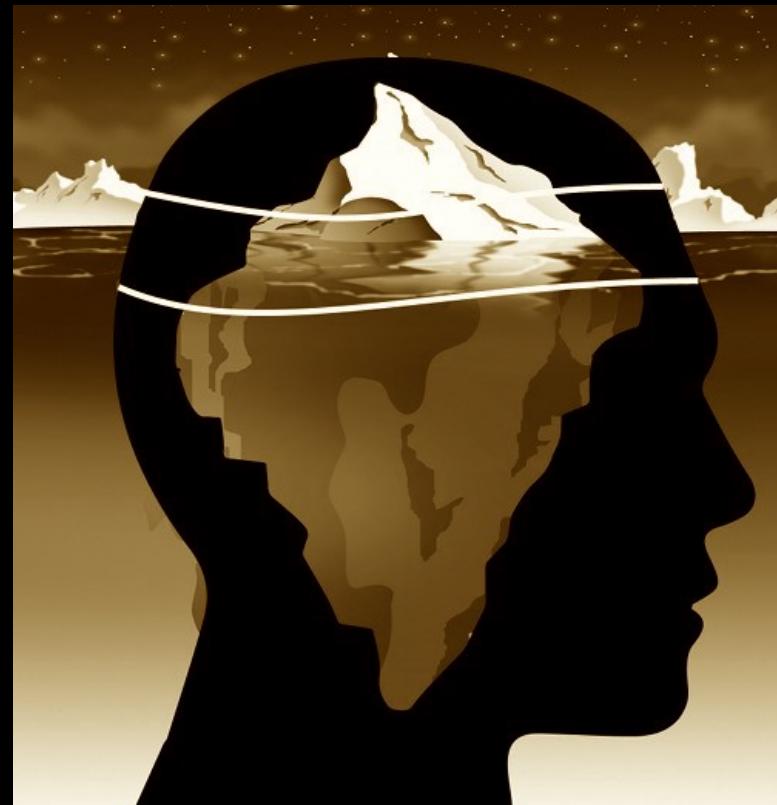


# On Benders decomposition (and on Jung's individuation process)

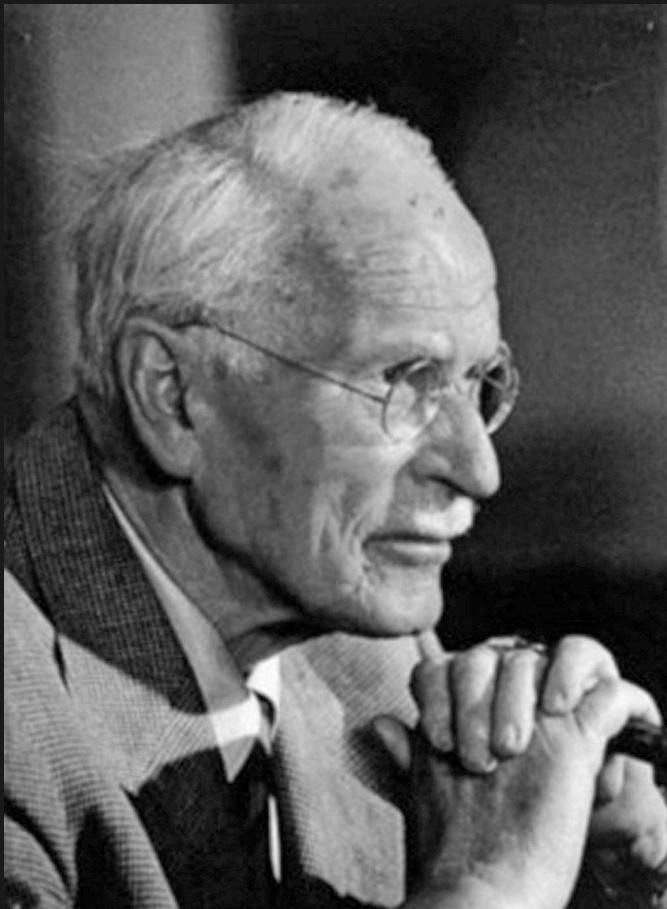
Andreas Ernst  
Alysson M. Costa



May 29, 2023

OPTIMA GR Conference

## The individuation process



“The only meaningful life is a life that strives for the individual realization — absolute and unconditional— of its own particular law ... To the extent that a person who is untrue to the law of his being ... has failed to realise his own life's meaning.”

— C.G. Jung

# Jung's word association Test

1. Transmission of sound to the ear of the recipient.
  2. Neural conduction to the auditory centre.
  3. Word-recognition (primary identification).
  4. Word-comprehension (secondary identification).
  5. Evocation of the associated image, i.e., pure association.
  6. Naming of the idea evoked.
  7. Excitation of the motor speech-apparatus or the motor-centre of the hand when measurement is made by means of a Morse telegraph key.
  8. Neural conduction to the muscle.

# Jung's word association Test

Jung's word association  
Test

Head

Jung's word association  
Test

Green

Jung's word association  
Test

Water

Jung's word association  
Test

To pierce

Jung's word association  
Test

Angel

Jung's word association  
Test

To hit

Jung's word association  
Test

Law

# Jung's word association Test

**Jung's Word Association Test Form**

---

NAME \_\_\_\_\_ DATE \_\_\_\_\_

INTERVIEWER: \_\_\_\_\_

(ANSWER AS QUICKLY AS POSSIBLE WITH THE FIRST WORD THAT OCCURS TO YOUR MIND)

Word	RT	Response	Word	RT	Response
head			frog		
green			to part		
water			hunger		
to sing			white		
dead			child		
long			to take care		
ship			pencil		
pay			sad		
window			plum		
friendly			to marry		
to cook			house		
to ask			sweatheart		
cold			glass		
stem			to quarrel		

1. Transmission of sound to the ear of the recipient.
2. Neural conduction to the auditory centre.
3. Word-recognition (primary identification).
4. Word-comprehension (secondary identification).
5. Evocation of the associated image, i.e., pure association.
6. Naming of the idea evoked.
7. Excitation of the motor speech-apparatus or the motor-centre of the hand when measurement is made by means of a Morse telegraph key.
8. Neural conduction to the muscle.

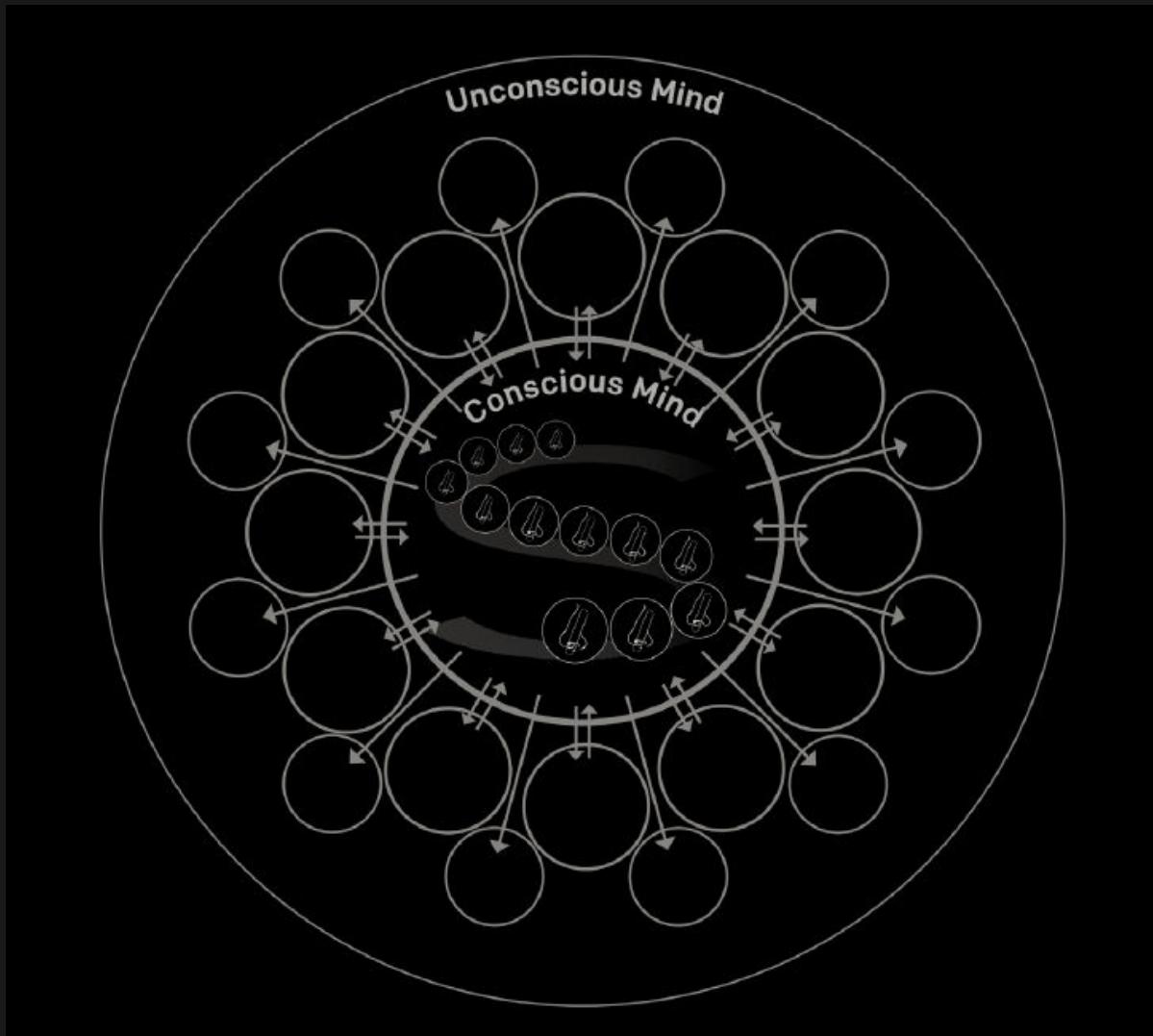
# Jung's word association Test

1. head	-scarf	1.0
2. green	grass	0.8
3. water	-fall	1.0
4. to pierce	to cut	0.8
5. angel	-heart	0.8
60. to hit	marksman	1.2
61. law	not set	4.8

