To overcome this challenge, the following step is introduced.

2.) Application of Integer Cuts for the Pools: In order to limit the number of pools to m, an integer cut is used to select m pools from among the N_p discretized pools. The details will be given later. Consequently, the formulation becomes a mixed-integer linear program "MILP" that can be globally solved using branch and bound method.

3.) Convex Hull Search: A potential problem with the large number of discretizations for several qualities is the large size of the resulting MILP. In order to reduce the problem dimensionality, a convex hull is constructed by invoking physical limits on the possible combinations of pool qualities.

The following sections provide more details on the concepts, rationale, and implementation of the above-mentioned steps.

3.1 Motivating example

Consider an example of the pooling problem adapted from Greenberg [8] and summarized in Figure 4, Table 2 and Table 3.

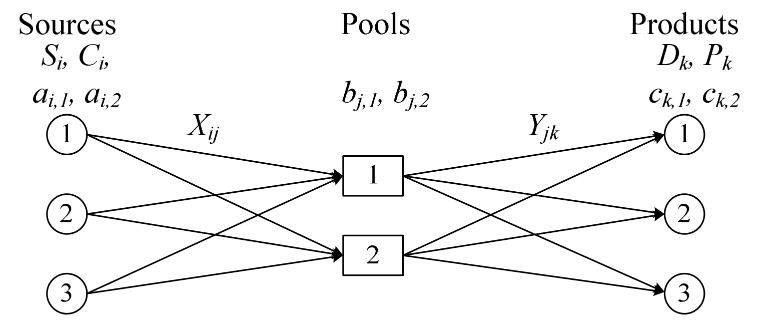


Figure 4 An example of pooling problem adapted from Greenberg [8]

Table 2 Source information for the example

Source	1	2	3
Supply limit (S_i)	100	200	100
RON (a_{i1})	82	92	82
Sulfur content (a_{i2})	1	2	1.5
Cost (C_i)	7	9	6

Table 3 Product constraints for the example

Product	1	2	3
Demand (D_k)	100	100	200
RON (min) (c_{k1})	84	87	90
Sulfur content (max) (c_{k2})	1.9	2	2
Price (P_k)	10	15	17

This example will be used to demonstrate the proposed approach and implementation throughout the following sections.

3.2 Discretization by exhaustively enumerating b_j's

The pooling problem (P) is linearized by listing the values of one of the two variables in the bilinear terms $b_{jq}.Y_{jk}$ and $b_{jq}.X_{jk}$. This work proposes the use of quality as the discretized variable.

The quality of any pool is bounded by qualities of the blended sources. Using the following notation: $a_{q,min} = \arg\min_i \{a_{iq}\}$ and $a_{q,max} = \arg\max_i \{a_{iq}\}$, then the domain of b_{jq} is $a_{q,min} \leq b_{jq} \leq a_{q,max}$.

This discretization approach limits the search space to pools whose values are bounded between $a_{q,min}$ and $a_{q,max}$. When the quality range is discretized into known values, the bilinear terms become linear which is conducive to the global solution of the problem.

The quality range may be discretized in various ways (e.g., random, structured). One way of discretizing the range of quality q is to divide it into t_q equal intervals. In such cases, the values of discretized quality q are calculated through the following expression:

$$b_{qr} = a_{q,\min} + (r_q - 1) \cdot \frac{a_{q,\max} - a_{q,\min}}{t_q} \quad \text{for } r_q = 1,\ldots,(t_q+1) \qquad (1)$$

When $t_q \to \infty$, the solution of the discretized and the original formulations become equivalent. However, for all practical purposes, there is a large enough number of discretizations that strikes the right balance between computational time and proximity of the discretized solution to the original solution.

For N_q qualities, let us denote the number of interval discretizations for qualities $1,2,\ldots,N_q$ by $t_1, t_2,\ldots,t_{Nq}$, respectively. Therefore, the total number of discretized pools is $(t_1 + 1).(t_2 + 1)\ldots(t_{Nq} + 1)$. If $t_1 = t_2 = \ldots = t_{Nq} = t$, there are $(t+1)^{Nq}$ pools.

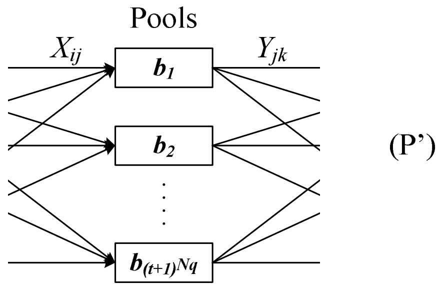


Figure 5 Pools are enumerated on their assigned quality values

Figure 5 shows $(t+1)^{Nq}$ pools, where $b_j = \{b_{q,r} \mid q = 1,\ldots,N_q$ and $r = 1,\ldots,(t+1)\}$. This number of pools $(t+1)^{Nq}$ is independent of the number of pools in the original problem m. Instead, t only depends on how small the quality increment is selected.

At this point, the problem is to blend l sources into $(t+1)^{Nq}$ pools with known qualities. It is a linear optimization formulation and a global result can be obtained. Pools which have zero flow rate calculated are not used. It is possible that the resulting number of used pools may exceed the allowable number of pools (m). To resolve this issue, a 0/1 binary integer variable f_j is introduced. If the pool j is used in the solution of (P'), then f_j is assigned a value of 1, otherwise f_j is 0. This constraint may be formulated using the following linear constraint:

$$L_j.f_j \le \sum_{i=1}^{l} X_{ij} \le U_j.f_j \quad \text{for } j = 1,\ldots,(t+1)^{Nq}.$$

where L_j and U_j are given positive real numbers that correspond to the minimum and maximum capacities of the pools. If the original problem (P) has no constraint on pool capacity, U_j should be at least the largest available supply of total sources. From these two inequalities, f_j is forced to be 0 when $\sum_{i=1}^{l} X_{ij} = 0$, i.e. no flow rate to discretized pool j, or 1 when $\sum_{i=1}^{l} X_{ij} > 0$. As a characteristic of pooling problems, $\sum_{i=1}^{l} X_{ij}$ tends to be positive to contribute to the objective function. Hence, the inequality $f_j \le \sum_{i=1}^{l} X_{ij}$ can be removed without affecting the solutions.

Next, the restriction on the number of pools is taken into formulation simply as $\sum_{j=1}^{(t+1)^{Nq}} f_j \le m$. For simplicity in notation, let us denote $(t+1)^{Nq}$ by M. Therefore, reformulation is given by:

Objective function: $\sum_{k=1}^{n} P_k \cdot \sum_{j=1}^{M} Y_{jk} - \sum_{i=1}^{l} C_i \sum_{j=1}^{M} X_{ij}$ is maximized

Available supply: $\sum_{j=1}^{M} X_{ij} \le S_i$ for $i = 1,\ldots,l$

Mass balance on pools: $\sum_{i=1}^{l} X_{ij} = \sum_{k=1}^{n} Y_{jk}$ for $j = 1,\ldots,M$

Product demand: $\sum_{j=1}^{M} Y_{jk} \le D_k$ for $k = 1,\ldots,n$ (P')

Mixing rule for pools: $b_{jq} \cdot \sum_{i=1}^{l} X_{ij} = \sum_{i=1}^{l} X_{ij} \cdot a_{iq}$ for $j = 1,\ldots,M$

Number of pools: $f_j = \{0, 1\}$ for $j = 1,\ldots,M$

$\sum_{i=1}^{l} X_{ij} \le U_j.f_j$ for $j = 1,\ldots,M$

$\sum_{j=1}^{M} f_j \le m$

Mixing rule for products: $c_{kq} \cdot \sum_{i=1}^{l} Y_{jk} \ge \sum_{i=1}^{l} Y_{jk} \cdot b_{jq}$ for $k = 1,\ldots,n$

Positive variables: $X_{ij} \ge 0$, $Y_{jk} \ge 0$ for $i = 1,\ldots,l$

$j = 1,\ldots,m$

$k = 1,\ldots,n$

where b_{jq}'s are known parameters defined by (1).

This formulation is a mixed integer linear program (MILP). The main drawback of this approach is the potential large size integer of the MILP. The number of pools has increased from *m* to *M*, resulting in (*M* -*m*) more variables for b_j, *M* variables of f_j and *l*.(*M* -*m*) more variables for each set of X_{ij} and Y_{jk}. It also increases rapidly when the increment decreases. Consequently, the selection of the increment size is critically important for the problem dimensionality and computational efficiency of the devised discretization approach. The following section is a discussion on the selection of the discretization scheme.

Exhaustive enumeration approach to the discretized space

Let us start by a pooling problem involving two qualities. A graphical technique is proposed to reduce these variable amounts. This technique is discussed using the following example. Consider a pooling problem with research octane number (RON) and sulfur content as the two primary qualities. If the search space for qualities is discretized in an exhaustive enumeration manner, then the number of discretized pools will correspond to the number of all possible quality combinations. The range of RON is [82, 92] and for the sulfur content (expressed as %sulfur) is [1, 2]. The number of RON intervals is taken to be 40 and the number of the sulfur content is also taken to be 40. Therefore, the increments of 0.25 for RON and 0.025 for sulfur content percentage are used. hence, the discretized set of RON {82.00; 82.25; …; 92.00} has 41 components and that of sulfur concentration, {1; 1.025; 1.05; …; 1.975; 2} has 41 components. Using the exhaustive enumeration, we need 41x41 = 1,681 pools qualities of which are described in Table 4. Figure 6 is a schematic representation of the exhaustive enumeration of the discretized search space described by Table 4.

Table 4 Exhaustive enumeration approach to discretizing the motivating example

Pool index	Quality 1: RON (b_{j1})	Quality 2: Sulfur (b_{j2})
1	82.00	1.000
2	82.25	1.000
3	82.50	1.000
⋮	⋮	⋮
1680	91.75	2.000
1681	92.00	2.000

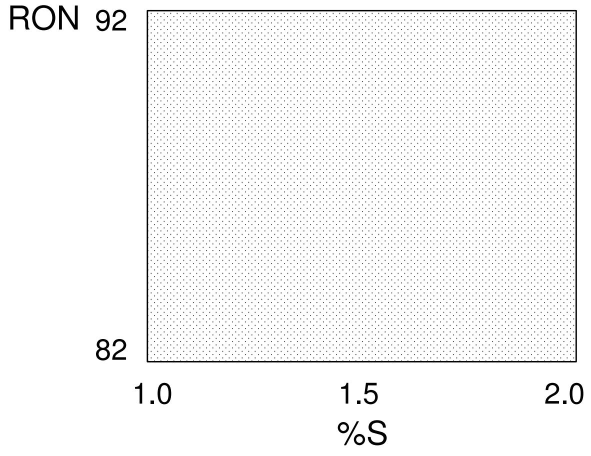


Figure 6 Exhaustive enumeration of a discretized search space for pool qualities

The problem formulation is given by:

$$\text{Maximize } \left(10\sum_{j=1}^{1681} Y_{j1} + 15\sum_{j=1}^{1681} Y_{j2} + 17\sum_{j=1}^{1681} Y_{j3}\right) - \left(7\sum_{j=1}^{1681} X_{1j} + 9\sum_{j=1}^{1681} X_{2j} + 6\sum_{j=1}^{1681} X_{3j}\right)$$

$$\text{Available supply: } \sum_{j=1}^{1681} X_{1j} \le 100;\ \sum_{j=1}^{1681} X_{2j} \le 200;\ \sum_{j=1}^{1681} X_{3j} \le 100$$

Mass balance on pools: $\sum_{i=1}^{3} X_{ij} = \sum_{k=1}^{3} Y_{jk}$ for j = 1,…,1,681

Product demand: $\sum_{j=1}^{1681} Y_{j1} \le 100$; $\sum_{j=1}^{1681} Y_{j2} \le 100$; $\sum_{j=1}^{1681} Y_{j3} \le 200$

Mixing rule for pools: RON $b_{j,1} \cdot \sum_{i=1}^{3} X_{ij} = \sum_{i=1}^{3} X_{ij} \cdot a_{i1}$ for j = 1,…,1,681

Sulfur $b_{j2} \cdot \sum_{i=1}^{3} X_{ij} = \sum_{i=1}^{3} X_{ij} \cdot a_{i2}$ for j = 1,…,1,681

The values of a_{i1} and a_{i2} are given in Table 2.

Number of used pools: $\sum_{i=1}^{3} X_{ij} \le 400 \cdot f_j$ for j = 1,…, 1,681

$$\sum_{j=1}^{1681} f_j \le 2$$

Mixing rule for products: RON $c_{k1} \cdot \sum_{j=1}^{1681} Y_{jk} \le \sum_{j=1}^{1681} Y_{jk} \cdot b_{j1}$ for k = 1,…,3

Sulfur $c_{k2} \cdot \sum_{j=1}^{1681} Y_{jk} \ge \sum_{j=1}^{1681} Y_{jk} \cdot b_{j2}$ for k = 1,…,3

where b_{j1} and b_{j2} are defined as in *Table 4*

The model is formulated in LINGO software (see Appendix B6). Calculation on Optiplex GX 620 personal computer gives the results that pool 116 and 689 are used with the qualities (83.25; 1.5) and (90; 1.8) respectively. A global optimum of 2,425 is found in a runtime of 1,235 seconds.

According to the foregoing exhaustive enumeration of the discretized qualities, 11,767 variables and 6,738 constraints are included in the formulation. The question is whether physical insights be used to reduce the number of variables and constraints. The answer is yes and will be detailed in the following section.

New discretization approach using the quality diagram (attainable regions)

Consider the q^{th} quality, for which the set of qualities for the sources is given by: Set_Quality_Sources$_q$ = {a_{1q}, a_{2q}, …,a_{iq},…,a_{lq}}. In order to reduce the size of the search space of the discretized qualities for the pool, it is proposed to use physical insights from mixing rules. The key idea is that the quality of any pool composed by mixing several sources will be enclosed in the convex hull constructed by the convex combination of the qualities of the individual sources. i.e.

$$b_{jq} \in H_{Convex} = \{ \sum_{i=1}^{l} x_{ij} \cdot a_{iq} \mid a_{iq} \in Set\_Quality\_Sources_q, x_{ij} = \frac{X_{ij}}{\sum_{i=1}^{l} X_{ij}} \}$$

To illustrate this concept graphically, consider the case of the three sources and two qualities (RON and sulfur content) whose data are given in Table 2. The three sources are represented by S_1, S_2, S_3 and the discretized pools are shown as dots. Figure 7 illustrates the convex hull (triangle) constructed from the three sources. This convex hull is referred to as the attainable region for the pool.

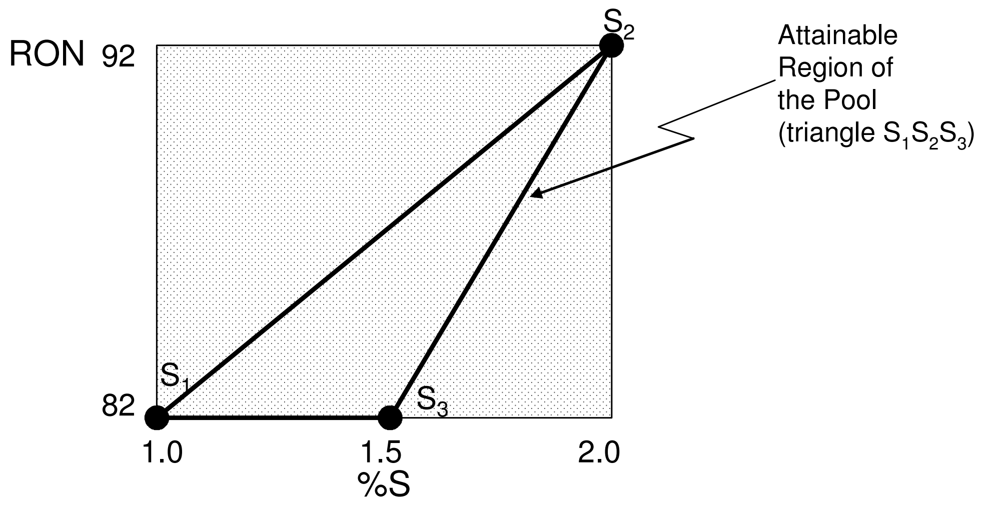


Figure 7 Attainable region (convex hull) of the pool qualities for two qualities

Therefore, for given qualities of sources S_1, S_2 and S_3 any possible blend will lie within the attainable region characterized by the triangle $S_1S_2S_3$. Indeed, it is unnecessary to search for the pool qualities outside this convex hull. The construction of the convex hull as the search space for the pool qualities leads to significant reduction in the size of the search space. In this example, the ratio of triangle $S_1S_2S_3$ area and the rectangular area for exhaustive enumeration is 0.25. In other words, equivalently, the convex hull includes 420 discretized pools compared to 1681 discretized pools in the case of exhaustive enumeration. This leads to a 75% reduction in the search space for the qualities of the pools.

While the construction of the convex hull with its enclosed points is relatively simple for the case of two qualities and three sources, it is more challenging for higher orders. There are several algorithms for determination of convex hulls and enclosed points (see section 4.3 for a discussion). The next section provides useful mathematical approaches to constructing the convex hull with its enclosed points based on implicit enumeration of qualities using flow rates.

3.3 Implicit enumeration of discretized qualities b_j's using flow rate proportion

This section introduces a preprocessing step to enumerate the discretized pool qualities b_j's within the convex hull of the attainable region of the pools. For those pools inside the convex hulls, the pool qualities relate to the source qualities by the mixing rule constraints: $b_{jq} \cdot \sum_{i=1}^{l} X_{ij} = \sum_{i=1}^{l} X_{ij} \cdot a_{iq}$.

Divide both sides of this equation by $\sum_{i=1}^{l} X_{ij}$ and say $x_{ij} = \frac{X_{ij}}{\sum_{i=1}^{l} X_{ij}}$, we have: $b_{jq} = \sum_{i=1}^{l} x_{ij} \cdot a_{iq}$ where x_{ij}'s are the flow rate proportion from source i to pool j and $\sum_{i=1}^{l} x_{ij} = 1$. This is a convex hull condition.

One way of enumerating the b_{jq} points within the convex hull is to discretize the values of x_{ij} within the interval [0, 1] While satisfying the condition of and $\sum_{i=1}^{l} x_{ij} = 1$. Therefore, the values of the discretized x_{ij}'s cannot be randomly selected. In each set of discretization, they must add up to 1.

The following section describes a systematic way for enumerating the x_{ij}'s while maintaining the unit summation condition. It is worth noting that the more number of the sources, the more complicated the discretization scheme of x_{ij}'s. We start with the simplest case in which only 2 sources are investigated.

For two sources:

Consider the following:

Pool index: j

$x_{1j} = (j-1)/t$

$x_{2j} = 1 - x1j = 1 - (j-1)/t$

$b_{jq} = a_{2q} + a_{1q}.(j-1)/t$ for $q = 1,.., N_q$.

Let j run from 1 to (t+1). The total number of discretized pools is simply (t+1).

For example, when t = 5 then the interval width $\Delta x_{ij} = 1/t = 0.2$. Table 5 shows the calculation.

Table 5 An example of implicit enumeration for 2 sources

Pools	Flow fraction: Source 1 (x_{1j})	Flow fraction: Source 2 (x_{2j})	Quality q: Pool j (b_{jq})	Quality interval: (Δb_{jq})
1	0	1.0	a_{2q}	
				$0.2(a_{2q} - a_{1q})$
2	0.2	0.8	$0.2a_{2q} + 0.8a_{1q}$	
				$0.2(a_{2q} - a_{1q})$
3	0.4	0.6	$0.4a_{2q} + 0.6a_{1q}$	
				$0.2(a_{2q} - a_{1q})$
4	0.6	0.4	$0.6a_{2q} + 0.4a_{1q}$	
				$0.2(a_{2q} - a_{1q})$
5	0.8	0.2	$0.8a_{2q} + 0.2a_{1q}$	
				$0.2(a_{2q} - a_{1q})$
6	1.0	0	a_{1q}	

For three sources:

For two or more sources, the condition $\sum_{i=1}^{l} x_{ij} = 1$ must be satisfied while discretizing. This constraint reduces the discretized space from the large hyper rectangle to the much smaller convex hull of the sources, but makes difficult to calculate the total number of discretized pools as well as to program the pool index in some programming languages. In the exhaustive enumeration over a hyper rectangle gives a total of $(t+1)^{Nq}$ discretized pools, i.e. proportional to a power of N_q. We will calculate the number of discretized pools in the convex hull and see the improvement.

For three sources, enumeration of x_{ij}'s is

$x_{1j} = (u - 1)/t$ for $u = 1,\ldots, (t+1)$

$x_{2j} = (v - 1)/t$ for $v = 1,\ldots, (t+2-u)$

$x_{3j} = 1 - x_{1j} - x_{2j} = 1 - (u + v - 2)/t$

The index v runs from 1 to (t+2-u), depending on the value of the index u in the outer loop.

When u = 1, the value of v runs from 1 to (t+1). That means the number of discretized pools in this loop is (t+1), the same to that of the 2 source discretization. The pool index j equals to v.

When u = 2, the index v runs from 1 to t in order to satisfy the constraints $\sum_{i=1}^{l} x_{ij} = 1$. In other words, the point representing the pool with u = 1 and v = t+1 (i.e. x_{1j} = 1/t, x_{2j} = 1, x_{3j} arbitrary) is outside the convex hull of the 3 sources in the quality space. The pool index is $j = v + t + 1$ due to addition of (t+1) pools as u = 1.

Similarly for the next step of u, the pool index is equal to

$$(u - 1)(t + 1) - [1 + 2 + \ldots + (u - 2)] + v \qquad \text{for } u = 2,\ldots, (t+1)$$

where the first two groups account for cumulative number of pools in the previous loop of u. It can be rewritten as

$$(u - 1)(t + 1) - (u - 2)(u - 1)/2 + v \qquad \text{for } u = 1,\ldots, (t+1)$$

and now also true for the case u = 1.

The total number of discretized pools is $(t + 1) + t + \ldots + 2 + 1 = (t + 1).(t + 2)/2$. Each term in the left hand side of this equation is the number of pools in each loop of u, starting from 1 to (t+1). Equivalently, it is the last pool index.

Revisit the example of pooling problem in the previous section, the mixing rule constraints for the RON and sulfur properties are respectively:

$$b_{j1} = \sum_{i=1}^{3} x_{ij} \cdot a_{i1} = x_{1j} \cdot 82 + x_{2j} \cdot 92 + x_{3j} \cdot 82$$

$$b_{j2} = \sum_{i=1}^{3} x_{ij} \cdot a_{i2} = x_{1j} \cdot 1 + x_{2j} \cdot 2 + x_{3j} \cdot 1.5 \ .$$

Note that the x_{ij}'s are identical for both sets of constraints.

In the quality diagram, the discretized pools represented by dots in Figure 8 are inside the convex hull of the 3 sources. The convex hull – triangle $S_1S_2S_3$ – is smaller than the hyper rectangle in Figure 7.

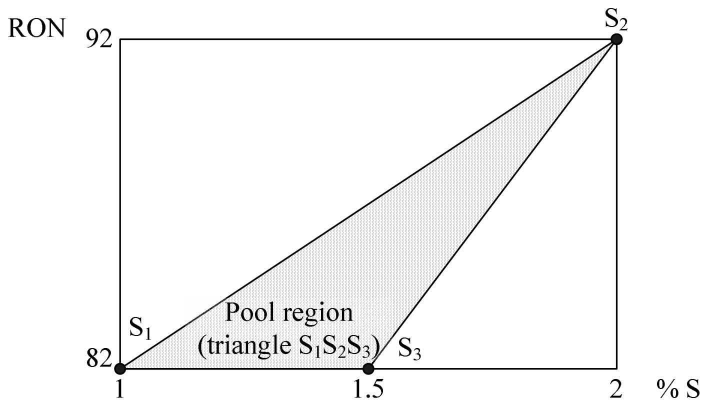


Figure 8 Convex hull discretizing space in quality diagram

An increment of 0.025 for x_{ij} is chosen, there are (1-0)/0.025 = 40 intervals, the values of b_{jq}'s are then discretized and assigned for (40+1).(40+2)/2 = 861 pools as in Table 6.

Table 6 An example of implicit discretization for three sources

Pools	Flow fraction: Source 1 (x_{1j})	Flow fraction: Source 2 (x_{2j})	Flow fraction: Source 3 (x_{3j})	Quality 1: RON (b_{j1})	Quality 2: Sulfur (b_{j2})
1	0	0	1	82	1.5
2	0	0.025	0.975	82.25	1.5125
3	0	0.050	0.95	82.5	1.525
⋮	⋮	⋮	⋮	⋮	⋮
41	0	1	0	92	2
42	0.025	0	0.975	82	1.4875
43	0.025	0.025	0.95	82.25	1.5
⋮	⋮	⋮	⋮	⋮	⋮
861	1	0	0	82	1

In LINGO, this formulation has 6,027 variables and 3,458 constraints (see Appendix B7). With a runtime of 542 seconds, a global optimum of 2,425 is found as pool 41 (92; 2.0) and pool 512 (82.25; 1.325) are used.

