Synthesis Lectures on
Operations Research and Applications

SYNTHESIS
COLLECTION OF TECHNOLOGY

John Hooker

Logic-Based Benders Decomposition

Theory and Applications

Springer

Synthesis Lectures on Operations Research and Applications

John Hooker

Logic-Based Benders Decomposition

Theory and Applications

Springer

John Hooker
Carnegie Mellon University
Pittsburgh, PA, USA

ISSN 2770-6303 ISSN 2770-6311 (electronic)
Synthesis Lectures on Operations Research and Applications
ISBN 978-3-031-45038-9 ISBN 978-3-031-45039-6 (eBook)
https://doi.org/10.1007/978-3-031-45039-6

Preface

Benders decomposition is a well-known and often very effective tool for solving hard optimization problems. It is especially advantageous when a problem reduces to a much simpler subproblem after certain variables are fixed. It benefits from an ingenious learning mechanism based on Benders cuts.

Despite its success, the classical Benders method has limited applicability, because the subproblem must be a linear programming problem. Fortunately, its underlying problem-solving strategy is much more general than may be evident at first. Benders cuts can be viewed as arising from logical inference rather than the specific properties of linear programming. This enables a substantial generalization to *logic-based Benders decomposition* (LBBD), in which the subproblem can, in principle, be any optimization problem. This, in turn, opens the door to a much broader range of applications.

This book is intended as a comprehensive guide to the LBBD user. It presents a unified account of LBBD theory as it has developed over the last two decades. It provides an in-depth tutorial on how to develop effective logic-based cuts for a given problem. It explains such related ideas as branch-and-check methods and combinatorial Benders cuts, the latter being a special case of logic-based cuts. It offers practical suggestions for crafting a successful LBBD implementation for a given application.

LBBD must, in fact, frequently be tailored to the specific application to harness its full potential. With this in mind, nearly half the book is devoted to a compendium of 147 different LBBD applications, ranging from transportation and supply chain management to hospital scheduling and disaster preparation. It describes how 226 published articles adapt the LBBD framework to these problems. This repository of domain-specific solutions not only demonstrates LBBD's wide applicability, but it can serve as a source of ideas for addressing the problem at hand.

Pittsburgh, USA
June 2023

John Hooker

Contents

Introduction 1

1.1 Generalizing Benders Decomposition

Benders decomposition, introduced in 1962 by Jacques Benders [20], is one of the best known and most successful strategies for solving hard optimization problems. It is designed for problems that become easier when certain variables are assigned fixed values, thus creating a more tractable *subproblem*. It solves a problem by enumerating various assignments to the fixed variables and solving the subproblems that result, with the aim of identifying an optimal solution.

While Benders decomposition has many successful applications, it is restricted by the fact that the subproblem must be a linear programming problem—or a convex nonlinear programming problem in Geoffrion's 1972 extension of the method [103]. There are a wide range of potential applications in which the subproblem simplifies without yielding a linear or nonlinear programming problem, often by decoupling into smaller problems. Benders decomposition in its classical form is not suitable for applications of this sort. Nonetheless, its underlying problem-solving idea is much more general than may be evident at first, and it can be extended to a substantially broader class of problems.

Logic-based Benders decomposition (LBBD), which dates from the 1990s, carries out this extension by allowing the subproblem to be *any* optimization problem, at least in principle. The method is "logic-based" in the sense that logical inference plays a key role in its conception, and not because there is any need for the problem statement to consist of logical formulas or have any other relation to formal logic. The enhanced generality of LBBD has enabled its application to a greatly expanded range of real-world problems. This can result in reductions of solution time of several orders of magnitude relative to the previous state of the art.

This book is intended to show how LBBD can be applied to a wide range of optimization problems. It begins by laying out the basic theory of LBBD, along with some variations

J. Hooker, *Logic-Based Benders Decomposition*, Synthesis Lectures on Operations Research and Applications, https://doi.org/10.1007/978-3-031-45039-6_1

and enhancements. Because LBBD can, and often must, be tailored to fit the structure of a given application to obtain fast convergence, the book proceeds to illustrate how this may be accomplished for a variety of problem structures.

1.2 The Fundamental Idea

A Benders method benefits from the fact that subproblems can be solved relatively quickly as it searches for an optimal solution. Yet its primary problem-solving power derives from *Benders cuts* that guide the search. These cuts are constraints that bound the quality of solutions that would result from fixed variable assignments other than those already tried, based on an analysis of previous subproblem solutions. The Benders cuts are accumulated in a *master problem* that is solved to identify the next set of assignments and thereby the next subproblem to solve. The process continues until the optimal value of the master problem, and the best optimal subproblem value obtained so far, converge to the same value. This normally occurs after only a small fraction of the solution space has been explored.

