Benders Decomposition: Solving Binary Master Problems by Enumeration

D. Antony Tarvin^a, R. Kevin Wood^b, Alexandra M. Newman^{c,*}

^aWhite Sands Missile Range, NM 88002, USA

^bOperations Research Department, Naval Postgraduate School, Monterey, CA 93943, USA

^cMechanical Engineering Department, Colorado School of Mines, Golden, CO 80401, USA

Abstract

We develop a variant of Benders decomposition for mixed-integer programming that solves each master problem by explicit enumeration. By storing the master problem's current objective-function value for each potential solution, computational effort remains essentially constant across iterations. Using both serial and parallel processing, tests against competing methods show computational speedups that exceed two orders of magnitude.

Keywords: Benders decomposition, explicit enumeration, facility location

1. Introduction

Many mixed-integer programs (MIPs) solve efficiently only through decomposition (e.g., [1]). Benders decomposition is typical [2], and has been employed in such applications as distribution planning [3], power-flow optimization [4] and preventive-maintenance scheduling [5]. Magnanti and Wong [6] note that a key computational challenge for the Benders decomposition algorithm (**BD**) is solving the (*relaxed*) *mixed-integer master problem* repeatedly. Each iteration of **BD** introduces a new constraint, a *Benders cut*, which increases the master problem's size and, thus, its worst-case solution time [7, p. 125]. A typical **BD** implementation solves the master problems by standard, linear-programming-based branch-and-bound (**B&B**). Our paper investigates how the decomposition algorithm's empirical efficiency may improve by solving those master problems through the explicit enumeration of feasible solutions.

A number of techniques have been developed to improve **BD**'s computational efficiency, for example: (i) strengthening master-problem cuts by exploiting interior-point subproblem solutions [8]; (ii) adding knapsack cover cuts based on Benders cuts [9]; (iii) deleting Benders cuts from early iterations

that become “unattractive” [10]; and (iv) reformulating to yield a “multicut master problem,” which possesses one group of cuts for each independent subproblem [11]. Rather than modifying master-problem formulations, however, we seek a faster method to solve master problems, a method that applies to standard and multicut formulations, and which should benefit from enhancements such as the inclusion of knapsack cuts or other integer cutting planes. Because of this focus, we compare our method against two important, existing variants of **BD** that also modify how master problems are solved: (i) *master-problem suboptimization*, which relaxes standard optimality requirements for the master problem [3]; and (ii) *Benders branch-and-cut*, which adds Benders cuts within the branch-and-bound enumeration tree of a single master problem [12]. We describe both techniques in more detail later.

Salmeron and Wood [13] first suggest the technique investigated in our paper: converting standard **BD** into “**BD-E**” by solving **BD**'s master problem by explicit enumeration. This technique has clear computational limitations, but those limitations may not be relevant in many settings. For example, models for infrastructure protection and/or interdiction [14] often describe a “target-rich but resource-poor environment,” in which resources limit the number of targets (e.g., bridges, electric power substations) that might be attacked or protected. In this case, a feasible master-problem solution corresponds exactly to a resource-feasible

*Corresponding author

Email addresses: dtarvin@mines.edu (D. Antony Tarvin), kwood@nps.edu (R. Kevin Wood), anewman@mines.edu (Alexandra M. Newman)

attack or protection plan, so enumeration of each such plan and evaluation in the relaxed master problem may be computationally viable [13]. (Evaluating each plan exactly through the Benders “subproblem” would be computationally prohibitive, however.)

The implementation of the enumeration procedure in [13] produces a decomposition algorithm that does not consistently outperform standard **BD**. That paper restricts itself, however, to programming the enumeration using a non-compiled, algebraic modeling language, which employs only serial processing. To examine the technique more completely, we implement both serial and parallel algorithms using a compiled programming language and test in the context of an interdiction model, as well as a two-stage stochastic program.

2. Background on Benders Decomposition

To facilitate subsequent development, this section describes a standard **BD** to solve a MIP. For simplicity, we limit attention to a MIP whose discrete variables are binary and which is assumed to have a finite optimal solution:

$$\begin{aligned} \text{MIP} \quad z^* = \min_{\mathbf{x} \in \mathcal{X}, \mathbf{y} \in \mathcal{R}_+^{n_y}} \quad & \mathbf{c}\mathbf{x} + \mathbf{f}\mathbf{y} \\ \text{s.t.} \quad & B\mathbf{x} + D\mathbf{y} \geq \mathbf{d}, \end{aligned} \quad (1)$$

where $\mathcal{X} \equiv \{\mathbf{x} \in \{0, 1\}^{n_x} | A\mathbf{x} \leq \mathbf{b}\}$; $\mathcal{R}_+^n$ denotes the n -dimensional, non-negative real space; A , B , and D are dimensioned $m_1 \times n_x$, $m_2 \times n_x$, and $m_2 \times n_y$, respectively; and the vectors $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$, $\mathbf{f}$, $\mathbf{x}$, and $\mathbf{y}$ conform. In addition, we require an efficient enumeration algorithm for $\mathcal{X}$, which should also have modest cardinality, say $|\mathcal{X}| < 10^9$. (Parallel processing might relax the cardinality limit by several orders of magnitude.) Our two test problems define $\mathcal{X}$ through single knapsack constraints, so efficient enumeration is straightforward. More complex constraint sets defining $\mathcal{X}$ are certainly possible, and we suggest those used in [3, 15] as good examples.