In the exhaustive discretization approach, only around 420 pools out of the listed 1,681 pools are inside the triangle $S_1S_2S_3$ and they are promising candidates for the solution. Meanwhile, this implicit enumeration gives 861 discretized pools, all inside the convex hull triangle. The number of promising candidates increases but the total number of variables and constraints deceases.

For four sources

The discretization procedure needs one more loop:

$$x_{1j} = (u-1)/t \qquad \text{for } u = 1,\dots,(t+1)$$

$$x_{2j} = (v-1)/t \qquad \text{for } v = 1,\dots,(t+2-u)$$

$$x_{3j} = (r-1)/t \qquad \text{for } r = 1,\dots,(t+3-v)$$

$$x_{4j} = 1 - x_{1j} - x_{2j} - x_{3j} = 1 - (u+v+r-3)/t$$

In each loop of u (each value of x_{1j}), the amount of pools results from the discretization of flow rate portions of three sources. Therefore, it is $(t+1)(t+2)/2$ for the first loop and $t(t+1)/2$ for the second loop (because the number of intervals decreases by 1 to satisfy $\sum_{i=1}^{l} x_{ij} = 1$) and so on.

The total number of discretized pools is

$$(t+2).(t+1)/2 + (t+1).t/2 + \dots + (3).(2)/2 + (2).(1)/2$$

or $$(t+1).(t+2).(t+3)/6$$

The pool index is

$(t+1).(t+2).(t+3)/6-(t-u+1).(t-u+2).(t-u+3)/6-(t+1).(t+2)/2+(v-1).(t-u+1)-(v-1).(v-2)/2+r$

This complicated pool index needs sufficient effort to be derived for programming with FOR loop in LINGO 10 or older. In other programming software, there is a simple trick to deal with the pool indices in which *j* is added by 1 in each step of *r* in the appropriate loop command, e.g. WHILE, DO, REPEAT etc.

For *l* sources

In general, if there are *l* sources and *t* intervals are chosen to discretize, the total number of discretized pools is given by

$\frac{(t+1).(t+2)...(t+l-1)}{1.2.3...(l-1)} = \frac{(t+l-1)!}{(l-1)!.t!}$ which is not dependent on the number of qualities.

The discretization of b_j's which is considered as a preprocessing step has been performed. These discretized values are the input data and play a role of parameters in the formulation (P').

3.4 Implemented formulation for implicit enumeration of quality discretization for the pools

This part provides an implementation on the formulation using the same implicit discretization approach described earlier. The implementation reduces the number of variables and constraints in the formulation without compromising the optimization results.

In the proposed implicit enumeration approach, the values of b_j's are calculated from $b_{jq} = \sum_{i=1}^{l} x_{ij} \cdot a_{iq}$ where a_{iq}'s are known parameters and x_{ij}'s are discretized on the range [0, 1]. As shown, this set of equalities is actually another form of the mixing rule constraints for the pools. Hence, when the values of x_{ij}'s associating with b_j's are stored in the preprocessing step and supplied as input data for the formulation, all the terms in the mixing rule equations are known and can be moved from the formulation to the preprocessing discretization step. x_{ij}'s can substitute X_{ij}'s to refer the flow rates from each source to pools. Those flow rates are the multiplication products between the fractional flow rates x_{ij}'s and the total flow rate through pool j, say Z_j. Therefore, X_{ij}'s are removed from the formulation, leaving Y_{jk}'s and Z_j as the decision variables.

This implementation takes $l.(t+1)^{Nq}$ variables X_{ij}'s and $(t+1)^{Nq}$ constraints on mixing rule for pools off, and introduces $(t+1)^{Nq}$ variable Z_j's and $l.(t+1)^{Nq}$ parameter x_{ij}'s to the formulation. The constraints and unknown terms are sufficiently reduced.

The formulation is given by:

Objective function: $\sum_{k=1}^{n} P_k \cdot \sum_{j=1}^{M} Y_{jk} - \sum_{i=1}^{l} [C_i \sum_{j=1}^{M} (x_{ij} \cdot Z_j)]$ is maximized

Available supply: $\sum_{j=1}^{M} x_{ij} \cdot Z_j \le S_i$ for $i = 1,\ldots,l$

Mass balance on pools: $Z_j = \sum_{k=1}^{n} Y_{jk}$ for $j = 1,\ldots,M$

Product demand: $\sum_{j=1}^{M} Y_{jk} \le D_k$ for $k = 1,\ldots,n$ (P')

Number of pools: $f_j = \{0, 1\}$ for $j = 1,\ldots,M$

$Z_j \le U_j.f_j$ for $j = 1,\ldots,M$

$\sum_{j=1}^{M} f_j \le m$

Mixing rule for products: $c_{kq} \cdot \sum_{i=1}^{l} Y_{jk} \ge \sum_{i=1}^{l} Y_{jk} \cdot b_{jq}$ for $k = 1,\ldots,n$

Positive variables: $Z_j \ge 0$, $Y_{jk} \ge 0$ for $i = 1,\ldots,l$

$j = 1,\ldots,m$

$k = 1,\ldots,n$

where b_{jq}'s and x_{ij}'s are known parameters defined in the section 4.3.

By discarding the mixing rule constraints in the formulation (but discretization is based on those constraints), is there anything missing? To answer this question, consider the following case shown by Figure 9.

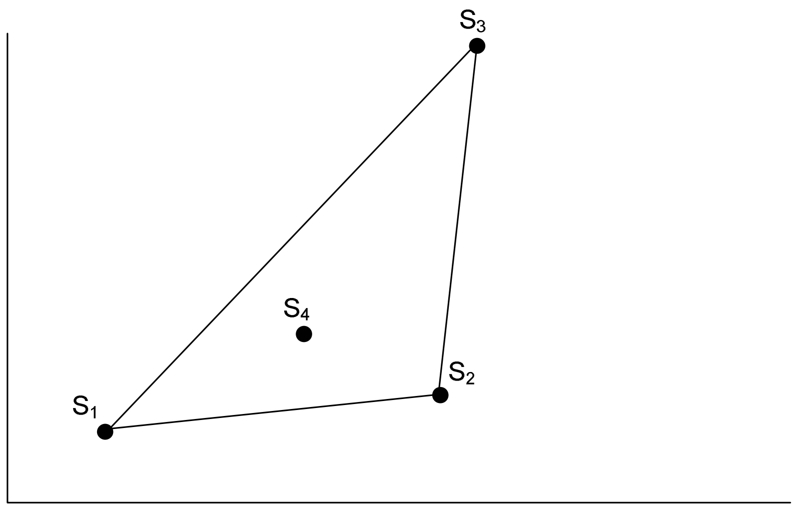


Figure 9 Attainable region for four sources

From Figure 8, it can be seen that a discretized quality may result from more than one flow fraction discretization. For instance, the pool represented by the point at S_4 can be obtained from source S_4 only or from a certain combination of S_1, S_2 and S_3.

This non-uniqueness in discretization leads to redundancy which may be avoided if we realize that S_4 is inside the convex hull of the sources; therefore is not a component of the convex hull boundary. So we discretize on the set of three sources S_1, S_2 and S_3 only, i.e. considering flow rate fraction of S_4 as zero when discretizing.

If the constraints on the mixing rules are removed, something is missing. Since the fractional flow rate from S_4 is zero as discretizing, it is always stored as zero in formulation input data. The solver is then not allowed to use S_4. This may mislead the solution when the cost of S_4 is cheaper.

If the constraints on the mixing rules are still kept in the formulation and discretized values of x_{ij}'s are not stored, nothing is missed. The qualities are discretized without records of associated flow rate fractions. Then the solver is free to calculate the optimum flow rate (or flow rate fraction), i.e. the feasible region covers both duplicated discretizations.

For the above-stated reasons, we should keep all the sources to discretize flow rate fractions x_{ij}'s if the constraints on the mixing rules are moved to the preprocessing step. The implemented formulation of the example is reported in Appendix B8. Calculation runtime is improved (i.e. deceased) and shown in Figure 24.

3.5 Comparison of two discretization approaches

For discretizing b_{jq}'s, the number of discretized pools as well as reformulation size mostly depends on the number of investigated qualities. This approach favors pooling problems with only one quality. For problems with many qualities, the reformulated problem may be large enough to overwhelm the capacity memory of linear programming software.

For discretizing b_{jq}'s by implicitly enumerating x_{ij}'s, the discretizing space is the convex hull of sources, which is smaller than the hyper rectangle in the quality space. Although the approach is applicable for pooling problems with one quality, it does not reduce the discretizing space, even may lead to the case that some pools have a same quality. The amount of discretized pools drastically increases when the number of sources increases and does not depend on the number of qualities.

Given a pooling problem, the latter approach produces a reformulation with a much smaller number of pools as well as the formulation size. However, it takes sufficient efforts in programming to relate to pool index j to assigned values of x_{ij}'s when dealing with many qualities in LINGO language.

3.6 Distinction from discretization in piecewise linear RLT

Meyer and Floudas [2] proposed the use of discretizing the quality space in solving pooling problems using reformulation linear transformation (RLT). It is important to distinguish the RLT approach from the new discretization approach proposed in this work. As discussed in the Literature Review section, Meyer and Floudas [2] proposed a piecewise linear reformulation based on the RLT technique to solve the superstructure model of generalized pooling problems. Before the RLT is applied, the continuous space of each quality is partitioned into many subintervals that are placed between discretized points. Some binary variables are introduced along with integer-cut constraints to indicate which interval includes the optimum quality. Then, RLT is used with a note that the mixing rule constraints are excluded from the reformulation step but included in the linearization step.

Meyer and Floudas' discretization of quality space augments the upper bound on the global maximum of the original problem. Therefore, the calculation time is reduced despite the addition of variables. This contribution is depicted in Figure 10.

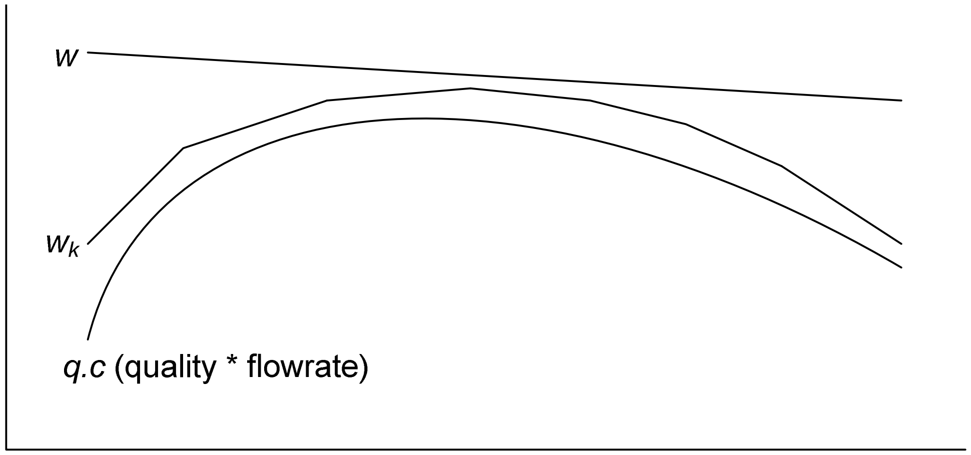


Figure 10 Piecewise linearization vs. regular linearization of RLT

Assume that the curve of bilinear term *q.c* is represented as the bottom curve. The linearization step of Sherali's [21] RLT replaces *q.c* by new variable w. This means that the curve *q.c* is substituted by a straight line for the whole investigated range. Meyer and

Floudas [2] proposed a new way of linearization. The curve *q.c* is replaced by many connected line segments w_k's which are closer to *q.c* curve than the line *w* in the former approach. This is called piecewise linear relaxation. To obtain this piecewise function, Meyer and Floudas [2] discretized the *q* domain and investigated the subintervals between those discretized points.

The global optimum of the relaxed problem provides an upper bound on the original problem. But the found solution could or could not be a feasible solution for the original problem, i.e. it does not guarantee one can find a set of quality and flow rate values associating to the optimum of the objective function. Meanwhile, the discretization approach in this research gives a lower bound along with the blending strategy in perfect mass balance. Figure 11 shows the difference between the outcomes of the two approaches. This is the first difference between the two approaches.

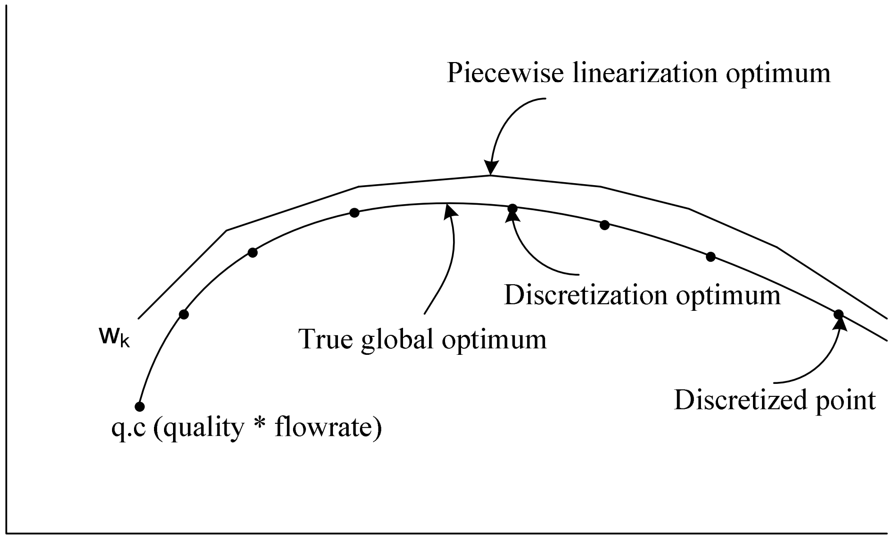


Figure 11 Comparison of optima for the two discretization approaches

The RLT approach of Meyer and Floudas [2] does not produce a lower bound. The authors suggested a way to verify the ε-global optimum by performing a series of runs in which the quality partition scheme is restructured after each run. But the authors did not demonstrate this algorithm in the example of a large scale industrial problem. Instead, the lower bound was found by using DICOPT. The approach can reduce the gap between the lower and upper bound of the example to 1.2%.

Another distinction between the two approaches is that Meyer and Floudas [2] discretized the hyper-rectangle space of qualities. This approach is in the same manner to the exhaustive enumeration discretization described earlier in this research. As has been discussed previously, this work has introduced the concept of implicit enumeration of the discretized qualities within the convex hull (attainable region). This approach typically results in much smaller linearized problems.

CHAPTER IV

METHODOLOGY

4.1 Discretization approach in optimization

Discretization is a useful way of transforming optimization problems with infinite search points (over a continuous space) to a search space with finite points (over a discrete space). Discretization may be conducted to fix values of complicating variables in a way that transforms a nonlinear program to a linear program which is amenable to global solution. With sufficient discretization, the global optimum of the discretized problem can approximate (or coincide with) the true global optimum of the original problem. Hence, a discretization approach is intended to give an ε- global optimum.

In order to reduce the mismatch between the two optima, discretization schemes should be fine-enough to approximate the true solution. However, increased discretization yields problem sizes that increase dramatically, resulting in increased computational cost (time). This trade-off needs to be carefully balanced so that the practically acceptable tolerance is found in a reasonable computational time.

Another way to reduce the tolerance is to solve the problems in series of runs. The first run is for raw discretization. The next run uses updated input data which are finer discretization around the optimum point from the previous step and so on.

Due to highly computational time (or solution error), discretization approach has not been widely applied. However, it is hoped that the results of this research show that the application to the pooling problems gives optima with acceptable tolerance and runtime.

4.2 Mixed integer linear programs

A mixed integer linear program (MILP) is a mathematical program of optimization problems in which the objective function and constraints are linear and with some of the variables taking on integer values while other variables are continuous. As stated by Kallrath, [22] there are often many ways to formulate an MILP and which way to choose should be carefully considered because the computational costs may vary, which is a well-known characteristic of MILPs. To transform a nonlinear program into MILP problems, the following techniques are usually applied:

- Binary variables are used to formulate logical conditions and disjunctive constraints.

- Binary variables can be also introduced to formulations to represent bounds and signify the values of semi-continuous variables defined as $\{x \mid x = 0 \text{ or lower} < x < \text{upper}\}$. The reformulated model is lower*$f < x <$ upper*f and f = {0, 1} that is appropriate to branch and bound framework.

- When piecewise linear functions are present, special ordered set is a good choice to model them. There are two types of special ordered sets. In type 1, only one variable is non-zero. For instance, a problem is to choose one of mutually exhaustive alternatives. In type 2, there are not more than 2 variables in the set may be non-zero. Approximate linearization of nonlinear function is often done using special ordered set type 2.

There have been many algorithms developed to solve MILPs. These include cutting planes, branch and cut, dynamic programming, decomposition, logic-based methods and the widely used branch and bound approach which is discussed in details below.

4.3 Convex hull algorithm

Given is a set of discrete points. The convex hull of a set is the smallest polytope containing all points in the set.

Algorithms to determine the convex hull is a field of geometrical mathematics. There have been many algorithms, shown in Table 7 (Sunday [23]). In this table, n is the number of points and h is the number of vertices on the convex hull. Among these algorithms, Graham Scan and Divide-and-Conquer are the most popular ones.

Table 7 Convex hull algorithms (Sunday [23])

Algorithm	Speed
Brute Force	O(n^4)
Gift Wrapping	O(nh)
Graham Scan	O(n log n)
Jarvis March	O(nh)
QuickHull	O(nh)
Divide-and-Conquer	O(n log n)
Monotone Chain	O(n log n)
Incremental	O(n log n)
Marriage-before-Conquest	O(n log h)

These convex hull determining procedures can be integrated with the proposed exhaustive discretization approach in order to eliminated useless discretized pools, which reduces the problem size.

However, the implicit discretization approach needs not use those algorithms. The equalities $b_{jq} = \sum_{i=1}^{l} x_{ij} \cdot a_{iq}$ are a form of convex hull condition. These constraints guarantee that the discretized pools are only inside the convex hull of sources in the quality diagram.

4.4 Branch and bound approach

The branch and bound approach is a common (but not the only) procedure to solve discrete and combinatorial optimization problems.

According to Murty, [24] the main difficulty for solving discrete optimization and combinatorial optimization problems is that there has not been a global optimality condition to check whether a feasible solution is optimal or not. Instead, a systematic way is to enumerate all of the feasible solutions and then choose the best. However, this approach is only applicable to small problems. For large enough problems, even the fastest computer can not handle the huge amount of calculation for this total enumeration. Unfortunately, practical optimization problems are usually in this large size. Another better approach is to partially enumerate the feasible solutions so that computational cost is affordable and optimality is guaranteed. Branch and bound is such an approach.

The branch and bound approach was first published in the beginning of the 1960s, as a result of two independent contributions of Land and Doig, [25] which is focused on general discrete optimization problem, and Murty [24] on a specific type of discrete optimization problems.

The approach name itself implies the solution procedure. From a feasible heuristic solution (a root node), *branching* on the feasible regions results in a set of subproblems looked like a tree, nodes of which are subproblems. But, branching a set of nodes until the ending node is not necessary if we check and see the best optimal in the descent nodes can not be better than a known optimal which is called *bound*. When we have many branches *pruned* like this, the enumerating load is sufficiently reduced.

Because we safely stop branching at the nodes, the branch and bound approach is still an exactly global optimization procedure despite of partial enumeration.

Certainly, we prefer to prune off as many branches as possible. Nevertheless, it highly depends on the data as well as how to branch and bound. A specific way of branch and bound is referred as an algorithm. A branch and bound algorithm, therefore, may be suitable for a set of data but poorly behaves to other data or problems. There is no unique branch and bound algorithm that is good for every discrete or combinatorial optimization problem.

A good branch and bound algorithm is the one that gives an optimum tight bound. The bound should not be too tight or too loose. If the bound is loose, many nodes need to be calculated, leading to expensive calculation cost. On the other hand, a highly tight bound is a result of more calculation effort at each node and it may exceed the benefit from reducing node amount.

Many branch and bound algorithms become more effective when integrated with other techniques, such as Lagrangian, Reformulation-Linearization Technique as discussed in Chapter III - Literature Review.

CHAPTER V

A SELECTION OF PUBLISHED POOLING PROBLEMS

The application of the discretization approach will be demonstrated using some published pooling problems. These include Haverly's (3 problems), Ben-Tal's (2 problems), Foulds' (4 problems) and Adhya's (4 problems).

Figure 12 and Table 8 show the input data for Haverly's [3] pooling problems, the first published and simplest series with only a one actual pool and one quality. to the problems referred to as Haverly 1 and Haverly 2 have the updated first product demand and Haverly 3 has the adjusted cost of the second source. Other parameters are identical in the three problems.

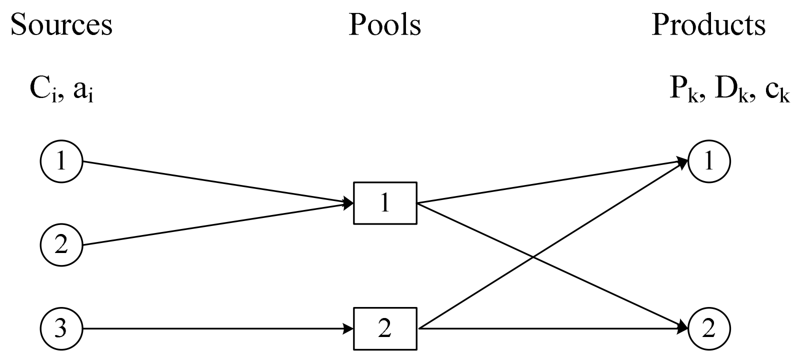


Figure 12 Haverly's [3] pooling problems

Table 8 Haverly's [3] pooling problems

Pooling problem	Haverly 1	Haverly 2	Haverly 3
Source 1 (cost; quality)	(6;3)	(6;3)	(6;3)
Source 2 (cost; quality)	(16;1)	(16;1)	(13;1)
Source 3 (cost; quality)	(10;2)	(10;2)	(10;2)
Product 1 (price; demand; max quality)	(9;100;2.5)	(9;600;2.5)	(9;100;2.5)
Product 2 (price; demand; max quality)	(15;200;1.5)	(15;200;1.5)	(15;200;1.5)

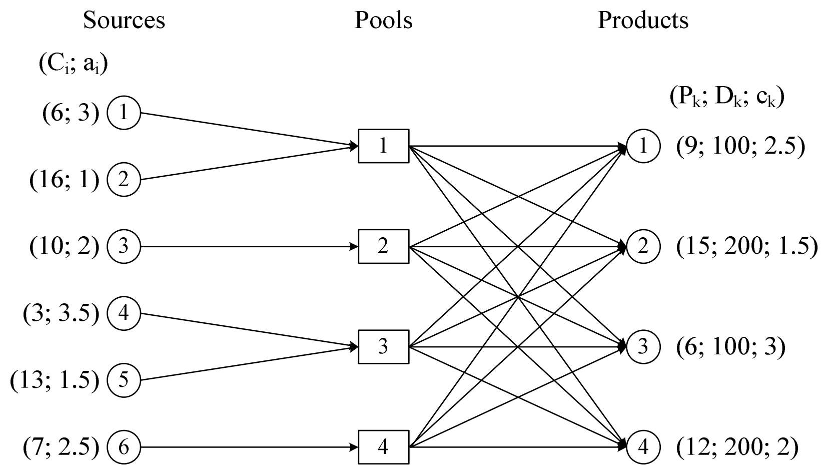


Figure 13 Foulds 2 pooling problem (Foulds [12])

Table 9 Foulds 3 pooling problem (Foulds [12])

Pool	Source	Cost (C_i)	Quality (a_i)	Product	Price (P_k)	Min quality (c_k)
1	1	20	1.0	1	20.0	1.05
	2	19	1.1	2	19.5	1.10
	3	18	1.2	3	19.0	1.15
	4	17	1.3	4	18.5	1.20
2	2	19	1.1	5	18.0	1.25
	3	18	1.2	6	17.5	1.30
	4	17	1.3	7	17.0	1.35
	5	16	1.4	8	16.5	1.40
3	3	18	1.2	9	16.0	1.45
	4	17	1.3	10	15.5	1.50
	5	16	1.4	11	15.0	1.55
	6	15	1.5	12	14.5	1.60
4	4	17	1.3	13	14.0	1.65
	5	16	1.4	14	13.5	1.70
	6	15	1.5	15	13.0	1.75
	7	14	1.6	16	12.5	1.80
5	5	16	1.4			
	6	15	1.5			
	7	14	1.6			
	8	13	1.7			
6	6	15	1.5			
	7	14	1.6			
	8	13	1.7			
	9	12	1.8			
7	7	14	1.6			
	8	13	1.7			
	9	12	1.8			
	10	11	1.9			
8	8	13	1.7			
	9	12	1.8			
	10	11	1.9			
	11	10	2.0			

Foulds et al. [12] developed a pooling problem (referred to as Foulds 2 in Figure 13) from Haverly's problem and generated three larger problems. Due to the complexity of the jumble diagrams, Foulds 3, Foulds 4 and Foulds 5 examples are only presented in table forms (Table 9, Table 10 and Table 11.) For these problems, each of the pools is blended from some selective sources and may be mixed in any product.