## A **model** of the mind

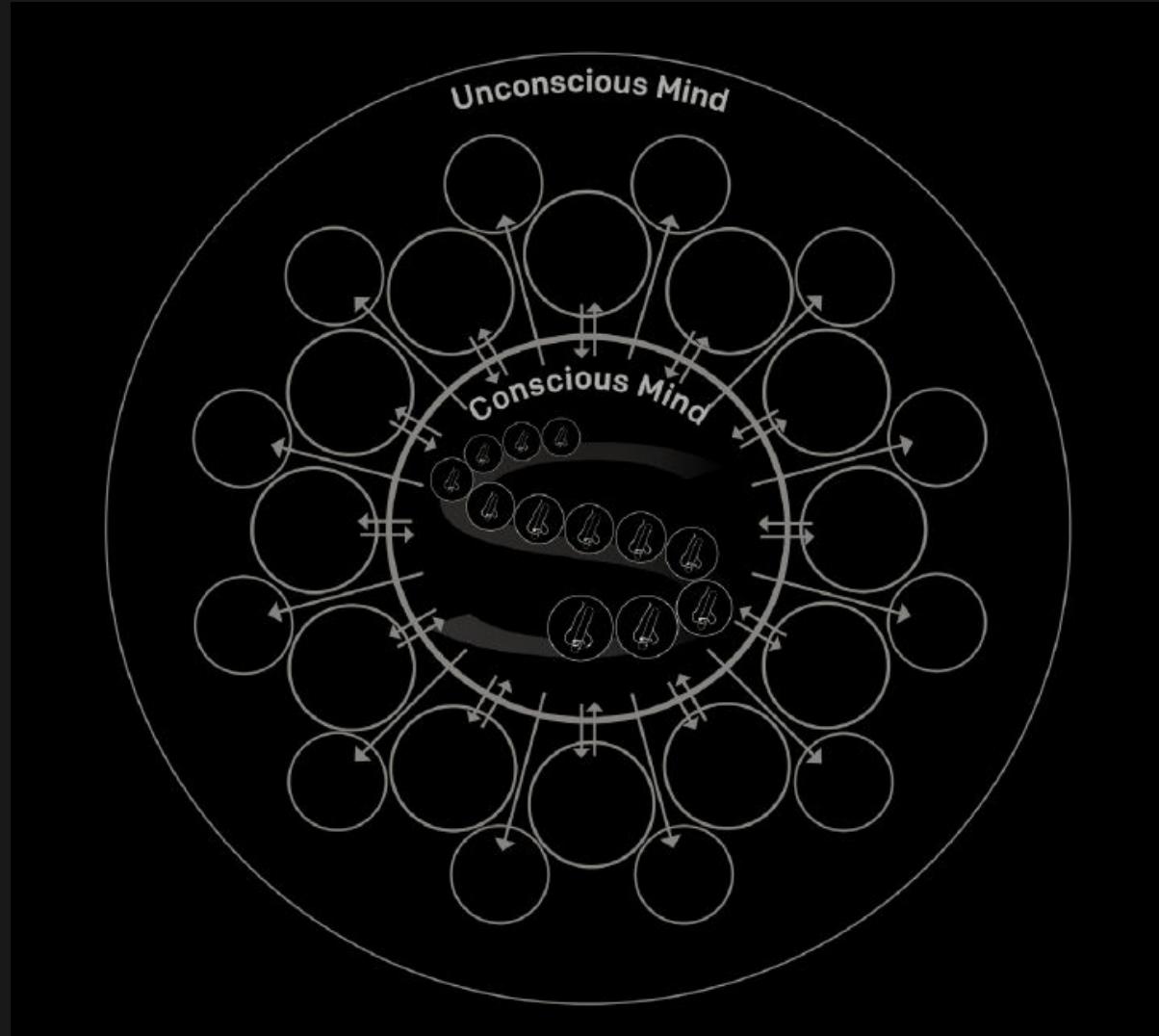


## A model of the mind



The mind illuminated  
Culadasa (John Yates, PhD)

And where is Optimisation ?



The mind illuminated  
Culadasa (John Yates, PhD)

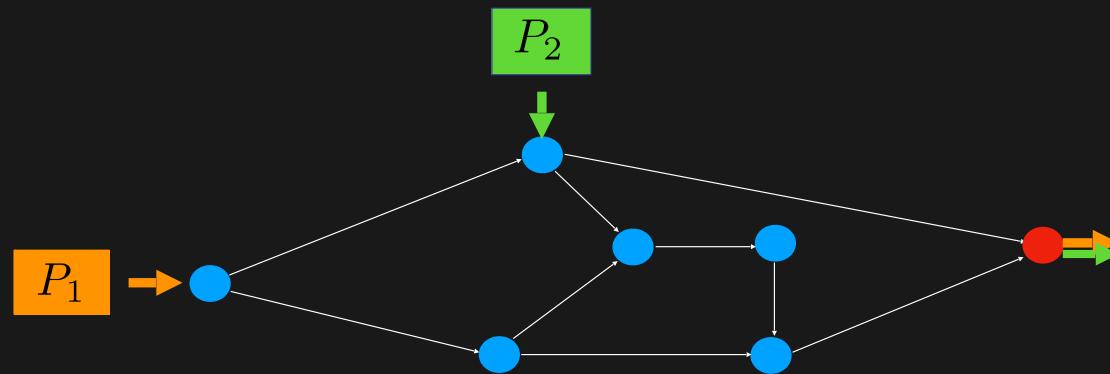
## Network design

Network design problems are central to a large number of contexts.  
(transportation, telecommunications, power systems)

The idea is to establish a network of links  
(roads, optical fibres, electric lines)

to enable the flow of commodities  
(people, data packets, electricity)

## Network design

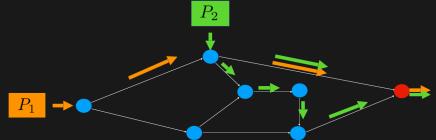


fixed cost to build a link (road, fibre, power line)



variable cost to move commodity

## Network design



$$\text{Minimize} \quad \sum_{(i,j) \in A} \left( f_{ij} y_{ij} + \sum_{k \in K} c_{ij}^k x_{ij}^k \right)$$

subject to

$$\begin{aligned} \sum_{j \in N_i^+} x_{ij}^k - \sum_{j \in N_i^-} x_{ji}^k &= \begin{cases} d_k, & i = O(k), \\ 0, & i \notin \{O(k), D(k)\}, \\ -d_k, & i = D(k), \end{cases} \quad \forall i \in N, \quad \forall k \in K, \\ \sum_{k \in K} x_{ij}^k &\leq u_{ij} y_{ij}, \quad \forall (i, j) \in A, \\ x_{ij}^k &\geq 0, \quad \forall (i, j) \in A, \quad \forall k \in K, \\ y_{ij} &\in \{0, 1\}, \quad \forall (i, j) \in A, \end{aligned}$$

where

$$N_i^+ = \{j | (i, j) \in A\} \text{ and } N_i^- = \{j | (j, i) \in A\}.$$

# 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.

### 1. INTRODUCTION

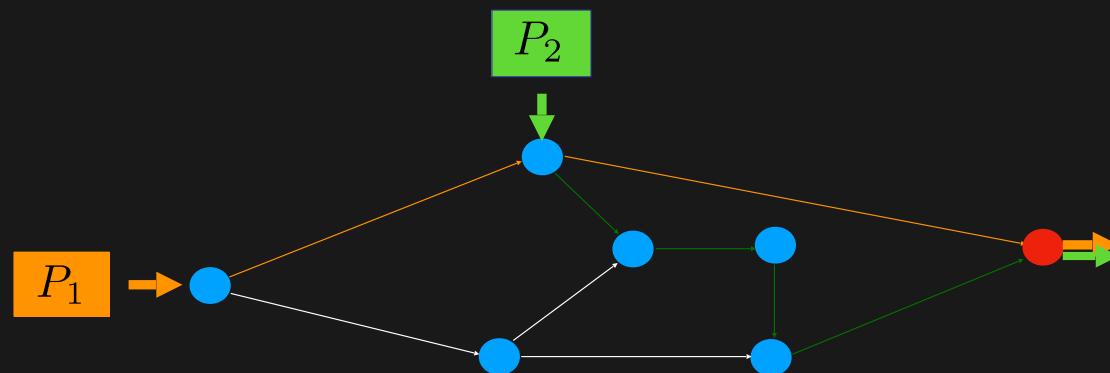
THERE IS A growing literature [1, 3, 5, 6] about optimization problems which could be formulated as linear programming problems with additional constraints that some or all of the variables may take only integral values. This form of linear programming arises whenever there are indivisibilities. It is not meaningful, for instance, to schedule 3–7/10 flights between two cities, or to undertake only 1/4 of the necessary setting up operation for running a job through a machine shop. Yet it is basic to linear programming that the variables are free to take on any positive value,<sup>1</sup> and this sort of answer is very likely to turn up.

In some cases, notably those which can be expressed as transport problems, the linear programming solution will itself yield discrete values of the variables. In other cases the percentage change in the maximand<sup>2</sup> from common sense rounding of the variables is sufficiently small to be neglected. But there remain many problems where the discrete variable constraints are significant and costly.

Until recently there was no general automatic routine for solving such

Branch-and-bound

## Network design



fixed cost to build a link (road, fibre, power line)

