OPTIMA Graduate Research Workshop
On Benders decomposition

Andreas Ernst
Alysson M. Costa

Mixed-Integer Programming (MIP) is a highly effective method for solving optimisation problems. The remarkable progress made by MIP solvers such as CPLEX and GUROBI in the last three decades has resulted in the widespread use of the technique as a black-box approach for various industrial applications. All that is required of the practitioner is to model the problem as a Mixed-Integer Programming problem, and the solver takes care of the rest.

However, the complexity of some industrial applications exceeds the capabilities of these solvers. In order to advance the use of MIP, decomposition techniques are a popular approach. These methods divide the problem into smaller sub-problems that can be efficiently handled by the solvers. Benders decomposition is one of the most successful techniques in this regard.

The purpose of this workshop is to provide an introduction to the main concepts of Benders decomposition in a user-friendly manner. Participants will receive a hands-on tutorial on implementing an algorithm for a toy problem, which will serve as a gentle introduction to the technique. In the second part of the workshop, we will discuss recent advances in the field and demonstrate how this technique can be applied to solve large-scale industrial problems.

Please note that participants are required to bring their laptops and register for an account at Google Colaboratory (https://colab.research.google.com/) in order to participate in the hands-on portion of the workshop.

Mixed-Integer Programming

$$
\begin{aligned}
\text{Maximise} \quad & z = c^t x + d^t y \\
\text{s.t.} \quad & Ax + By \geq b, \\
& Dy \geq e, \\
& x \geq 0, y \geq 0 \text{ and integer.}
\end{aligned}
$$

Example

$$
\begin{aligned}
\text{Maximise} \quad & z = x_1 - x_2 + y_1 + y_2 \\
\text{s.t.} \quad & -x_1 - x_2 - y_2 \geq -2, \\
& x_1 x_2 - y_1 \geq 1, \\
& x_1, x_2 \geq 0, \\
& y_1, y_2 \in 0, 1.
\end{aligned}
$$

Give me a list of applications of Mixed-Integer Programming

1. **Production planning**: Mixed-Integer Programming can be used to optimize production planning, such as determining the optimal mix of products to manufacture or the best schedule for production runs.
2. **Supply chain management**: Mixed-Integer Programming can be used to optimize supply chain decisions, such as transportation and inventory management.
3. **Facility location and network design**: Mixed-Integer Programming can be used to determine the optimal location of facilities and the design of supply chain networks.
4. **Resource allocation**: Mixed-Integer Programming can be used to optimize resource allocation decisions, such as the allocation of labor and equipment.
5. **Financial planning and portfolio optimization**: Mixed-Integer Programming can be used to optimize financial planning and portfolio optimization decisions, such as asset allocation and investment selection.
6. **Energy and environmental systems**: Mixed-Integer Programming can be used to optimize energy systems, such as the design and operation of power grids, and to optimize environmental systems, such as waste management and pollution reduction.
7. **Telecommunications and network optimization**: Mixed-Integer Programming can be used to optimize network design and routing decisions, such as the placement of cell towers and the routing of network traffic.
8. **Healthcare management**: Mixed-Integer Programming can be used to optimize healthcare decisions, such as the allocation of hospital resources and the design of clinical trials.
9. **Transportation and logistics**: Mixed-Integer Programming can be used to optimize transportation and logistics decisions, such as route optimization and vehicle scheduling.
10. **Scheduling and timetabling**: Mixed-Integer Programming can be used to optimize scheduling and timetabling decisions, such as employee scheduling and course timetabling.

Example

$$
\begin{aligned}
\text{Maximise} \quad & z = x_1 - x_2 + y_1 + y_2 \\
\text{s.t.} \quad & -x_1 - x_2 - y_2 \geq -2, \\
& x_1 + x_2 - y_1 \geq 1, \\
& x_1, x_2 \geq 0, \\
& y_1, y_2 \in 0, 1.
\end{aligned}
$$