Fixing $\mathbf{x} = \hat{\mathbf{x}} \in \mathcal{X}$ in *MIP* yields the following *subproblem*:

$$\text{SUB}(\hat{\mathbf{x}}) \quad z_{\text{SUB}}^*(\hat{\mathbf{x}}) = \min_{\mathbf{y} \in \mathcal{R}_+^{n_y}} \quad \mathbf{c}\hat{\mathbf{x}} + \mathbf{f}\mathbf{y} \quad (3)$$

$$\text{s.t.} \quad D\mathbf{y} \geq \mathbf{d} - B\hat{\mathbf{x}}. \quad [\alpha(\hat{\mathbf{x}})] \quad (4)$$

where: (i) in brackets and next to its corresponding constraints, $\alpha(\hat{\mathbf{x}})$ represents the optimal dual vector produced by our linear-programming solver; (ii)

we define $\mathcal{A} = \cup_{\hat{\mathbf{x}} \in \mathcal{X}} \{\alpha(\hat{\mathbf{x}})\}$; (iii) $\hat{\alpha} \equiv \alpha(\hat{\mathbf{x}})$, with the argument suppressed for simplicity; and (iv) $\hat{\mathbf{y}}$ denotes the optimal primal solution given $\hat{\mathbf{x}}$ (with no argument for the sake of simplicity).

Assumption 1. *SUB*($\hat{\mathbf{x}}$) is feasible $\forall \hat{\mathbf{x}} \in \mathcal{X}$. ■

Assumption 1 corresponds to “relatively complete recourse” in the stochastic-programming literature (e.g., [16]). While convenient for exposition, the assumption is not limiting. If it does not hold, then **BD** can (i) introduce “feasibility cuts” to the master problem as necessary [1], or (ii) solve a standard, reformulated model that penalizes constraint violations in the subproblem. Section 6 explains how to implement feasibility cuts in **BD-E**.

Under Assumption 1, **BD** reformulates *MIP* into this *equivalent master problem*:

$$\begin{aligned} \text{MP}(\mathcal{A}) \quad z^* = z_{\text{MP}}^*(\mathcal{A}) = \min_{\mathbf{x} \in \mathcal{X}, \eta} \quad & \eta \\ \text{s.t.} \quad & \eta \geq \mathbf{c}\mathbf{x} + \hat{\alpha}(\mathbf{d} - B\mathbf{x}) \quad \forall \hat{\alpha} \in \mathcal{A}. \end{aligned} \quad (5)$$

Rather than exhaustively computing the elements of $\mathcal{A}$ and then solving *MP*($\mathcal{A}$) directly, a general version of **BD** begins with an arbitrary set $\hat{\mathcal{A}} \subset \mathcal{A}$ and with an initial feasible solution $\hat{\mathbf{x}} \in \mathcal{X}$. Then, iteratively, **BD** (i) solves *SUB*($\hat{\mathbf{x}}$) for $\hat{\mathbf{y}}$ and $\hat{\alpha}$, (ii) adds $\hat{\alpha}$ to $\hat{\mathcal{A}}$, and (iii) solves the *relaxed master problem* *MP*($\hat{\mathcal{A}}$) for $\hat{\mathbf{x}}$. The algorithm terminates once *MP*($\hat{\mathcal{A}}$) approximates *MP*($\mathcal{A}$) well enough to prove near-optimality of some previously discovered $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ [2]. “Algorithm 1” summarizes **BD** for later reference; we refer the reader to Benders’ original paper [2] for a proof of correctness.

3. Solving *MP*($\hat{\mathcal{A}}_K$) by Enumeration

Typically, **BD** solves the K th relaxed master problem *MP*($\hat{\mathcal{A}}_K$) by **B&B**, but if $|\mathcal{X}|$ is sufficiently small, we will demonstrate that explicit enumeration can solve *MP*($\hat{\mathcal{A}}_K$) more efficiently. The comparative efficiency of enumeration will become apparent as the number of cuts in *MP*($\hat{\mathcal{A}}_K$) increases with increasing K and **B&B** slows dramatically.

Let $\hat{\alpha}_k$ denote the dual vector generated in **BD**’s k th iteration. The master problem in iteration K is then

$$\text{MP}(\hat{\mathcal{A}}_K) \quad z_{\text{MP}}^*(\hat{\mathcal{A}}_K) = \min_{\mathbf{x} \in \mathcal{X}, \eta} \quad \eta \quad (7)$$

$$\text{s.t.} \quad \eta \geq g_0(\hat{\mathbf{x}}_k) + \sum_{j=1}^{n_x} g_j(\hat{\mathbf{x}}_k) x_j \quad \forall k \in \{1, \dots, K\}, \quad (8)$$

Algorithm 1 - Benders decomposition algorithm (**BD**)