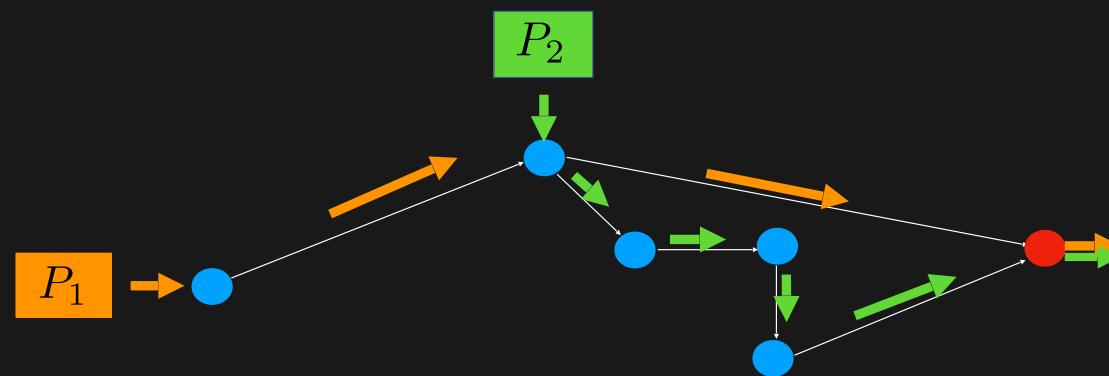


variable cost to move commodity

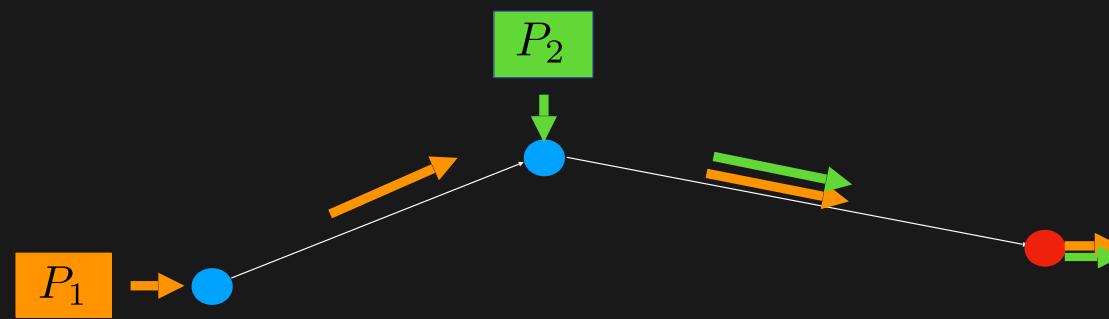
# Network design

A decomposition process

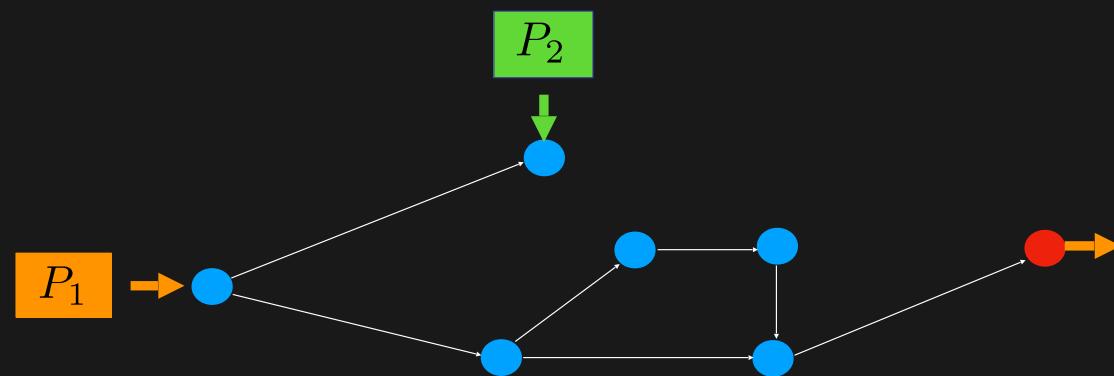
## Network design



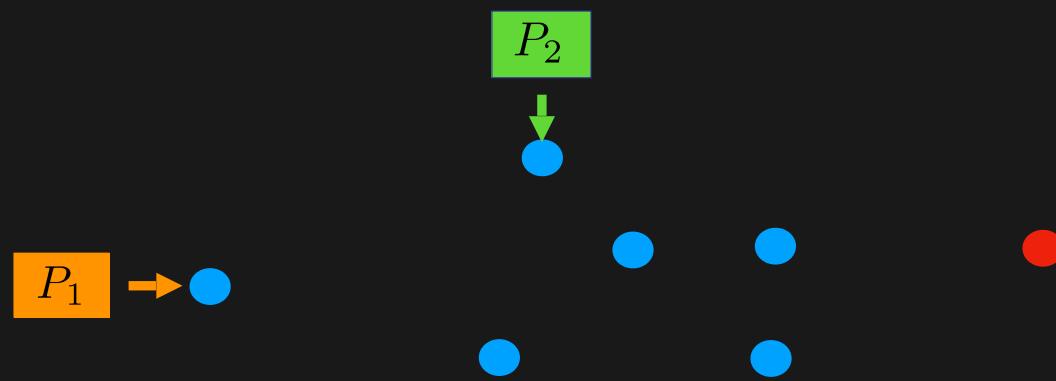
## Network design



## Network design

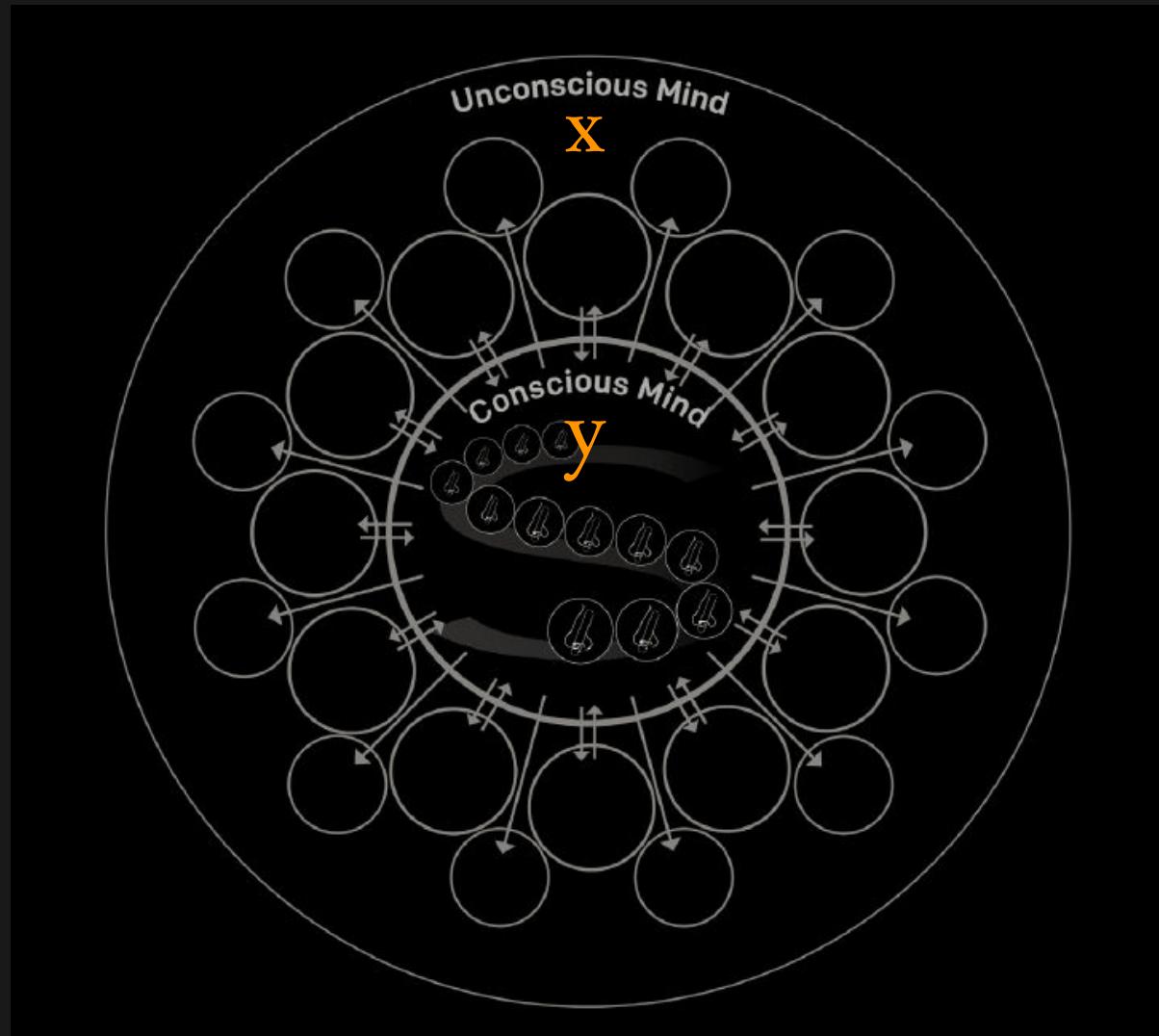


## Network design



Naïve decomposition is not **individuated**.

The conscious **Y** has failed to realise the true natural law of **X**.



# Benders decomposition

Numerische Mathematik 4, 238–252 (1962)

## Partitioning procedures for solving mixed-variables programming problems\*



By

J. F. BENDERS\*\*

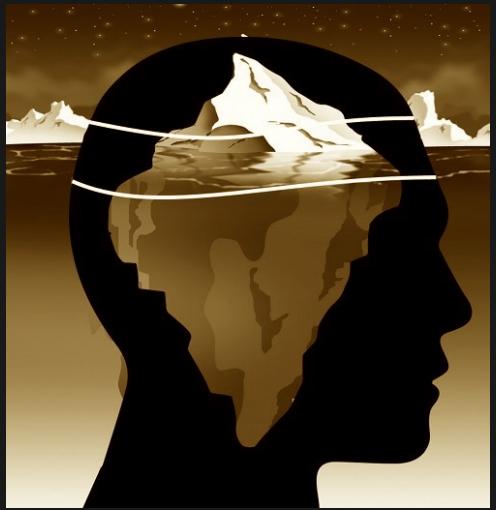
### I. Introduction

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

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

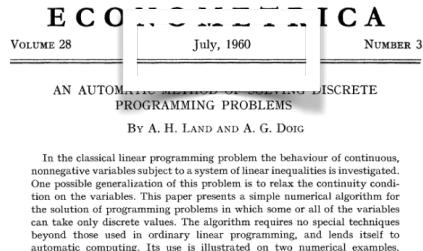
where  $x \in R_p$  (the  $p$ -dimensional Euclidean space),  $y \in R_q$ , and  $S$  is an arbitrary subset of  $R_q$ . Furthermore,  $A$  is an  $(m, p)$  matrix,  $f(y)$  is a scalar function and  $F(y)$  an  $m$ -component vector function both defined on  $S$ , and  $b$  and  $c$  are fixed vectors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



**Branch-and-bound** was computationally much ahead of its time.

Benders even more.



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.

#### I. INTRODUCTION

THERE IS A growing literature [1, 3, 5, 6] about optimization problems which could be formulated as linear programming problems with additional constraints that some or all of the variables may take only integral values. This form of linear programming arises whenever there are indivisibilities. It is not meaningful, for instance, to schedule 3–7/10 flights between two cities, or to undertake only 1/4 of the necessary setting up operation for running a job through a machine shop. Yet it is basic to linear programming that the variables are free to take on any positive value,<sup>1</sup> and this sort of answer is very likely to turn up.

In some cases, notably those which can be expressed as transport problems, the linear programming solution will itself yield discrete values of the variables. In other cases the percentage change in the maximand<sup>2</sup> from common sense rounding of the variables is sufficiently small to be neglected. But there remain many problems where the discrete variable constraints are significant and costly.

Until recently there was no general automatic routine for solving such

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

$$\max \{c^T x + f(y) \mid Ax + F(y) \leq b, x \in R_p, y \in S\}, \quad (1.1)$$

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].



Available online at [www.sciencedirect.com](http://www.sciencedirect.com)



Computers & Operations Research 32 (2005) 1429–1450

---

computers &  
operations  
research

---

[www.elsevier.com/locate/dsw](http://www.elsevier.com/locate/dsw)

## A survey on benders decomposition applied to fixed-charge network design problems

Alysson M. Costa\*

*Canada Research Chair in Distribution Management, HEC Montréal, 3000 Chemin de la Côte-Sainte-Catherine,  
Montréal, Canada H3T 2A7*

---

### Abstract

Network design problems concern the selection of arcs in a graph in order to satisfy, at minimum cost, some flow requirements, usually expressed in the form of origin–destination pair demands. Benders decomposition methods, based on the idea of partition and delayed constraint generation, have been successfully applied to many of these problems. This article presents a review of these applications.