Implementation

```python
import gurobipy as gp
from gurobipy import GRB

m = gp.Model("A Linear Programming model")


#variables

indices = [1, 2]
x = m.addVars(indices, name = 'x',)
y = m.addVars(indices, name = 'y', vtype = GRB.BINARY)

#objective
m.setObjective( x[1] - x[2] + y[1] + y[2] , GRB.MAXIMIZE)

#constraints
cons1 = m.addConstr( -x[1] - x[2] - y[2]   >= -2)
cons2 = m.addConstr( x[1] + x[2] - y[1]   >= 1)

#solve
m.optimize()

#Optimal solution
for i in indices:
  print("x[{}] = ".format(i), x[i].x)
for i in indices:
  print("y[{}] = ".format(i), y[i].x)
print("Optimal solution value:", m.objVal )
```


ECONOMETRICA

VOLUME 28 July, 1960 NUMBER 3

AN AUTOMATIC METHOD OF SOLVING DISCRETE PROGRAMMING PROBLEMS

BY A. H. LAND AND A. G. DOIG

In the classical linear programming problem the behaviour of continuous, nonnegative variables subject to a system of linear inequalities is investigated. One possible generalization of this problem is to relax the continuity condition on the variables. This paper presents a simple numerical algorithm for the solution of programming problems in which some or all of the variables can take only discrete values. The algorithm requires no special techniques beyond those used in ordinary linear programming, and lends itself to automatic computing. Its use is illustrated on two numerical examples.

```
Username
Academic license - for non-commercial use only - expires 2023-11-04
Gurobi Optimizer version 9.5.0 build v9.5.0rc5 (mac64[x86])
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 2 rows, 4 columns and 6 nonzeros
Model fingerprint: 0x252cfa2e
Variable types: 2 continuous, 2 integer (2 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [1e+00, 2e+00]
Presolve removed 2 rows and 4 columns
Presolve time: 0.00s
Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)
Thread count was 1 (of 8 available processors)

Solution count 1: 3

Optimal solution found (tolerance 1.00e-04)
Best objective 3.000000000000e+00, best bound 3.000000000000e+00, gap 0.0000%
x[1] =  2.0
x[2] =  0.0
y[1] =  1.0
y[2] =  0.0
Optimal solution value: 3.0
[Finished in 0.5s]
```

Benders decomposition

For many years, Benders decomposition [Benders, 1962] was a brilliant theoretical result with little applicability in practice.

Numerische Mathematik 4, 238—252 (1962)

Partitioning procedures for solving mixed-variables programming problems*

By

J. F. BENDERS**

I. Introduction

In this paper two slightly different procedures are presented for solving mixed-variables programming problems of the type

$$\max\{c^T x + f(y) \mid A x + F(y) \leqq b,\ x \in R_p,\ y \in S\}, \tag{1.1}$$

where $x \in R_p$ (the p-dimensional Euclidean space), $y \in R_q$, and S is an arbitrary subset of R_q. Furthermore, A is an (m, p) matrix, $f(y)$ is a scalar function and $F(y)$ an m-component vector function both defined on S, and b and c are fixed vectors in R_m and R_p, respectively.

An example is the mixed-integer programming problem in which certain variables may assume any value on a given interval, whereas others are restricted to integral values only. In this case S is a set of vectors in R_q with integral-valued components. Various methods for solving this problem have been proposed by BEALE [1], GOMORY [9] and LAND and DOIG [11]. The use of integer variables, in particular for incorporating in the programming problem a choice from a set of alternative discrete decisions, has been discussed by DANTZIG [4].

Industrial applications

It took 12 years for Benders decomposition to be first used on a practical application [Geoffrion and Graves, 1974].

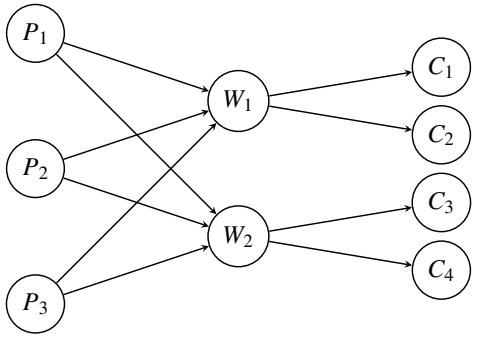
MANAGEMENT SCIENCE
Vol. 20, No. 5, January, 1974
Printed in U.S.A.

