# MATHEMATICAL OPTIMIZATION AND THE MIXED INTEGER LINEAR PROGRAMMING IN BLENDING STEEL

S.Sathyapriya[1] ,R.Varun velraj[2], M.Mukesh[3], R.Lokesh[4], P.Jeevanantham[5],A.GeorgeAntonyRaja

[1] Assistant Professor, Department of Mathematics, Sri Krishna Arts and Science College, Coimbatore.

UG Scholar, Department of Mathematics, Sri Krishna Arts and Science College, Coimbatore.

## ABSTRACT

This project dispenses the brief knowledge about the Mathematical Optimization along with the various methods and programming options that are utilised to achieve that optimization. It insights the concept of Mixed Integer Linear Programming and their esteemed applications. Further, the methods for solving the Mixed Integer and Constraint programming such as Branch and Bound, Strong Branching, Pre-processing and Probing, Cut Generation and Integer Heuristics are investigated

**KEYWORDS**: Mathematical Optimization, Mixed Integer Linear Programming, Branch and Bound, Strong Branching, Pre-processing and Probing, Cut Generation and Integer Heuristics.

## INTRODUCTION

Mathematical optimization isthe choice of a best component, with respect to few criterion, from some set of approachable alternatives. Optimisation issues of sorts arise all quantitativedomains from technology and engineering to research and social science, and also the development of answer strategies has been of interest towards mathematics for hundreds of years.

In the easiest case, an optimization problem comprises of maximizing or minimizing a real function by systematically selecting input values from within apermitted set and solving the value of the function. The generalization of optimization theory and methods to other formulations contains a large area of applied mathematics. More commonly, optimization consists ofseeking “best available” values of few objective function lent a defined domain (or input), including various types of objective functions and different types of domains.

## NOTATION

Optimization problems are commonly expressed using special notation. Some of their examples are:

## MAXIMUM AND MINIMUM VALUE OF A FUNCTION

Consider the subsequent notation:

$$max_{X \in \mathbb{R}} 2x$$

Demands for the maximum value of the objective function 2x, And let x may be any real number. There is no such maximum as the objective function is unbounded in this case, so we can conclude the answer as “infinity” or “undefined”.

Likewise,

$$\min_{X\in\mathbb{R}}(x^2+1)$$

The above notationindicates the minimum value of the objective function $x^2 + 1$, whiletaking x from the set of real numbers $\mathbb{R}$. In this case,the minimum value is 1, resulting at x = 0.

## OPTIMAL INPUT ARGUMENTS

Consider the subsequent notation:

$$\operatorname{argmin}_{X\in(-\infty,-1]} x^2+1$$

It denotes the value (or values) of the argument x that minimizes (or minimize) the objective function x2 + 1 between the interval $(-\infty,-1]$ (what the problem asks for is not the actual minimum value of that function we get). In suchcases, the answer will be x = −1, since x = 0 is infeasible, which means, it is not a component of the feasible set.

Likewise,

$$\operatorname{argmin}_{X\in[-5,-5],Y\in\mathbb{R}} x\cos y$$

Denotes the {x, y} pair (or pairs) with the added constraint that x lie in the interval [−5,5], that maximizes (or maximize) the value of the objective function x cos y, (again, the actual maximum value of the expression does not matter). In such case, the answers are the pairs of the form {5, 2kπ} and {−5, (2k + 1)π}, where k lies over all the integers.

Operators argmax and arg min are oftenwritten as argmax and argmin, and they stand forargument of the maximumand argument of the minimum**.**

## MAJOR SUBFIELDS

## CONVEX PROGRAMMING

Convex programmingdeals with the case when the constraint set is convex and the objective function is concave (maximization) or convex (minimization). Which can be overviewed as a particular case of nonlinear programming otherwise as convex quadratic programming or colligation of linear.

- Linear programming (LP), deals with the case,where the constraints are described using only linear equalities and inequalities and the objective function f is linear. It is a kind of convex programming. If that is bounded of the sort a constraint set is knownas polytope or apolyhedron
- Second-order cone programming (SOCP) is a convex program, and itincludes certainkinds of quadratic programs.