© 2003 Elsevier Ltd. All rights reserved.

---

**Keywords:** Benders decomposition; Network design; Fixed charge

---

## 4. Summary and conclusions

Numerous practical applications can be formulated as network design problems. In these problems, the idea is to obtain a least cost network in order to satisfy some flow constraints, commonly expressed in the form of origin–destination demands. We have presented a review on Benders decomposition methods applied to network design. Formulations for these problems usually contain one set of integer variables associated with the selection of the arcs in the network, and one set of continuous variables associated with commodity flows. This structure offers a natural framework for the decomposition approach which consists of isolating the integer variables in the master problem and the flow variables in the auxiliary subproblem. Moreover, the relative ease of solving the auxiliary subproblem in network design formulations make of Benders decomposition one of the most appropriate approaches. Indeed, in most of the surveyed articles (see Table 1 for a summary), validations tests have indicated that Benders decomposition is an efficient method for solving network design problems, and may outperform traditional techniques such as Branch-and-Bound or Lagrangian relaxation. Efficient solution methodologies have also been obtained by combining Benders decomposition with other techniques, as proposed by Magnanti et al. [24]. A rich variety of these successful hybrid approaches is available in the literature.

In spite of this success, Benders decomposition has been mostly ignored for many years, not only for network design problems but for some of the other applications mentioned in Section 2. We believe that this tendency is slowly changing, given the increasing number of researchers using this technique, as shown in this article.

## 2005 C&OR Survey

In spite of this success, Benders decomposition has been mostly ignored for many years, not only for network design problems but for some of the other applications mentioned in Section 2. We believe that this tendency is slowly changing, given the increasing number of researchers using this technique, as shown in this article.

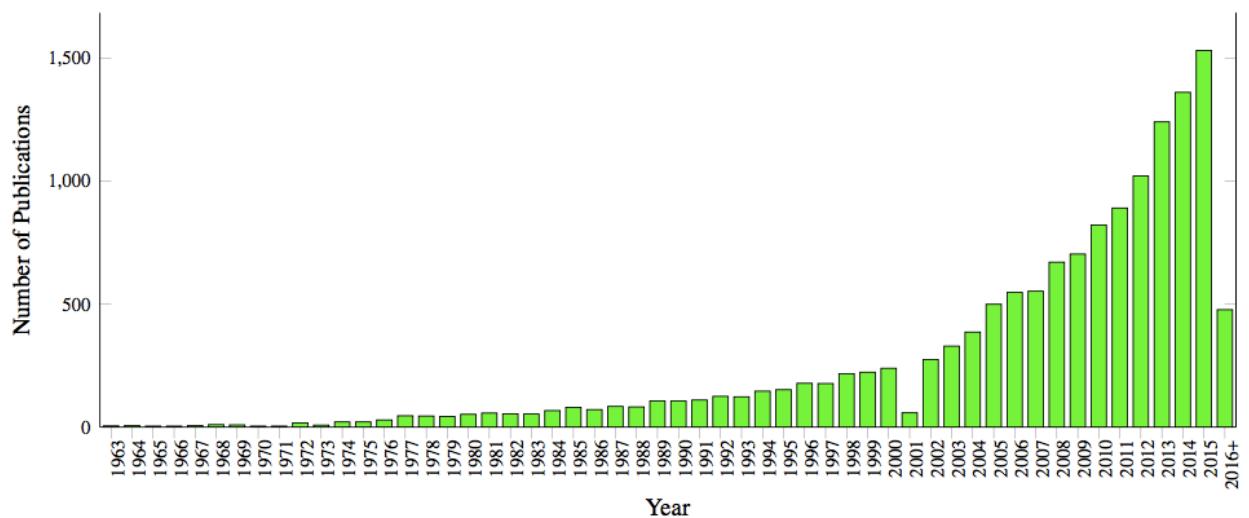
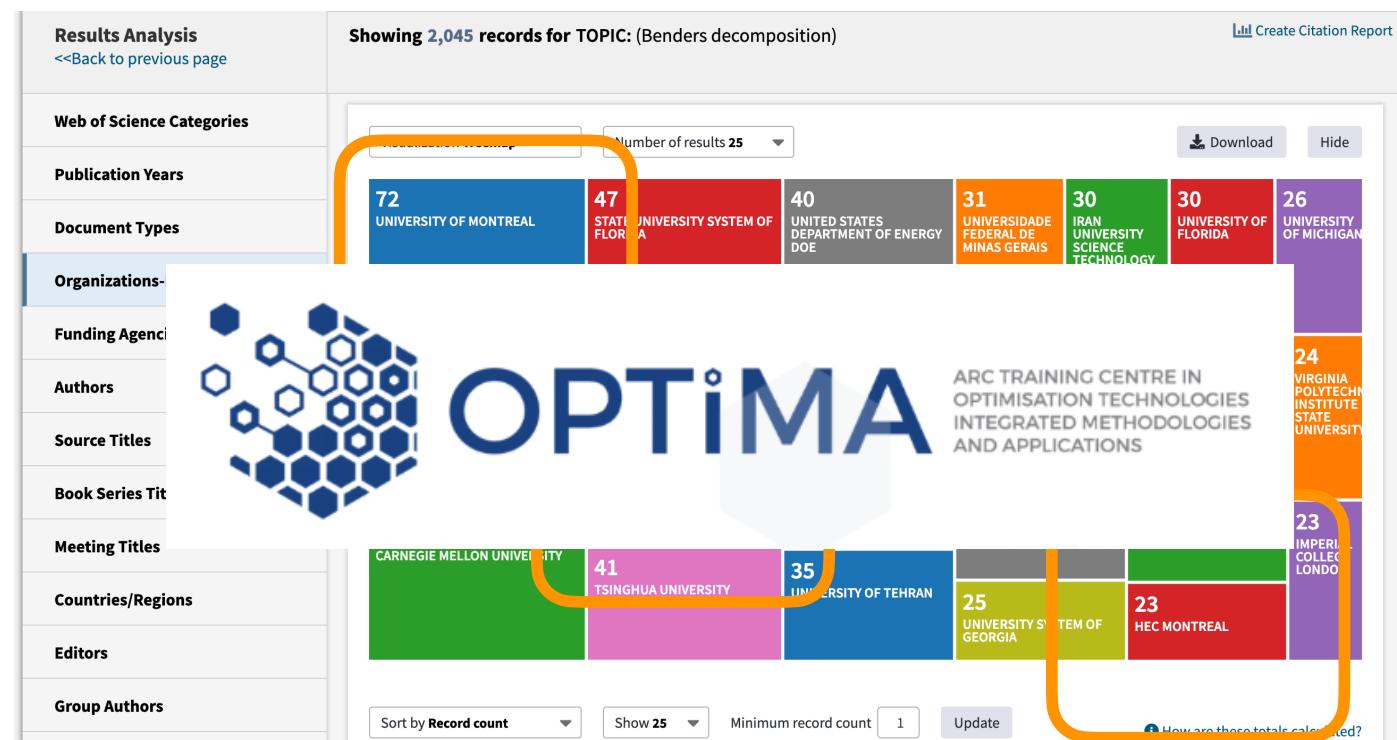


Figure 1: Publication–year distribution of BD research according to <https://scholar.google.ca/>.

# 2005 C&OR Survey

In spite of this success, Benders decomposition has been mostly ignored for many years, not only for network design problems but for some of the other applications mentioned in Section 2. We believe that this tendency is slowly changing, given the increasing number of researchers using this technique, as shown in this article.



## Benders reformulation

## A generic MIP

Minimize  $cx + dy$

subject to  $Ax + By \geq b,$

$Dy \geq e,$

$x \geq 0, \quad y \geq 0 \quad \text{and integer.}$

## Benders reformulation

## A generic MIP

$$\min_{\bar{y} \in Y} \left\{ d^T \bar{y} + \min_{x \geq 0} \{ c^T x : Ax \geq b - B\bar{y} \} \right\},$$

Minimize  $cx + dy$

subject to  $Ax + By \geq b,$

$Dy \geq e,$

$x \geq 0, \quad y \geq 0 \quad \text{and integer.}$

Benders reformulation

The ace in the hole / ‘the jump of the cat’

$$\min_{\bar{y} \in Y} \{ d\bar{y} + \min_{x \geq 0} \{ cx : Ax \geq b - B\bar{y} \} \},$$



## Benders reformulation

$$\min_{\bar{y} \in Y} \{ d^T \bar{y} + \min_{x \geq 0} \{ c^T x : Ax \geq b - B\bar{y} \} \},$$

$$\max_{u \geq 0} \{ u(b - B\bar{y}) : uA \leq c \}.$$

But how ?

**Theorem 1** (Fundamental Theorem of Duality)

With regard to the primal and dual linear programming problems, exactly one of the following statements is true.

1. Both possess optimal solutions  $x^*$  and  $w^*$  with  $c x^* = w^* b$ .
2. One problem has unbounded objective value, in which case the other problem must be infeasible.
3. Both problems are infeasible.

From this theorem we see that duality is not completely symmetric. The best we can say is that (here optimal means finite optimal, and unbounded means having an unbounded optimal objective):

P	OPTIMAL	$\Leftrightarrow$	D	OPTIMAL
P	UNBOUNDED	$\Rightarrow$	D	INFEASIBLE
D	UNBOUNDED	$\Rightarrow$	P	INFEASIBLE
P	INFEASIBLE	$\Rightarrow$	D	UNBOUNDED OR INFEASIBLE
D	INFEASIBLE	$\Rightarrow$	P	UNBOUNDED OR INFEASIBLE



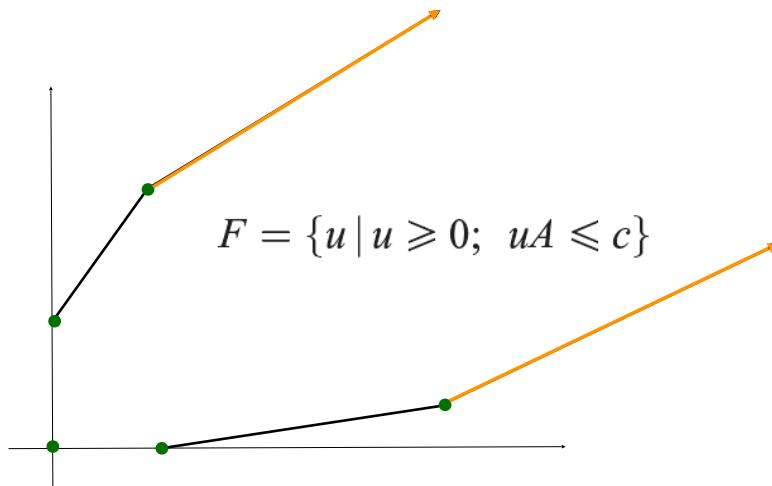
From Bazaraa - Linear Programming and Network flows