MULTICOMMODITY DISTRIBUTION SYSTEM DESIGN BY BENDERS DECOMPOSITION*†

A. M. GEOFFRION AND G. W. GRAVES§

University of California, Los Angeles

A commonly occurring problem in distribution system design is the optimal location of intermediate distribution facilities between plants and customers. A multicommodity capacitated single-period version of this problem is formulated as a mixed integer linear program. A solution technique based on Benders Decomposition is developed, implemented, and successfully applied to a real problem for a major food firm with 17 commodity classes, 14 plants, 45 possible distribution center sites, and 121 customer zones. An essentially optimal solution was found and proven with a surprisingly small number of Benders cuts. Some discussion is given concerning why this problem class appears to be so amenable to solution by Benders' method, and also concerning what we feel to be the proper professional use of the present computational technique.



The Benders reformulation

Consider again a generic MIP model:

$$\begin{aligned} \text{Maximise} \quad & z = c^t x + d^t y \\ \text{s.t.} \quad & Ax + By \geq b, \\ & Dy \geq e, \\ & x \geq 0, y \geq 0 \text{ and integer.} \end{aligned}$$

This problem can be rewritten as:

$$\text{Maximise}_{y \in Y} \{ dy + \text{Maximise}_{x \geq 0} \{ c^t x : Ax \leq b - By \} \},$$

with $Y = \{y \mid Dy \geq e,\ y \geq 0 \text{ and integer}\}$.

The inner problem is a linear programming model and is known as the Benders decomposition **subproblem**.

Consider a tentative solution $\overline{y}$. Associating dual variables u to constraints $Ax \geq b - B\overline{y}$, we can write the dual version of subproblem as

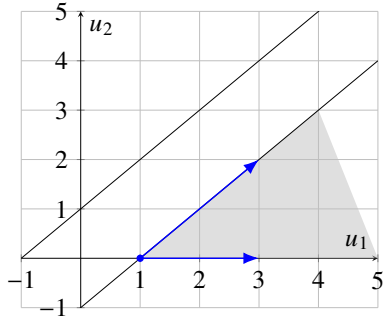
$$\text{Minimise}_{u \geq 0} \{ u(b - B\overline{y}) : uA \geq c \}.$$

Let $\mathbb{F} = \{u \mid u \geq 0 \ ; \ uA \geq c\}$. We assume that $\mathbb{F}$ is not empty for it would correspond to a primal problem either infeasible or unbounded.

$\mathbb{F}$ is therefore composed of extreme points u^p (for $p = 1 \ldots P$) and extreme rays r^q (for $q = 1 \ldots Q$).

Example

$$\begin{aligned} \text{Minimise} \quad & (2 - \overline{y}_2)u_1 + (-1 - \overline{y}_1)u_2 \\ & u_1 - u_2 \geq 1, \\ & u_1 - u_2 \geq -1, \\ & u_1, u_2 \geq 0. \end{aligned}$$

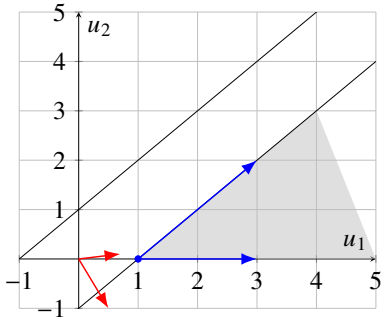


Using duality theory, the primal and dual formulations can be interchanged.

$$\text{Maximise}_{\overline{y}\in Y}\{\, d\overline{y} + \text{Minimise}_{u\geq 0}\{u(b - B\overline{y}) : uA \geq c\}\,\}.$$

The feasible space of the dual subproblem does not depend on the choice of $\overline{y}$.

The objective function of the subproblem depends on the choice $\overline{y}$ and can be either bounded or unbounded.



In the first case, we obtain an extreme point u^p. In the latter situation, there is a direction r^q for which $r^q(b - B\overline{y}) > 0$. [Costa et al., 2009].