Table 10 Foulds 4 pooling problem (Foulds [12])

Pool	Source	Cost (C_i)	Quality (a_i)	Product	Price (P_k)	Min quality (c_k)	Demand (D_k)
1	1	20	1.0	1	20.0	1.05	1
	4	17	1.3	2	19.5	1.10	1
	7	14	1.6	3	19.0	1.15	1
	10	11	1.9	4	18.5	1.20	1
2	2	19	1.1	5	18.0	1.25	1
	5	16	1.4	6	17.5	1.30	1
	8	13	1.7	7	17.0	1.35	1
	11	10	2.0	8	16.5	1.40	1
3	3	18	1.2	9	16.0	1.45	1
	2	19	1.1	10	15.5	1.50	1
	5	16	1.4	11	15.0	1.55	1
	6	15	1.5	12	14.5	1.60	1
4	4	17	1.3	13	14.0	1.65	1
	3	18	1.2	14	13.5	1.70	1
	6	15	1.5	15	13.0	1.75	1
	7	14	1.6	16	12.5	1.80	1
5	5	16	1.4				
	6	15	1.5				
	3	18	1.2				
	8	13	1.7				
6	6	15	1.5				
	7	14	1.6				
	4	17	1.3				
	9	12	1.8				
7	7	14	1.6				
	8	13	1.7				
	9	12	1.8				
	4	17	1.3				
8	8	13	1.7				
	9	12	1.8				
	10	11	1.9				
	5	16	1.4				

Table 11 Foulds 5 pooling problem (Foulds [12])

Pool	Source	Cost (C_i)	Quality (a_i)	Product	Price (P_k)	Min quality (c_k)
1	1	20	1.0	1	20.0	1.05
	2	19	1.1	2	19.5	1.10
	3	18	1.2	3	19.0	1.15
	4	17	1.3	4	18.5	1.20
	8	13	1.7	5	18.0	1.25
	9	12	1.8	6	17.5	1.30
	10	11	1.9	7	17.0	1.35
	11	10	2.0	8	16.5	1.40
2	2	19	1.1	9	16.0	1.45
	3	18	1.2	10	15.5	1.50
	4	17	1.3	11	15.0	1.55
	5	16	1.4	12	14.5	1.60
	7	14	1.6	13	14.0	1.65
	8	13	1.7	14	13.5	1.70
	9	12	1.8	15	13.0	1.75
	10	11	1.9	16	12.5	1.80
3	4	17	1.3			
	5	16	1.4			
	6	15	1.5			
	7	14	1.6			
	8	13	1.7			
	9	12	1.8			
	10	11	1.9			
	11	10	2.0			
4	1	20	1.0			
	2	19	1.1			
	3	18	1.2			
	4	17	1.3			
	5	16	1.4			
	6	15	1.5			
	7	14	1.6			
	8	13	1.7			

Ben-Tal et al. [16] proposed two pooling problems by introducing one more source with a constraint on its capacity (problem Ben-Tal 4 in Figure 14) to Haverly's example or developed a larger problems with additional 3 sources, 2 pools and 3 products (Ben-Tal 5 in Figure 15).

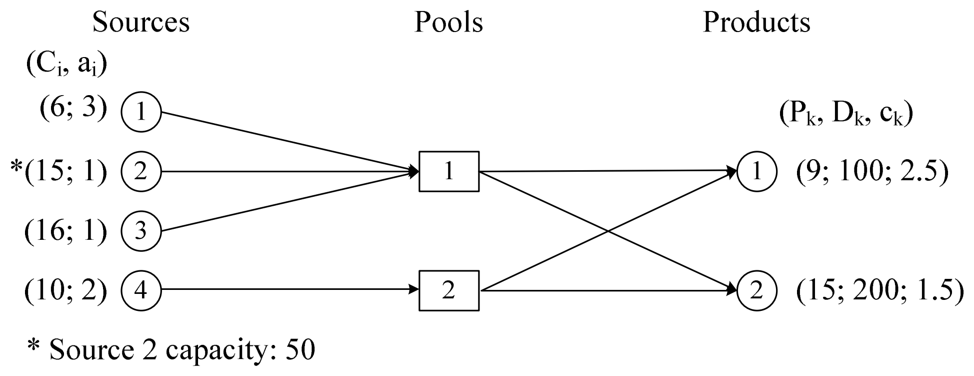


Figure 14 Ben-Tal 4 pooling problem (Ben-Tal et al. [16])

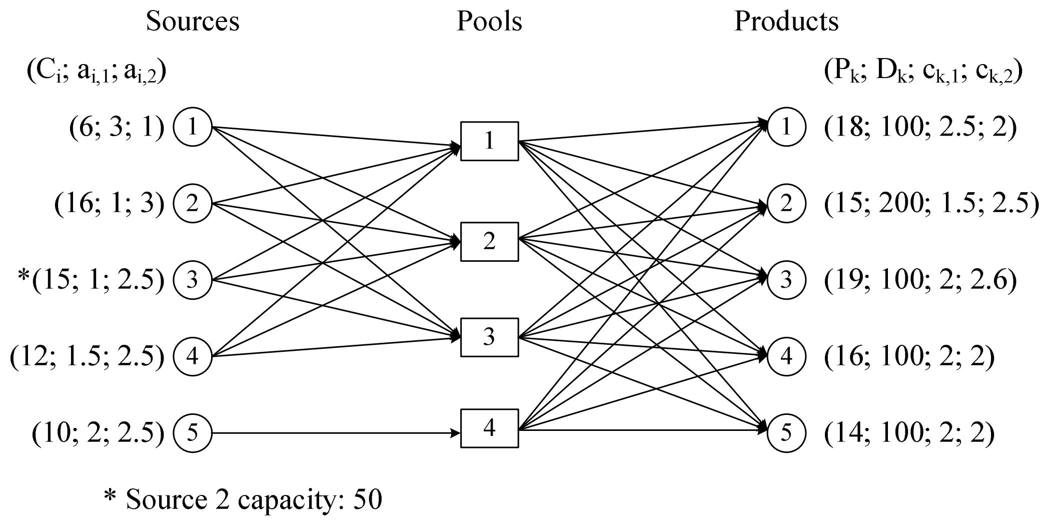


Figure 15 Ben-Tal 5 pooling problem (Ben-Tal et al. [16])

Haverly, Foulds and Ben-Tal's pooling problems may be referred as small problems for dealing with only one or two qualities. Meanwhile, the Adhya's examples with many investigated qualities are larger in size. These problems' data are presented in Figure 16, Figure 17, Figure 18 and Figure 19.

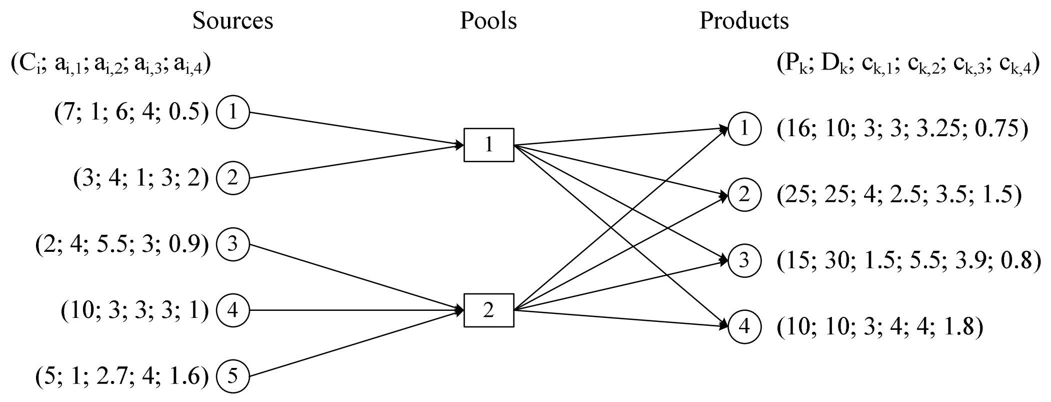


Figure 16 Adhya 1 pooling problem (Adhya et al. [18])

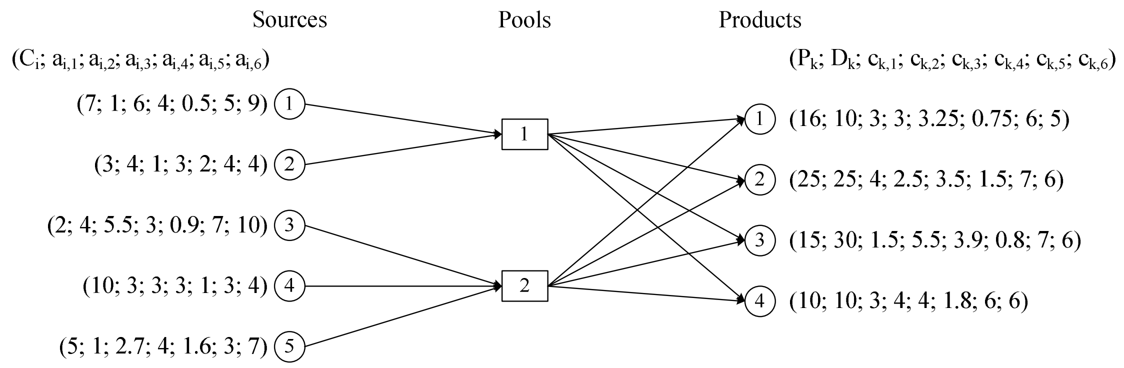


Figure 17 Adhya 2 pooling problem (Adhya et al. [18])

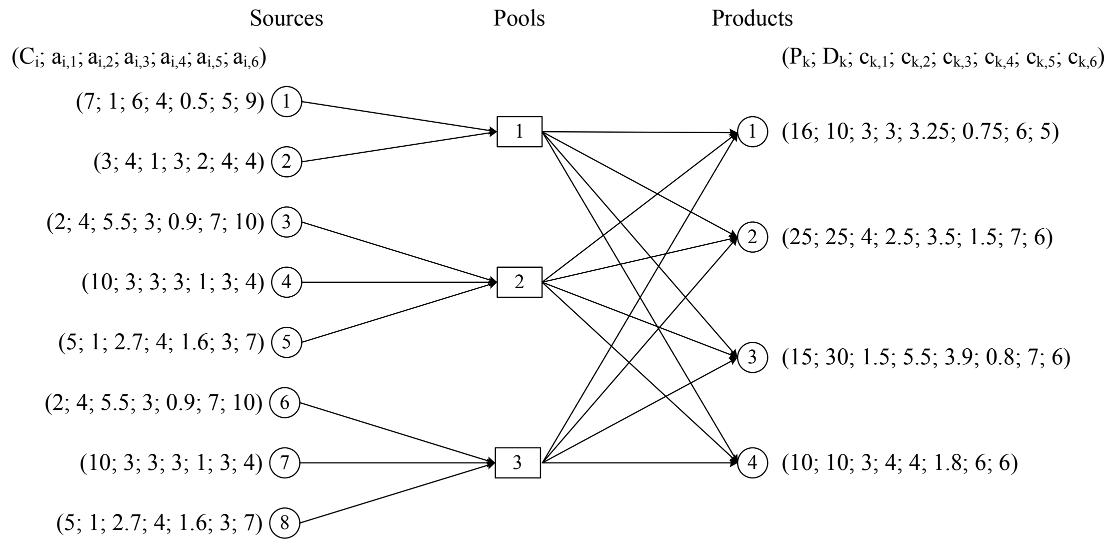


Figure 18 Adhya 3 pooling problem (Adhya et al. [18])

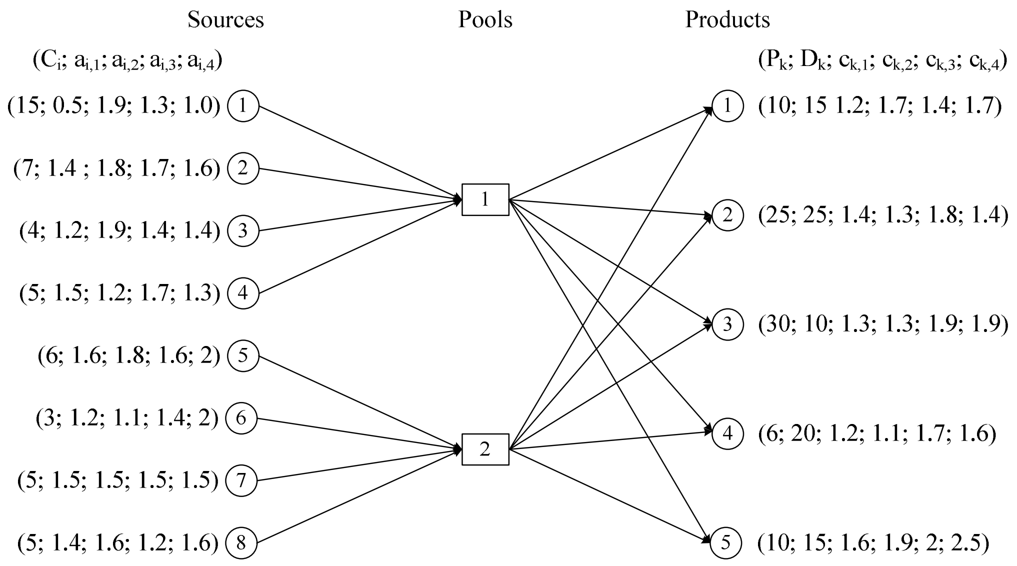


Figure 19 Adhya 4 pooling problem (Adhya et al. [18])

CHAPTER VI

RESULTS AND DISCUSSION

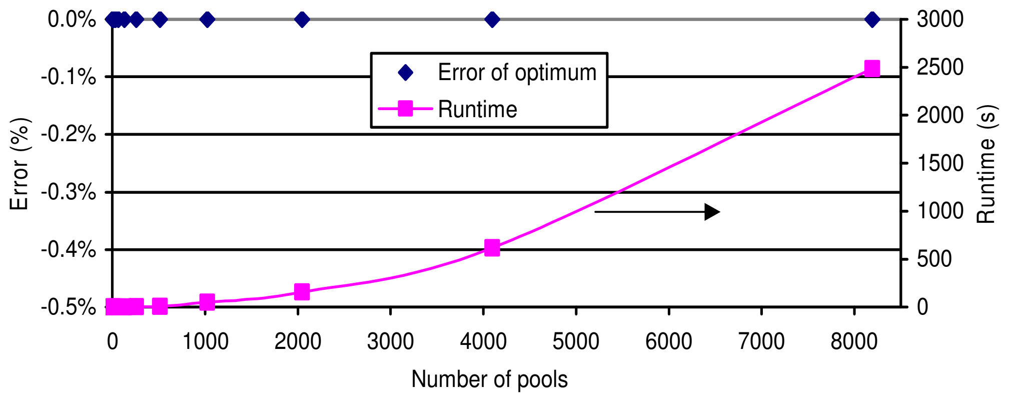
In this chapter, the test problems described in Chapter V are solved using the proposed new approach. As mentioned earlier, the new approach transforms the nonlinear pooling formulation to an MILP. The resulting MILP is solved using the commercial optimization software LINGO [26] developed by of LINDO Systems. The programs are solved using a personal computer Dell Optiplex GX 620 with the processor Pentium 4 3.6Ghz and a RAM of 1 Gb. The new version of LINGO (version 10) has a global solver tool for NLP problems. This option will also be used to solve the same test problems to compare with the solution results and characteristics of the proposed approach.

For each test problem, several runs are made to account for the effect of increasing the number of intervals on the quality and computation time of the solution. A series of runs are made such that the number of intervals in each run is taken as double the number of intervals in the previous run. Consequently, all discretized points one run are kept in the subsequent run.

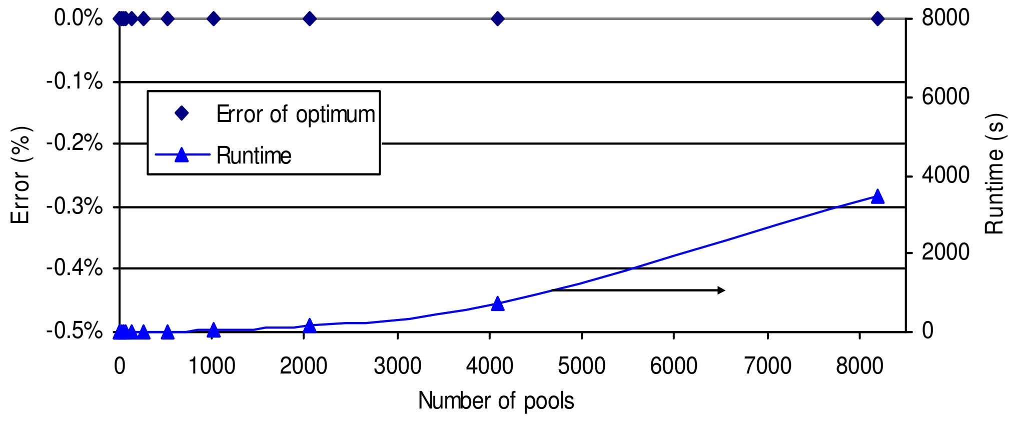
The results are grouped according to four categories: results for problems with one quality, two qualities, more than two qualities, and results using the global solver tool of LINGO.

6.1 All global optimum found in one-quality problems

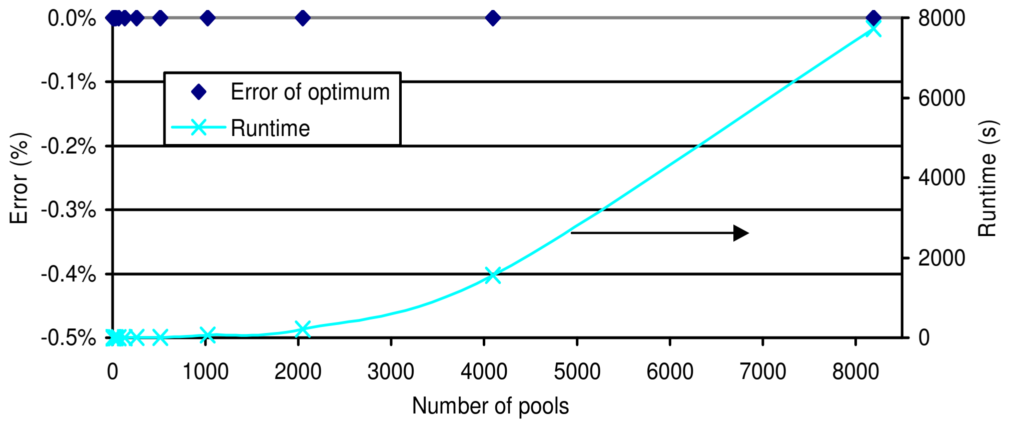
Among the investigated pooling problems, there are those with one quality such as Haverly's (3 problems), Foulds' (4 problems) and Ben-Tal 4. As discussed in Section 4.5, for problems with one quality, it is preferable to use the exhaustive enumeration approach. The codes are presented in Appendix A. The calculation results (run time and deviation from global solution) are shown by Figure 20 - 21.



(a)



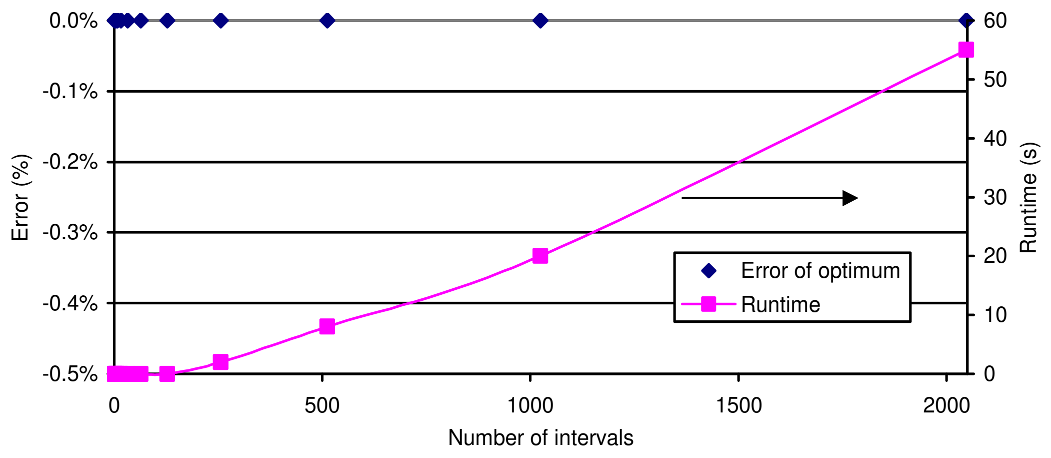
(b)



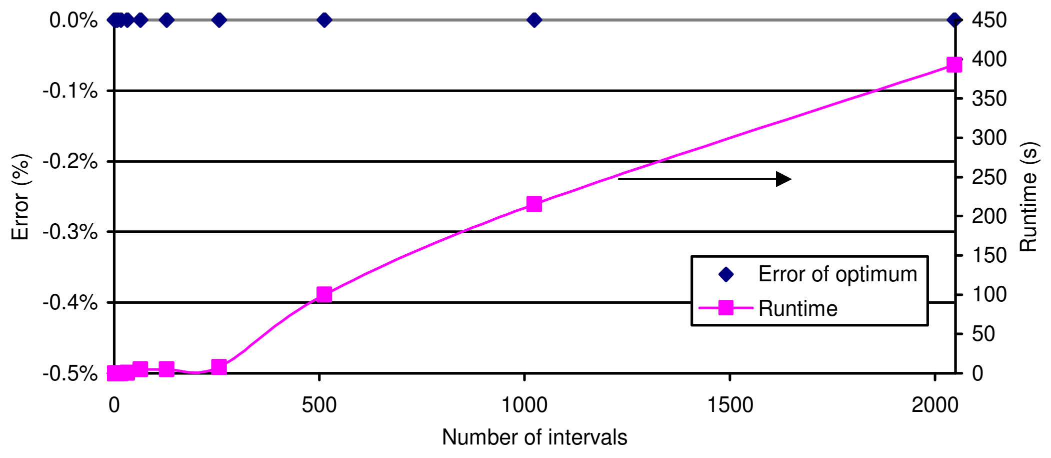
(c)

Figure 20 Results on Haverly's problems with exhaustive enumeration of discretization (a) Haverly 1, (b) Haverly 2, (c) Haverly 3

As can be seen from the results, all three problems were solved globally using the new discretization approach. It is worth noting that as the number of intervals is doubled, the computational time increases and the results constitute a series of non-inferior solutions (improving or staying the same).