## Benders reformulation



$$\min_{\bar{y} \in Y} \{ d^T \bar{y} + \min_{x \geq 0} \{ c^T x : Ax \geq b - B\bar{y} \} \},$$

$$\max_{u \geq 0} \{ u(b - B\bar{y}) : uA \leq c \}.$$

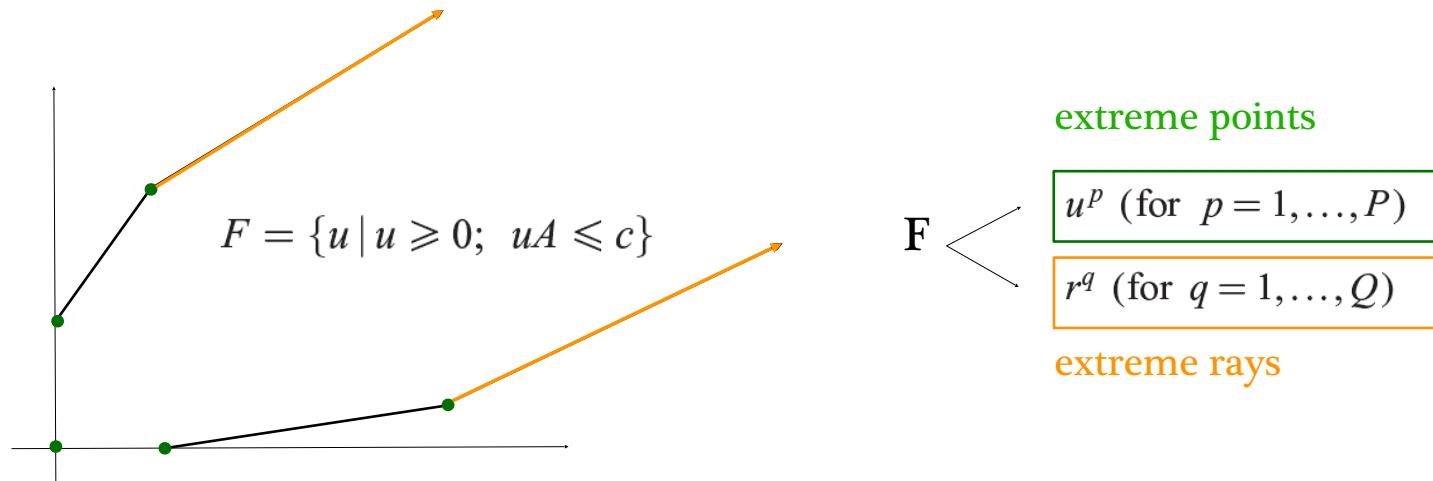


## Benders reformulation



$$\min_{\bar{y} \in Y} \{ d^T \bar{y} + \min_{x \geq 0} \{ c^T x : Ax \geq b - B\bar{y} \} \},$$

$$\max_{u \geq 0} \{ u(b - B\bar{y}) : uA \leq c \}.$$



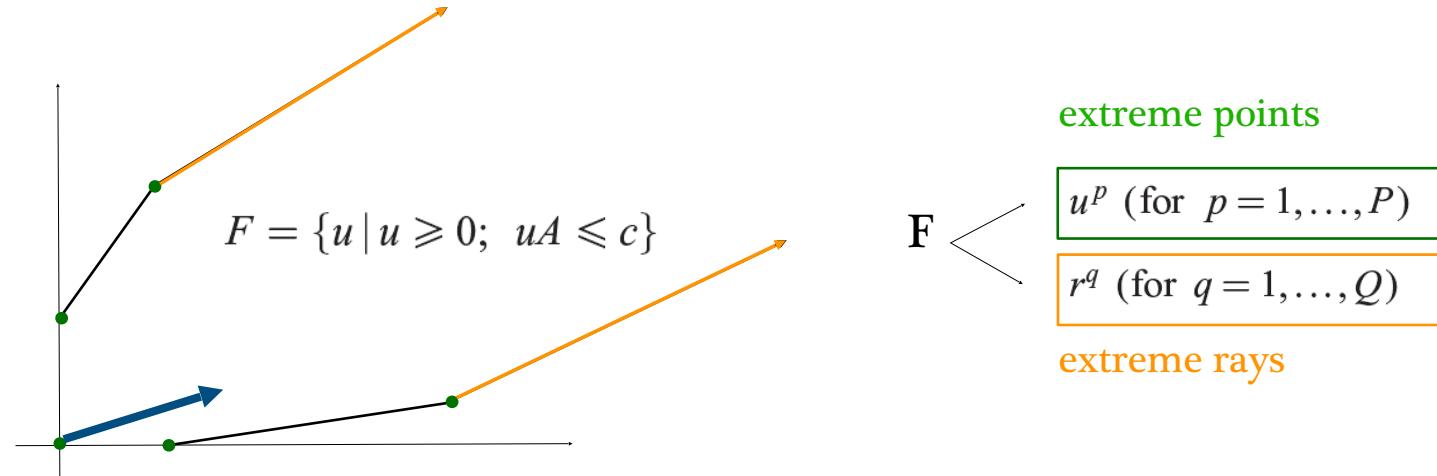
## Benders reformulation



## A generic MIP

$$\min_{\bar{y} \in Y} \{ d^T \bar{y} + \min_{x \geq 0} \{ c^T x : Ax \geq b - B\bar{y} \} \},$$

$$\max_{u \geq 0} \{ u(b - B\bar{y}) : uA \leq c \}.$$



## Benders reformulation



## A generic MIP

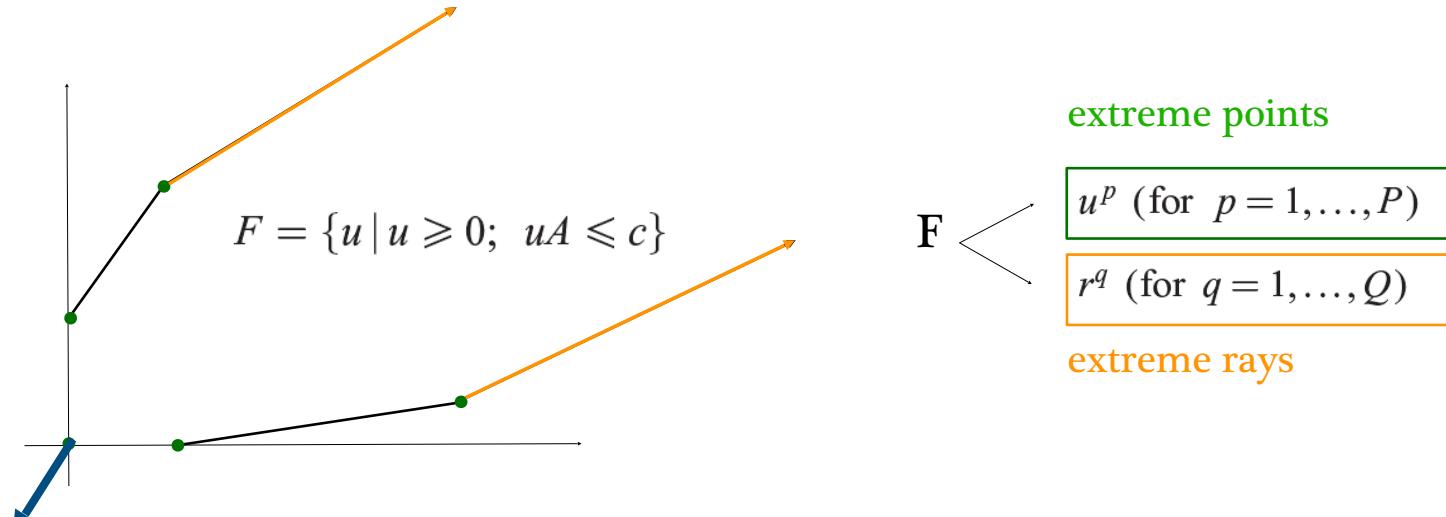
$$\max_{u \geq 0} \{u(b - B\bar{y}) : uA \leq c\}.$$

Z

$$\min_{\bar{y} \in Y} \{d\bar{y} + \min_{x \geq 0} \{cx : Ax \geq b - B\bar{y}\}\},$$

$$\text{s.t } r^q(b - B\bar{y}) \leq 0, \quad q = 1, \dots, Q,$$

$$z \geq u^p(b - B\bar{y}), \quad p = 1, \dots, P,$$



## Benders reformulation

Minimize  $dy + z$

subject to  $z \geq u^p(b - B\bar{y}), \quad p = 1, \dots, P,$

$r^q(b - B\bar{y}) \leq 0, \quad q = 1, \dots, Q,$

$y \in Y, \quad z \geq 0.$

The number of constraints **stresses** me.

## Benders algorithm

Minimize  $dy + z$

subject to  $\underline{z} \geq u^p(b - B\bar{y}), \quad p = 1, \dots, P,$

$\underline{r}^q(b - B\bar{y}) \leq 0, \quad q = 1, \dots, Q,$

$y \in Y, \quad z \geq 0.$

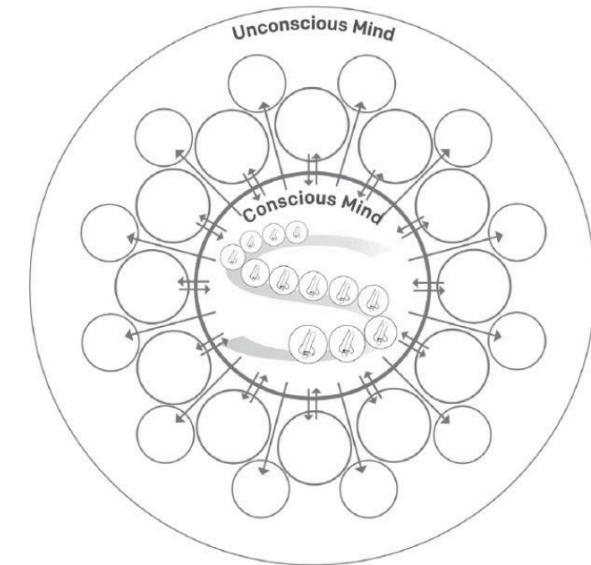
Relax



$u^p$  (for  $p = 1, \dots, P$ )