Input: An instance of *MIP* and allowable optimality gap $\varepsilon \geq 0$.
Output: An ε -optimal solution to *MIP*.
Step 0: Initialization
 a. $K \leftarrow 1$; $\underline{z} \leftarrow -\infty$; $\bar{z} \leftarrow \infty$; $\hat{\mathcal{A}}_0 \leftarrow \emptyset$;
 $\hat{\mathbf{x}}_1 \leftarrow \text{any } \mathbf{x} \in \mathcal{X}$;
 b. Dummy step for initialization;
Step 1: Subproblem
 Solve *SUB*($\hat{\mathbf{x}}_K$) for $\hat{\mathbf{y}}_K$, $\hat{\alpha}_K$ and $z_{\text{SUB}}^*(\hat{\mathbf{x}}_K)$;
 $\hat{\mathcal{A}}_K \leftarrow \hat{\mathcal{A}}_{K-1} \cup \{\hat{\alpha}_K\}$;
 If ($z_{\text{SUB}}^*(\hat{\mathbf{x}}_K) < \bar{z}$) { $\bar{z} \leftarrow z_{\text{SUB}}^*(\hat{\mathbf{x}}_K)$;
 $(\mathbf{x}^*, \mathbf{y}^*) \leftarrow (\hat{\mathbf{x}}_K, \hat{\mathbf{y}}_K)$; }
 If ($\bar{z} - \underline{z} \leq \varepsilon$) go to Step 3;
Step 2: Master Problem
 a. Solve *MP*($\hat{\mathcal{A}}_K$) for $\hat{\mathbf{x}}_{K+1}$ and $z_{\text{MP}}^*(\hat{\mathcal{A}}_K)$;
 b. $\underline{z} \leftarrow z_{\text{MP}}^*(\hat{\mathcal{A}}_K)$;
 c. If ($\bar{z} - \underline{z} \leq \varepsilon$) go to Step 3;
 d. $K \leftarrow K + 1$; Go to Step 1;
Step 3: Print Solution
 Print (“ ε -optimal solution to *MIP* is”,
 $(\bar{z}, \mathbf{x}^*, \mathbf{y}^*)$); Halt.

where, letting $B_{\cdot j}$ denote B ’s j th column, $g_j(\hat{\mathbf{x}}_k) \equiv -\hat{\alpha}_k B_{\cdot j} + c_j$ for $j = 1, \dots, n_x$, and $g_0(\hat{\mathbf{x}}_k) \equiv \hat{\alpha}_k \mathbf{d}$. Equivalently, *MP*($\hat{\mathcal{A}}_K$) may be written as

$$MP(\hat{\mathcal{A}}_K) \quad z_{\text{MP}}^*(\hat{\mathcal{A}}_K) = \min_{\mathbf{x} \in \mathcal{X}} \eta_K(\mathbf{x}), \text{ where} \quad (9)$$

$$\eta_K(\mathbf{x}) = \max_{k=1, \dots, K} \left\{ g_0(\hat{\mathbf{x}}_k) + \sum_{j=1}^{n_x} g_j(\hat{\mathbf{x}}_k) x_j \right\}, \quad (10)$$

and where $\hat{\mathbf{x}}_{K+1} = \text{argmin}_{\mathbf{x} \in \mathcal{X}} \eta_K(\mathbf{x})$.

Now, rather than solving *MP*($\hat{\mathcal{A}}_K$) “from scratch” in each iteration as a mathematical program defined by (7) and (8), equations (9) and (10) lead to a simple “update computation” for evaluating $\eta_K(\mathbf{x})$, and thus solving *MP*($\hat{\mathcal{A}}_K$):

$$\eta_K(\mathbf{x}) \leftarrow \max \left\{ \eta_{K-1}(\mathbf{x}), g_0(\hat{\mathbf{x}}_K) + \sum_{j=1}^{n_x} g_j(\hat{\mathbf{x}}_K) x_j \right\} \forall \mathbf{x} \in \mathcal{X}. \quad (11)$$

By storing $\eta_{K-1}(\mathbf{x})$ for each $\mathbf{x} \in \mathcal{X}$, the computational effort to solve *MP*($\hat{\mathcal{A}}_K$) will not increase with K , as it tends to with standard **B&B**. Algorithm 2 presents the modifications of Algorithm 1 that convert standard **BD** into **BD-E**. **BD-E** is simply a particular implementation of **BD**, so Benders’ proof of correctness [2] still applies.

Algorithm 2 - Benders decomposition algorithm with master problems solved by explicit enumeration (**BD-E**)

Proceed as in **BD** (Algorithm 1), with these variations:
Replace **Step 0b** with:
 $\eta_0(\mathbf{x}) \leftarrow -\infty \forall \mathbf{x} \in \mathcal{X}$;
Replace **Step 2a** with:
 $z_{\text{MP}}^*(\hat{\mathcal{A}}_K) \leftarrow \infty$;
 For (all $\mathbf{x} \in \mathcal{X}$) {
 $\eta_K(\mathbf{x}) \leftarrow \max\{\eta_{K-1}(\mathbf{x}), g_0(\hat{\mathbf{x}}_K) + \sum_{j=1}^{n_x} g_j(\hat{\mathbf{x}}_K) x_j\}$;
 Dummy step for an integer cut; /* See Section 6. */
 If ($\eta_K(\mathbf{x}) < z_{\text{MP}}^*(\hat{\mathcal{A}}_K)$) {
 $z_{\text{MP}}^*(\hat{\mathcal{A}}_K) \leftarrow \eta_K(\mathbf{x})$; $\hat{\mathbf{x}}_{K+1} \leftarrow \mathbf{x}$;
 }
 }

To solve *MP*($\hat{\mathcal{A}}$) as described above, **BD-E** must generate every $\mathbf{x} \in \mathcal{X}$. Let us assume that $\mathcal{X}$ is defined through a single knapsack constraint

$$\mathcal{X} \equiv \{\mathbf{x} \in \{0, 1\}^{n_x} \mid \mathbf{a}\mathbf{x} \leq b\}, \quad (12)$$

where all data are positive integers. To produce an algorithm to enumerate $\mathcal{X}$, we (i) remove the optimality tests from the branch-and-bound algorithm described by Horowitz and Sahni for solving a 0-1 knapsack problem [17, pp. 350-354], and (ii) improve efficiency by introducing a stack to represent partial solutions. Also, for the test problem in Section 4 (but not Section 5), we reduce computational effort by exploiting the fact that an optimal solution is always maximal for the knapsack constraint.