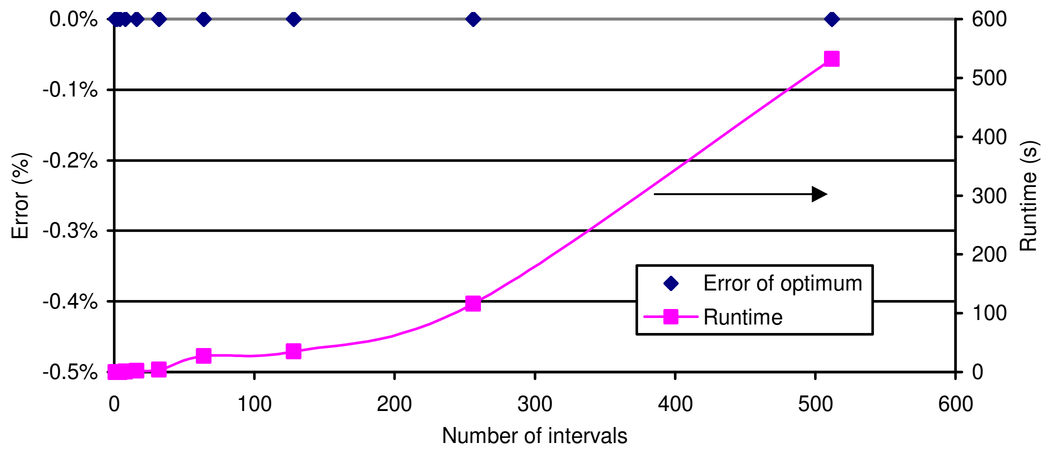


(a)

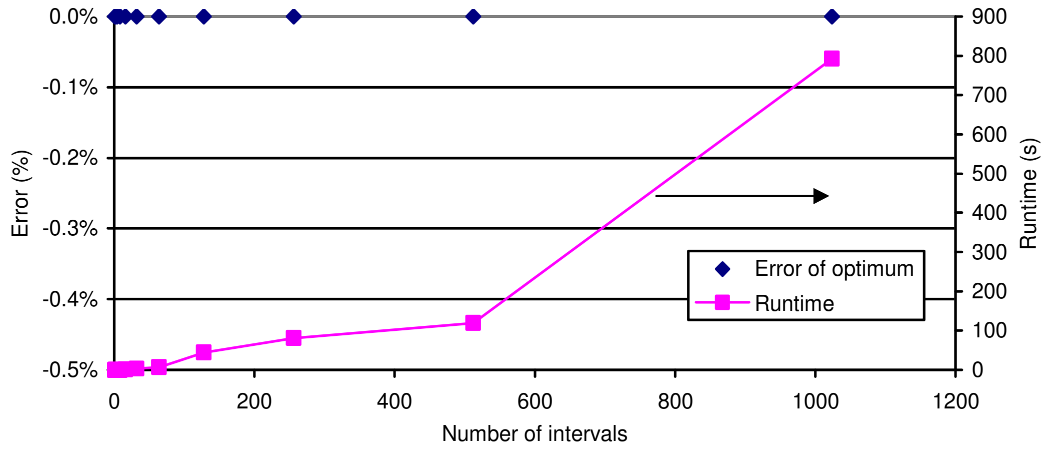


(b)

Figure 21 Results on Foulds' problems with implicit enumeration of discretization (a) Foulds 2, (b) Foulds 3, (c) Foulds 4, (d) Foulds 5



(c)



(d)

Figure 21 Continued

The runtime curve in Figure 22 for problem Ben-Tal 4 is a typical runtime curve of discretization approach. The curve is not linear and its slope increases quickly as the discretized points are fine enough.

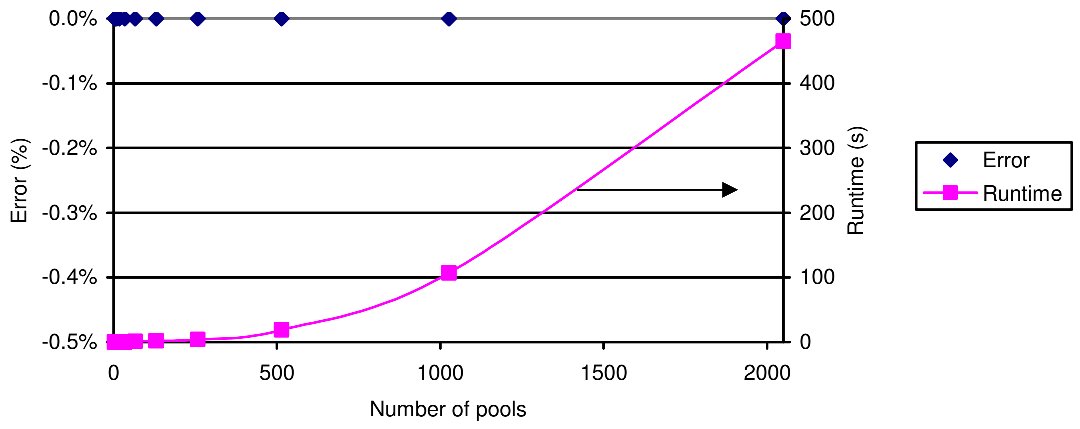


Figure 22 Result on problem Ben-Tal 4 with exhaustive discretization

6.2 Enhancing the performance with implicit enumeration for discretization

The two-quality pooling problems, Ben-Tal 5 and the motivating example (described in section 3.1), are investigated to see the calculation results using different discretization and formulation approaches:

- Scenario 1: Exhaustive enumeration for discretization.
- Scenario 2: Implicit enumeration for discretization with mixing rule constraints.
- Scenario 3: Implicit enumeration for discretization without mixing rule constraints.

The codes are reported in Appendix B. The calculation results are presented in Figure 23 and Figure 24. For problem Ben-Tal 5, since the number of discretized pools increases quickly as the number of intervals doubles, an axis of logarithmic scale is used. As for the example, because the global optimum has not been proven, the axis of found optimum instead of error is used. The found optima from scenarios 2 and 3 are the same because the preprocessing discretization steps are identical.

From the results, it is shown that discretization via the implicit enumeration approach is more advantageous than the exhaustive enumeration approach for discretization. With the same number of discretized pools, the former gives better solutions and requires less runtime for calculations.

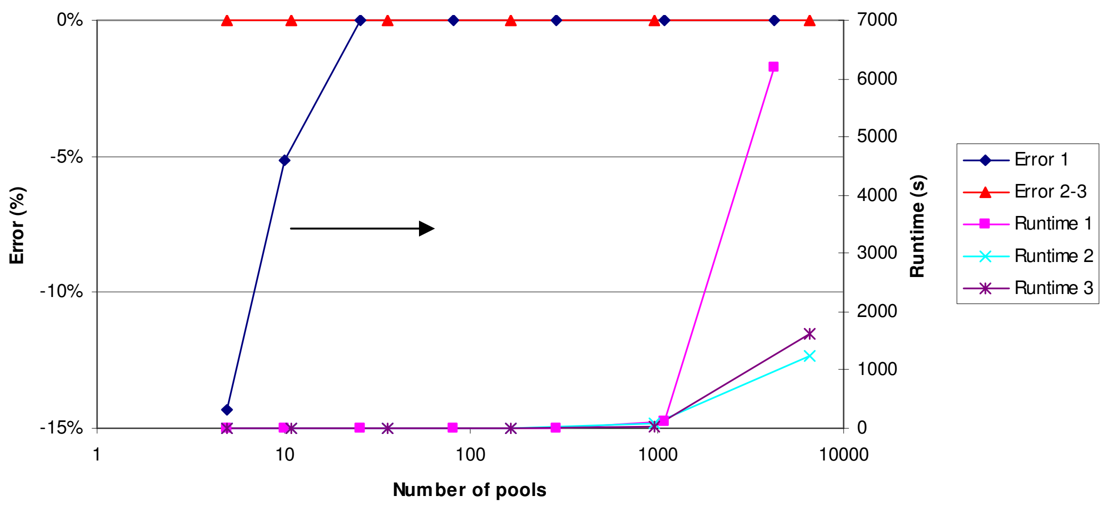
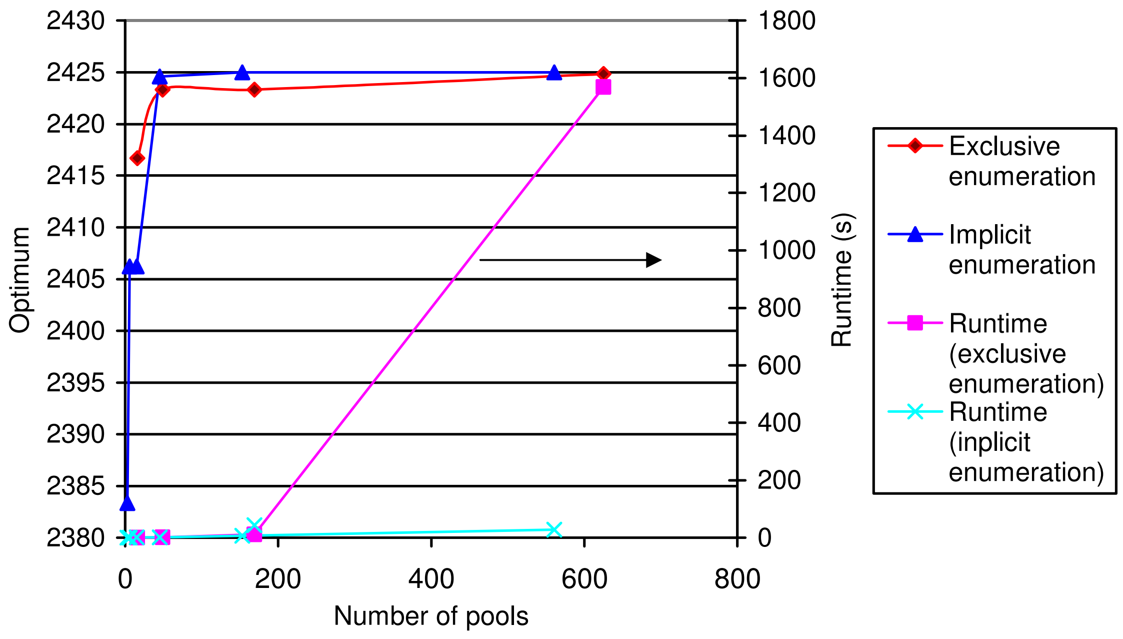
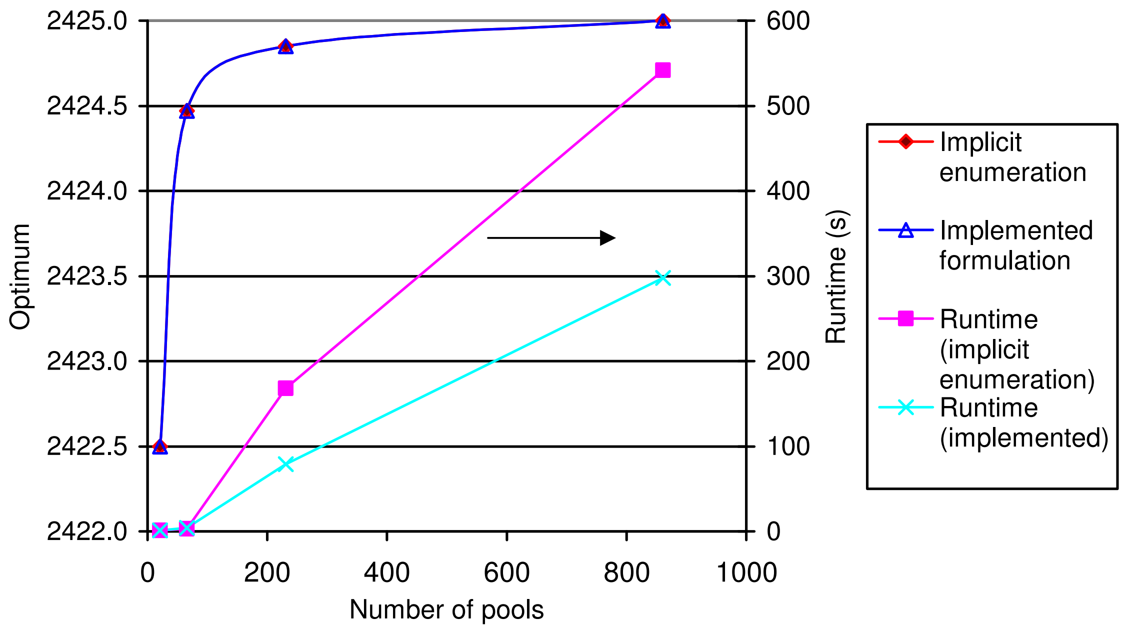


Figure 23 Results on problem Ben-Tal 5 with various approaches



(a)

Figure 24 Results on the example problem with various approaches
(a) Exhaustive and implicit enumeration
(b) Implicit enumeration and its implemented formulation



(b)

Figure 24 Continued

Between the two implicit enumeration discretization approaches, there is no clear improvement of the formulation without the mixing constraints over the one with those constraints. However, the improvement will become more observable for pooling problems with more-than-two qualities, as in the results of solving Adhya's pooling problems in the following section.

6.3 Results of implicit enumeration for discretization of pooling problems with multiple qualities

In this section, two sets of scenarios are carried out:

- Scenario 1: Implicit enumeration for discretization with mixing rule constraints.
- Scenario 2: Implicit enumeration for discretization without mixing rule constraints.

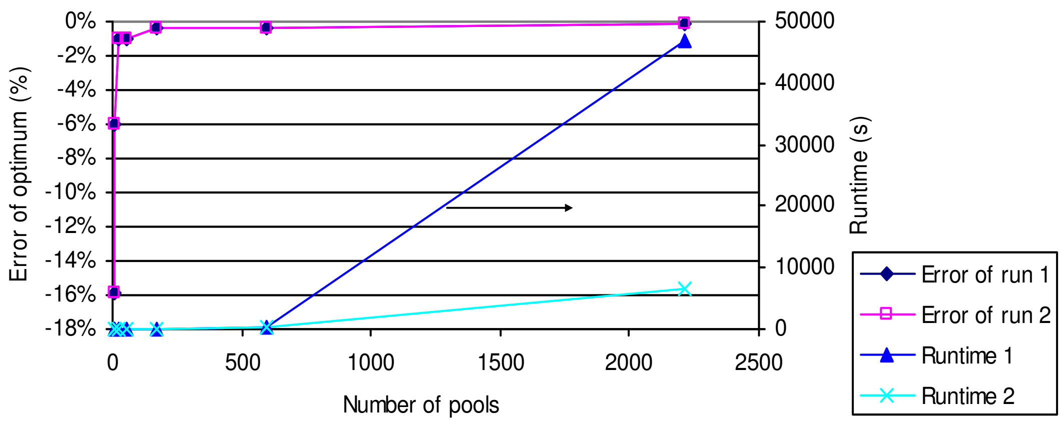


Figure 25 Results on problem Adhya 1 with two formulations

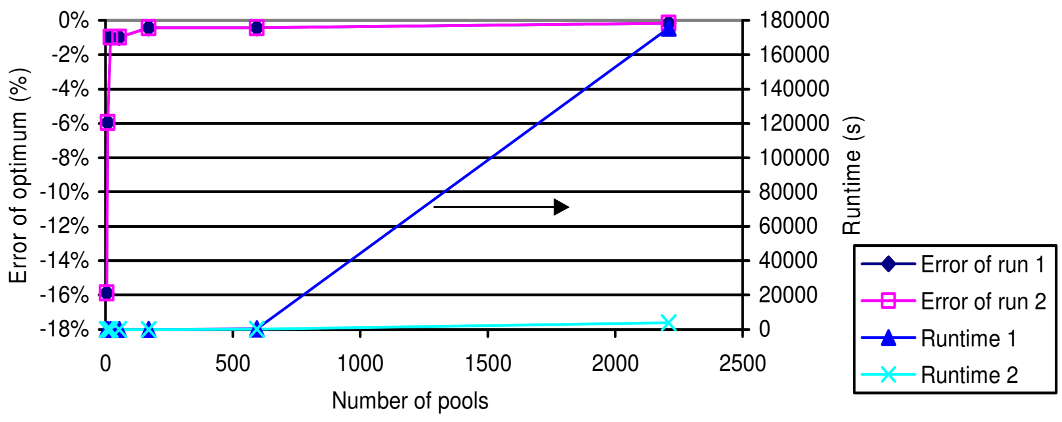


Figure 26 Results on problem Adhya 2 with two formulations

The results are shown in Figure 25, Figure 26, Figure 27 and Figure 28. Again, the found optima and the numbers of discretized pools are the same because of identical discretized points. The difference is with or without the mixing rule constraints, resulting in the different number of variables and constraints that the solver has to deal with. Therefore, the runtimes become a key factor in analyzing the results.

The Adhya's pooling problems have 4 or 6 qualities. If the exhaustive enumeration approach is applied, a linear formulation of a huge size is derived. The number of variables may be up to millions, which may exceed the capabilities of the solver.

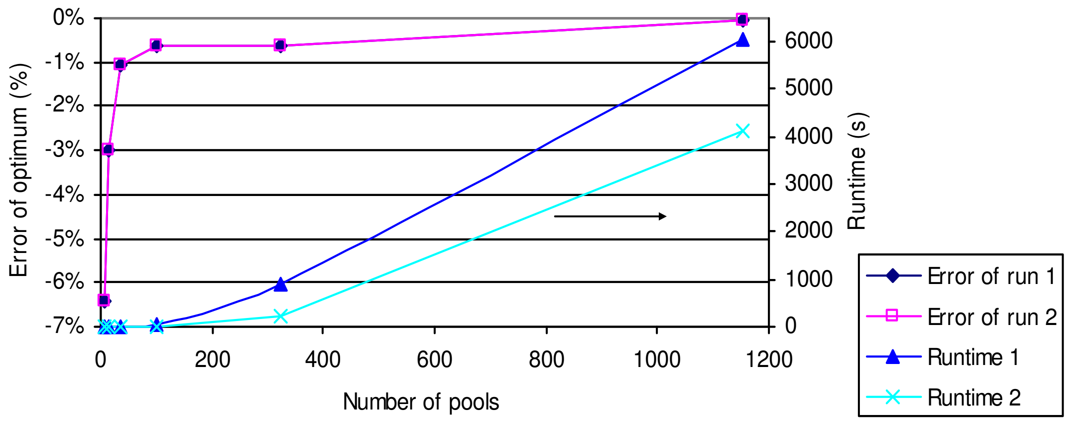


Figure 27 Results on problem Adhya 3 with two formulations

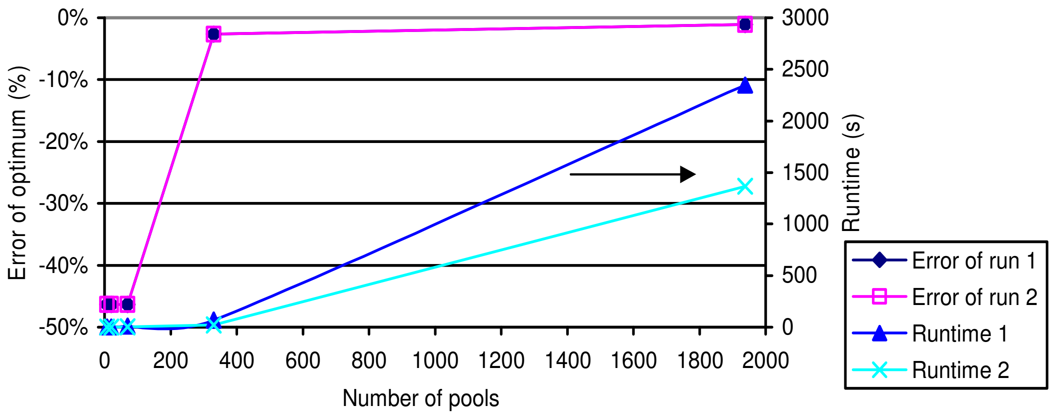


Figure 28 Results on problem Adhya 4 with two formulations

The runtimes of scenario 2 are always less than the corresponding one of scenario 1. The larger the number of discretized pools, the better the improvement is. This outcome is observed in all Adhya's problems.

6.4 Comparison to the results using the Global Solver in LINGO

The Global Solver is an add-on toolkit in LINGO. When the solver converges it provides a guaranteed global optimum through the technique of range bound and reduce embedded in the branch and bound method. The range bounding techniques, for example, are interval analysis and convex analysis. One of the reduce techniques is linearization.

All the investigated pooling problems are nonlinearly formulated without discretization and solved for the global optima. The formulations are in the appendices. Table 12 shows the results from these runs.

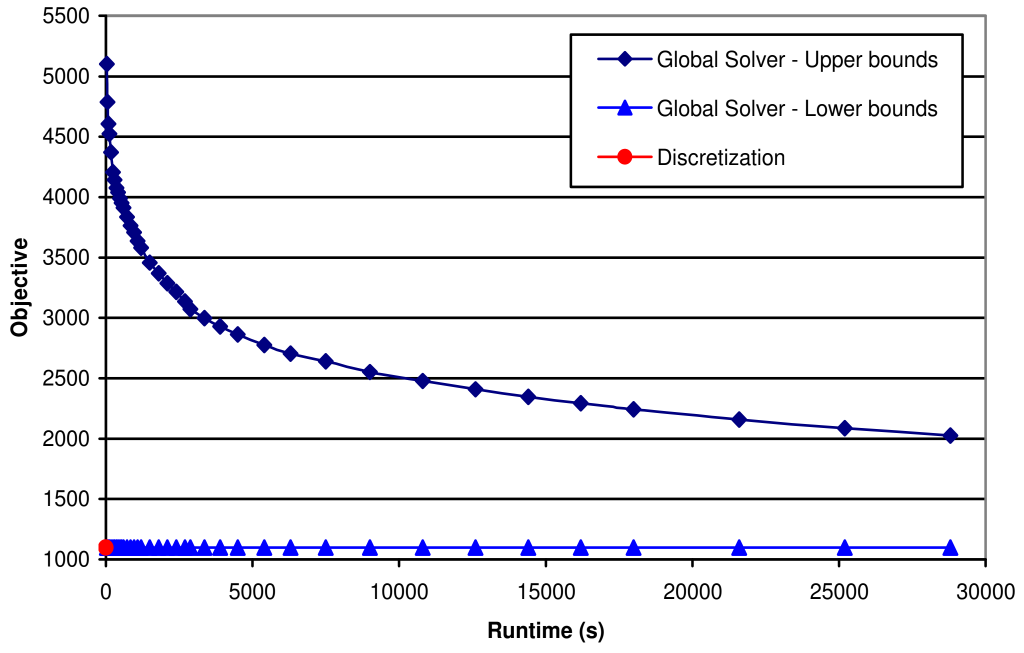
Table 12 Runs using Global Solver tool

Pooling Problems	Global optimum	Runtime (s)
Haverly 1	400	38
Haverly 2	600	< 1
Haverly 3	750	1
Ben-Tal 4	450	1
Adhya 1	549.803	196
Adhya 2	549.803	193
Adhya 3	561.045	6,446
Adhya 4	877.646	143

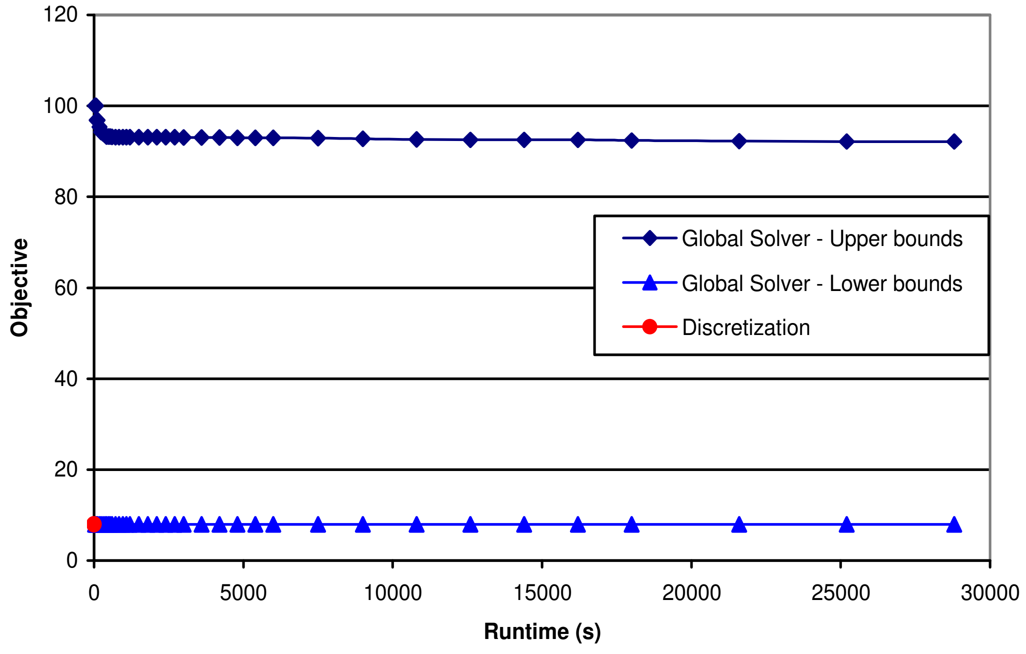
The exact global optima are found on these pooling problems within relatively short runtime.

However, the Global Solver does not always perform well. In some tested pooling problems that are not reported in Table 12, Global Solver converges very slowly, resulting in much more expensive calculations than discretization does.

Figure 28 depicts the calculation status with respect to the runtime when the LINGO Global Solver is used to solve Foulds' pooling problems. One run is done for each problem. At certain runtimes of a run, the found objective bound (upper bound) and best objective (lower bound) are recorded. The true global optimum is somewhere between the two bounds. Therefore, the gap between bounds should be as small as possible.