$r^q$  (for  $q = 1, \dots, Q$ )

feasibility/optimality  
**cut**



# Benders algorithm

Convergence

Benders

Master: **LB**

Feasible Subproblem: **UB**

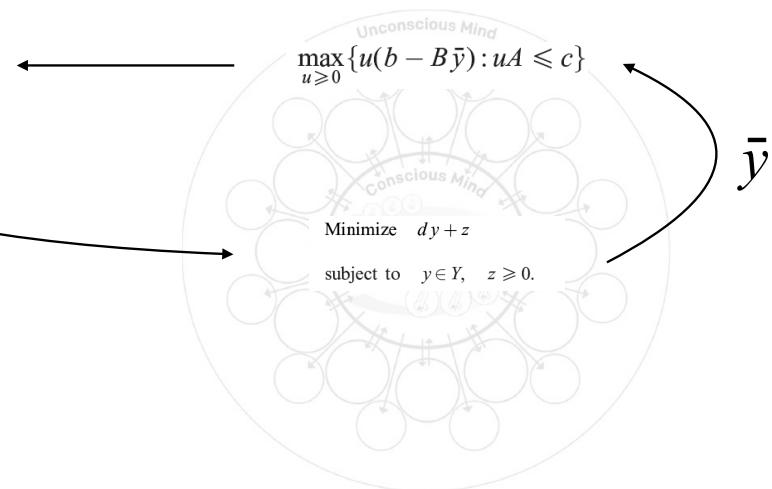
Jung

Individuation

$u^p$  (for  $p = 1, \dots, P$ )

$r^q$  (for  $q = 1, \dots, Q$ )

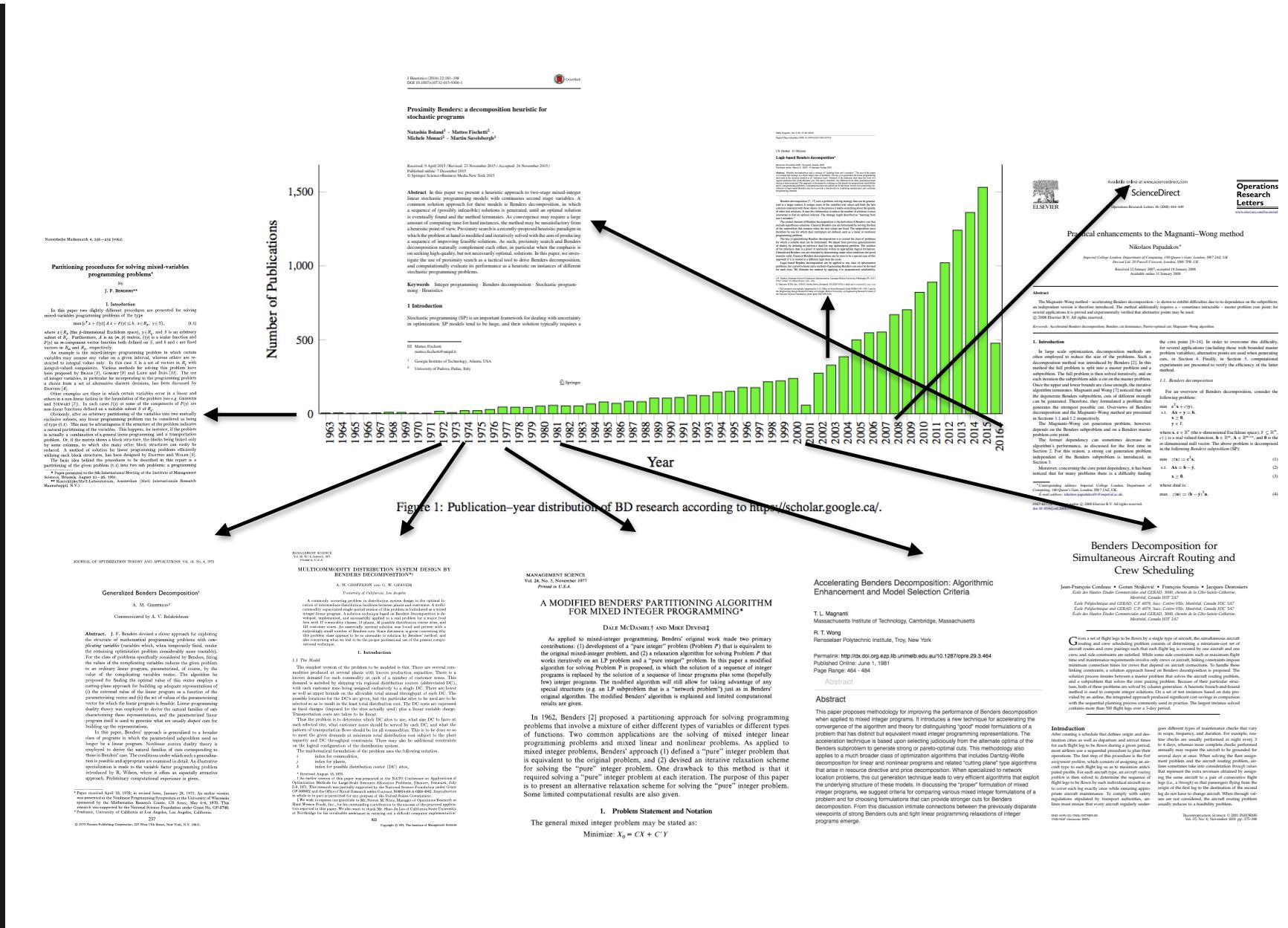
feasibility/optimality  
**cut**



# Example

# What else you should know?

## (Part 2)



# Clear applications

What  
else  
you  
should  
know.

*Naukische Mathematik* 4, 261–272 (1961)

Partitioning procedures for solving mixed-variables programming problems\*

J. T. Benders\*\*

### 1. Introduction

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

$$\min_{x \in \mathbb{R}^n} f(x) + g(x) \text{ s.t. } Ax \leq b, x \in S, x \geq 0, \quad (1)$$

where  $x \in \mathbb{R}^n$  (the admissible Euclidean space),  $A \in \mathbb{R}^{m \times n}$ , and  $S$  is an arbitrary subset of  $\mathbb{R}^n$ . Furthermore,  $f$  is an  $(n, m)$  matrix,  $g(x)$  is a nonlinear function and  $A$  is a  $m \times n$  matrix. The vectors  $b \in \mathbb{R}^m$  and  $x$  are the right-hand side and the decision variables respectively.

An example of a linear programming problem in which certain variables may assume values in a given interval, whereas others are restricted to integer values, is the knapsack problem [1]. It consists of  $n$  integer-valued components. Various methods for solving this problem have been proposed [2–4]. In this paper we shall consider the case where some of integer variables, in particular those incorporated in the programming problem as shown in (1), are subject to restrictions of the type  $x_i \in \mathbb{Z}, i = 1, \dots, r$ .

One examples are those in which certain variables occur in a linear and often in a non-linear fashion in the formulation of the problem (see e.g. Gassmann [5] and Dantzig [6]). Another example is the knapsack problem [1] to non-linear functions defined on a suitable subset  $S$  of  $\mathbb{R}^n$ .

As far as the author is concerned, the linear programming problem has naturally exclusive subsets, any linear programming problem can be considered as being of type (1). This implies, for instance, that the problem is not necessarily a natural partitioning of the variables. This happens, for instance, if the problem is such that the integer variables are not necessarily included in the same partition. On the other hand, there is a block structure, the blocks being linked only by some linear constraints, in which the integer variables are included in the relevant block. A method of solving linear programming problems based on this idea was proposed by M. W. Wong [7] and by J. T. Benders [8].

The basic idea behind the procedure to be described in this report is a partitioning of the variables into integer and continuous parts, followed by the solution of the integer part by the Benders' decomposition method [9].

\* Paper presented to the 5th International Meeting of the Institute of Management Sciences, Brussels, Belgium, 1960.

\*\* Head of the Laboratory, Associate (then) International Research Institute, Paris, France.

Journal of Optimization Theory and Applications 56, No. 4, 493–507 (1988)

Generalized Benders Decomposition<sup>\*</sup>

A. M. Gherardi<sup>†</sup>

Commissariato Nazionale per il Volo, Roma, Italy

**Abstract.** J. F. Benders devised a clear approach for explaining the structure of mathematical programming problems with constraints involving variables that are discrete and continuous. This approach is particularly useful for solving mixed-integer programming problems, which are considerably more tractable than the class of problems specifically treated by Benders, fixing the integer variables at their lower bounds. In this paper, the Benders' approach is extended to an integer linear programming problem, of course, by the introduction of a new constraint. This constraint is used to propose a method for finding the optimal value of the fixed variables. This problem is solved by a modified Benders' decomposition method, which produces short programs to be available to the Benders' method, and which is able to handle integer variables in a very simple way.

**1. Introduction**

A commonly occurring problem in discrete optimization is the optimal integer programming problem, which involves both integer and continuous variables. This class of problems is particularly well suited to the Benders' decomposition method [1], which has been developed, implemented, and successfully applied to a wide range of integer programming problems [2–10].

In this paper, the Benders' decomposition method is generalized to the integer linear programming problem. The main idea is to propose a method for finding the optimal value of the integer variables. This problem is solved by a modified Benders' decomposition method, which produces short programs to be available to the Benders' method, and which is able to handle integer variables in a very simple way.

In this paper, Benders' approach is generalized to a broader class of problems, namely, the partitioning of variables into integer and continuous parts. The basic idea behind the Benders' approach is to decompose the pattern of transportation flows which should be for all commodities. This is to be done so as to obtain a set of integer variables and a set of continuous variables. The integer and DC constraint statements. These may also be additional constraints in this class of problems.

The mathematical formulation of the problem uses the following notations.

**1.1. The Model**

The model of the problem to be modeled is the following:

$$\min_{x \in \mathbb{R}^n} f(x) + g(x) \text{ s.t. } Ax \leq b, x \in S, x \geq 0, \quad (1)$$

where  $x \in \mathbb{R}^n$  (the admissible Euclidean space),  $A \in \mathbb{R}^{m \times n}$ , and  $S$  is an arbitrary subset of  $\mathbb{R}^n$ .

As applied to mixed-integer programming, Benders' original work made two primary contributions: (i) the definition of a "pure" integer problem ( $P$ ) that is equivalent to the original mixed-integer problem; and (ii) a decomposition algorithm for solving Problem  $P$  that works iteratively on an LP problem and a "pure" integer problem. In this paper a modified algorithm for solving Problem  $P$  is proposed, in which the solution of a sequence of integer programs is used to obtain the solution of the original mixed-integer problem. The modified algorithm will still allow for taking advantage of any special structures (e.g. LP subproblems that is a "network problem") just as in Benders' original work. The modified Benders' algorithm is explained and limited computational results are given.

**1.2. Problem Statement and Notation**

The general mixed integer problem may be stated as:

$$\text{Minimize: } X_0 = CX + C'Y$$



Figure 1: Publication-year distribution of BD research according to <https://scholar.google.ca>.