Parallel processing seems ideally suited for reducing computation times when explicit enumeration is involved. To solve *MP*($\hat{\mathcal{A}}_K$) on P parallel processors, we first use the knapsack enumeration algorithm to enumerate $\mathcal{X}$ and evaluate its cardinality. Next, we implicitly partition $\mathcal{X}$ into ordered subsets $\mathcal{X}^1, \mathcal{X}^2, \dots, \mathcal{X}^P$, such that: (i) each subset’s elements are defined in decreasing lexicographic order; (ii) $|\mathcal{X}^p| \approx |\mathcal{X}|/P$ for $p = 1, \dots, P$; and (iii) with $>_L$ indicating the lexicographic inequality, $\mathbf{x} >_L \mathbf{x}'$ for all $p < p'$, $\mathbf{x} \in \mathcal{X}^p$, and $\mathbf{x}' \in \mathcal{X}^{p'}$. This partitioning scheme permits the enumeration algorithm to regenerate an entire subset $\mathcal{X}^p$ while only storing the subset’s first element and its cardinality.

For each master problem, a parallel version of **BD-E** can then: (i) use a separate processor labeled p to evaluate $\eta_K(\mathbf{x})$ for each $\mathbf{x} \in \mathcal{X}^p$; (ii) compute $z_{\text{MP}}^{*p}(\hat{\mathcal{A}}_K) \leftarrow \min_{\mathbf{x} \in \mathcal{X}^p} \eta_K(\mathbf{x})$ and $\hat{\mathbf{x}}_{K+1}^p \leftarrow \text{argmin}_{\mathbf{x} \in \mathcal{X}^p} \eta_K(\mathbf{x})$ for each $p \in \{1, \dots, P\}$; and finally (iii) compute $z_{\text{MP}}^*(\hat{\mathcal{A}}_K) \leftarrow \min_{p \in \{1, \dots, P\}} z_{\text{MP}}^{*p}(\hat{\mathcal{A}}_K)$ and $\hat{\mathbf{x}}_{K+1} \leftarrow \text{argmin}_{p \in \{1, \dots, P\}} \eta_K(\hat{\mathbf{x}}_{K+1}^p)$.

4. Computational Results: Solving Master Problems Only

We first test explicit enumeration by solving the sequence of 1400 master problems that a generalized version of **BD**, “global Benders decomposition,” generates in [18] as it solves a network-interdiction problem “*NI*.” We focus on master problems only, solving $MP(\hat{A}_K)$ using explicit enumeration or standard **B&B** for $K = 1, \dots, 1400$. “**MP-E**” and “**MP-B&B**” denote the respective master-problem solution methods.

In *NI*, an attacker seeks to maximize the estimated cost of operating an electric power transmission grid by disabling up to five of the grid’s 113 substations. Converting the master problem in [18] to a minimizing model, the resulting Benders cuts have the form of (8), where: (i) $x_j = 1$ if the attacker disables substation j , and $x_j = 0$, otherwise; (ii) $-g_0(\hat{\mathbf{x}})$ represents the estimated network operating cost after the attacker executes interdiction plan $\hat{\mathbf{x}}$; (iii) if $\hat{x}_j = 0$, then $-g_j(\hat{\mathbf{x}})$ represents an upper bound on the additional cost that would result from disabling substation j ; and (iv) if $\hat{x}_j = 1$, then $g_j(\hat{\mathbf{x}}) = 0$. Since disabling a substation cannot decrease overall operating cost, we may assume that an optimal attack plan will always be maximal with respect to “attack resource.” Thus, both solution methods enforce the resource constraint “ $\mathbf{1x} = 5$ ” rather than “ $\mathbf{1x} \leq 5$.” For **MP-E**, this implies that each master problem requires the evaluation of $|\mathcal{X}| = \binom{113}{5} \approx 1.4 \times 10^8$ solutions.

We implement **MP-E** and **MP-B&B** in C++, using the GNU G++ compiler and OpenMP parallelization [19] on a computer with four Intel Xeon E5520 quad-core processors and 12 GB of memory. CPLEX v.12.4.0 [20] solves the mixed-integer master problems for **MP-B&B**. CPLEX’s default parameters apply except that (i) the relative optimality gap is relaxed to 0.1%, (ii) parallel computation uses “deterministic parallelism,” and (iii) the number of parallel threads P is specified. **MP-E** solves all problems exactly. Reported times exclude model generation.

Basic computational tests compare the time required by **MP-E** and **MP-B&B** to solve the entire series of 1400 master problems from [18]. For $q = 1, \dots, 14$, Figure 1 displays the total seconds required by **MP-E** and **MP-B&B** to solve $100q$ master problems using either one or 16 threads. The results are promising and can be summarized as follows: with a single thread, **MP-E** reduces to-

tal solution time by a factor of 18 over **MP-B&B** and, with 16 threads, that factor becomes 58.