A tentative solution $\overline{y}$ that provides an unbounded solution to the dual subproblem implies an infeasible subproblem and must be avoided. This is done by adding constraints:

$$r^q(b - B\overline{y}) \leq 0, \qquad q = 1 \ldots Q. \tag{1}$$

Example: $y = (1, 1)$

$$\begin{aligned} \text{Minimise} \quad & u_1 - 2u_2 \\ \text{s.t.} \quad & u_1 - u_2 \geq 1, \\ & u_1 - u_2 \geq -1, \\ & u_1, u_2 \geq 0. \end{aligned}$$

Adding constraints (1) to the formulation.

$$\begin{aligned} &\text{Maximise}_{\overline{y}\in Y}\ d^t y + \{\text{Minimise}_{u\geq 0}\{u(b - B\overline{y}) : uA \geq c\} \\ &\text{s.t.} \quad r^q(b - By) \leq 0. \end{aligned}$$

Now the solution must be one of the extreme points $u_p,\ p = 1, \ldots, P$.

$$\begin{aligned} &\text{Maximise}_{\overline{y}\in Y}\ d^t y + \{\text{Minimise}\{u^p(b - By) : p = 1, \ldots, P\} \\ &\text{s.t.} \quad r^q(b - By) \leq 0, \quad q = 1, \ldots, Q. \end{aligned}$$

Which can be linearised with the use of a continuous variable z as:

$$\begin{aligned} &\text{Maximise}_{\overline{y}\in Y}\ d^t y + \{\text{Minimise}\{u^p(b - By) : p = 1, \ldots, P\} \\ &\text{s.t.} \quad r^q(b - By) \leq 0, \quad q = 1, \ldots, Q, \\ &\qquad z \leq u^p(b - By), \quad p = 1, \ldots, P. \end{aligned}$$

This is the **Benders reformulation**.

A delayed generation of constraints

The idea of the **Benders decomposition algorithm** is to ignore most of the initial constraints of the Benders reformulation and generate them as needed.

We start with a relaxed version of the reformulation called the *relaxed master problem.*

$$\begin{aligned} \text{Maximise}_{\overline{y} \in Y} \; & d^t y + z \\ \text{s.t.} \quad & z \geq -M. \end{aligned}$$

This gives us tentative values for the integer variables, $\overline{y}$, that are sent to the subproblem:

$$\begin{aligned} \text{Minimise}_{u \geq 0} \; & u(b - B\overline{y}) \\ \text{s.t.} \quad & uA \geq c. \end{aligned}$$

which returns us an extreme ray r^q or an extreme point u^p that can be used to generate a feasibility or optimality cut for the master problem (and the method iterates).

End of the method

Every time the Master problem is run, we have a dual bound for the problem (i.e., we have solved a relaxation of the problem). Every time the subproblem finds an extreme point, we have found a primal bound for the problem (i.e.,we have found a feasible solution).

The method converges when the last dual bound found is equal (up to a tolerance) to the best primal bound found during the process.

Improving convergence

There are many ways to improve the convergence of the Benders decomposition algorithm. See, for example, [Costa, 2005, Rahmaniani et al., 2017].

Example

Solve the optimisation problem below using Benders decomposition

$$\begin{aligned} \text{Maximise} \quad & z = x_1 - x_2 + y_1 + y_2 \\ \text{s.t.} \quad & -x_1 - x_2 - y_2 \geq -2, \\ & x_1 + x_2 - y_1 \geq 1, \\ & x_1, x_2 \geq 0, \\ & y_1, y_2 \in 0, 1. \end{aligned}$$

Master Problem template:

```
import gurobipy as gp
from gurobipy import GRB

m = gp.Model(#fill)
y = m.addVars(#fill)
z = m.addVar(#fill, ub=10000)
m.setObjective(#fill)
```

Subproblem template:

```
import gurobipy as gp
from gurobipy import GRB

s = gp.Model(#fill)
s.Params.InfUnbdInfo = 1 #allow to get dual ray
x = s.addVars(#fill)
cons1 = s.addConstr(#fill, name="cons1")
cons2 = s.addConstr(#fill, name="cons2")
```