*Industrial Mathematics* (2016) 22:181–198  
DOI 10.1007/s10288-015-0496-4  
© Springer Science+Business Media New York 2015

### Proximity Benders: a decomposition heuristic for stochastic programs

Natalia Boland<sup>1</sup> · Matteo Fischetti<sup>2</sup> ·  
Michèle Monaci<sup>2</sup> · Martin Savelsbergh<sup>3</sup>

Received: 9 April 2015 / Revised: 21 November 2015 / Accepted: 24 November 2015  
Published online: 7 December 2015  
© Springer Science+Business Media New York 2015

**Abstract** In this paper we present a heuristic approach to two-stage mixed-integer linear and stochastic programming models with continuous second stage variables. A common solution approach for these models is Benders decomposition, in which a sequence of master problems are solved until a feasible solution is found and the method terminates. As convergence may require a large amount of computing time for hard instances, the method may be unsatisfactory from a computational perspective. We propose a proximity-based decomposition approach, in which the problem at hand is modified and iteratively solved with the aim of producing a sequence of feasible solutions that are close to each other. The proximity-based decomposition naturally complements each other, in particular when the emphasis is on seeking high-quality, but not necessarily optimal, solutions. In this paper, we investigate the performance of the proximity-based decomposition and compare it with Benders decomposition and computationally evaluate its performance as a heuristic on instances of difficult stochastic programming problems.

**Keywords** Integer programming · Benders decomposition · Stochastic programming · Heuristics

Mark P. Magnanti · Ming Tang ·  
David P. Trichart · Michael R. Trick

*Operations Research Letters* 36 (2008) 644–649

Received 21 January 2007; accepted 31 January 2008

Available online 31 January 2008

### Abstract

In this paper we introduce a proximity-based decomposition heuristic for two-stage mixed-integer linear and stochastic programming models with continuous second stage variables. The proximity-based decomposition approach is a modification of the standard Benders decomposition, in which the problem at hand is modified and iteratively solved with the aim of producing a sequence of feasible solutions that are close to each other. The proximity-based decomposition naturally complements each other, in particular when the emphasis is on seeking high-quality, but not necessarily optimal, solutions. In this paper, we investigate the performance of the proximity-based decomposition and compare it with Benders decomposition and computationally evaluate its performance as a heuristic on instances of difficult stochastic programming problems.

**Keywords** Integer programming · Benders decomposition · Stochastic programming · Heuristics

*Operations Research Letters* 36 (2008) 644–649  
Received 21 January 2007; accepted 31 January 2008  
Available online 31 January 2008

### Practical enhancements to the Magnanti–Wong method

Nikolaos Papalambros\*

Imperial College London, Department of Computing, 100 Queen's Gate, London, SW7 2AZ, UK

Received 21 January 2007; accepted 31 January 2008

Available online 31 January 2008

### Abstract

The Magnanti–Wong method—accelerating Benders decomposition—is known to exhibit difficulties due to dependence on the subproblem cost point [9–14].

In several applications (including those with bounded master problem costs) the subproblem cost point is constant or changes little over time.

We propose practical enhancements to the Magnanti–Wong method to reduce the dependence on the subproblem cost point.

These enhancements are tested on a set of test problems.

© 2008 Elsevier B.V. All rights reserved.

**Keywords** Accelerated Benders decomposition · Benders cut dominance · Proximity-based decomposition · Magnanti–Wong algorithm

For large scale optimization, decomposition methods are often employed to reduce the size of the problem. Such a decomposition method is the Benders decomposition, in which the total problem is split into a master problem and a subproblem. The subproblem is solved at each iteration and the solution is used to update the master problem. A well-known difficulty with the Benders decomposition is that the subproblem cost point depends on the subproblem cost point of the previous iteration. Therefore, the Magnanti–Wong [1] method is proposed that generates a sequence of subproblems with a constant subproblem cost point. The Magnanti–Wong method is presented in Sections 1 and 2 respectively.

The basic idea behind the decomposition problem, however, depends on the Benders subproblem and on a Benders master problem.

The former dependency can sometimes be eliminated by using a decomposition method that does not depend on the subproblem cost point [2].

The latter dependency can sometimes be removed by using a decomposition method that does not depend on the subproblem cost point [3].

For the decomposition problem, however, it has been noticed that for many problems there is a difficulty related to the subproblem cost point [4–6].

The following decomposition subproblem (SP):

$$\min_{x \in \mathbb{R}^n} c^T x + d^T z \quad (1)$$

where  $c \in \mathbb{R}^n$ ,  $d \in \mathbb{R}^m$ ,  $z \in \mathbb{R}^m$ ,  $A \in \mathbb{R}^{m \times n}$ ,  $B \in \mathbb{R}^{m \times m}$  is to be solved at each iteration. The subproblem cost point is determined by the following formula:

$$\text{max}_{z \in \mathbb{R}^m} (c^T x + d^T z) \quad (2)$$

where  $x$  is the current solution of the master problem.

When  $c^T x + d^T z$  is constant, the subproblem cost point is constant.

Otherwise, the subproblem cost point is determined by the following formula:

$$\text{max}_{z \in \mathbb{R}^m} (c^T x + d^T z - \epsilon) \quad (3)$$

where  $\epsilon$  is a small positive number.

© 2008 Elsevier B.V. All rights reserved.

Journal of Optimization Theory and Applications 139, No. 2, 647–657 (2008)  
DOI 10.1007/s10957-007-9377-1

Published online: 04 October 2007

© 2008 Springer Science+Business Media, LLC

Journal of Optimization Theory and Applications is covered by Science Citation Index, Current Contents/Physics, Chemistry, Mathematics, Mathematical Reviews, Zentralblatt für Mathematik/Mathematics Abstracts, Mathematical Reviews/MathSciNet, Mathematical Citation Quotient, Science Citation Index, Science Citation Index-Expanded, SCOPUS, and SCIE.

**Keywords** Accelerated Benders decomposition · Benders cut dominance · Proximity-based decomposition · Magnanti–Wong algorithm

For an overview of Benders decomposition, consider the following problem:

$$\min_{x \in \mathbb{R}^n} f(x) \quad (1)$$

where  $f$  is convex. Therefore, the Benders decomposition is a decomposition method that splits the problem into a master problem and a subproblem. At each iteration, the subproblem adds cuts on the master problem. Once a solution is found, the subproblem is discarded. The subproblem cost point depends on the master problem cost point. The subproblem cost point is determined by the following formula:

$$\text{max}_{x \in \mathbb{R}^n} f(x) \quad (2)$$

where  $x$  is the current solution of the master problem.

When  $f(x)$  is constant, the subproblem cost point is constant.

Otherwise, the subproblem cost point is determined by the following formula:

$$\text{max}_{x \in \mathbb{R}^n} f(x) - \epsilon \quad (3)$$

where  $\epsilon$  is a small positive number.

© 2008 Springer Science+Business Media, LLC

Journal of Optimization Theory and Applications is covered by Science Citation Index, Current Contents/Physics, Chemistry, Mathematics, Mathematical Reviews/MathSciNet, Mathematical Citation Quotient, Science Citation Index, Science Citation Index-Expanded, SCOPUS, and SCIE.

**Keywords** Accelerated Benders decomposition · Benders cut dominance · Proximity-based decomposition · Magnanti–Wong algorithm

After creating a schedule that defines origin and destination for each flight, we usually plan for eight every 2 days, when the computation time is

not too long. However, when the computation time is

longer, we usually plan for ten flights every 2 days.

When the computation time is longer than 10 days, we usually plan for 12 flights every 2 days.

When the computation time is longer than 15 days, we usually plan for 14 flights every 2 days.

When the computation time is longer than 20 days, we usually plan for 16 flights every 2 days.

When the computation time is longer than 25 days, we usually plan for 18 flights every 2 days.

When the computation time is longer than 30 days, we usually plan for 20 flights every 2 days.

When the computation time is longer than 35 days, we usually plan for 22 flights every 2 days.

When the computation time is longer than 40 days, we usually plan for 24 flights every 2 days.

When the computation time is longer than 45 days, we usually plan for 26 flights every 2 days.

When the computation time is longer than 50 days, we usually plan for 28 flights every 2 days.

When the computation time is longer than 55 days, we usually plan for 30 flights every 2 days.

When the computation time is longer than 60 days, we usually plan for 32 flights every 2 days.

When the computation time is longer than 65 days, we usually plan for 34 flights every 2 days.

When the computation time is longer than 70 days, we usually plan for 36 flights every 2 days.

When the computation time is longer than 75 days, we usually plan for 38 flights every 2 days.

When the computation time is longer than 80 days, we usually plan for 40 flights every 2 days.

When the computation time is longer than 85 days, we usually plan for 42 flights every 2 days.

When the computation time is longer than 90 days, we usually plan for 44 flights every 2 days.

When the computation time is longer than 95 days, we usually plan for 46 flights every 2 days.

When the computation time is longer than 100 days, we usually plan for 48 flights every 2 days.

When the computation time is longer than 105 days, we usually plan for 50 flights every 2 days.

When the computation time is longer than 110 days, we usually plan for 52 flights every 2 days.

When the computation time is longer than 115 days, we usually plan for 54 flights every 2 days.

When the computation time is longer than 120 days, we usually plan for 56 flights every 2 days.

When the computation time is longer than 125 days, we usually plan for 58 flights every 2 days.

When the computation time is longer than 130 days, we usually plan for 60 flights every 2 days.

When the computation time is longer than 135 days, we usually plan for 62 flights every 2 days.

When the computation time is longer than 140 days, we usually plan for 64 flights every 2 days.

When the computation time is longer than 145 days, we usually plan for 66 flights every 2 days.

When the computation time is longer than 150 days, we usually plan for 68 flights every 2 days.

When the computation time is longer than 155 days, we usually plan for 70 flights every 2 days.

When the computation time is longer than 160 days, we usually plan for 72 flights every 2 days.

When the computation time is longer than 165 days, we usually plan for 74 flights every 2 days.

When the computation time is longer than 170 days, we usually plan for 76 flights every 2 days.

When the computation time is longer than 175 days, we usually plan for 78 flights every 2 days.

When the computation time is longer than 180 days, we usually plan for 80 flights every 2 days.

When the computation time is longer than 185 days, we usually plan for 82 flights every 2 days.

When the computation time is longer than 190 days, we usually plan for 84 flights every 2 days.

When the computation time is longer than 195 days, we usually plan for 86 flights every 2 days.

When the computation time is longer than 200 days, we usually plan for 88 flights every 2 days.

When the computation time is longer than 205 days, we usually plan for 90 flights every 2 days.

When the computation time is longer than 210 days, we usually plan for 92 flights every 2 days.

When the computation time is longer than 215 days, we usually plan for 94 flights every 2 days.

When the computation time is longer than 220 days, we usually plan for 96 flights every 2 days.

When the computation time is longer than 225 days, we usually plan for 98 flights every 2 days.

When the computation time is longer than 230 days, we usually plan for 100 flights every 2 days.

When the computation time is longer than 235 days, we usually plan for 102 flights every 2 days.