Figure 1: Cumulative solution times for Benders master problems: branch-and-bound (**MP-B&B**) versus explicit enumeration (**MP-E**) using one or 16 threads.

The reader should consider two caveats with respect to the apparent superiority of **MP-E** over **MP-B&B** and what that means for **BD-E** versus **BD**. First, we only compare master-problem solution times here, and the corresponding decomposition algorithms, **BD-E** and **BD**, might produce distinct solution trajectories and different numbers of iterations. However, if the number of decomposition iterations K (and the associated subproblem solution times) differs only modestly, while master problems solve much faster with **MP-E**, then **BD-E** would certainly be the preferred decomposition algorithm. Second, in our experience, the average and maximum integrality gaps for these master problems are unusually large, being 87.6% and 124.6%, respectively. Thus, the long solution times observed here for **MP-B&B** might not be representative of “usual” master problems. But, the test problems in Section 5.3 exhibit similar differences in solution times for **MP-E** and **MP-B&B**, yet the average and maximum integrality gaps there are only 4.1% and 6.0%, respectively.

To explore more fully the use of parallel computation in solving the master problems from *NI*, we have run **MP-B&B** and **MP-E** using four, eight, and 12 threads, in addition to the one and 16 already covered. Table 1 lists total solution times and empirical parallel-processing efficiencies [21]. Clearly, **MP-E** benefits more from parallel processing than does **MP-B&B**.

As a final point on these computational tests, we note that the solution times for **MP-E** do remain

Algorithm		Number of threads (P)				
		1	4	8	12	16
MP-B&B	T_P (sec.)	22352	11521	9449	9095	8864
	Efficiency	1.00	0.49	0.30	0.20	0.16
MP-E	T_P (sec.)	1208	414	236	192	151
	Efficiency	1.00	0.73	0.64	0.52	0.50

Table 1: Solution times (T_P) and efficiencies using P parallel threads to solve all 1400 master problems for NI . Efficiency for P threads is defined as $T_1/(P \cdot T_P)$.

roughly constant, at about 0.86 seconds with one thread and 0.11 seconds with 16. By contrast, solution times increase dramatically for **MP-B&B** as the size of $MP(\mathcal{A}_K)$ increases. For example, using one thread, the solution time averages 0.40 seconds for each of the first 100 master problems, but the corresponding value becomes 63.74 seconds for the final 100 master problems.

5. Computational Results: A Complete Solution by Benders Decomposition

This section more definitively investigates the computational effectiveness of **BD-E** and comparable decomposition algorithms by solving complete instances of the stochastic facility location problem (*SFLP*) described in [22]:

$$\min_{\mathbf{x}, \mathbf{y}, \mathbf{v}} \sum_{i \in I} c_i x_i + \sum_{s \in S, i \in I, j \in J} p_s f_{ij} y_{ij}^s + \sum_{s \in S, j \in J} p_s r_j v_j^s \quad (13)$$

$$\text{s.t.} \quad -u_i x_i + \sum_{j \in J} y_{ij}^s \leq 0 \quad \forall s \in S, i \in I \quad (14)$$

$$\sum_{i \in I} y_{ij}^s + v_j^s \geq d_j^s \quad \forall s \in S, j \in J \quad (15)$$

$$\sum_{i \in I} x_i \leq b \quad (16)$$

$$x_i \in \{0, 1\} \quad \forall i; \quad y_{ij}^s \geq 0 \quad \forall s, i, j; \quad v_j^s \geq 0 \quad \forall s, j; \quad (17)$$

where $x_i = 1$ if facility $i \in I$ is constructed, and $x_i = 0$, otherwise; y_{ij}^s is the amount of commodity shipped from facility i to customer $j \in J$ in probabilistic scenario $s \in S$; v_j^s is unserved demand at j in scenario s ; and $p_s \equiv 1/|S|$ for all s . We use the parameters in [22] to define the probability distribution for demands d_j^s , and to define plant capacities u_i , plant construction costs c_i , production and shipping costs f_{ij} , and unmet-demand penalties r_j .

We cannot show that **BD-E** will always be faster than competing techniques, given that the combinatorics of some problems may be prohibitive. Thus,

our purpose here is to show that, for a realistic class of problems for which the technique does apply, **BD-E** can be the best choice.

5.1. Solution Techniques

Naturally, we compare **BD-E** to **BD** as a baseline. Because **BD-E** replaces the standard algorithm that solves **BD**'s master problem, we also compare to two existing variants of **BD** that make similarly substantial changes to how the master problem is solved:

(i) **Benders decomposition with suboptimized master problems (BD-SO)**. In an early paper on Benders decomposition, Geoffrion and Graves [3] solve a multicommodity distribution system design model using **BD** and improve computational efficiency by accepting suboptimal master problem solutions. (For examples of more recent applications of **BD-SO**, see [23, 24].) Specifically, given a global optimality tolerance of $100\epsilon\%$, the decomposition will converge provided that, in every iteration K , the master-problem solution $\hat{\mathbf{x}}$ satisfies $\eta_K(\hat{\mathbf{x}}) \leq \bar{z}(1 - \epsilon)$ or it is proven that no such solution exists. (For efficiency, the factor $1 - \epsilon$ might be adjusted as the algorithm proceeds, but we find no consistent computational advantage in doing so here.) **BD-SO** attempts to reduce overall computational effort by making the master problem easier to “solve.” This reduction in computational effort may not come for free, however, if less exact master-problem solutions lead to more iterations which, in turn, lead to more time spent solving subproblems. The net benefit must be evaluated.