External loop template:

```
import gurobipy as gp
from gurobipy import GRB

#Benders loop
LB = -100000; UB = 100000

while(UB - LB >= 0.00001):
  #optimize Master
  m.optimize()
  UB = m.objVal

  #update the right hand side of the subproblem
  cons1.rhs = #fill
  cons2.rhs = #fill

  #optimize sub
  s.optimize()

  #generate cuts
  if s.status == 3:
    print("Infeasibility cut")
    # get the dual ray
    u1 = cons1.getAttr('FarkasDual')
    u2 = cons2.getAttr('FarkasDual')
    m.addConstr(#fill)  #add the new cut
  else:
    print("Feasibility cut")
    # get the dual ray
    u1 = cons1.getAttr('Pi')
    u2 = cons2.getAttr('Pi')
    m.addConstr( #fill)

    #fill (update the best lower bound if necessary).

  print(LB, " ", UB)

#Print solution
```

An implementation using lazy constraints:

```
#!/usr/bin/env python3.7
import gurobipy as gp
from gurobipy import GRB


#Initial Subproblem:
def initial_sub():
  s = gp.Model()
  s.Params.InfUnbdInfo = 1
  s.Params.OutputFlag= 0
  x = s.addVars(2, name="x")
  cons1 = s.addConstr(x[0] + x[1] <= 0, name="cons1")
  cons2 = s.addConstr(-x[0] -x[1] <= 0, name="cons2")
  s.setObjective(x[0] - x[1],GRB.MAXIMIZE)
  return s

#Update subproblem
def update_subproblem(s,y1,y2):
  cons1 = s.getConstrByName("cons1")
  cons2 = s.getConstrByName("cons2")
  cons1.rhs = 2 - y2
  cons2.rhs = -1 - y1

def generate_cuts(model,where):
  if where == GRB.Callback.MIPSOL:
    valsy = model.cbGetSolution(model._y)
    print(valsy)
    update_subproblem(s,valsy[0],valsy[1])
    s.optimize()
    if s.status == 3:
      print("Infeasibility constraint")
      cons1 = s.getConstrByName("cons1")
      cons2 = s.getConstrByName("cons2")
      u1 = cons1.getAttr('FarkasDual')
      u2 = cons2.getAttr('FarkasDual')
      expr = u1*(2 - y[1]) + u2*(-1  - y[0])
      print(expr, ">=", "0")
```



```
      model.cbLazy( expr >= 0)
    else:
      print("Feasibility constraint")
      cons1 = s.getConstrByName("cons1")
      cons2 = s.getConstrByName("cons2")
      u1 = cons1.getAttr('Pi')
      u2 = cons2.getAttr('Pi')
      expr = u1*(2 - y[1]) + u2*(-1  - y[0])
      print(expr, ">=", "z")
      model.cbLazy( expr >= m._z)

s = initial_sub()
s.write("model.lp")

#master
m = gp.Model()
y = m.addVars(2, vtype = GRB.BINARY, name="y")
z = m.addVar(name="z", ub=1000)

m.setObjective(y[0]  + y[1] + z, GRB.MAXIMIZE)

m.Params.LazyConstraints = 1
m._y = y
m._z = z
m.Params.OutputFlag= 0
m.optimize(generate_cuts)

s.write("model.lp")


# Display optimal values of decision variables
for v in m.getVars():
    if v.x > 1e-6:
        print(v.varName, v.x)

# Display optimal solution value
print('Total profit: ', m.objVal)
```

Bibliography

[Benders, 1962] Benders, J. F. (1962). Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252.

[Costa, 2005] Costa, A. M. (2005). A survey on Benders decomposition applied to fixed-charge network design problems. *Computers & Operations Research*, 32:1429–1450.

[Costa et al., 2009] Costa, A. M., Cordeau, J.-F., and Gendron, B. (2009). Benders, metric and cutset inequalities for multicommodity capacitated network design. *Computational Optimization and Applications*, 42:371–392.

[Geoffrion and Graves, 1974] Geoffrion, A. M. and Graves, G. W. (1974). Multicommodity Distribution System Design by Benders Decomposition. *Management Science*, 20:822–844.

[Rahmaniani et al., 2017] Rahmaniani, R., Crainic, T. G., Gendreau, M., and Rei, W. (2017). The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259:801–817.