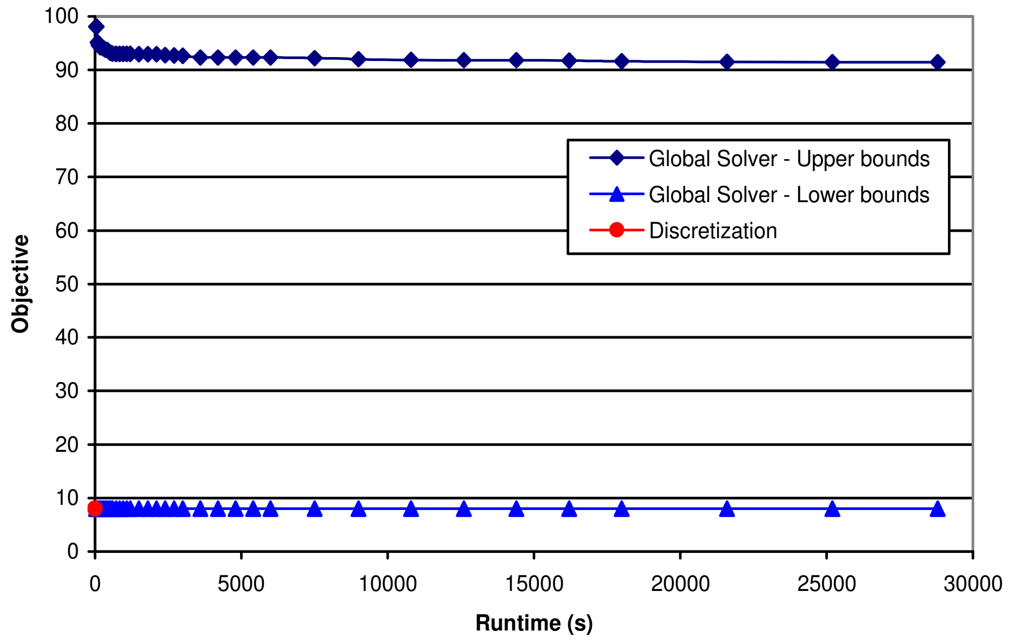


(a)

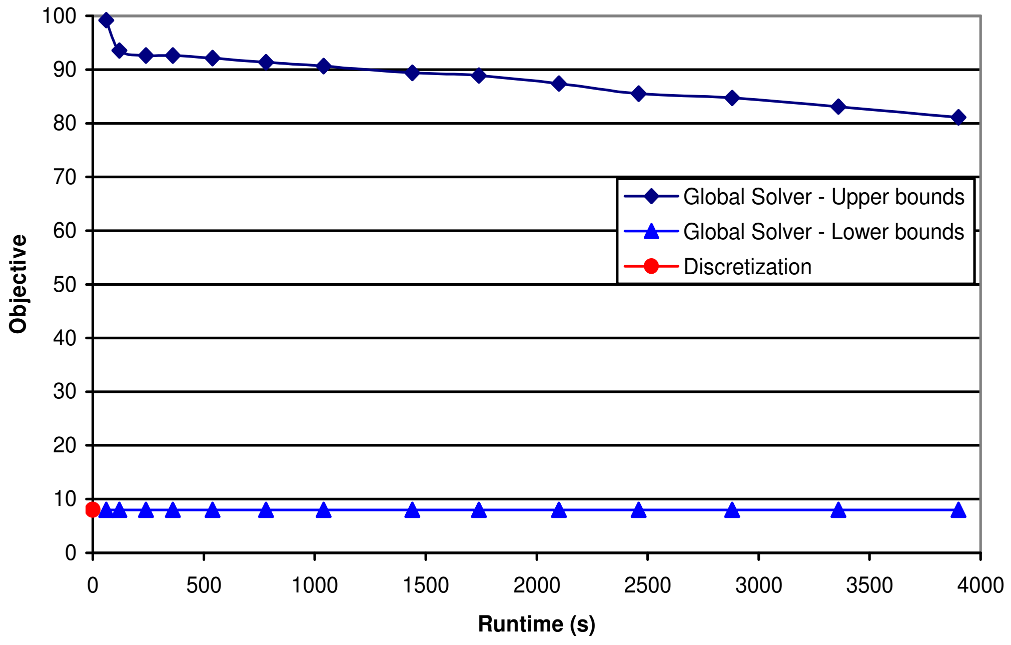


(b)

Figure 29 Calculations using Global Solver for Foulds' pooling problems
(a) *problem Foulds 2; (b) Foulds 3, (c) Foulds 4 and (d) Foulds 5*



(c)

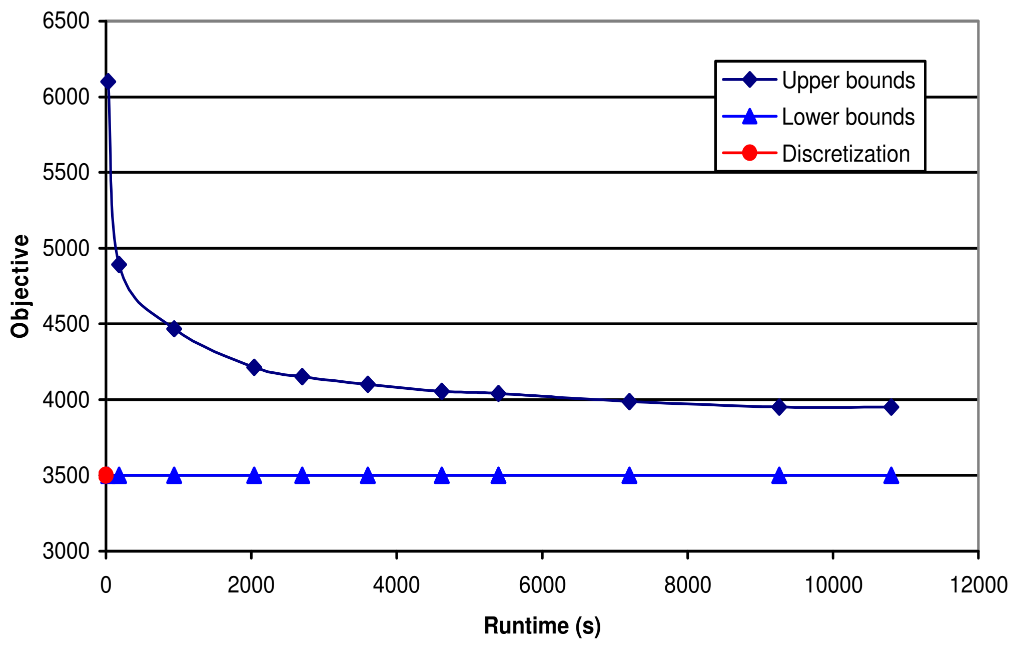


(d)

Figure 29 Continued

In

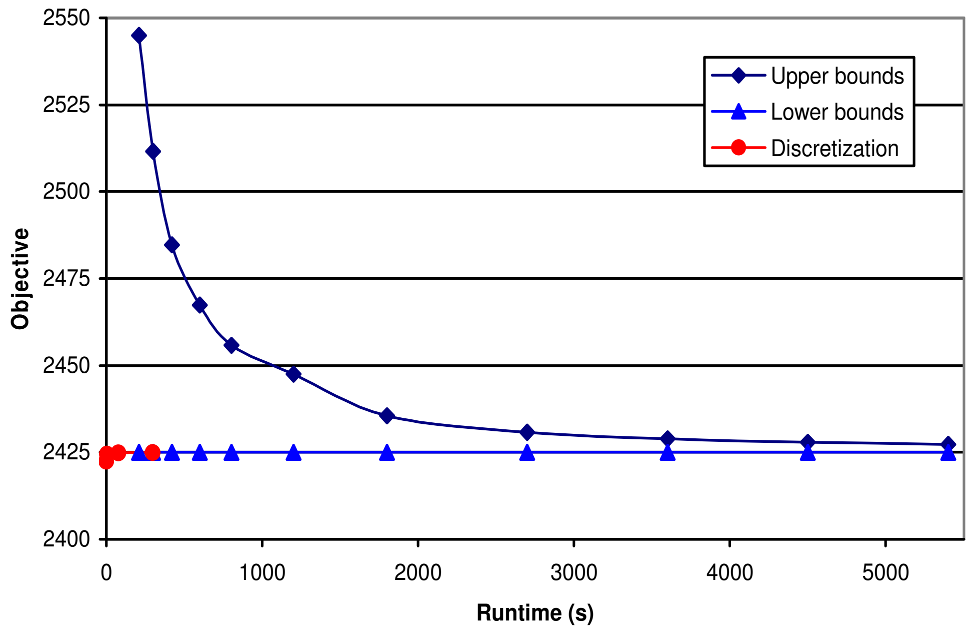
Figure 29, the top lines are the upper bounds; the bottom lines are lower bounds which are also the global optima in these cases. The best runs from the discretization approach are also plotted for comparison purposes. The results show that discretization approach finds the global optima of these problems in a very short runtime while the LINGO Global Solver needs much more runtime to locate the global optima. The same slow performance of Global Solver is also observed in solving problem Ben-Tal 5 and the example (Figure 30.). This discussion illustrates the merits of the new discretization approach.



(a)

Figure 30 Calculation using Global Solver for problem Ben-Tal 5 and the example

(a) problem Ben-Tal 5; (b) the example



(b)

Figure 30 Continued

CHAPTER VII
CONCLUSIONS AND RECOMMENDATION FOR FUTURE WORK

7.1 Conclusions

This work has introduced a new approach for the global solution of pooling problems. The approach is based on discretization of the pooling qualities to eliminate the nonconvex bilinear terms. An integer cut is added to provide equivalence between the original problem with a given number of pools and the discretized problem with an exceeding number of pools. The result is an MILP which can be solved globally. In order to reduce the size of the search space, an implicit enumeration scheme is proposed using fractions of flow rates from sources to pools. A convex hull representation is adopted and an algorithm is proposed for enumerating the pool qualities enclosed in the attainable region of pool qualities. Both exhaustive and implicit enumeration techniques are compared. It is shown that for problems with more than one pooling quality, the implicit enumeration typically results in a significant reduction in the problem dimensionality. Several test problems have been examined. The results indicate that the new discretization approach is capable of attaining global or near-global solutions while maintaining efficient computing times.

7.2 Recommendations for future work

The devised approach can be extended to address the following cases:

- Pooling problems with very large number of sources and qualities. It will be beneficial to identify the limits of the discretization approach and to add new elements to the procedure to overcome these limits.

- Pooling problems which allow flows among the pools. In such cases, a superstructure will have to be developed to embed all configurations of interest.

Then, the mathematical formulation will be developed. The discretization approach developed in this work may be generalized to address the resulting mathematical formulation.

- Pooling problems which are integrated with the rest of process optimization. In such cases, the exact values of the qualities of the process sources (intermediate streams) are not known. A decomposition approach may be used whereby process optimization is handled using process integration techniques while the pooling problem is handled through the approach developed in this work.
- Synthesis of water networks where the bilinear terms (similar to the quality*flow rate) are encountered.

LITERATURE CITED

1. Audet C, Hansen P, Jaumard B, Savard G. A Branch and Cut Algorithm for Nonconvex Quadratically Constrained Quadratic Programming. *Math. Program.* 2000; 878(1): 131 – 152.
2. Meyer C, Floudas CA. Global Optimization of a Combinatorially Complex Generalized Pooling Problem. *AIChE J.* 2006; 52: 1027-1037
3. Haverly CA. Studies of the Behaviour of Recursion for the Pooling Problem. *ACM SIGMAP Bull.* 1978; 25: 29-32.
4. Lasdon LS, Waren AD, Sarkar S, Palacios-Gomez F. Solving the Pooling Problem Using Generalized Reduced Gradient and Successive Linear Programming Algorithms. *ACM SIGMAP Bull*. 1979; 27: 9-15.
5. Griffith RE, Stewart RA. A Nonlinear Programming Technique for the Optimization of Continuous Processing Systems. *Management Science*. 1961; 7: 379-392.
6. Palacios-Gomez F, Lasdon LS, Engquist M. Nonlinear Optimization by Successive Linear Programming. *Management Science*. 1982; 28: 1106-1120.
7. Baker TE, Lasdon LS. Successive Linear Programming at Exxon. *Management Science.* 1985; 31: 264-274.
8. Greenberg HJ. Analysing the Pooling Problem. *ORSA J. Comput.* 1995; 7(2): 205-217.
9. Floudas CA, Aggarwal A. A Decomposition Strategy for Global Optimum Search in the Pooling Problem. *ORSA J.Comput*. 1990; 2(3): 225–235.
10. Floudas CA, Aggarwal A, Ciric AR. Global Optimum Search for Nonconvex NLP and MINLP Problems. *Computers and Chemical Engineering*. 1989; 13: 1117-1132.
11. Androulakis IP, Visweswaran V, Floudas C. Distributed Decomposition-Based Approaches in Global Optimization. *State of Art in Global Optimization: Computational Methods and Applications*; In Fouldas C, Parados PM; Kluwer Academic Publishers: Dordrecht. 1996: 1-17.
12. Foulds LR, Haugland D, Jornsten K. A Bilinear Approach to the Pooling Problem. *Optimization.* 1992; 24: 165-180.
13. McCormick GP, Computability of Global Solutions to Factorable Nonconvex Programs, Part I. Convex underestimating problems. *Math Prog.* 1976; 10: 147–175.

14. Al-Khayyal FA, Falk JE. Jointly Constrained Biconvex Programming, *Math Oper Res*. 1983; 8: 273–286.

15. Sherali HD, Alameddine A. A New Reformulation-Linearization Technique for Bilinear Programming Problems. *J. Global Optimiz.* 1992; 2: 379-410.

16. Ben-Tal A, Eiger G, Gershovitz V. Global Minimization by Reducing the Duality Gap. *Math. Program.* 1994; 63: 193-212.

17. Quesada I, Grossmann IE. Global Optimization of Bilinear Process Networks with Multicomponent Streams. *Comp. Chem. Eng.* 1995; 19: 1219-1242.

18. Adhya N, Sahinidis NV, Tawarmalani M. A Lagrangian Approach to the Pooling Problem. *Industrial Engineering, Chemistry Resources*. 1999; 38**:** 1956–1972.

19. Audet C, Brimberg J, Hansen P, Le Digabel S, Mladenovic N. Pooling Problem: Alternative Formulations And Solution Methods. *Management Science*. 2004; 50: 761 – 776.

20. Sahinidis N.V. Global optimization and constraint satisfaction: The Branch-and Reduce Approach. In Ch. Bliek et al., editor, *Global Optimization and Constraint Satisfaction.* Springer, Berlin. 2003**:** 1-16.

21. Sherali H. Tight Relaxations for Nonconvex Optimization Problems Using the Reformulation-Linearization/Convexification Technique (RLT). In Pardalos P, Romeijn H, editor, *Handbook of Global Optimization*. Dordrecht, Kluwer Academic Publishers. 2002; 2: 1–63.

22. Kallrath J. Mixed Integer Optimization in the Chemical Process Industry: Experience, Potential and Future Perspectives. *Chemical Engineering Research and Design*. 2000; 78(6): 809–822.

23. Sunday D. http://softsurfer.com/Archive/algorithm_0109/algorithm_0109.htm (accessed June 4, 2007).

24. Murty KG. *Operations Research: Deterministic Optimization Models.* Englewood Cliffs: Prentice-Hall. 1995: chapter 10.

25. Land AH, Doig, AG. An Automatic Method of Solving Discrete Programming Problems. *Econometrica*. 1960; 28: 497-520.

26. LINDO Systems Inc. *LINGO User's Guide*. LINDO Systems Inc. Chicago, IL. 2006.

APPENDIX A

LINGO CODES FOR ONE-QUALITY POOLING PROBLEMS

A 1 Haverly: Not linearized

The following two models are written for the pooling problem Haverly 1. For codes of problem Haverly 2 and Haverly 3, update D(1) to 600 and Cost(2) to 13 respectively.

```
SETS:
      SOURCE /1..3/: a, Cost;
      POOL /1..2/: b;
      PROD /1..2/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS

DATA:
      Cost =  6, 16, 10;
      a =  3,  1,  2;
      P =  9, 15;
      D = 100, 200;
      c = 2.5, 1.5;
ENDDATA

! Profit is maximized;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance for pools;
      X(1,1)=0; X(2,1)=0;
      @FOR(POOL(j): @SUM(SOURCE(i):X(i,j)) = @SUM(PROD(k):Y(j,k)));

! Mixing rule for pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));

! Constraints on product demands;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Mixing rule for products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
END
```

A 2 *Haverly: Exhaustively discretized*

```
SETS:
      SOURCE /1..3/: a, Cost;
      POOL /1..22/: b, f; ! The number of pools is equal to t+2 (see below for t value);
      PROD /1..2/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
DATA:
      Cost =  6, 16, 10;
      a =  3,  1,  2;
      P =  9, 15;
      D = 100, 200;
      c = 2.5, 1.5;
      t = 20; ! Number of discretized intervals;
ENDDATA
! Profit is maximized;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));
! Mass balance for pools;
      X(1,1)=0; X(2,1)=0;
      @FOR(POOL(j): @SUM(SOURCE(i):X(i,j)) = @SUM(PROD(k):Y(j,k)));
! Mixing rule for pools;
      @FOR(POOL(j)|j#GE#2: @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));
! Constraints on product demands;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));
! Mixing rule for products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
! Discretize the qualities of the pools;
      b(1) = a(3); ! Pool 1 is reserved for source 3;
      @FOR(POOL(j)|j#GE#2: b(j) = 1 + (j-2)*(3-1)/t);
! Use only 1 pool;
      @FOR(POOL(j)|j#GE#2:
            @BIN(f(j));
            @SUM(SOURCE(i):X(i,j)) <= @SUM(PROD(k):D(k))*f(j));
      @SUM(POOL(j)|j#GE#2:f(j))<=1;
END
```

A 3 Foulds 2: Not linearized

```
SETS:
      SOURCE /1..6/: a, Cost;
      POOL /1..4/: b;
      PROD /1..4/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
DATA:
      a =  3, 1,  2, 3.5, 1.5, 2.5;
      Cost = 6, 16, 10, 3, 13, 7;
      P = 9, 15, 6, 12;
      D = 100, 200, 100, 200;
      c = 2.5, 1.5, 3, 2;
ENDDATA
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance on the pools;
      X(3,1) = @SUM(PROD(k):Y(1,k)); !Pool 1 here represents pool 2 in Foulds' statement;
      X(6,2) = @SUM(PROD(k):Y(2,k)); !Pool 2 here represents pool 4 in Foulds' statement;
      @FOR(POOL(j)|j#GE#3: X(1,j)+X(2,j)+X(4,j)+X(5,j) = @SUM(PROD(k):Y(j,k)));

! Mass balance on the sale products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
END
```

```
A 4 Foulds 2: Exhaustive discretization
SETS: SOURCE /1..6/: a, Cost;
      POOL /1..2051/: b, f1, f2; ! The number of pools is equal to t+3 (see below for t value);
      PROD /1..4/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
DATA: a =  3, 1,  2, 3.5, 1.5, 2.5;
      Cost = 6, 16, 10, 3, 13, 7;
      P = 9, 15, 6, 12;
      D = 100, 200, 100, 200;
      c = 2.5, 1.5, 3, 2;
      t = 2048; ! Number of discretized intervals;
ENDDATA
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));
! Mass balance on the pools;
      X(3,1) = @SUM(PROD(k):Y(1,k)); !Pool 1 here represents pool 2 in Foulds' statement;
      X(6,2) = @SUM(PROD(k):Y(2,k)); !Pool 2 here represents pool 4 in Foulds' statement;
      @FOR(POOL(j)|j#GE#3: X(1,j)+X(2,j)+X(4,j)+X(5,j) = @SUM(PROD(k):Y(j,k)));
! Mass balance on the sale products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));
! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));
! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
! Linearize the problem by discretizing qualities of pools;
      b(1)= a(3); !Pool 1 is reserved for source 3;   b(2)= a(6); !Pool 2 is reserved for source 6;
      aU = @MAX(SOURCE(i):a(i));    aL = @MIN(SOURCE(i):a(i));
      @FOR(POOL(j)|j#GE#3:    b(j)=aL+(aU-aL)*(j-3)/t);
! Source 1&2 are forced to feed to one same pool;
      @FOR(POOL(j): @BIN(f1(j)); X(1,j) + X(2,j) <= @SUM(PROD(k):D(k))*f1(j));
      @SUM(POOL(j):f1(j))<=1;
! Source 4&5 are forced to feed to one same pool;
      @FOR(POOL(j):     @BIN(f2(j));      X(4,j) + X(5,j) <= @SUM(PROD(k):D(k))*f2(j));
      @SUM(POOL(j):f2(j))<=1;
END
```

A 5 Foulds 3: Not linearized

```
SETS:
      SOURCE /1..11/: a, Cost;
      POOL /1..8/: b;
      PROD /1..16/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
!DATA;
      @FOR(SOURCE(i):a(i) = 0.9+i*0.1);
      @FOR(SOURCE(i):Cost(i) = 21-i);
      @FOR(PROD(k):P(k) =  (41-k)/2);
      @FOR(PROD(k):D(k) = 1);
      @FOR(PROD(k): c(k) = 1+0.05*k);

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance on the pools;
      @FOR(POOL(j): @SUM(SOURCE(i)|(i#GE#j) #AND# (i#LE#j+3):X(i,j)) = @SUM(PROD(k):Y(j,k)));

! Mass balance on the sale products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
END
```

A 6 Foulds 3: Exhaustive discretization

```
SETS:
      SOURCE /1..11/: a, Cost;
      REALPOOL/1..8/;
      POOL /1..2049/: b; ! The number of pools is equal to t+1 (see below for t value);
      SWITCH (REALPOOL,POOL):f;
      PROD /1..16/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
!DATA;@FOR(SOURCE(i):a(i) = 0.9+i*0.1);
      @FOR(SOURCE(i):Cost(i) = 21-i);
      @FOR(PROD(k):P(k) =  (41-k)/2);
      @FOR(PROD(k):D(k) = 1);
      @FOR(PROD(k): c(k) = 1+0.05*k);
      t = 2048; ! Number of discretized intervals;
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));
! Mass balance on the pools;
      @FOR(REALPOOL(i): @FOR(POOL(j)|(b(j)#GE#a(i)) #AND# (b(j)#LE#a(i+3)):
                        @SUM(SOURCE(l)|(l#GE#i) #AND# (l#LE#i+3):X(l,j)) = @SUM(PROD(k):Y(j,k))));
! Mass balance on the sale products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));
! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));
! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
! Linearize the problem by discretizing qualities of pools;
      aU = @MAX(SOURCE(i):a(i));    aL = @MIN(SOURCE(i):a(i));
      @FOR(POOL(j): b(j)=aL+(aU-aL)*(j-1)/t);
! Only 8 or less pools needed;
      @FOR(REALPOOL(i): @FOR(POOL(j)|(b(j)#LE#a(i+3)) #AND# (b(j)#GE#a(i)):
                              @BIN(f(i,j));
                              @SUM(SOURCE(k)|k#LE#4:X(i+k-1,j)) <= @SUM(PROD(l):D(l))*f(i,j));
                        @SUM(POOL(j):f(i,j))<=1);
END
```

A 7 Foulds 4: Not linearized

```
SETS:
      SOURCE /1..11/: a, Cost;
      POOL /1..8/: b;
      PROD /1..16/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
!DATA;
      @FOR(SOURCE(i):   a(i) = 0.9+i*0.1;
                        Cost(i) = 21-i);
      @FOR(PROD(k):P(k) =  (41-k)/2;
                   D(k) = 1;
                   c(k) = 1+0.05*k);

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance on the pools;
      ! Pool 1;   X(1,1)+X(4,1)+X(7,1)+X(10,1)= @SUM(PROD(k):Y(1,k));
      ! Pool 2;   X(2,2)+X(5,2)+X(8,2)+X(11,2)= @SUM(PROD(k):Y(2,k));
      ! Pool 3;   X(2,3)+X(3,3)+X(5,3)+X(6,3) = @SUM(PROD(k):Y(3,k));
      ! Pool 4;   X(3,4)+X(4,4)+X(6,4)+X(7,4) = @SUM(PROD(k):Y(4,k));
      ! Pool 5;   X(3,5)+X(5,5)+X(6,5)+X(8,5) = @SUM(PROD(k):Y(5,k));
      ! Pool 6;   X(4,6)+X(6,6)+X(7,6)+X(9,6) = @SUM(PROD(k):Y(6,k));
      ! Pool 7;   X(4,7)+X(7,7)+X(8,7)+X(9,7) = @SUM(PROD(k):Y(7,k));
      ! Pool 8;   X(6,8)+X(8,8)+X(9,8)+X(10,8)= @SUM(PROD(k):Y(8,k));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
END
```