(ii) **Benders branch-and-cut (BB&C)**. Branch-and-cut in its original form [25] solves *MIP* using standard **B&B**, but generates and applies integer cutting planes within the branch-and-bound enumeration tree to tighten linear-programming lower bounds; the cutting planes may or may not be globally valid. In essence, **BB&C** applies the branch-and-cut paradigm to the equivalent master problem for *MIP*, generating Benders cuts within a single branch-and-bound tree (e.g., [12, 26]). A cut can be generated for $\hat{\mathbf{x}}$ that is fractional or integral, even though the nominal equivalent master problem only defines cuts for integral $\hat{\mathbf{x}}$. Using the same intuition behind **BD-SO**, **BB&C** attempts to reduce computational effort over **BD** by spending less time “in the master problem,” at the possible expense of generating more cuts and therefore spending more time solving subproblems.

5.2. Hardware, Software and Implementation

We program algorithms for solving *SFLP* using C++ and version 12.6.0 of CPLEX [27]; default settings apply, except as noted. Computations are carried out on a Lenovo W530 laptop computer (i) with an Intel i7-3820QM processor having four cores, (ii) with 32 GB of RAM, and (iii) running the 64-bit Windows 7 Professional operating system. All decomposition algorithms apply a 1% relative optimality tolerance. We note that solution times for even the smallest monolithic MIPs are not competitive, so those times are not reported.

Programming for **BD**, **BD-SO** and **BD-E** is straightforward. We base our code for **BB&C** on the code template “ilobendersatp.cpp” provided with CPLEX. That template implements cuts as lazy constraints [27] for integral $\hat{\mathbf{x}}$, and can create cuts using non-integral solutions, also. The latter type of cut increases run times by a factor of at least 3 for *SFLP*, however, so we do not use it.

We solve the subproblem for any $\hat{\mathbf{x}} \in \mathcal{X}$ as a set of $|S|$, independent, “scenario subproblems.” CPLEX’s primal simplex algorithm solves explicit dual formulations. (A primal scenario subproblem is a pure network, but applying CPLEX’s network simplex algorithm to such subproblems does not yield a competitive solution method.)

5.3. Basic Results

We first solve problem instances with $|I| = 20$, $b = 8$, $|J| = 100$ and $|S| = 2^n$, for $n = 6, \dots, 11$. The parameters $|I| = 20$ and $b = 8$ produce $|\mathcal{X}| = \sum_{b'=0}^8 \binom{20}{b'} \approx 2.6 \times 10^5$. (We present results with a larger set $\mathcal{X}$ later.) Figure 2 shows average solution times for *SFLP* with varying values of $|S|$. The figure shows that **BD-E** is the fastest of the four methods, and that **BB&C** is more than competitive when $|S|$ is small, but becomes the slowest of all the methods for $|S| \geq 512$. The detailed numerical results in Table 2 indicate why: **BB&C** spends little time in its master problem, but it generates about twice as many cuts as the other methods. Thus, it must spend more time solving subproblems, and this becomes computationally burdensome as $|S|$ grows. Clearly, **BD**, **BD-SO** and **BD-E** generate fewer cuts than **BB&C** and, therefore, spend less time solving subproblems. **BD** and **BD-E** generate about the same number of cuts, but **BD** must spend much more time than **BD-E** solving its master problems. **BD-SO** makes a worthwhile trade-off that significantly reduces the

amount of time spent solving master problems, at the expense of only a modest increase in the number of subproblems solved.

Figure 2: Solution times, averaged over 30 instances, for *SFLP* having $|I| = 20$, $b = 8$ and $|J| = 100$. See Table 2 for numerical data and note that parallel processing does not apply here.

5.4. Parallel Processing

BB&C and **BD-E** do not spend much time solving their master problems in the instances of *SFLP* tested here, so there is little or nothing to be gained in parallelizing master-problem algorithms. And, in fact, CPLEX cannot currently parallelize the **BB&C** master problem because of synchronization issues. The large number of independent scenario subproblems does lead us to hope, however, that a massively parallel solution of those subproblems could reduce times dramatically [9]. The last column of Table 2 shows estimated run times for all algorithms, assuming that each scenario subproblem could be solved on a separate processor, in parallel, with perfect efficiency. These estimates use a maximum subproblem solution time of 0.0030 seconds, which is a conservative estimate established using observations for $|S| = 2048$. (Naturally, these maximum times tend to be shorter with smaller values of $|S|$, but we use a single, conservative value for simplicity.) **BB&C** becomes the second best algorithm, but **BD-E** remains the fastest by a factor of at least 3. The other two algorithms would not be competitive, so corresponding results are omitted.