When the computation time is longer than 240 days, we usually plan for 104 flights every 2 days.

When the computation time is longer than 245 days, we usually plan for 106 flights every 2 days.

When the computation time is longer than 250 days, we usually plan for 108 flights every 2 days.

When the computation time is longer than 255 days, we usually plan for 110 flights every 2 days.

When the computation time is longer than 260 days, we usually plan for 112 flights every 2 days.

When the computation time is longer than 265 days, we usually plan for 114 flights every 2 days.

When the computation time is longer than 270 days, we usually plan for 116 flights every 2 days.

When the computation time is longer than 275 days, we usually plan for 118 flights every 2 days.

When the computation time is longer than 280 days, we usually plan for 120 flights every 2 days.

When the computation time is longer than 285 days, we usually plan for 122 flights every 2 days.

When the computation time is longer than 290 days, we usually plan for 124 flights every 2 days.

When the computation time is longer than 295 days, we usually plan for 126 flights every 2 days.

When the computation time is longer than 300 days, we usually plan for 128 flights every 2 days.

When the computation time is longer than 305 days, we usually plan for 130 flights every 2 days.

When the computation time is longer than 310 days, we usually plan for 132 flights every 2 days.

When the computation time is longer than 315 days, we usually plan for 134 flights every 2 days.

When the computation time is longer than 320 days, we usually plan for 136 flights every 2 days.

When the computation time is longer than 325 days, we usually plan for 138 flights every 2 days.

When the computation time is longer than 330 days, we usually plan for 140 flights every 2 days.

When the computation time is longer than 335 days, we usually plan for 142 flights every 2 days.

When the computation time is longer than 340 days, we usually plan for 144 flights every 2 days.

When the computation time is longer than 345 days, we usually plan for 146 flights every 2 days.

When the computation time is longer than 350 days, we usually plan for 148 flights every 2 days.

When the computation time is longer than 355 days, we usually plan for 150 flights every 2 days.

When the computation time is longer than

What  
else  
you  
should  
know.

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

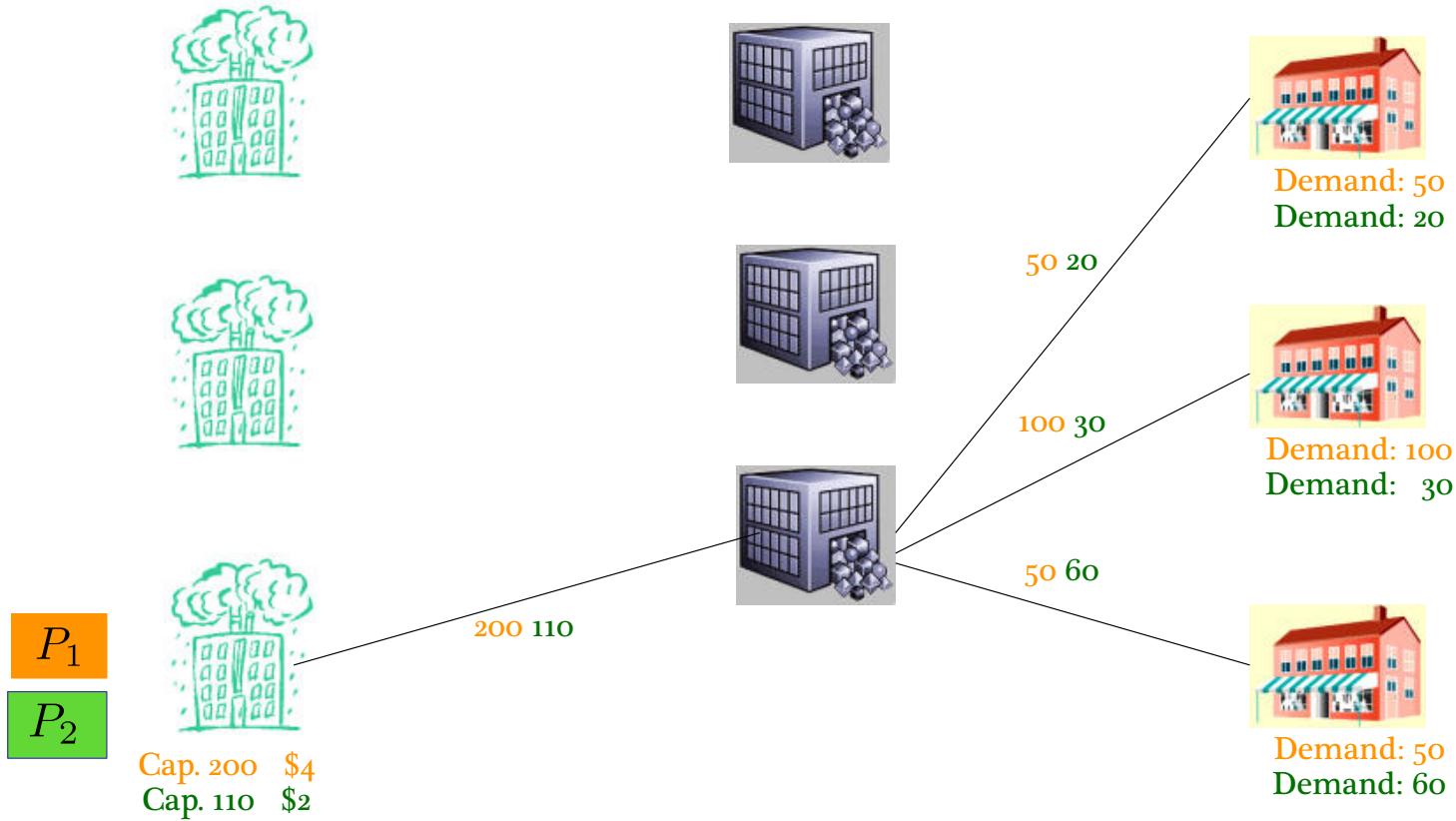
## MULTICOMMODITY DISTRIBUTION SYSTEM DESIGN BY BENDERS DECOMPOSITION\*†

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 multi-commodity 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.

What  
else  
you  
should  
know.



# What else you should know.

$x_{ijkl}$	a variable denoting the amount of commodity $i$ shipped from plant $j$ through DC $k$ to customer zone $l$ ,
$y_{kl}$	a 0–1 variable that will be 1 if DC $k$ serves customer zone $l$ , and 0 otherwise
$z_k$	a 0–1 variable that will be 1 if a DC is acquired at site $k$ , and 0 otherwise.

$$(1) \quad \text{Minimize}_{x \geq 0; y, z=0,1} \sum_{ijkl} c_{ijkl} x_{ijkl} + \sum_k [f_k z_k + v_k \sum_{il} D_{il} y_{kl}]$$

subject to

$$(2) \quad \sum_{kl} x_{ijkl} \leq S_{ij}, \quad \text{all } ij$$

$$(3) \quad \sum_j x_{ijkl} = D_{il} y_{kl}, \quad \text{all } ikl$$

$$(4) \quad \sum_k y_{kl} = 1, \quad \text{all } l$$

$$(5) \quad \underline{V}_k z_k \leq \sum_{il} D_{il} y_{kl} \leq \bar{V}_k z_k, \quad \text{all } k$$

(6) Linear configuration constraints on  $y$  and/or  $z$ .

# What else you should know.

Third, as indicated previously, the LP subproblem (10) is most easily solved by solving an equivalent collection of independent classical transportation problems—one for each commodity. This can be demonstrated by observing that since  $y^{H+1}$  satisfies (4), (3) implies

$$x_{ijkl}^{H+1} = 0 \text{ for all } i j k l \text{ with } k \neq \bar{k}(l)$$

where  $\bar{k}(l)$  is the  $k$ -index for which  $y_{kl}^{H+1} = 1$ . Thus (10) simplifies to

$$\text{Minimize } \sum_i (\sum_{jl} c_{ij\bar{k}(l)l} x_{ij\bar{k}(l)l})$$

subject to

$$\sum_l x_{ij\bar{k}(l)l} \leq S_{ij}, \quad \text{all } ij$$

$$\sum_j x_{ij\bar{k}(l)l} = D_{il}, \quad \text{all } il$$

$$x_{ij\bar{k}(l)l} \geq 0, \quad \text{all } ij l.$$

This problem obviously separates on  $i$  into independent transportation problems of the form (7i). If the optimal value of (7i) is denoted by  $T_i(y^{H+1})$ , then  $T(y^{H+1}) = \sum_i T_i(y^{H+1})$ .

The reduction of (10) to independent problems of the form (7i) greatly simplifies Step 2a, but Step 2b then becomes less straightforward. The required optimal dual solution for (10) must be synthesized from the optimal dual solutions of (7i). The relationship between the optimal primal solutions of (10) and (7i) is obvious, but the relationship between the optimal dual solutions requires some analysis. This analysis is as follows.

# Convergence

In Benders

Master: **LB**

Feasible Subproblem: **UB**

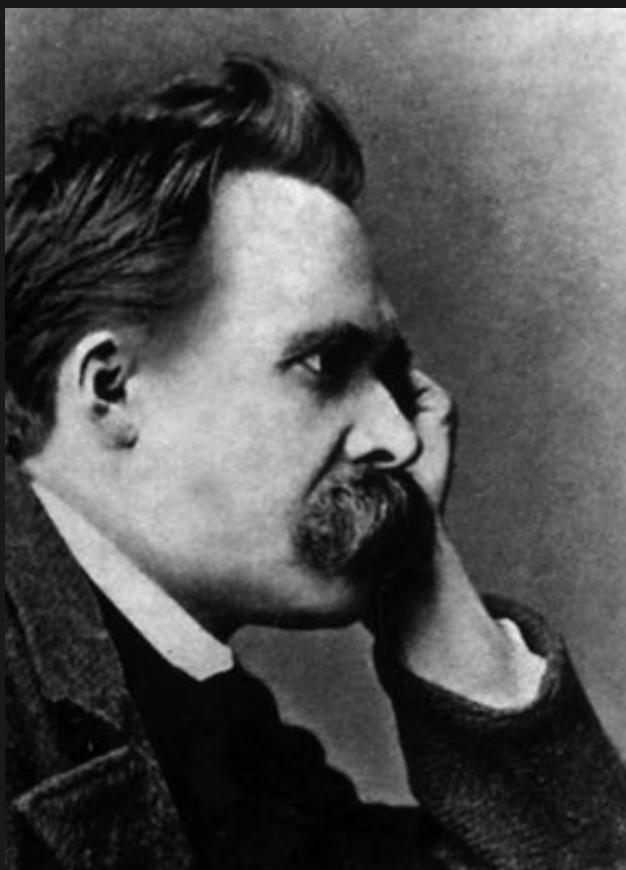
In the mind

**Individuation**

Solving MIPs e LPs



## Beyond good and evil



“No price is too high to pay for the privilege of owning yourself”.

Thank you