A 8 Foulds 4: Exhaustive discretization

```
SETS:
      SOURCE /1..11/: a, Cost;
      REALPOOL/1..8/;
      POOL /1..4104/: b; ! Number of pools is equal to 8*(t+1) (see below for t value);
      SWITCH (REALPOOL,POOL):f;
      PROD /1..16/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
!DATA;
      @FOR(SOURCE(i):   a(i) = 0.9+i*0.1;
                        Cost(i) = 21-i);
      @FOR(PROD(k):P(k) =  (41-k)/2;
                   D(k) = 1;
                   c(k) = 1+0.05*k);
      t = 512; ! Number of discretized intervals;

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance on the pools;
      ! Pool 1;   @FOR(POOL(j)|j#LE#t+1: X(1,j)+X(4,j)+X(7,j)+X(10,j)= @SUM(PROD(k):Y(j,k)));
      ! Pool 2;   @FOR(POOL(j)|(j#GT#t+1) #AND# (j#LE#2*(t+1)):
                        X(2,j)+X(5,j)+X(8,j)+X(11,j)= @SUM(PROD(k):Y(j,k)));
      ! Pool 3;   @FOR(POOL(j)|(j#GT#2*(t+1)) #AND# (j#LE#3*(t+1)):
                        X(2,j)+X(3,j)+X(5,j)+X(6,j)= @SUM(PROD(k):Y(j,k)));
      ! Pool 4;   @FOR(POOL(j)|(j#GT#3*(t+1)) #AND# (j#LE#4*(t+1)):
                        X(3,j)+X(4,j)+X(6,j)+X(7,j)= @SUM(PROD(k):Y(j,k)));
      ! Pool 5;   @FOR(POOL(j)|(j#GT#4*(t+1)) #AND# (j#LE#5*(t+1)):
                        X(3,j)+X(5,j)+X(6,j)+X(8,j)= @SUM(PROD(k):Y(j,k)));
      ! Pool 6;   @FOR(POOL(j)|(j#GT#5*(t+1)) #AND# (j#LE#6*(t+1)):
                        X(4,j)+X(6,j)+X(7,j)+X(9,j)= @SUM(PROD(k):Y(j,k)));
      ! Pool 7;   @FOR(POOL(j)|(j#GT#6*(t+1)) #AND# (j#LE#7*(t+1)):
                        X(4,j)+X(7,j)+X(8,j)+X(9,j)= @SUM(PROD(k):Y(j,k)));
      ! Pool 8;   @FOR(POOL(j)|(j#GT#7*(t+1)): X(6,j)+X(8,j)+X(9,j)+X(10,j)= @SUM(PROD(k):Y(j,k)));
! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));

! Linearize the problem by discretizing qualities of pools;
      aU = @MAX(SOURCE(i):a(i));    aL = @MIN(SOURCE(i):a(i));
      @FOR(REALPOOL(l): @FOR(POOL(j)|j#LE#t+1:  b(j+(l-1)*(t+1))=aL+(aU-aL)*(j-1)/t));

! Constraint on numbers of real pools and flows;
      MAXD = @SUM(PROD(k):D(k));
! Pool 1;   @FOR(POOL(j)|j#LE#t+1:
                  @BIN(f(1,j));     X(1,j)+X(4,j)+X(7,j)+X(10,j) <= MAXD*f(1,j));
! Pool 2;   @FOR(POOL(j)|(j#GT#t+1) #AND# (j#LE#2*(t+1)):
                  @BIN(f(2,j));     X(2,j)+X(5,j)+X(8,j)+X(11,j) <= MAXD*f(2,j));
! Pool 3;   @FOR(POOL(j)|(j#GT#2*(t+1)) #AND# (j#LE#3*(t+1)):
                  @BIN(f(3,j));     X(2,j)+X(3,j)+X(5,j)+X(6,j) <= MAXD*f(3,j));
! Pool 4;   @FOR(POOL(j)|(j#GT#3*(t+1)) #AND# (j#LE#4*(t+1)):
                  @BIN(f(4,j));     X(3,j)+X(4,j)+X(6,j)+X(7,j) <= MAXD*f(4,j));
! Pool 5;   @FOR(POOL(j)|(j#GT#4*(t+1)) #AND# (j#LE#5*(t+1)):
                  @BIN(f(5,j));     X(3,j)+X(5,j)+X(6,j)+X(8,j) <= MAXD*f(5,j));
! Pool 6;   @FOR(POOL(j)|(j#GT#5*(t+1)) #AND# (j#LE#6*(t+1)):
                  @BIN(f(6,j));     X(4,j)+X(6,j)+X(7,j)+X(9,j) <= MAXD*f(6,j));
! Pool 7;   @FOR(POOL(j)|(j#GT#6*(t+1)) #AND# (j#LE#7*(t+1)):
                  @BIN(f(7,j));     X(4,j)+X(7,j)+X(8,j)+X(9,j) <= MAXD*f(7,j));
! Pool 8;   @FOR(POOL(j)|(j#GT#7*(t+1)):
                  @BIN(f(8,j));     X(6,j)+X(8,j)+X(9,j)+X(10,j) <= MAXD*f(8,j));
      @FOR(REALPOOL(l):@SUM(POOL(j):f(l,j))<=1);
END
```

A 9 *Foulds 5: Not linearized*

```
SETS:
      SOURCE /1..11/: a, Cost;
      POOL /1..4/: b;
      PROD /1..16/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
!DATA;
      @FOR(SOURCE(i):   a(i) = 0.9+i*0.1;
                        Cost(i) = 21-i);
      @FOR(PROD(k):P(k) =  (41-k)/2;
                   D(k) = 1;
                   c(k) = 1+0.05*k);
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance on the pools;
! Pool 1;   @SUM(SOURCE(i)|i#LE#4:X(i,1))+@SUM(SOURCE(i)|i#GE#8:X(i,1))= @SUM(PROD(k):Y(1,k));
! Pool 2;   @SUM(SOURCE(i)|(i#GE#2)#AND#(i#LE#10):X(i,2))-X(6,2)= @SUM(PROD(k):Y(2,k));
! Pool 3;   @SUM(SOURCE(i)|i#GE#4:X(i,3))= @SUM(PROD(k):Y(3,k));
! Pool 4;   @SUM(SOURCE(i)|i#LE#8:X(i,4))= @SUM(PROD(k):Y(4,k));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) = D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
END
```

A 10 Foulds 5: Exhaustive discretization

```
SETS:
      SOURCE /1..11/: a, Cost;
      REALPOOL/1..4/;
      POOL /1..44/: b; ! Number of pools is equal to 4*(t+1) (see below for (t+1) value);
      SWITCH (REALPOOL,POOL):f;
      PROD /1..16/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
!DATA;
      @FOR(SOURCE(i):   a(i) = 0.9+i*0.1;
                        Cost(i) = 21-i);
      @FOR(PROD(k):P(k) =  (41-k)/2;
                   D(k) = 1;
                   c(k) = 1+0.05*k);
      t = 10; ! Number of discretized intervals;

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance on the pools;

! Pool 1;   @FOR(POOL(j)|j#LE#t+1:
                  @SUM(SOURCE(i)|i#LE#4:X(i,j))+@SUM(SOURCE(i)|i#GE#8:X(i,j))= @SUM(PROD(k):Y(j,k)));

! Pool 2;   @FOR(POOL(j)|(j#GT#(t+1)) #AND# (j#LE#2*(t+1)):
                  SUM(SOURCE(i)|(i#GE#2)#AND#(i#LE#10):X(i,j))-X(6,j)= @SUM(PROD(k):Y(j,k)));

! Pool 3;   @FOR(POOL(j)|(j#GT#2*(t+1)) #AND# (j#LE#3*(t+1)):
                  @SUM(SOURCE(i)|i#GE#4:X(i,j))= @SUM(PROD(k):Y(j,k)));

! Pool 4;   @FOR(POOL(j)|(j#GT#3*(t+1)):
                  @SUM(SOURCE(i)|i#LE#8:X(i,j))= @SUM(PROD(k):Y(j,k)));
```

```
! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) = D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));

! Linearize the problem by discretizing qualities of pools;
      aU = @MAX(SOURCE(i):a(i));
      aL = @MIN(SOURCE(i):a(i));
      @FOR(REALPOOL(l): @FOR(POOL(j)|j#LE#t+1: b(j+(l-1)*(t+1))=aL+(aU-aL)*(j-1)/t));

! Constraints on numbers of real pools and flows;
      MAXD = @SUM(PROD(k):D(k));

! Pool 1;   @FOR(POOL(j)|j#LE#t+1:
                  @BIN(f(1,j));
                  @SUM(SOURCE(i)|i#LE#4:X(i,j))+@SUM(SOURCE(i)|i#GE#8:X(i,j)) <= MAXD*f(1,j));

! Pool 2;   @FOR(POOL(j)|(j#GT#(t+1)) #AND# (j#LE#2*(t+1)):
                  @BIN(f(2,j));
                  @SUM(SOURCE(i)|(i#GE#2)#AND#(i#LE#10):X(i,j))-X(6,j)<= MAXD*f(2,j));

! Pool 3;   @FOR(POOL(j)|(j#GT#2*(t+1)) #AND# (j#LE#3*(t+1)):
                  @BIN(f(3,j));
                  @SUM(SOURCE(i)|i#GE#4:X(i,j))<= MAXD*f(3,j));

! Pool 4;   @FOR(POOL(j)|(j#GT#3*(t+1)) #AND# (j#LE#4*(t+1)):
                  @BIN(f(4,j));
                  @SUM(SOURCE(i)|i#LE#8:X(i,j)) <= MAXD*f(4,j));

      @FOR(REALPOOL(l):@SUM(POOL(j):f(l,j))<=1);
END
```

A 11 *Ben-Tal 4: Not linearized*

```
SETS:
      SOURCE /1..4/: a, Cost;
      POOL /1..2/: b;
      PROD /1..2/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
DATA:
      a =  3, 1, 1, 2;
      Cost = 6, 15, 16, 10;
      P = 9, 15;
      D = 100, 200;
      c = 2.5, 1.5;
ENDDATA
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Constraint on source 2's flow rate;
      X(2,1) <= 50;

! Mass balance on the pools;
      X(4,2) = @SUM(PROD(k):Y(2,k));
      @FOR(SOURCE(i)|i#LE#3: X(i,2)=0);
      @SUM(SOURCE(i)|i#LE#3: X(i,1)) = @SUM(PROD(k):Y(1,k));
      X(4,1)=0;

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
END
```

A 12 Ben-Tal 4: Exhaustive discretization

```
SETS: SOURCE /1..4/: a, Cost;
      POOL /1..82/: b, f; ! The number of pools is equal to t+2 (see below for t value);
      PROD /1..2/: P, D, c;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
DATA: a =  3, 1, 1, 2;
      Cost = 6, 15, 16, 10;
      P = 9, 15;
      D = 100, 200;
      c = 2.5, 1.5;
      t = 80; ! Number of discretized intervals;
ENDDATA
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));
! Constraint on source 2's flow rate;
      @SUM(POOL(j)|j#LE#t+1:X(2,j)) <= 50;
! Mass balance on the pools;
      X(4,t+2) = @SUM(PROD(k):Y(t+2,k));
      @FOR(SOURCE(i)|i#LE#3: X(i,t+2)=0);
      @FOR(POOL(j)|j#LE#t+1:  X(4,j)=0;
                              @SUM(SOURCE(i)|i#LE#3: X(i,j)) = @SUM(PROD(k):Y(j,k)));
! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));
! Quality blending for the pools;
      @FOR(POOL(j)|j#LE#t+1: @SUM(SOURCE(i): a(i)*X(i,j)) = b(j)*@SUM(SOURCE(i):X(i,j)));
! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b(j)*Y(j,k)) <= c(k)*@SUM(POOL(j):Y(j,k)));
! Linearize the problem by discretizing qualities of pools;
      b(t+2)= a(4);
      aU = @MAX(SOURCE(i):a(i));    aL = @MIN(SOURCE(i):a(i));
      @FOR(POOL(j)|j#LE#t+1:  b(j)=aL+(aU-aL)*(j-1)/t);
! Use only 1 pool for first three sources;
      @FOR(POOL(j)|j#LE#t+1:  @BIN(f(j)); @SUM(SOURCE(i)|i#LE#3: X(i,j)) <= @SUM(PROD(k):D(k))*f(j));
      @SUM(POOL(j):f(j))<=1;
END
```

APPENDIX B

LINGO CODES FOR MULTIPLE-QUALITY POOLING PROBLEMS

B 1 Ben-Tal 5: Not linearized

```
SETS: SOURCE /1..5/: a1, a2, Cost;
      POOL /1..4/: b1, b2;
      PROD /1..5/: c1, c2, D, P;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
DATA: a1 =  3, 1, 1, 1.5, 2;
      a2 =  1, 3, 2.5, 2.5, 2.5;
      Cost = 6, 16, 15, 12, 10;
      D = 100, 200, 100, 100, 100;
      c1 = 2.5, 1.5, 2, 2, 2;
      c2 = 2, 2.5, 2.6, 2, 2;
      P = 18, 15, 19, 16, 14;
ENDDATA
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Constraint on the 3th source's flow rate;
      @SUM(POOL(j):X(3,j)) <= 50;

! Mass balance on the pools;
      X(5,4) = @SUM(PROD(k):Y(4,k));
      @FOR(POOL(j): @SUM(SOURCE(i)|i#LE#4: X(i,j)) = @SUM(PROD(k):Y(j,k)));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a1(i)*X(i,j)) = b1(j)*@SUM(SOURCE(i):X(i,j)));
      @FOR(POOL(j): @SUM(SOURCE(i): a2(i)*X(i,j)) = b2(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b1(j)*Y(j,k)) <= c1(k)*@SUM(POOL(j):Y(j,k)));
      @FOR(PROD(k): @SUM(POOL(j): b2(j)*Y(j,k)) <= c2(k)*@SUM(POOL(j):Y(j,k)));
END
```

B 2 Ben-Tal 5: Exhaustive discretization

```
SETS:
      SOURCE /1..5/: a1, a2, Cost;
      POOL /1..442/: b1, b2, f; ! The number of pools is (t1+1)*(t2+1)+1 (see below for t1, t2);
      PROD /1..5/: c1, c2, D, P;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
DATA:
      a1 =  3, 1, 1, 1.5, 2;
      a2 =  1, 3, 2.5, 2.5, 2.5;
      Cost = 6, 16, 15, 12, 10;
      D = 100, 200, 100, 100, 100;
      c1 = 2.5, 1.5, 2, 2, 2;
      c2 = 2, 2.5, 2.6, 2, 2;
      P = 18, 15, 19, 16, 14;
      t1 = 20;! Number of discretized intervals for quality 1;
      t2 = 20;! Number of discretized intervals for quality 2;
ENDDATA

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Linearize the problem by discretizing qualities of pools;
! Discretize quality 1 for pools;
      b1((t1+1)*(t2+1)+1)= a1(5);
      a1U = @MAX(SOURCE(i):a1(i));
      a1L = @MIN(SOURCE(i):a1(i));
      @FOR(POOL(j)|j#LE#t2+1: b1(j)=a1L+(a1U-a1L)*(j-1)/t2;
                              @FOR(POOL(l)|l#LE#t1:b1(j+l*(t2+1))=b1(j)));
! Discretize quality 2 for pools;
      b2((t1+1)*(t2+1)+1)= a2(5);
      a2U = @MAX(SOURCE(i):a2(i));
      a2L = @MIN(SOURCE(i):a2(i));
      @FOR(POOL(j)|j#LE#t2+1: b2(j)=a2L;
                              @FOR(POOL(l)|l#LE#t1:b2(j+l*(t2+1))=a2L+(a2U-a2L)*l/t1));
```

```
! Constraint on the 3th source's flow rate;
      @SUM(POOL(j)|j#LE#(t1+1)*(t2+1):X(3,j)) <= 50;

! Mass balance on the pools;
      X(5,(t1+1)*(t2+1)+1) = @SUM(PROD(k):Y((t1+1)*(t2+1)+1,k));
      @FOR(POOL(j)|j#LE#(t1+1)*(t2+1): @SUM(SOURCE(i)|i#LE#4: X(i,j)) = @SUM(PROD(k):Y(j,k)));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a1(i)*X(i,j)) = b1(j)*@SUM(SOURCE(i):X(i,j)));
      @FOR(POOL(j): @SUM(SOURCE(i): a2(i)*X(i,j)) = b2(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b1(j)*Y(j,k)) <= c1(k)*@SUM(POOL(j):Y(j,k)));
      @FOR(PROD(k): @SUM(POOL(j): b2(j)*Y(j,k)) <= c2(k)*@SUM(POOL(j):Y(j,k)));

! Use only 3 pools;
      @FOR(POOL(j)|j#LE#(t1+1)*(t2+1):    @BIN(f(j));
                                          @SUM(SOURCE(i):X(i,j)) <= @SUM(PROD(k):D(k))*f(j));
      @SUM(POOL(j)|j#LE#(t1+1)*(t2+1):f(j))<=3;

END
```

B 3 Ben-Tal 5: Implicit discretization

```
SETS:
      SOURCE /1..5/: a1, a2, Cost;
      POOL /1..47906/: b1, b2, f; ! The number of pools is (t+1)*(t+2)*(t+3)/6+1 (see below for t);
      PROD /1..5/: c1, c2, D, P;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
DATA:
      a1 =  3, 1, 1, 1.5, 2;
      a2 =  1, 3, 2.5, 2.5, 2.5;
      Cost = 6, 16, 15, 12, 10;
      D = 100, 200, 100, 100, 100;
      c1 = 2.5, 1.5, 2, 2, 2;
      c2 = 2, 2.5, 2.6, 2, 2;
      P = 18, 15, 19, 16, 14;
      t = 64;
ENDDATA
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Linearize the problem by discretizing qualities of pools;
      b1(1)=a1(5);!Pool 1 in this formulation represents pool 4 in Ben-Tal's statement;
      b2(1)=a2(5);

@FOR(POOL(u)|u#LE#t+1:
      @FOR(POOL(v)|v#LE#t-u+2:
            @FOR(POOL(r)|r#LE#t-v-u+3:
 b1(1+(t+2)*(t+3)*(t+4)/6-((t-u+2))*((t-u+3))*((t-u+4))/6-(t+2)*(t+3)/2+(v-1)*((t-u+2))-(v-1)*(v-2)/2+r)
= a1(1)*(u-1)/t + a1(2)*(v-1)/t + a1(3)*(r-1)/t + a1(4)*(1-(u-1)/t-(v-1)/t-(r-1)/t);

 b2(1+(t+2)*(t+3)*(t+4)/6-((t-u+2))*((t-u+3))*((t-u+4))/6-(t+2)*(t+3)/2+(v-1)*((t-u+2))-(v-1)*(v-2)/2+r)
= a2(1)*(u-1)/t +a2(2)*(v-1)/t +a2(3)*(r-1)/t +a2(4)*(1-(u-1)/t-(v-1)/t-(r-1)/t))));

! Constraint on the 3th source's flow rate;
      @SUM(POOL(j)|j#GE#2: X(3,j)) <= 50;
```

```
! Mass balance on the pools;
      X(5,1) = @SUM(PROD(k):Y(1,k));
      @FOR(POOL(j)|j#GE#2: @SUM(SOURCE(i)|i#LE#4: X(i,j)) = @SUM(PROD(k):Y(j,k)));

! Constraints on product demands;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a1(i)*X(i,j)) = b1(j)*@SUM(SOURCE(i):X(i,j)));
      @FOR(POOL(j): @SUM(SOURCE(i): a2(i)*X(i,j)) = b2(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b1(j)*Y(j,k)) <= c1(k)*@SUM(POOL(j):Y(j,k)));
      @FOR(PROD(k): @SUM(POOL(j): b2(j)*Y(j,k)) <= c2(k)*@SUM(POOL(j):Y(j,k)));

! Use only 3 pools for sources 1,2,3&4;
      @FOR(POOL(j)|j#GE#2:
            @BIN(f(j));
            @SUM(SOURCE(i):X(i,j)) <= @SUM(PROD(k):D(k))*f(j));
      @SUM(POOL(j): f(j))<=3;
END
```

B 4 Ben-Tal 5: Implicit discretization – Formulation in flow rate fraction

```
SETS:
      SOURCE /1..5/: a1, a2, Cost;
      POOL /1..287/: b1, b2, f, Z; ! The number of pools is (t+1)*(t+2)*(t+3)/6+1 (see below for t);
      PROD /1..5/: c1, c2, D, P;
      INPOOL (SOURCE,POOL): x;
      OUTPOOL (POOL,PROD): Y;
ENDSETS

DATA:
      a1 =  3, 1, 1, 1.5, 2;
      a2 =  1, 3, 2.5, 2.5, 2.5;
      Cost = 6, 16, 15, 12, 10;
      D = 100, 200, 100, 100, 100;
      c1 = 2.5, 1.5, 2, 2, 2;
      c2 = 2, 2.5, 2.6, 2, 2;
      P = 18, 15, 19, 16, 14;
      t = 10;
ENDDATA

! Maximize the revenue;
MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):x(i,j)*Z(j)));

! Linearize the problem by discretizing qualities of pools;
      b1(1)=a1(5);!Pool 1 in this formulation represents pool 4 in Ben-Tal's statement;
      b2(1)=a2(5);
      @FOR(SOURCE(i)|i#LE#4: x(i,1) = 0);
      x(5,1)=1;

@FOR(POOL(u)|u#LE#t+1:
      @FOR(POOL(v)|v#LE#t-u+2:
            @FOR(POOL(r)|r#LE#t-v-u+3:
 x(1,1+(t+2)*(t+3)*(t+4)/6-((t-u+2))*((t-u+3))*((t-u+4))/6-(t+2)*(t+3)/2+(v-1)*((t-u+2))-(v-1)*(v-2)/2+r)
= (u-1)/t;
```

```
 x(2,1+(t+2)*(t+3)*(t+4)/6-((t-u+2))*((t-u+3))*((t-u+4))/6-(t+2)*(t+3)/2+(v-1)*((t-u+2))-(v-1)*(v-2)/2+r)
= (v-1)/t;

 x(3,1+(t+2)*(t+3)*(t+4)/6-((t-u+2))*((t-u+3))*((t-u+4))/6-(t+2)*(t+3)/2+(v-1)*((t-u+2))-(v-1)*(v-2)/2+r)
= (r-1)/t;

 x(4,1+(t+2)*(t+3)*(t+4)/6-((t-u+2))*((t-u+3))*((t-u+4))/6-(t+2)*(t+3)/2+(v-1)*((t-u+2))-(v-1)*(v-2)/2+r)
= 1-(u-1)/t-(v-1)/t-(r-1)/t;

 x(5,1+(t+2)*(t+3)*(t+4)/6-((t-u+2))*((t-u+3))*((t-u+4))/6-(t+2)*(t+3)/2+(v-1)*((t-u+2))-(v-1)*(v-2)/2+r)
= 0;

 b1(1+(t+2)*(t+3)*(t+4)/6-((t-u+2))*((t-u+3))*((t-u+4))/6-(t+2)*(t+3)/2+(v-1)*((t-u+2))-(v-1)*(v-2)/2+r)
= a1(1)*(u-1)/t + a1(2)*(v-1)/t + a1(3)*(r-1)/t + a1(4)*(1-(u-1)/t-(v-1)/t-(r-1)/t);

 b2(1+(t+2)*(t+3)*(t+4)/6-((t-u+2))*((t-u+3))*((t-u+4))/6-(t+2)*(t+3)/2+(v-1)*((t-u+2))-(v-1)*(v-2)/2+r)
= a2(1)*(u-1)/t + a2(2)*(v-1)/t + a2(3)*(r-1)/t + a2(4)*(1-(u-1)/t-(v-1)/t-(r-1)/t))));

! Constraint on the 3th source's flow rate;
      @SUM(POOL(j)|j#GE#2: x(3,j)*Z(j)) <= 50;

! Mass balance on the pools;
      @FOR(POOL(j): Z(j) = @SUM(PROD(k):Y(j,k)));

! Constraints on product demands;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b1(j)*Y(j,k)) <= c1(k)*@SUM(POOL(j):Y(j,k)));
      @FOR(PROD(k): @SUM(POOL(j): b2(j)*Y(j,k)) <= c2(k)*@SUM(POOL(j):Y(j,k)));

! Use only 3 pools for sources 1,2,3&4;
      @FOR(POOL(j)|j#GE#2:
            @BIN(f(j));
            Z(j) <= @SUM(PROD(k):D(k))*f(j));
      @SUM(POOL(j): f(j))<=3;
END
```