- Semidefinite programming (SDP) is a optimization where the underlying variables are semidefinite matrices, It is a subfield of convex optimization. It is a generalization of convex and linear quadratic programming.
- Conic programming SOCP,SDP, and LP can all be viewed as conic programs with the appropriate type of cone. Itis a general form of convex programming.
- Geometric programming is a technique wherebyequality constraints as monomials and objective and inequality constraints expressed as polynomials, can be transformed into a convex program.

**INTEGER PROGRAMMING**

Integer programming deals with linear programs in which all or some variables are constrained to take on integer values. And basically much more difficult than regular linear programming. This is not convex.

**QUADRATIC PROGRAMMING**

In Quadratic programming,this is a type of convex programming. Quadratic terms are allowed for objective function, while the specification of linear inequalities and equalities in the feasible set is must . For certaintypes of the quadratic term.

**FRACTIONAL PROGRAMMING**

Fractional programming deals with the ratios of two nonlinear functions and its optimization. The convex optimization problem can be transformed from a special class of concave fractional programs.

**NONLINEAR PROGRAMMING**

Nonlinear programming deals with the general case where the constraints or the objective function or both consists nonlinear parts.Basically, whether the program is convex affects the hardness of solving it. This can or cannot not be a convex program.

**STOCHASTIC PROGRAMMING**

Stochastic programming deals with the case, where some of theparameters or constraints depend on random variables.

**ROBUST OPTIMIZATION**

Similar to stochastic programming, Robust optimization isantry to capture uncertainty in the data lying under the optimization problem. Robust optimization aims to find answers that are valid for all possible realizations of the uncertainties that are defined by an uncertainty set.

**COMBINATORIAL OPTIMIZATION**

Combinatorial optimization deals with problems in which the set of feasible solutions is discrete or can be transformed to a discrete set.

**STOCHASTIC OPTIMIZATION**

Stochastic optimization is dealt with random (noisy) parameters of function measurements or random inputs in the process ofsearching.

**INFINITE-DIMENSIONAL OPTIMIZATION**

Infinite-dimensional optimization deals with the case where the set of feasible solutions is an infinite-dimensionalspace's subset, likely a space of functions.

**HEURISTICS AND METAHEURISTICS**

They make some or no predictionson the problem being optimized. In general, heuristics do not guarantees that any optimal solution want be Sought. On the oppositeside, heuristics are used to seek appropriateanswers for lot ofdifficult optimization problems.

Constraint satisfaction deal with the typeswhere the objective function *f* is fixed or constant (this is harnessed in artificial intelligence, specifically in automated reasoning).

- Constraint programming is a type of programming paradigm in which relations between variables are declared in the format of constraints.

**DISJUNCTIVE PROGRAMMING**

Disjunctive programming is harnessedin whichminimum one constraint should be pleased but not all. It is of certainusage in scheduling.

**SPACE MAPPINGS**

It is anidea for framing and optimizing an engineering unit to top-fidelity (fine) sample accuracy spoiling a fitting physically significant coarse or surrogate sample. Amongnumerous subfields, the techniques are framedbasically for optimization in dynamic contexts (that is called as decision making over the time).

**CALCULUS OF VARIATIONS**

They looks to optimize an action integral towardsfew space to an extreme by changing a function of the coordinate values.

**OPTIMAL CONTROL THEORY**

It is an abstraction of the calculus of change which paves way to control policies.

**DYNAMIC PROGRAMMING**

It is the proceedings towards to solve the stochastic optimization problem besides, randomness, stochastic,and unknown model parameters. It deals the typewhere the optimization strategy is dependent on dividing the problem into smallsub problems. The equation that explains the link between those sub problems is known as the Bellman equation.

**MATHEMATICAL PROGRAMMING WITH EQUILIBRIUM CONSTRAINTS**

It is a programming in which the constraints contains complementarities or variational inequalities.

**MIXED-INTEGER LINEAR PROGRAMMING**

Mixed integer linear programming (MILP) describes an efficient mathematical modelling proceedings to solve difficult optimisation tasks and find the potential trade-offs among conflicting objectives, that can lend a better knowledge of bioenergy systems and hold up decision-makers briefing the sustainable roots towards bioenergy destinations.