Classical Benders decomposition obtains cuts by solving the linear programming dual of the subproblem. LBBD extends the classical method by observing that the linear programming dual is a special case of an *inference dual* that can be defined for any optimization problem, thus placing no restriction on the form of the subproblem. The inference dual seeks the best bound on a problem's optimal value that can be *logically deduced* from its constraint set. The brilliant maneuver that underlies Benders decomposition is to ask what bound *this same proof* can deduce from the subproblem constraints if the master problem variables are fixed to different values. In the classical Benders method, the proof is encoded as a set of dual multipliers that give rise to a Benders cut in the form of a linear inequality constraint. Logical inference therefore lies at the heart of classical Benders decomposition, and once this is recognized, an extension to LBBD becomes possible. One need only replace linear programming duality with inference duality and write a Benders cut that is based on logical inference.

Formulating a classical Benders cut is straightforward because it always follows the same pattern. By contrast, constructing other types of logic-based cuts typically requires a separate analysis (and often some ingenuity) for every problem class. This can be viewed as a weakness of LBBD, but it also provides an opportunity to design cuts that exploit a problem's special structure. Indeed, one might argue that solution of hard combinatorial problems typically requires problem-specific methods. While we are beginning to see general-purpose LBBD solvers that rely on generic logic-based cuts, we can nonetheless expect many problem classes to require or benefit from hand-crafted cuts.

A major objective of this book is to assist cut design by mining the rapidly growing LBBD literature for ideas on how to exploit the mathematical structure of a particular application. There is a large body of experience to draw from, due to the remarkable number and variety of existing applications, covering such diverse areas as supply chain logistics, computer processor scheduling, organ transplantation, wind turbine maintenance, search-and-rescue operations, and many others.

1.3 Early Developments

The logic-based approach of LBBD was foreshadowed in a 1990 paper of Jeroslow and Wang [138], who showed that the linear programming dual of a Horn clause system can be viewed as an inference problem. A Horn system is a set of specially structured logical propositions whose satisfiability can be checked by linear programming, as well as by an inference method known as unit resolution. When the system is unsatisfiable, a solution of the classical linear programming dual can be read directly from the structure of a unit resolution proof of infeasibility. The key implication of this work for future developments is that the dual problem can be treated as a logical inference problem.

Horn clauses in fact comprise the subproblem in what might be interpreted, in retrospect, as the first clear application of LBBD. It is described in a 1995 paper of Hooker and Yan [127], who solve logic circuit verification problems using a specialized Benders method in which the inputs to the circuit are master problem variables, and Horn clauses represent the circuit design. The first explicit description of LBBD as a general method appeared in the year 2000 [122]. Early computational experiments (for machine assignment and scheduling) are reported in a 2001 paper of Jain and Grossmann [136], and LBBD was further developed and tested computationally in a 2003 paper of Hooker and Ottosson [126].

Two additional developments occurred during the early years. One is *branch and check*, a variation of LBBD that can be used when the master problem is a mixed integer/linear programming (MILP) problem solved by a branching method. Branch and check solves the master problem only once, rather than repeatedly as in standard LBBD. Integer solutions encountered during the branching process are sent to the subproblem to obtain logic-based Benders cuts on the fly. These cuts are used alongside traditional cutting planes while solving the master problem. They differ from the cutting planes in a traditional branch-and-cut method, however, partly because they are valid only when one takes into account the subproblem constraints. Branch and check was initially described in Section 19.6 of [122] and first tested computationally in 2001 by Thorsteinsson [244], who coined the term "branch and check."

A second development is that of *combinatorial Benders cuts*, introduced in 2006 by Codato and Fischetti [60]. These are a type of logic-based cut used in a specialized version of branch and check. Codato and Fischetti use the cuts to solve MILP problems with "big-M" constraints, which are very common but notorious for making problems hard to solve. Their article shows how to get rid of the big-M constraints by accounting for them in the way a linear programming subproblem is constructed, thus substantially accelerating solution. The term "combinatorial Benders cut" is sometimes used to denote logic-based Benders cuts in general, but we reserve the term for logic-based cuts derived from a linear programming subproblem, as this is how it seems most often to be understood.

Surveys of the Benders decomposition literature can be found in [125, 210], the latter dealing specifically with LBBD.

1.4 A Motivating Example

One of the earliest and most frequent applications of LBBD is to assignment and scheduling problems, in which the master problem assigns tasks to facilities or agents, and the subproblem schedules the tasks assigned to each agent. A small example of this kind will illustrate the basic ideas of LBBD.