We do not have access to a massively parallel computer, but have begun to use a limited number of parallel threads to solve the independent scenario subproblems in *SFLP*. To test this and parallel processing for **BD-E**’s master problem, we solve 10 instances of *SFLP* having $|I| = 30$, $b = 10$, $|J| = 100$

Alg	Scen (num.)	Tot (sec.)	Sub (sec.)	MP ⁺ (sec.)	Iter (num.)	Para (sec.)
BD	64	194.6	14.5	180.1	242	180.8
	128	174.2	26.8	147.4	226	148.1
	256	137.3	44.9	92.4	189	93.0
	512	170.3	92.1	78.2	193	78.7
	1024	295.9	208.2	87.6	225	88.3
	2048	462.7	383.2	79.5	213	80.1
BD-SO	64	58.0	15.7	42.4	266	43.1
	128	65.1	28.8	36.3	244	37.0
	256	71.3	46.3	24.8	200	25.4
	512	123.3	96.6	26.6	213	27.2
	1024	251.2	220.8	30.4	245	31.2
	2048	440.4	412.3	28.1	230	28.8
BB&C	64	33.2	25.5	7.7	470	9.1
	128	57.9	54.6	3.4	438	4.7
	256	102.8	100.7	2.1	394	3.3
	512	207.6	205.5	2.1	399	3.3
	1024	439.1	437.1	2.0	412	3.3
	2048	990.2	987.6	2.5	427	3.8
BD-E	64	15.3	15.0	0.2	241	1.0
	128	28.2	28.0	0.2	226	0.9
	256	45.7	45.1	0.2	188	0.8
	512	92.6	92.4	0.2	193	0.8
	1024	215.7	215.4	0.2	225	0.9
	2048	425.3	425.1	0.2	227	0.9

Table 2: Data used to produce Figure 2. “Alg” = algorithm and “Scen” = number of scenarios. Averaged over 30 instances then, “Tot” = total solution time, “Sub” = time spent solving subproblems, “MP⁺” = time spent solving master problems plus overhead (i.e., MP⁺ = Tot – Sub), “Iter” = number of Benders iterations, or number of cuts generated in **BB&C**, and “Para” estimates the average solution time if all scenario subproblems were solved in parallel, with perfect efficiency.

and $|S| = 256$. The first and second parameters yield $|\mathcal{X}| \approx 5.3 \times 10^7$ (cf. $|\mathcal{X}|$ in Section 5.3). Table 3 presents results.

Table 3 clearly shows shorter average solution times for **BD-E** compared to **BB&C**, although we cannot explain the relative inefficiency of parallel processing for subproblems in **BB&C**. However, assuming the same parallel-processing efficiency as that of **BD-E**, namely 0.77, the average solution time for **BB&C** would reduce only to 1348.0 seconds compared to the fastest time for **BD-E**, which is 380.2 seconds.

6. Potential Enhancements

Multicut versions of **BD** and **BD-E** may improve solution efficiency for some models, but Tarvin [28] reports that multicut methods have the opposite effect when applied to *SFLP*. Also, generating Benders cuts from interior-point dual subproblem solutions may produce more effective cuts for some models [8] but, at least for *SFLP*, relatively slow

Alg	Threads (num.)		Tot (sec.)	Sub (sec.)	MP ⁺ (sec.)	Iter (num.)
	Subs	MP				
BB&C	1	1	3057.9	2531.9	526.0	3379
	4	1	2112.9	1672.6	440.3	3257
	4	4	NA	NA	NA	NA
BD-E	1	1	1065.1	681.9	383.2	1860
	4	1	663.6	221.0	442.7	1860
	4	4	380.2	102.4	277.8	1860

Table 3: Computational results, averaged over 10 instances, for **BB&C** and **BD-E** applied to *SFLP* with $|I| = 30$, $b = 10$, $|J| = 100$ and $|S| = 256$. “Threads” gives the number of threads used to solve the scenario subproblems (Subs) or a master problem (MP). “NA” reflects the fact that, currently, **BB&C**’s master problem cannot exploit parallel processing.

interior-point solutions from CPLEX negate any value such cuts might have.

One unexplored research topic is the use in **BD-E**’s master problem of integer cutting planes (“integer cuts”) that do not involve η , e.g., [9]. (Super-valid inequalities [29] would also use the techniques described here, while integer cuts involving both η and $\mathbf{x}$ could be scaled and treated as standard Benders cuts.) Such integer cuts should be straightforward to implement. For example, suppose that in each iteration K of **BD-E**, a single integer cut is generated along with a standard Benders cut. The integer cut may be expressed as

$$\bar{g}_0(\hat{\mathbf{x}}_K) + \sum_{j=1}^{n_x} \bar{g}_j(\hat{\mathbf{x}}_K) x_j \leq 0. \quad (18)$$

Thus, if $\bar{g}_0(\hat{\mathbf{x}}_K) + \sum_{j=1}^{n_x} \bar{g}_j(\hat{\mathbf{x}}_K) x_j > 0$, then $\mathbf{x} \notin \mathcal{X}$. If setting $\eta_K(\mathbf{x}) \equiv M$ for some sufficiently large value $M \in R_+^1$ indicates that $\mathbf{x} \notin \mathcal{X}$, then (18) may be implemented in **BD-E** by replacing “Dummy step for integer cut” in Algorithm 2 with this:

$$\text{If } \left(\bar{g}_0(\hat{\mathbf{x}}_K) + \sum_{j=1}^{n_x} \bar{g}_j(\hat{\mathbf{x}}_K) x_j > 0 \right) \eta_K(\mathbf{x}) \leftarrow M;$$

By extension, adding h integer cuts at each iteration of **BD-E** would increase the master problem’s solution time by a factor of at most $1 + h$ because: (i) the arithmetic operations associated with evaluating an integer cut and a Benders cut are analogous; and (ii) just as a Benders cut, once an integer cut is evaluated, it need not be considered again. Note also that a Benders feasibility cut [1] could be implemented as just described.