**MIXED-INTEGER PROGRAMMING (MIP) PROBLEMS**

A mixed-integer programming (MIP) problem is the one in which constraining few of the decision variables to the values of integer occurs (i.e. non fractional or whole numbers such as -2, -1, 0, 1, 2, 3 etc.) at the optimal solution. The usage of integer variables majorlyelaborates the view of helpful optimization sums that you can declare and solve.

Aprimary special type is a decision variable X1 whichshould be either 0 or 1 at the answer. That variables are known asbinary integer or 0-1 variables and can be utilizedfor the modelmodelling yes/no decisions, such as whether to build a plant or buy a piece of equipment. However, integer variables make an optimization problem non-convex, and therefore far more difficult to solve. Memory and solution time may rise exponentially as you add more integer variables.

Even with highly sophisticated algorithms and modern supercomputers, there are models with just a few hundred integer variables that have never been solved to optimality. This is because many combinations of specific integer values for the variables must be tested, and each combination requires the solution of a "normal" linear or nonlinear optimization problem. The number of combinations can rise exponentially with the size of the problem.

Optimizers solve mixed-integer and constraint programming problems using these methods:

- Branch and Bound
- Strong Branching
- Pre-processing and Probing
- Cut Generation
- Integer Heuristics
- Non-traditionalMethods

## BRANCH AND BOUND

The standard Microsoft Excel Solver uses a basic implementation of the Branch and Bound method to solve MIP problems. Its speed limitations make it suitable only for problems with a small number (perhaps 50 to 100) integer variables.

The Premium Solver and Premium Solver Platform use an extended Branch and Bound method that supports the alldifferent constraint as a native type, as well as reduced cost fixing for integer variables. It also uses more sophisticated rules for choosing the next node to explore and the next variable to branch upon, based upon pseudocosts which are estimates of the change in the objective that will result from branching on a given variable.

The Large-Scale GRG Solver, Large-Scale SQP Solver, KNITRO Solver, MOSEK Solver, and LGO Global Solver make use of the Premium Solver Platform's Branch and Bound method to handle integer variables and the alldifferent constraint.

The Large-Scale LP Solver an integrated Branch and Bound plus Cut Generation strategy, often called Branch and Cut. It supports the alldifferent constraint by generating an equivalent matrix of 0-1 variables and incorporating these into the problem. Its Branch and Bound method uses pseudocosts, degradation factors and strong branching, and it implements a number of cuts.

The XPRESS Solver Engine uses an integrated and highly tuned Branch and Cut strategy. It uses a variety of node selection and branch variable selection strategies, including pseudocosts, degradation factors and strong branching, and offers many user options for controlling the search strategy. Like the Large-Scale LP Solver, it supports the alldifferent constraint by generating an equivalent matrix of 0-1 variables and incorporating these into the problem.

The Gurobi Solver Engine also uses an integrated and highly tuned Branch and Cut strategy, with a variety of node selection and branch variable selection strategies. It was designed to take maximum advantage of multi-core processors by parallelizing the Branch and Bound search.

Like the XPRESS Solver Engine, it supports the alldifferent constraint by generating an equivalent matrix of 0-1 variables and incorporating these into the problem.

**STRONG BRANCHING**

Strong Branching is a method used to estimate the impact of branching on each integer variable on the objective function (itspseudocost), by performing a few iterations of the Dual Simplex method. Such pseudocosts are used to guide the choice of the next subproblem to explore, and the next integer variable to branch upon, throughout the Branch and Bound process.

The Large-Scale LP Solver, XPRESS Solver Engine and Gurobi Solver Engine use Strong Branching techniques in their own Branch and Bound methods.

**PREPROCESSING AND PROBING**

Preprocessing and probing strategies exploit the special properties of 0-1 or binary integer variables. For example, they use the constrained settings of certain 0-1 variables to determine settings for other 0-1 variables, without solving an optimization subproblem.

The standard Microsoft Excel Solver and the Premium Solver do not employ any such strategies. The Premium Solver Platform uses several Preprocessing and Probing methods includingfeasibility testing, optimality fixing, bounds improvement and variable reordering for Branch and Bound.

The Large-Scale LP Solver, Large-Scale SQP Solver, and MOSEK Solver make full use of the Premium Solver Platform's Preprocessing and Probing methods.