B 5 *The example: Not linearized*

```
SETS:
      SOURCE /1..3/: a1, a2, S, Cost;
      POOL /1..2/: b1, b2;
      PROD /1..3/: c1, c2, D, P;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS
DATA: a1 = 82, 92, 82 ;
      a2 =  1,  2, 1.5;
      S = 100,200,100;
      D = 100, 100, 200;
      c1 = 84, 87, 90;
      c2 = 1.9, 2, 2;
      Cost = 7, 9, 6;
      P = 10, 15, 17;
ENDDATA
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k)))-@SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Constraints on source's capacities;
      @FOR(SOURCE(i): @SUM(POOL(j):X(i,j)) <= S(i));

! Mass balance on the pools;
      @FOR(POOL(j):@SUM(SOURCE(i):X(i,j)) = @SUM(PROD(k):Y(j,k)));

! Mass balance on the products' demands;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a1(i)*X(i,j)) = b1(j)*@SUM(SOURCE(i):X(i,j)));
      @FOR(POOL(j): @SUM(SOURCE(i): a2(i)*X(i,j)) = b2(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b1(j)*Y(j,k)) >= c1(k)*@SUM(POOL(j):Y(j,k)));
      @FOR(PROD(k): @SUM(POOL(j): b2(j)*Y(j,k)) <= c2(k)*@SUM(POOL(j):Y(j,k)));
END
```

B 6 The example: Exhaustive discretization

```
SETS:
      SOURCE /1..3/: a1, a2, S, Cost;
      POOL /1..1681/: b1, b2, f; ! The number of pools is (t1+1)*(t2+1) (see below for t1, t2);
      PROD /1..3/: c1, c2, D, P;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS

DATA:
      a1 = 82, 92, 82 ;
      a2 =  1,  2, 1.5;
      a1U = 92; a1L = 82; ! Maximum and Minimum of a1;
      a2U =  2; a2L =  1; ! Maximum and Minimum of a2;
      S = 100,200,100;
      D = 100, 100, 200;
      c1 = 84, 87, 90;
      c2 = 1.9, 2, 2;
      Cost = 7, 9, 6;
      P = 10, 15, 17;
      t1=40;! Number of discretized intervals for quality 1;
      t2=40;! Number of discretized intervals for quality 2;
ENDDATA

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k)))-@SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Constraints on source's capacities;
      @FOR(SOURCE(i): @SUM(POOL(j):X(i,j)) <= S(i));

! Mass balance on the pools;
      @FOR(POOL(j):@SUM(SOURCE(i):X(i,j)) = @SUM(PROD(k):Y(j,k)));

! Mass balance on the products' demands;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));
```

```
! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a1(i)*X(i,j))  = b1(j)*@SUM(SOURCE(i):X(i,j)));
      @FOR(POOL(j): @SUM(SOURCE(i): a2(i)*X(i,j))  = b2(j)*@SUM(SOURCE(i):X(i,j)));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b1(j)*Y(j,k))  >= c1(k)*@SUM(POOL(j):Y(j,k)));
      @FOR(PROD(k): @SUM(POOL(j): b2(j)*Y(j,k))  <= c2(k)*@SUM(POOL(j):Y(j,k)));

! Linearize the problem by specifying qualities of pools;

! Assign quality 1 for pools;
      @FOR(POOL(u)|u#LE#t2+1: b1(u)=a1L;
                              @FOR(POOL(v)|v#LE#t1:b1(u+v*(t2+1))=a1L+(a1U-a1L)*v/t1));
! Assign quality 2 for pools;
      @FOR(POOL(u)|u#LE#t2+1: b2(u)=a2L+(a2U-a2L)*(u-1)/t2;
                              @FOR(POOL(v)|v#LE#t1:b2(u+v*(t2+1))=b2(u)));

! Use only 2 pools;
      @FOR(POOL(j):      @BIN(f(j));
                         @SUM(SOURCE(i):X(i,j))  <= @SUM(SOURCE(i):S(i))*f(j));
      @SUM(POOL(j):f(j))<=2;

END
```

B 7 *The example: Implicit discretization*

```
SETS:
      SOURCE /1..3/: a1, a2, S, Cost;
      POOL /1..231/: b1, b2, f; ! The number of pools is equal to (t+1)*(t+2)/2 (see below for t);
      PROD /1..3/: c1, c2, D, P;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
ENDSETS

DATA:
      a1 = 82, 92, 82 ;
      a2 =  1,  2, 1.5;
      S = 100,200,100;
      D = 100, 100, 200;
      c1 = 84, 87, 90;
      c2 = 1.9, 2, 2;
      Cost = 7,9,6;
      P = 10, 15, 17;
ENDDATA

! Minimize the costs of sources;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k)))-@SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Constraints on source's capacities;
      @FOR(SOURCE(i): @SUM(POOL(j):X(i,j)) <= S (i));

! Mass balance on the pools;
      @FOR(POOL(j):@SUM(SOURCE(i):X(i,j)) = @SUM(PROD(k):Y(j,k)));

! Mass balance on the products' tanks;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @SUM(SOURCE(i): a1(i)*X(i,j)) = b1(j)*@SUM(SOURCE(i):X(i,j)));
      @FOR(POOL(j): @SUM(SOURCE(i): a2(i)*X(i,j)) = b2(j)*@SUM(SOURCE(i):X(i,j)));
```

```
! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b1(j)*Y(j,k)) >= c1(k)*@SUM(POOL(j):Y(j,k)));
      @FOR(PROD(k): @SUM(POOL(j): b2(j)*Y(j,k)) <= c2(k)*@SUM(POOL(j):Y(j,k)));

! Linearize the problem by specifying qualities of pools;

! Number of discretized intervals; t=20;
@FOR(POOL(u)|u#LE#t+1:
      @FOR(POOL(v)|v#LE#t-u+2:
      b1((u-1)*(t+1)-(u-1)*(u-2)/2+v)=a1(1)*(u-1)/t + a1(2)*(v-1)/t + a1(3)*(1-(u-1)/t-(v-1)/t);
      b2((u-1)*(t+1)-(u-1)*(u-2)/2+v)=a2(1)*(u-1)/t + a2(2)*(v-1)/t + a2(3)*(1-(u-1)/t-(v-1)/t)));

! Use only 2 pools;
      @FOR(POOL(j):     @BIN(f(j));
                        @SUM(SOURCE(i):X(i,j)) <= @SUM(SOURCE(i):S(i))*f(j));
      @SUM(POOL(j):f(j))<=2;
END
```

B 8 *The example: Implicit discretization – Flow rate fraction*

```
SETS:
      SOURCE /1..3/: a1, a2, S, Cost;
      POOL /1..231/: b1, b2, f, Z; ! The number of pools is equal to (t+1)*(t+2)/2 (see below for t);
      PROD /1..3/: c1, c2, D, P;
      INPOOL (SOURCE,POOL): x;
      OUTPOOL (POOL,PROD): Y;
ENDSETS

DATA:
      a1 = 82, 92, 82 ;
      a2 =  1,  2, 1.5;
      S = 100,200,100;
      D = 100, 100, 200;
      c1 = 84, 87, 90;
      c2 = 1.9, 2, 2;
      Cost = 7,9,6;
      P = 10, 15, 17;
ENDDATA

! Minimize the costs of sources;
MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k)))-@SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):x(i,j)*Z(j)));

! Constraints on source's capacities;
      @FOR(SOURCE(i): @SUM(POOL(j):x(i,j)*Z(j)) <= S (i));

! Mass balance on the pools;
      @FOR(POOL(j):Z(j) = @SUM(PROD(k):Y(j,k)));

! Mass balance on the products' tanks;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the products;
      @FOR(PROD(k): @SUM(POOL(j): b1(j)*Y(j,k)) >= c1(k)*@SUM(POOL(j):Y(j,k)));
      @FOR(PROD(k): @SUM(POOL(j): b2(j)*Y(j,k)) <= c2(k)*@SUM(POOL(j):Y(j,k)));
```

```
! Linearize the problem by specifying qualities of pools;
! Number of discretized intervals; t=20;
@FOR(POOL(u)|u#LE#t+1:
      @FOR(POOL(v)|v#LE#t-u+2:
      x(1,(u-1)*(t+1)-(u-1)*(u-2)/2+v) = (u-1)/t;
      x(2,(u-1)*(t+1)-(u-1)*(u-2)/2+v) = (v-1)/t;
      x(3,(u-1)*(t+1)-(u-1)*(u-2)/2+v) = 1-(u-1)/t-(v-1)/t;
      b1((u-1)*(t+1)-(u-1)*(u-2)/2+v) = a1(1)*(u-1)/t + a1(2)*(v-1)/t + a1(3)*(1-(u-1)/t-(v-1)/t);
      b2((u-1)*(t+1)-(u-1)*(u-2)/2+v) = a2(1)*(u-1)/t + a2(2)*(v-1)/t + a2(3)*(1-(u-1)/t-(v-1)/t)));

! Use only 2 pools;
      @FOR(POOL(j):      @BIN(f(j));
                         Z(j) <= @SUM(SOURCE(i):S(i))*f(j));
      @SUM(POOL(j):f(j))<=2;
END
```

B 9 *Adhya 1: Not linearized*

```
SETS: SOURCE /1..5/: Cost;
      POOL /1..2/;
      PROD /1..4/: D, P;
      QUAL /1..4/;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL):c;
ENDSETS
DATA: a =   1,   6, 4, 0.5,
            4,   1, 3,   2,
            4, 5.5, 3, 0.9,
            3,   3, 3,   1,
            1, 2.7, 4, 1.6;
      Cost = 7, 3, 2, 10, 5;
      D = 10, 25, 30, 10;
      P = 16, 25, 15, 10;
      c =     3,   3, 3.25, 0.75,
              4, 2.5,  3.5,  1.5,
            1.5, 5.5,  3.9,  0.8,
              3,   4,    4,  1.8;
ENDDATA
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));
! Mass balance on the pools;
      X(1,1) + X(2,1) = @SUM(PROD(k):Y(1,k));   X(3,1)=0;   X(4,1) = 0;    X(5,1) = 0;
      X(3,2) + X(4,2) + X(5,2)= @SUM(PROD(k):Y(2,k));       X(1,2)=0; X(2,2) = 0;
! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));
! Quality blending for the pools;
      @FOR(POOL(j):@FOR(QUAL(q):    @SUM(SOURCE(i): a(i,q)*X(i,j)) = b(j,q)*@SUM(SOURCE(i):X(i,j))));
! Quality blending for the products;
      @FOR(PROD(k):@FOR(QUAL(q):    @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));
END
```

B 10 Adhya 1: Implicit discretization

```
SETS:
      SOURCE /1..5/: Cost;
      POOL /1..77/: f1, f2; ! The number of pools is t1+1+(t2+1)(t2+2)/2 (see below for t1, t2);
      PROD /1..4/: D, P;
      QUAL /1..4/;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL):c;
ENDSETS

DATA:
      a =   1,   6, 4, 0.5,
            4,   1, 3,   2,
            4, 5.5, 3, 0.9,
            3,   3, 3,   1,
            1, 2.7, 4, 1.6;
      Cost = 7, 3, 2, 10, 5;
      D = 10, 25, 30, 10;
      P = 16, 25, 15, 10;
      c =     3,   3, 3.25, 0.75,
              4, 2.5,  3.5,  1.5,
            1.5, 5.5,  3.9,  0.8,
              3,   4,    4,  1.8;
      t1 = 10; ! Number of discretized intervals for pool group 1;
      t2 = 10; ! Number of discretized intervals for pool group 2;
ENDDATA

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));
```

```
! Linearize the problem by discretizing qualities of pools;
@FOR(POOL(j)|j#LE#t1+1: @FOR(QUAL(q): b(j,q)=a(1,q)*(j-1)/t1 + a(2,q)*(1-(j-1)/t1)));

@FOR(POOL(u)|u#LE#t2+1:
      @FOR(POOL(v)|v#LE#t2-u+2:
            @FOR(QUAL(q):
b(t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v,q)=a(3,q)*(u-1)/t2 + a(4,q)*(v-1)/t2 + a(5,q)*(1-(u-1)/t2-(v-1)/t2))));

! Mass balance on the pools;
      @FOR(POOL(j)|j#LE#t1+1: X(1,j) + X(2,j)  = @SUM(PROD(k):Y(j,k));
                              @FOR(SOURCE(i)|i#GE#3:X(i,j)=0));
      @FOR(POOL(j)|j#GT#t1+1: X(1,j)=0; X(2,j)  = 0;
                              X(3,j) + X(4,j) + X(5,j)= @SUM(PROD(k):Y(j,k)));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j):@FOR(QUAL(q):
            @SUM(SOURCE(i): a(i,q)*X(i,j)) = b(j,q)*@SUM(SOURCE(i):X(i,j))));

! Quality blending for the products;
      @FOR(PROD(k):@FOR(QUAL(q):
            @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));

! Source 1&2 are forced to one same pool;
      @FOR(POOL(j)|j#LE#t1+1:
            @BIN(f1(j));
            X(1,j)+X(2,j) <= @SUM(PROD(k):D(k))*f1(j));
      @SUM(POOL(j):f1(j))<=1;

! Source 3,4&5 are forced to one same pool;
      @FOR(POOL(j)|j#GT#t1+1:
            @BIN(f2(j));
            X(3,j)+X(4,j)+X(5,j)<= @SUM(PROD(k):D(k))*f2(j));
      @SUM(POOL(j):f2(j))<=1;
END
```

B 11 Adhya 1: Implicit discretization – Formulation in flow rate fraction

```
SETS:
      SOURCE /1..5/: Cost;
      POOL /1..5/: f1, f2, Z; ! The number of pools is t1+1+(t2+1)(t2+2)/2 (see below for t1, t2);
      PROD /1..4/: D, P;
      QUAL /1..4/;
      INPOOL (SOURCE,POOL): x;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL):c;
ENDSETS

DATA:
      a =   1,   6, 4, 0.5,
            4,   1, 3,   2,
            4, 5.5, 3, 0.9,
            3,   3, 3,   1,
            1, 2.7, 4, 1.6;

      Cost = 7, 3, 2, 10, 5;
      D = 10, 25, 30, 10;
      P = 16, 25, 15, 10;
      c =     3,   3, 3.25, 0.75,
              4, 2.5,  3.5,  1.5,
            1.5, 5.5,  3.9,  0.8,
              3,   4,    4,  1.8;
      t1 = 1; ! Number of discretized intervals for pool group 1;
      t2 = 1; ! Number of discretized intervals for pool group 2;
ENDDATA

! Maximize the revenue;
MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):x(i,j)*Z(j)));
```

```
! Linearize the problem by discretizing qualities of pools;
@FOR(POOL(j)|j#LE#t1+1:
      x(1,j)=(j-1)/t1;
      x(2,j)=1-(j-1)/t1;
      x(3,j)=0; x(4,j)=0; x(5,j)=0;
      @FOR(QUAL(q): b(j,q)=a(1,q)*(j-1)/t1 + a(2,q)*(1-(j-1)/t1)));

@FOR(POOL(u)|u#LE#t2+1:
      @FOR(POOL(v)|v#LE#t2-u+2:
      x(1,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=0;
      x(2,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=0;
      x(3,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=(u-1)/t2;
      x(4,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=(v-1)/t2;
      x(5,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=1-(v-1)/t2-(u-1)/t2;
      @FOR(QUAL(q): b(t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v,q)
                    =a(3,q)*(u-1)/t2 + a(4,q)*(v-1)/t2 + a(5,q)*(1-(u-1)/t2-(v-1)/t2))));

! Mass balance on the pools;
      @FOR(POOL(j): Z(j)= @SUM(PROD(k):Y(j,k)));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the products;
      @FOR(PROD(k):@FOR(QUAL(q):    @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));

! Source 1&2 are forced to one same pool;
      @FOR(POOL(j)|j#LE#t1+1: @BIN(f1(j));
                              Z(j) <= @SUM(PROD(k):D(k))*f1(j));
      @SUM(POOL(j):f1(j))<=1;

! Source 3,4&5 are forced to one same pool;
      @FOR(POOL(j)|j#GT#t1+1: @BIN(f2(j));
                              Z(j) <= @SUM(PROD(k):D(k))*f2(j));
      @SUM(POOL(j):f2(j))<=1;
END
```

B 12 Adhya 2: Not linearized

```
SETS:
      SOURCE /1..5/: Cost;
      POOL /1..2/;
      PROD /1..4/: D, P;
      QUAL /1..6/;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL): c;
ENDSETS

DATA:
      a =   1,   6, 4, 0.5, 5,  9,
            4,   1, 3,   2, 4,  4,
            4, 5.5, 3, 0.9, 7, 10,
            3,   3, 3,   1, 3,  4,
            1, 2.7, 4, 1.6, 3,  7;

      Cost = 7, 3, 2, 10, 5;
      D = 10, 25, 30, 10;
      P = 16, 25, 15, 10;

      c =   3,   3, 3.25, 0.75, 6, 5,
            4, 2.5,  3.5,  1.5, 7, 6,
          1.5, 5.5,  3.9,  0.8, 7, 6,
            3,   4,    4,  1.8, 6, 6;
ENDDATA

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance on the pools;
      X(1,1) + X(2,1) = @SUM(PROD(k):Y(1,k));
      X(3,2) + X(4,2) + X(5,2)= @SUM(PROD(k):Y(2,k));
```

```
! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @FOR(QUAL(q):
            @SUM(SOURCE(i): a(i,q)*X(i,j)) = b(j,q)*@SUM(PROD(k):Y(j,k))));

! Quality blending for the products;
      @FOR(PROD(k): @FOR(QUAL(q):
            @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));
END
```

B 13 Adhya 2: Implicit discretization

```
SETS:
      SOURCE /1..5/: Cost;
      POOL /1..77/: f1, f2; ! The number of pools is t1+1+(t2+1)(t2+2)/2 (see below for t1, t2);
      PROD /1..4/: D, P;
      QUAL /1..6/;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL): c;
ENDSETS

DATA:
      a =   1,   6, 4, 0.5, 5,  9,
            4,   1, 3,   2, 4,  4,
            4, 5.5, 3, 0.9, 7, 10,
            3,   3, 3,   1, 3,  4,
            1, 2.7, 4, 1.6, 3,  7;

      Cost = 7,  3, 2,  10, 5;
      D = 10, 25, 30, 10;
      P = 16, 25, 15, 10;

      c =   3,   3, 3.25, 0.75, 6, 5,
            4, 2.5,  3.5,  1.5, 7, 6,
          1.5, 5.5,  3.9,  0.8, 7, 6,
            3,   4,    4,  1.8, 6, 6;

      t1 = 10; ! Number of discretized intervals for pool group 1;
      t2 = 10; ! Number of discretized intervals for pool group 2;
ENDDATA

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));
```

```
! Linearize the problem by discretizing qualities of pools;
      @FOR(POOL(j)|j#LE#t1+1: @FOR(QUAL(q):
            b(j,q)=a(1,q)*(j-1)/t1 + a(2,q)*(1-(j-1)/t1)));

      @FOR(POOL(u)|u#LE#t2+1: @FOR(POOL(v)|v#LE#t2-u+2: @FOR(QUAL(q):
b(t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v,q)=a(3,q)*(u-1)/t2 + a(4,q)*(v-1)/t2 + a(5,q)*(1-(u-1)/t2-(v-1)/t2))));

! Mass balance on the pools;
      @FOR(POOL(j)|j#LE#t1+1: X(1,j) + X(2,j) = @SUM(PROD(k):Y(j,k));
                              X(3,j) + X(4,j) + X(5,j)=0);

      @FOR(POOL(j)|j#GT#t1+1: X(1,j) + X(2,j) = 0;
                              X(3,j) + X(4,j) + X(5,j)= @SUM(PROD(k):Y(j,k)));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @FOR(QUAL(q):
            @SUM(SOURCE(i): a(i,q)*X(i,j)) = b(j,q)*@SUM(PROD(k):Y(j,k))));

! Quality blending for the products;
      @FOR(PROD(k): @FOR(QUAL(q):
            @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));

! Source 1&2 are forced to feed same pool;
      @FOR(POOL(j)|j#LE#t1+1:
            @BIN(f1(j));
            X(1,j)+X(2,j) <= @SUM(PROD(k):D(k))*f1(j));
      @SUM(POOL(j):f1(j))<=1;

! Source 3,4&5 are force to feed same pool;
      @FOR(POOL(j)|j#GT#t1+1:
            @BIN(f2(j));
            X(3,j)+X(4,j)+X(5,j)<= @SUM(PROD(k):D(k))*f2(j));
      @SUM(POOL(j):f2(j))<=1;
END
```

B 14 Adhya 2: Implicit discretization – Formulation in flow rate fraction

```
SETS:
      SOURCE /1..5/: Cost;
      POOL /1..77/: f1, f2, Z; ! The number of pools is t1+1+(t2+1)(t2+2)/2 (see below for t1, t2);
      PROD /1..4/: D, P;
      QUAL /1..6/;
      INPOOL (SOURCE,POOL): x;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL): c;
ENDSETS

DATA:
      a =   1,   6, 4, 0.5, 5,  9,
            4,   1, 3,   2, 4,  4,
            4, 5.5, 3, 0.9, 7, 10,
            3,   3, 3,   1, 3,  4,
            1, 2.7, 4, 1.6, 3,  7;
      Cost = 7, 3, 2, 10, 5;
      D = 10, 25, 30, 10;
      P = 16, 25, 15, 10;
      c =   3,   3, 3.25, 0.75, 6, 5,
            4, 2.5,  3.5,  1.5, 7, 6,
          1.5, 5.5,  3.9,  0.8, 7, 6,
            3,   4,    4,  1.8, 6, 6;
      t1 = 10; ! Number of discretized intervals for pool group 1;
      t2 = 10; ! Number of discretized intervals for pool group 2;
ENDDATA

! Maximize the revenue;
MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):x(i,j)*Z(j)));
```

```
! Linearize the problem by discretizing qualities of pools;
      @FOR(POOL(j)|j#LE#t1+1:
            x(1,j)=(j-1)/t1;
            x(2,j)=1-(j-1)/t1;
            x(3,j)=0; x(4,j)=0; x(5,j)=0;
            @FOR(QUAL(q):b(j,q)=a(1,q)*(j-1)/t1 + a(2,q)*(1-(j-1)/t1)));

      @FOR(POOL(u)|u#LE#t2+1:
            @FOR(POOL(v)|v#LE#t2-u+2:
            x(1,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=0;
            x(2,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=0;
            x(3,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=(u-1)/t2;
            x(4,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=(v-1)/t2;
            x(5,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=1-(v-1)/t2-(u-1)/t2;
            @FOR(QUAL(q):
                  b(t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v,q)=
                  a(3,q)*(u-1)/t2 + a(4,q)*(v-1)/t2 + a(5,q)*(1-(u-1)/t2-(v-1)/t2))));

! Mass balance on the pools;
      @FOR(POOL(j): Z(j)= @SUM(PROD(k):Y(j,k)));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the products;
      @FOR(PROD(k): @FOR(QUAL(q):   @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));

! Source 1&2 are forced to one same pool;
      @FOR(POOL(j)|j#LE#t1+1: @BIN(f1(j));
                              Z(j) <= @SUM(PROD(k):D(k))*f1(j));
      @SUM(POOL(j):f1(j))<=1;

! Source 3,4&5 are forced to one same pool;
      @FOR(POOL(j)|j#GT#t1+1: @BIN(f2(j));
                              Z(j) <= @SUM(PROD(k):D(k))*f2(j));
      @SUM(POOL(j):f2(j))<=1;
END
```

B 15 Adhya 3: Not linearized

```
SETS:
      SOURCE /1..8/: Cost;
      POOL /1..3/;
      PROD /1..4/:D, P;
      QUAL /1..6/;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL): c;
ENDSETS
DATA:
      a =   1,   6,   4, 0.5,   5,   9,
            4,   1,   3,   2,   4,   4,
            4, 5.5,   3, 0.9,   7,  10,
            3,   3,   3,   1,   3,   4,
            1, 2.7,   4, 1.6,   3,   7,
          1.8, 2.7,   4, 3.5, 6.1,   3,
            5,   1, 1.7, 2.9, 3.5, 2.9,
            3,   3,   3,   1,   5,   2;
      Cost = 7, 3, 2, 10, 5, 5, 9, 11;
      D = 10, 25, 30, 10;
      P = 16, 25, 15, 10;
      c =   3,   3, 3.25, 0.75, 6, 5,
            4, 2.5,  3.5,  1.5, 7, 6,
          1.5, 5.5,  3.9,  0.8, 7, 6,
            3,   4,    4,  1.8, 6, 6;
ENDDATA
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance on the pools;
      X(1,1) + X(2,1) = @SUM(PROD(k):Y(1,k));
      X(3,2) + X(4,2) + X(5,2)= @SUM(PROD(k):Y(2,k));
      X(6,3) + X(7,3) + X(8,3)= @SUM(PROD(k):Y(3,k));
```

```
! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @FOR(QUAL(q):   @SUM(SOURCE(i): a(i,q)*X(i,j)) = b(j,q)*@SUM(PROD(k):Y(j,k))));

! Quality blending for the products;
      @FOR(PROD(k): @FOR(QUAL(q):   @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));
END
```