We have four jobs, any of which can be processed in shop 1 or shop 2. Each job j has a processing time p_{ij} in shop i, and it must be processed within the time window $[r_j, d_j]$. We wish to assign the jobs to shops and schedule them so as to minimize makespan, which the finish time of the last job to finish. The specific problem data appear in Table 1.1. Note that shop 1 processes jobs 1 and 3 more slowly than shop 2, while jobs 2 and 4 have the same processing time in either shop.

To write an optimization model for a problem of this form, suppose there are n jobs, and let the binary variable $x_{ij} = 1$ when job j is assigned to shop i. Then if we let s_j denote the start time of job j, the problem is

$$\min_{M,\boldsymbol{x},\boldsymbol{s}} \left\{ M \;\middle|\; \begin{array}{l} M \geqslant s_j + \sum\limits_i p_{ij} x_{ij},\ \sum\limits_i x_{ij} = 1,\ s_j \geqslant r_j,\ \text{all } j \\ s_j + \sum\limits_i p_{ij} x_{ij} \leqslant s_k \ \text{ or } \ s_k + \sum\limits_i p_{ik} x_{ik} \leqslant s_j, \\ \qquad\qquad\qquad\qquad \text{all } j, k \text{ with } j < k \text{ and } x_{ij} = x_{ik} = 1,\ \text{all } i \\ \boldsymbol{x} \in S;\quad x_{ij} \in \{0, 1\},\ \text{all } i, j \end{array} \right\}$$

where $\boldsymbol{s} = (s_1, \ldots, s_n)$, and $\boldsymbol{x}$ is the matrix of variables x_{ij}. The first line of constraints define the makespan M, ensure that every job is assigned to exactly one agent, and require that jobs start no earlier than their release time. The second line prevents jobs from overlapping

Table 1.1 Data for a small example problem

Job j	r_j	d_j	p_{1j}	p_{2j}
1	3	6	3	2
2	3	5	1	1
3	0	5	3	2
4	3	6	1	1

Table 1.2 Benders iterations for a small example problem

Iteration	Master problem optimal value	Job assignments $(x_{11}, \ldots, x_{14})$	Subproblem optimal value[†]	Subproblem solution $(s_1, \ldots, s_4)$
1		(1, 1, 0, 0)	∞	
2	0	(0, 1, 0, 1)*	5	(3, 4, 0, 3)*
3	4	(1, 0, 0, 0)	6	(3, 3, 0, 4)
4	5	(0, 0, 1, 1)		

*optimal solution
[†]∞ indicates infeasible subproblem

by requiring, for every pair of jobs in the same shop, that one finish before the other starts. The constraint $\boldsymbol{x} \in S$ covers any restrictions on the assignments.

The problem decomposes naturally into an assignment problem and a scheduling problem. This is particularly advantageous because the scheduling subproblem decouples into a separate problem for each shop once assignments have been made. Thus for a particular assignment $\bar{\boldsymbol{x}}$, we have the following scheduling problem for each shop i:

$$\min_{M_i, s} \left\{ M_i \;\middle|\; \begin{array}{l} M_i \geqslant s_j + p_{ij}, \quad s_j \geqslant r_j, \quad \text{all } j \in J_i \\ s_j + p_{ij} \leqslant s_k \;\text{ or }\; s_k + p_{ik} \leqslant s_j, \quad \text{all } j, k \in J_i \text{ with } j < k \end{array} \right\}$$

where $J_i = \{j \mid \bar{x}_{ij} = 1\}$ is the set of jobs assigned to shop i. If M_i is the makespan in shop i, then $\max_i\{M_i\}$ is the overall makespan we wish to minimize.

We now proceed to solve the problem by LBBD, which enumerates assignments to the variables x_{ij} and observes the smallest makespan that results. The progress of the procedure is summarized in Table 1.2.

Iteration 1. Suppose we begin by assigning jobs 1 and 2 to shop 1, and jobs 3 and 4 to shop 2, so that $(x_{11}, x_{12}, x_{13}, x_{14}) = (1, 1, 0, 0)$, and $(x_{21}, x_{22}, x_{23}, x_{24})$ is the complement $(0, 0, 1, 1)$. The resulting minimum makespan problem for each shop is illustrated by Gantt charts in Fig. 1.1. The horizontal dimension is time, and the brackets indicate time windows. The processing time is shown by a heavy line within each bracket.

It is clear from Fig. 1.1a that assigning jobs 1 and 2 to shop 1 creates a scheduling problem with no feasible solution. We therefore write a simple *nogood cut* indicating that this assignment must be avoided:

$$x_{11} + x_{12} \leqslant 1 \tag{1.1}$$

This cut actually excludes four assignments of jobs to shops, since there are four assignments in which shop 1 receives jobs 1 and 2 (possibly among others). All of these assignments are infeasible, because assigning jobs 1 and 2 already creates infeasibility.