The XPRESS Solver Engine and Gurobi Solver Engine both use a variety of Preprocessing and Probing strategies including most of the logical preprocessing methods of the Premium Solver Platform, reduced cost fixing, and probing at the top node.

**CUT GENERATION**

Cut Generation involves the automatic generation of additional constraints, or "cuts," that reduce the size of the feasible region for the optimization subproblems that must be solved, without eliminating any potential integer solutions.

The LP/Quadratic Solver in the Premium Solver Platform can generate both Gomory Cuts and Lifted Cover Inequalities at the root node, using a "Cut and Branch" framework. The Large-Scale SQP Solver and the MOSEK Solver can generate Lifted Cover Inequalities at the root node.

The Large-Scale LP Solver uses a wide range of Cut Generation methods. It can generate Lift and Cover, Rounding, Knapsack, Gomory, Clique and "Odd Hole" cuts in several passes at any node in the Branch and Bound tree.

The XPRESS Solver Engine employs sophisticated Cut Generation methods in an integrated Branch and Cut framework. It can generate both Gomory Cuts and Lifted Cover Inequalities at

any node. User options make it possible to control cut frequency and the depth of nodes eligible for cut generation.

The Gurobi Solver Engine also employs many sophisticated Cut Generation methods in an integrated Branch and Cut framework. It gives users control of the overall degree and frequency of cut generation, but wherever possible it makes an automatic choice of the best methods for a specific problem.

**INTEGER HEURISTICS**

Heuristics are "rules of thumb" that may often, but not always, succeed in achieving a given result. The Evolutionary Solver and the LP/Quadratic Solver in the Premium Solver Platform, and the Large-Scale SQP Solver and MOSEK Solver each use heuristics to attempt to find an integer feasible solution, or "incumbent," early in the Branch and Bound search. Such an incumbent can be used to prune the search tree and save time later in the search.

The XPRESS Solver Engine and Gurobi Solver Engine both make sophisticated use of integer heuristics. User options make it possible to control the type and frequency of application of these heuristic rules.

**NONTRADITIONAL METHODS**

The Evolutionary Solver built into the Premium Solver Platform and the OptQuest Solver use "non-traditional methods" to handle integer variables and the alldifferent constraint. In both of these solvers, integer variables and permutations are represented directly, and candidate solutions are generated that always satisfy integer and alldifferent constraints. The Evolutionary Solver uses several different integer- and permutation-preserving mutation and crossover operators to generate new candidate solutions.

**PROBLEMS**

**MIXED-INTEGER LINEAR PROGRAMMINGBASICS: PROBLEM-BASED**

**PROBLEM DESCRIPTION**

You want to blend steels with various chemical compositions to obtain 25 tons of steel with a specific chemical composition. The result should have 5% carbon and 5% molybdenum by weight, meaning 25 tons*5% = 1.25 tons of carbon and 1.25 tons of molybdenum. The objective is to minimize the cost for blending the steel.

Four ingots of steel are available for purchase. Only one of each ingot is available.

| Ingot | Weight in Tons | %Carbon | %Molybdenum | Cost Tons |
|---|---|---|---|---|
| 1 | 5 | 5 | 3 | $350 |
| 2 | 4 | 4 | 3 | $330 |

| | | | | |
|---|---|---|---|---|
| **3** | **3** | **5** | **4** | **$310** |
| **4** | **2** | **3** | **4** | **$280** |

Three grades of alloy steel and one grade of scrap steel are available for purchase. Alloy and scrap steels can be purchased in fractional amounts.

| **Alloy** | **%Carbon** | **%Molybdenum** | **Cost Ton** |
|---|---|---|---|
| **1** | **8** | **6** | **$500** |
| **2** | **7** | **7** | **$450** |
| **3** | **6** | **8** | **$400** |
| **Scrap** | **3** | **9** | **$100** |

**FORMULATE PROBLEM:**

To formulate the problem, first decide on the control variables. Take variable ingots(1) = 1 to mean that you purchase ingot **1**, and ingots(1) = 0 to mean that you do not purchase the ingot. Similarly, variables ingots(2) through ingots(4) are binary variables indicating whether you purchase ingots **2** through **4**.