B 16 Adhya 3: Implicit discretization

```
SETS:
      SOURCE /1..8/: Cost;
      POOL /1..143/: f1, f2, f3;
      ! The number of pools is t1+1+(t2+1)(t2+2)/2+(t3+1)(t3+2)/2 (see below for t1, t2 and t3);
      PROD /1..4/: D, P;
      QUAL /1..6/;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL): c;
ENDSETS

DATA:
      a =   1,   6,   4, 0.5,   5,   9,
            4,   1,   3,   2,   4,   4,
            4, 5.5,   3, 0.9,   7,  10,
            3,   3,   3,   1,   3,   4,
            1, 2.7,   4, 1.6,   3,   7,
          1.8, 2.7,   4, 3.5, 6.1,   3,
            5,   1, 1.7, 2.9, 3.5, 2.9,
            3,   3,   3,   1,   5,   2;

      Cost = 7, 3, 2, 10, 5, 5, 9, 11;
      D = 10, 25, 30, 10;
      P = 16, 25, 15, 10;
      c =   3,   3, 3.25, 0.75, 6, 5,
            4, 2.5,  3.5,  1.5, 7, 6,
          1.5, 5.5,  3.9,  0.8, 7, 6,
            3,   4,    4,  1.8, 6, 6;

      t1 = 10; ! Number of discretized intervals for pool group 1;
      t2 = 10; ! Number of discretized intervals for pool group 2;
      t3 = 10; ! Number of discretized intervals for pool group 3;
ENDDATA
```

```
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Linearize the problem by discretizing qualities of pools;
@FOR(POOL(j)|j#LE#t1+1: @FOR(QUAL(q):
      b(j,q)=a(1,q)*(j-1)/t1 + a(2,q)*(1-(j-1)/t1)));

@FOR(POOL(u)|u#LE#t2+1: @FOR(POOL(v)|v#LE#t2-u+2: @FOR(QUAL(q):
b(t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v,q)=a(3,q)*(u-1)/t2 + a(4,q)*(v-1)/t2 + a(5,q)*(1-(u-1)/t2-(v-1)/t2))));

@FOR(POOL(u)|u#LE#t3+1: @FOR(POOL(v)|v#LE#t3-u+2: @FOR(QUAL(q):
 b(t1+1+(t2+1)*(t2+2)/2+(u-1)*(t3+1)-(u-1)*(u-2)/2+v,q)
= a(6,q)*(u-1)/t3 + a(7,q)*(v-1)/t3 + a(8,q)*(1-(u-1)/t3-(v-1)/t3))));

! Mass balance on the pools;
      @FOR(POOL(j)|j#LE#t1+1: X(1,j) + X(2,j) = @SUM(PROD(k):Y(j,k));
                              @FOR(SOURCE(i)|i#GE#3:X(i,j)=0));

      @FOR(POOL(j)|(j#GT#t1+1) #AND# (j#LE#t1+1+(t2+1)*(t2+2)/2):
                              @FOR(SOURCE(i)|i#LE#2 #OR# i#GE#6:X(i,j)=0);
                              X(3,j) + X(4,j) + X(5,j)= @SUM(PROD(k):Y(j,k)));

      @FOR(POOL(j)|j#GT#t1+1+(t2+1)*(t2+2)/2:
                              @FOR(SOURCE(i)|i#LE#5:X(i,j)=0);
                              X(6,j) + X(7,j) + X(8,j)= @SUM(PROD(k):Y(j,k)));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @FOR(QUAL(q):
            @SUM(SOURCE(i): a(i,q)*X(i,j)) = b(j,q)*@SUM(PROD(k):Y(j,k))));

! Quality blending for the products;
      @FOR(PROD(k): @FOR(QUAL(q):
            @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));
```

```
! Source 1&2 are forced to same pool;
      @FOR(POOL(j)|j#LE#t1+1:
            @BIN(f1(j));
            X(1,j)+X(2,j)  <= @SUM(PROD(k):D(k))*f1(j));
      @SUM(POOL(j):f1(j))<=1;

! Source 3,4&5 are forced to same pool;
      @FOR(POOL(j)|(j#GT#t1+1) #AND# (j#LE#t1+1+(t2+1)*(t2+2)/2):
            @BIN(f2(j));
            X(3,j)+X(4,j)+X(5,j)<= @SUM(PROD(k):D(k))*f2(j));
      @SUM(POOL(j):f2(j))<=1;

! Source 6,7&8 are forced to same pool;
      @FOR(POOL(j)|j#GT#t1+1+(t2+1)*(t2+2)/2:
            @BIN(f3(j));
            X(6,j)+X(7,j)+X(8,j)<= @SUM(PROD(k):D(k))*f3(j));
      @SUM(POOL(j):f3(j))<=1;
END
```

B 17 *Adhya 3: Implicit discretization – Formulation in flow rate fraction*

```
SETS:
      SOURCE /1..8/: Cost;
      POOL /1..143/: f1, f2, f3, Z;
      ! The number of pools is t1+1+(t2+1)(t2+2)/2+(t3+1)(t3+2)/2 (see below for t1, t2 and t3);
      PROD /1..4/: D, P;
      QUAL /1..6/;
      INPOOL (SOURCE,POOL): x;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL): c;
ENDSETS
DATA:
      a =   1,   6,   4, 0.5,   5,   9,
            4,   1,   3,   2,   4,   4,
            4, 5.5,   3, 0.9,   7,  10,
            3,   3,   3,   1,   3,   4,
            1, 2.7,   4, 1.6,   3,   7,
          1.8, 2.7,   4, 3.5, 6.1,   3,
            5,   1, 1.7, 2.9, 3.5, 2.9,
            3,   3,   3,   1,   5,   2;

      Cost = 7, 3, 2, 10, 5, 5, 9, 11;
      D = 10, 25, 30, 10;
      P = 16, 25, 15, 10;

      c =   3,   3, 3.25, 0.75, 6, 5,
            4, 2.5,  3.5,  1.5, 7, 6,
          1.5, 5.5,  3.9,  0.8, 7, 6,
            3,   4,    4,  1.8, 6, 6;

      t1 = 10; ! Number of discretized intervals for pool group 1;
      t2 = 10; ! Number of discretized intervals for pool group 2;
      t3 = 10; ! Number of discretized intervals for pool group 3;
ENDDATA
```

```
! Maximize the revenue;
MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):x(i,j)*Z(j)));

! Linearize the problem by discretizing qualities of pools;
      @FOR(POOL(j)|j#LE#t1+1:
            x(1,j)=(j-1)/t1;
            x(2,j)=1-(j-1)/t1;
            @FOR(SOURCE(i)|i#GE#3: x(i,j)=0);
            @FOR(QUAL(q):b(j,q)=a(1,q)*(j-1)/t1 + a(2,q)*(1-(j-1)/t1)));

      @FOR(POOL(u)|u#LE#t2+1:
            @FOR(POOL(v)|v#LE#t2-u+2:
            x(3,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=(u-1)/t2;
            x(4,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=(v-1)/t2;
            x(5,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=1-(v-1)/t2-(u-1)/t2;
            @FOR(SOURCE(i)|i#LE#2 #OR# i#GE#6: x(i,t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v)=0);
            @FOR(QUAL(q): b(t1+1+(u-1)*(t2+1)-(u-1)*(u-2)/2+v,q)
                         = a(3,q)*(u-1)/t2 + a(4,q)*(v-1)/t2 + a(5,q)*(1-(u-1)/t2-(v-1)/t2))));

      @FOR(POOL(u)|u#LE#t3+1:
            @FOR(POOL(v)|v#LE#t3-u+2:
            x(6,t1+1+(t2+1)*(t2+2)/2+(u-1)*(t3+1)-(u-1)*(u-2)/2+v)=(u-1)/t2;
            x(7,t1+1+(t2+1)*(t2+2)/2+(u-1)*(t3+1)-(u-1)*(u-2)/2+v)=(v-1)/t2;
            x(8,t1+1+(t2+1)*(t2+2)/2+(u-1)*(t3+1)-(u-1)*(u-2)/2+v)=1-(v-1)/t2-(u-1)/t2;
            @FOR(SOURCE(i)|i#LE#5: x(i,t1+1+(t2+1)*(t2+2)/2+(u-1)*(t3+1)-(u-1)*(u-2)/2+v)=0);
            @FOR(QUAL(q): b(t1+1+(t2+1)*(t2+2)/2+(u-1)*(t3+1)-(u-1)*(u-2)/2+v,q)
                        = a(6,q)*(u-1)/t3 + a(7,q)*(v-1)/t3 + a(8,q)*(1-(u-1)/t3-(v-1)/t3))));

! Mass balance on the pools;
      @FOR(POOL(j): Z(j)= @SUM(PROD(k):Y(j,k)));

! Mass demand on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));
```

```
! Quality blending for the products;
      @FOR(PROD(k): @FOR(QUAL(q):
            @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));

! Source 1&2 are forced to same pool;
      @FOR(POOL(j)|j#LE#t1+1: @BIN(f1(j));
                              Z(j)<= @SUM(PROD(k):D(k))*f1(j));
      @SUM(POOL(j):f1(j))<=1;

! Source 3,4&5 are forced to same pool;
      @FOR(POOL(j)|j#GT#t1+1 #AND# j#LE#t1+1+(t2+1)*(t2+2)/2:
            @BIN(f2(j));
            Z(j) <= @SUM(PROD(k):D(k))*f2(j));
      @SUM(POOL(j):f2(j))<=1;

! Source 6,7&8 are forced to same pool;
      @FOR(POOL(j)|j#GT#t1+1+(t2+1)*(t2+2)/2:
            @BIN(f3(j));
            Z(j)<= @SUM(PROD(k):D(k))*f3(j));
      @SUM(POOL(j):f3(j))<=1;
END
```

B 18 *Adhya 4: Not linearized*

```
SETS:
      SOURCE /1..8/: Cost;
      POOL /1..2/;
      PROD /1..5/: D, P;
      QUAL /1..4/;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL): c;
ENDSETS
DATA:
      a =   0.5, 1.9, 1.3, 1.0,
            1.4, 1.8, 1.7, 1.6,
            1.2, 1.9, 1.4, 1.4,
            1.5, 1.2, 1.7, 1.3,
            1.6, 1.8, 1.6, 2.0,
            1.2, 1.1, 1.4, 2.0,
            1.5, 1.5, 1.5, 1.5,
            1.4, 1.6, 1.2, 1.6;

      Cost = 15,  7,  4,  5,  6, 3, 5, 5;
      D =  15, 25, 10, 20, 15;
      P =  10, 25, 30,  6, 10;

      c =   1.2, 1.7, 1.4, 1.7,
            1.4, 1.3, 1.8, 1.4,
            1.3, 1.3, 1.9, 1.9,
            1.2, 1.1, 1.7, 1.6,
            1.6, 1.9, 2.0, 2.5;

      t1 = 12; ! Number of discretized intervals for pool group 1;
      t2 = 12; ! Number of discretized intervals for pool group 2;
ENDDATA
```

```
! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));

! Mass balance on the pools;
      @SUM(SOURCE(i)|i#LE#4: X(i,1)) = @SUM(PROD(k):Y(1,k));
      @FOR(SOURCE(i)|i#LE#4: X(i,2) = 0);

      @FOR(SOURCE(i)|i#GT#4: X(i,1) = 0);
      @SUM(SOURCE(i)|i#GT#4: X(i,2)) = @SUM(PROD(k):Y(2,k));

! Mass balance on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the pools;
      @FOR(POOL(j): @FOR(QUAL(q):
            @SUM(SOURCE(i): a(i,q)*X(i,j)) = b(j,q)*@SUM(SOURCE(i):X(i,j))));

! Quality blending for the products;
      @FOR(PROD(k): @FOR(QUAL(q):
            @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));
END
```

B 19 Adhya 4: Implicit discretization

```
SETS:
      SOURCE /1..8/: Cost;
      POOL /1..910/: f1, f2;
      ! The number of pools is (t1+1)(t1+2)(t1+3)/6 + (t2+1)(t2+2)(t2+3)/6 (see below for t1, t2);
      PROD /1..5/: D, P;
      QUAL /1..4/;
      INPOOL (SOURCE,POOL): X;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL): c;
ENDSETS
DATA:
      a =   0.5, 1.9, 1.3, 1.0,
            1.4, 1.8, 1.7, 1.6,
            1.2, 1.9, 1.4, 1.4,
            1.5, 1.2, 1.7, 1.3,
            1.6, 1.8, 1.6, 2.0,
            1.2, 1.1, 1.4, 2.0,
            1.5, 1.5, 1.5, 1.5,
            1.4, 1.6, 1.2, 1.6;
      Cost = 15,  7,  4,  5,  6, 3, 5, 5;
      D =  15, 25, 10, 20, 15;
      P =  10, 25, 30,  6, 10;
      c =   1.2, 1.7, 1.4, 1.7,
            1.4, 1.3, 1.8, 1.4,
            1.3, 1.3, 1.9, 1.9,
            1.2, 1.1, 1.7, 1.6,
            1.6, 1.9, 2.0, 2.5;
      t1 = 12; ! Number of discretized intervals for pool group 1;
      t2 = 12; ! Number of discretized intervals for pool group 2;
ENDDATA

! Maximize the revenue;
      MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):X(i,j)));
```

```
! Linearize the problem by specifying qualities of pools;
@FOR(POOL(u)|u#LE#t1+1: @FOR(POOL(v)|v#LE#t1-u+2: @FOR(POOL(r)|r#LE#t1-v-u+3: @FOR(QUAL(q):
b((t1+2)*(t1+3)*(t1+4)/6-((t1-u+2))*((t1-u+3))*((t1-u+4))/6-(t1+2)*(t1+3)/2+(v-1)*((t1-u+2))-(v-1)*(v-2)/2+r,q)
= a(1,q)*(u-1)/t1 + a(2,q)*(v-1)/t1 + a(3,q)*(r-1)/t1 + a(4,q)*(1-(u-1)/t1-(v-1)/t1-(r-1)/t1)))));

@FOR(POOL(u)|u#LE#t2+1: @FOR(POOL(v)|v#LE#t2-u+2: @FOR(POOL(r)|r#LE#t2-v-u+3: @FOR(QUAL(q):
b((t1+1)*(t1+2)*(t1+3)/6+(t2+2)*(t2+3)*(t2+4)/6-(t2-u+2)*(t2-u+3)*(t2-u+4)/6-(t2+2)*(t2+3)/2+(v-1)*(t2-u+2)-(v-1)*(v-2)/2+r,q)
= a(5,q)*(u-1)/t2 + a(6,q)*(v-1)/t2 + a(7,q)*(r-1)/t2 + a(8,q)*(1-(u-1)/t2-(v-1)/t2-(r-1)/t2)))));

! Mass balance on the pools;
      @FOR(POOL(j)|j#LE#(t1+1)*(t1+2)*(t1+3)/6:
                              X(1,j) + X(2,j) + X(3,j) + X(4,j) = @SUM(PROD(k):Y(j,k));
                              @FOR(SOURCE(i)|i#GE#5:X(i,j)=0));

      @FOR(POOL(j)|j#GT#(t1+1)*(t1+2)*(t1+3)/6:
                              @FOR(SOURCE(i)|i#LE#4:X(i,j)=0);
                              X(5,j) + X(6,j) + X(7,j) + X(8,j)= @SUM(PROD(k):Y(j,k)));
! Mass balance on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));
! Quality blending for the pools;
      @FOR(POOL(j): @FOR(QUAL(q):   @SUM(SOURCE(i): a(i,q)*X(i,j)) = b(j,q)*@SUM(SOURCE(i):X(i,j))));

! Quality blending for the products;
      @FOR(PROD(k): @FOR(QUAL(q):   @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));

! Source 1,2,3&4 are forced to feed same pool;
      @FOR(POOL(j)|j#LE#(t1+1)*(t1+2)*(t1+3)/6:
            @BIN(f1(j));
            X(1,j) + X(2,j) + X(3,j) + X(4,j) <= @SUM(PROD(k):D(k))*f1(j));
      @SUM(POOL(j):f1(j))<=1;

! Source 5,6,7&8 are forced to feed same pool;
      @FOR(POOL(j)|j#GT#(t1+1)*(t1+2)*(t1+3)/6:
            @BIN(f2(j));
            X(5,j) + X(6,j) + X(7,j) + X(8,j) <= @SUM(PROD(k):D(k))*f2(j));
      @SUM(POOL(j):f2(j))<=1;
END
```

B 20 Adhya 4: Implicit discretization – Formulation in flow rate fraction

```
SETS:
      SOURCE /1..8/: Cost;
      POOL /1..5850/: f1, f2, Z;
      ! The number of pools is (t1+1)(t1+2)(t1+3)/6 + (t2+1)(t2+2)(t2+3)/6 (see below for t1, t2);
      PROD /1..5/: D, P;
      QUAL /1..4/;
      INPOOL (SOURCE,POOL): x;
      OUTPOOL (POOL,PROD): Y;
      SOURQ (SOURCE,QUAL): a;
      POOLQ (POOL,QUAL): b;
      PRODQ (PROD,QUAL): c;
ENDSETS
DATA:
      a =   0.5, 1.9, 1.3, 1.0,
            1.4, 1.8, 1.7, 1.6,
            1.2, 1.9, 1.4, 1.4,
            1.5, 1.2, 1.7, 1.3,
            1.6, 1.8, 1.6, 2.0,
            1.2, 1.1, 1.4, 2.0,
            1.5, 1.5, 1.5, 1.5,
            1.4, 1.6, 1.2, 1.6;
      Cost = 15,  7,  4,  5,  6, 3, 5, 5;
      D =  15, 25, 10, 20, 15;
      P =  10, 25, 30,  6, 10;

      c =   1.2, 1.7, 1.4, 1.7,
            1.4, 1.3, 1.8, 1.4,
            1.3, 1.3, 1.9, 1.9,
            1.2, 1.1, 1.7, 1.6,
            1.6, 1.9, 2.0, 2.5;

      t1 = 24; ! Number of discretized intervals for pool group 1;
      t2 = 24; ! Number of discretized intervals for pool group 2;
ENDDATA
```

```
! Maximize the revenue;
MAX = @SUM(PROD(k):P(k)*@SUM(POOL(j):Y(j,k))) - @SUM(SOURCE(i):Cost(i)*@SUM(POOL(j):x(i,j)*Z(j)));

! Linearize the problem by specifying qualities of pools;
      @FOR(POOL(u)|u#LE#t1+1:
            @FOR(POOL(v)|v#LE#t1-u+2:
                  @FOR(POOL(r)|r#LE#t1-v-u+3:
 x(1,(t1+2)*(t1+3)*(t1+4)/6-((t1-u+2))*((t1-u+3))*((t1-u+4))/6-(t1+2)*(t1+3)/2+(v-1)*((t1-u+2))-(v-1)*(v-2)/2+r)
=(u-1)/t1;

 x(2,(t1+2)*(t1+3)*(t1+4)/6-((t1-u+2))*((t1-u+3))*((t1-u+4))/6-(t1+2)*(t1+3)/2+(v-1)*((t1-u+2))-(v-1)*(v-2)/2+r)
= (v-1)/t1;

 x(3,(t1+2)*(t1+3)*(t1+4)/6-((t1-u+2))*((t1-u+3))*((t1-u+4))/6-(t1+2)*(t1+3)/2+(v-1)*((t1-u+2))-(v-1)*(v-2)/2+r)
= (r-1)/t1;

 x(4,(t1+2)*(t1+3)*(t1+4)/6-((t1-u+2))*((t1-u+3))*((t1-u+4))/6-(t1+2)*(t1+3)/2+(v-1)*((t1-u+2))-(v-1)*(v-2)/2+r)
= 1-(u-1)/t1-(v-1)/t1-(r-1)/t1;

@FOR(SOURCE(i)|i#GE#5:
x(i,(t1+2)*(t1+3)*(t1+4)/6-((t1-u+2))*((t1-u+3))*((t1-u+4))/6-(t1+2)*(t1+3)/2+(v-1)*((t1-u+2))-(v-1)*(v-2)/2+r)
=0);

@FOR(QUAL(q):
 b((t1+2)*(t1+3)*(t1+4)/6-((t1-u+2))*((t1-u+3))*((t1-u+4))/6-(t1+2)*(t1+3)/2+(v-1)*((t1-u+2))-(v-1)*(v-2)/2+r,q)
= a(1,q)*(u-1)/t1 + a(2,q)*(v-1)/t1 + a(3,q)*(r-1)/t1 + a(4,q)*(1-(u-1)/t1-(v-1)/t1-(r-1)/t1)))));

@FOR(POOL(u)|u#LE#t2+1:
      @FOR(POOL(v)|v#LE#t2-u+2:
            @FOR(POOL(r)|r#LE#t2-v-u+3:
x(5,(t1+1)*(t1+2)*(t1+3)/6+(t2+2)*(t2+3)*(t2+4)/6-(t2-u+2)*(t2-u+3)*(t2-u+4)/6-(t2+2)*(t2+3)/2+(v-
1)*(t2-u+2)-(v-1)*(v-2)/2+r) = (u-1)/t2;

x(6,(t1+1)*(t1+2)*(t1+3)/6+(t2+2)*(t2+3)*(t2+4)/6-(t2-u+2)*(t2-u+3)*(t2-u+4)/6-(t2+2)*(t2+3)/2+(v-
1)*(t2-u+2)-(v-1)*(v-2)/2+r) = (v-1)/t2;
x(7,(t1+1)*(t1+2)*(t1+3)/6+(t2+2)*(t2+3)*(t2+4)/6-(t2-u+2)*(t2-u+3)*(t2-u+4)/6-(t2+2)*(t2+3)/2+(v-
1)*(t2-u+2)-(v-1)*(v-2)/2+r) = (r-1)/t2;

x(8,(t1+1)*(t1+2)*(t1+3)/6+(t2+2)*(t2+3)*(t2+4)/6-(t2-u+2)*(t2-u+3)*(t2-u+4)/6-(t2+2)*(t2+3)/2+(v-
1)*(t2-u+2)-(v-1)*(v-2)/2+r) = 1-(u-1)/t1-(v-1)/t1-(r-1)/t1;

@FOR(SOURCE(i)|i#LE#4: x(i,(t1+1)*(t1+2)*(t1+3)/6+(t2+2)*(t2+3)*(t2+4)/6-(t2-u+2)*(t2-u+3)*(t2-
u+4)/6-(t2+2)*(t2+3)/2+(v-1)*(t2-u+2)-(v-1)*(v-2)/2+r) = 0);

@FOR(QUAL(q):b((t1+1)*(t1+2)*(t1+3)/6+(t2+2)*(t2+3)*(t2+4)/6-(t2-u+2)*(t2-u+3)*(t2-u+4)/6-
(t2+2)*(t2+3)/2+(v-1)*(t2-u+2)-(v-1)*(v-2)/2+r,q) = a(5,q)*(u-1)/t2 + a(6,q)*(v-1)/t2 + a(7,q)*(r-
1)/t2 + a(8,q)*(1-(u-1)/t2-(v-1)/t2-(r-1)/t2)))));

! Mass balance on the pools;
      @FOR(POOL(j): Z(j)  = @SUM(PROD(k):Y(j,k)));

! Mass balance on the products;
      @FOR(PROD(k): @SUM(POOL(j):Y(j,k)) <= D(k));

! Quality blending for the products;
      @FOR(PROD(k): @FOR(QUAL(q):
            @SUM(POOL(j): b(j,q)*Y(j,k)) <= c(k,q)*@SUM(POOL(j):Y(j,k))));

! Source 1,2,3&4 are forced to feed same pool;
      @FOR(POOL(j)|j#LE#(t1+1)*(t1+2)*(t1+3)/6:
            @BIN(f1(j));
            Z(j) <= @SUM(PROD(k):D(k))*f1(j));
      @SUM(POOL(j):f1(j))<=1;

! Source 5,6,7&8 are forced to feed same pool;
      @FOR(POOL(j)|j#GT#(t1+1)*(t1+2)*(t1+3)/6:
            @BIN(f2(j));
            Z(j) <= @SUM(PROD(k):D(k))*f2(j));
      @SUM(POOL(j):f2(j))<=1;
END
```

VITA

Name:	Viet Pham
Address:	Department of Chemical Engineering, Texas A&M University 3122 TAMU, College Station, TX 77840
Email Address:	viet@tamu.edu
Education:	B.S., Chemical Engineering, Ho Chi Minh City University of Technology, 2002 M.S., Chemical Engineering, Texas A&M University, 2007