Acknowledgements

Kevin Wood thanks the Defense Threat Reduction Agency, under grant HDTRA1-10-1-0087, for

its support. He also thanks the Office of Naval Research for its support.

References

- [1] R. M. Van Slyke, R. Wets, L-shaped linear programs with applications to optimal control and stochastic programming, *SIAM J. Appl. Math.* 17 (4) (1969) 638–663.
- [2] J. F. Benders, Partitioning procedures for solving mixed-variables programming problems, *Numerische Mathematik* 4 (1) (1962) 238–252.
- [3] A. M. Geoffrion, G. W. Graves, Multicommodity distribution system design by Benders decomposition, *Manag. Sci.* 20 (5) (1974) 822–844.
- [4] N. Alguacil, A. J. Conejo, Multiperiod optimal power flow using Benders decomposition, *IEEE Trans. Pow. Syst.* 15 (1) (2000) 196–201.
- [5] S. P. Canto, Application of Benders’ decomposition to power plant preventive maintenance scheduling, *Eur. J. Oper. Res.* 184 (2) (2008) 759–777.
- [6] T. L. Magnanti, R. T. Wong, Accelerating Benders decomposition: Algorithmic enhancement and model selection criteria, *Oper. Res.* 29 (3) (1981) 464–484.
- [7] G. Nemhauser, L. Wolsey, *Integer and Combinatorial Optimization*, Wiley-Interscience, New York, 1999.
- [8] J. Naoum-Sawaya, S. Elhedhli, An interior-point Benders based branch-and-cut algorithm for mixed integer programs, *Ann. Oper. Res.* 210 (1) (2013) 33–55.
- [9] T. Santos, S. Ahmed, M. Goetschalckx, A. Shapiro, A stochastic programming approach for supply chain network design under uncertainty, *Eur. J. Oper. Res.* 167 (1) (2005) 96–115.
- [10] A. M. Geoffrion, Generalized Benders decomposition, *J. Optim. Theory Appl.* 10 (4) (1972) 237–260.
- [11] J. R. Birge, F. V. Louveaux, A multicut algorithm for two-stage stochastic linear programs, *Eur. J. Oper. Res.* 34 (3) (1988) 384–392.
- [12] V. Sridhar, J. S. Park, Benders-and-cut algorithm for Fixed-charge capacitated network design problem, *Eur. J. Oper. Res.* 125 (3) (2000) 622–632.
- [13] J. Salmeron, R. K. Wood, The value of recovery transformers in protecting an Electric transmission grid against attack, *IEEE Trans. Pow. Syst.* 30 (5) (2015) 2396 – 2403.
- [14] G. Brown, M. Carlyle, J. Salmerón, K. Wood, Defending critical infrastructure, *Interfaces* 36 (6) (2006) 530–544.
- [15] S. C. Sarin, J. West-Hansen, The long-term mine production scheduling problem, *IIE Trans.* 37 (2) (2005) 109–121.
- [16] R. J.-B. Wets, Stochastic programs with fixed recourse: The equivalent deterministic program, *SIAM Rev.* 16 (3) (1974) 309–339.
- [17] E. Horowitz, S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, Inc., Rockville, MD, 1978.
- [18] J. Salmeron, K. Wood, R. Baldick, Worst-case interdiction analysis of large-scale electric power grids, *IEEE Trans. Pow. Syst.* 24 (1) (2009) 96–104.
- [19] OpenMP Architecture Review Board, *OpenMP 4.0 Specifications*, URL openmp.org, 2014.
- [20] IBM, *IBM ILOG CPLEX Optimization Studio, Getting Started with CPLEX*, v12.4, IBM Corp., 2011.
- [21] Y. Xu, T. K. Ralphs, L. Ladányi, M. J. Saltzman, Computational experience with a software framework for parallel integer programming, *INFORMS J. Comp.* 21 (3) (2009) 383–397.
- [22] S. M. Sanchez, R. K. Wood, The “BEST” algorithm for solving stochastic mixed integer programs, in: *Proc. 2006 Winter Sim. Conf.*, Monterey, CA, 765–773, 2006.
- [23] G. Codato, M. Fischetti, Combinatorial Benders’ cuts for mixed-integer linear programming, *Oper. Res.* 54 (4) (2006) 756–766.
- [24] L. Bai, P. A. Rubin, Combinatorial Benders cuts for the minimum tollbooth problem, *Oper. Res.* 57 (6) (2009) 1510–1522.
- [25] H. Crowder, E. Johnson, M. Padberg, Solving large-scale zero-one linear programming problems, *Oper. Res.* 31 (5) (1983) 803–834.
- [26] B. Fortz, M. Poss, An improved Benders decomposition applied to a multi-layer network design problem, *Oper. Res. Lett.* 37 (5) (2009) 359–364.
- [27] IBM, *IBM ILOG CPLEX Optimization Studio, CPLEX User’s Manual*, v12.6, IBM Corp., 2013.
- [28] D. A. Tarvin, *Benders decomposition: An integer-programming extension with further computational enhancements*, Ph.D. thesis, Colorado School of Mines, Golden, CO, 2014.
- [29] E. Israeli, R. K. Wood, Shortest-path network interdiction, *Networks* 40 (2) (2002) 97–111.