Variables alloys(1) through alloys(3) are the quantities in tons of alloys **1**, **2**, and **3** that you purchase. scrap is the quantity of scrap steel that you purchase.

```
steelprob = optimproblem;
ingots = optimvar('ingots',4,'Type','integer','LowerBound',0,'UpperBound',1);
alloys = optimvar('alloys',3,'LowerBound',0);
scrap = optimvar('scrap','LowerBound',0);
```

Create expressions for the costs associated with the variables.

```
weightIngots = [5,3,4,6];
costIngots = weightIngots.*[350,330,310,280];
costAlloys = [500,450,400];
costScrap = 100;
cost = costIngots*ingots + costAlloys*alloys + costScrap*scrap;
```

Include the cost as the objective function in the problem

```
steelprob.Objective = cost;
```

The problem has three equality constraints. The first constraint is that the total weight is 25 tons. Calculate the weight of the steel.

```
totalWeight = weightIngots*ingots + sum(alloys) + scrap;
```

The second constraint is that the weight of carbon is 5% of 25 tons, or 1.25 tons. Calculate the weight of the carbon in the steel.

```
carbonIngots = [5,4,5,3]/100;
carbonAlloys = [8,7,6]/100;
carbonScrap = 3/100;
totalCarbon = (weightIngots.*carbonIngots)*ingots + carbonAlloys*alloys +
carbonScrap*scrap
;
```

The third constraint is that the weight of molybdenum is 1.25 tons. Calculate the weight of the molybdenum in the steel.

```
molybIngots = [3,3,4,4]/100;
molybAlloys = [6,7,8]/100;
molybScrap = 9/100;
totalMolyb = (weightIngots.*molybIngots)*ingots + molybAlloys*alloys +
molybScrap*scrap;
```

Include the constraints in the problem

```
steelprob.Constraints.conswt = totalWeight == 25;
steelprob.Constraints.conscarb = totalCarbon == 1.25;
steelprob.Constraints.consmolyb = totalMolyb == 1.25;
```

**SOLVE PROBLEM:**

Now that you have all the inputs call the solver

```
[sol,fval] = solve(steelprob);
```

```
Solving problem using intlinprog.
LP:             Optimal objective value is 8125.600000.

Cut Generation:   Applied 3 mir cuts.
                  Lower bound is 8495.000000.
                  Relative gap is 0.00%.


Optimal solution found.
```

Intlinprog stopped at the root node because the objective value is within a gap tolerance of the optimal value, options.AbsoluteGapTolerance = 0 (the default value). The intcon variables are integer within tolerance, options.IntegerTolerance = 1e-05 (the default value).

View the solution.

***sol.ingots***

ans = *4×1*

1.0000
1.0000
0
1.0000

***sol.alloys***

ans = *3×1*

7.2500
0
0.2500

***sol.scrap***

ans = 3.5000

***fval***

fval = 8.4950e+03

The optimal purchase costs $8,495. Buy ingots **1**, **2**, and **4**, but not **3**, and buy 7.25 tons of alloy **1**, 0.25 ton of alloy **3**, and 3.5 tons of scrap steel.

## CONCLUSION

In this Paper, we have made a detailed view about the Mathematical Optimization and their classifications. And we made a elaborate study over the Mixed Integer Linear Programming and its various methods for solving the problems. And we have solved a basic problem, in which a steel blending should be done in certain chemical composition and we were required to find the right combination that brings us minimal cost for blending the steel and we have successfully solved the Problem using Mixed Integer Linear Programming.

## REFERENCE:

[1]. Chanas.S and Kutcha.D, A concept of the optimal solution of the transportation problem . With fuzzy cost coefficients, Fuzzy sets and
systems, 82,299-305, (1996).
[2]. Chanas.S, Kolodziejczyk.W and Machaj.A, A fuzzy approach to the transportation problem, Fuzzy Sets and Systems, 13, 211-221,
(1984)
[3]. Chiang, J.: The Optimal Solution of the Transportation Problem with Fuzzy Demand and Fuzzy Product. J. Info. Science and
Engineering 21, 439–451 (2005)
[4]. Liu, S.-T.: Fuzzy Total Transportation Cost Measures for Fuzzy Solid Transportation Problem. Applied Math. and
Computation 174, 927–941 (2006)
[5]. Taha, H.A.: Operations Research. Macmillan, New York (1992)