Ancestral Benders' Cuts and Multi-term Disjunctions for Mixed-Integer Recourse Decisions in Stochastic Programming *

Yunwei Qi†, Suvrajeet Sen†‡

qi.47@osu.edu, s.sen@usc.edu

First Version: September 2, 2013, Revised: August 28, 2014

Abstract

This paper focuses on solving two-stage stochastic mixed integer programs (SMIPs) with general mixed integer decision variables in both stages. We develop a decomposition algorithm in which the first stage approximation is solved using a branch-and-bound tree with nodes inheriting Benders' cuts that are valid for their ancestor nodes. In addition, we develop two closely related convexification schemes which use multi-term disjunctive cuts to obtain approximations of the second stage mixed-integer programs. We prove that the proposed methods are finitely convergent. One of the main advantages of our decomposition scheme is that we use a Benders-based branch-and-cut approach in which linear programming approximations are strengthened sequentially. Moreover as in many decomposition schemes, these subproblems can be solved in parallel. We also illustrate these algorithms using several variants of an SMIP example from the literature, as well as a new set of test problems, which we refer to as Stochastic Server Location and Sizing. Finally, we present our computational experience with both classes of test problems, and demonstrate that under some restrictions on the growth of first stage decisions, the algorithm provides a realistic approach to solve SMIP models with a reasonable number of scenarios.

Keywords: Two-stage stochastic mixed-integer programs, cutting plane tree algorithm, multi-term disjunctive cut, Benders' decomposition.

1 Introduction

Stochastic mixed-integer programs (SMIPs) have long been recognized as an important class of models for many practical operational problems (see e.g. [Wollmer, 1980]). However, algorithmic advances to

*This work is supported in part by NSF-CMMI Grant 1100383 and AFOSR Grant FA9550-13-1-0015

†Integrated Systems Engineering, The Ohio State University,Columbus OH, 43210, USA

‡Epstein Department of Industrial and Systems Engineering, University of Southern California, Los Angeles CA, 90089-0193, USA

solve SMIP models have lagged behind other forms of stochastic programs. In addition to the standard difficulties associated with stochastic linear programming (e.g. designing scalable ways to approximate the expected recourse/value function), SMIP formulations with mixed-integer recourse decisions in the second stage encounter value functions that are possibly non-convex and discontinuous. Early work by [Carøe and Tind, 1998] presented a decomposition algorithm based on mixed integer programming (MIP) duality, and while it is conceptually applicable to SMIPs with general mixed-integer recourse decisions, the algorithm is not easily realizable because it requires calculations involving exact MIP value functions. Such value functions are not only difficult to construct in the second stage, but also lead to discontinuous and non-convex first stage approximations in general. Other decomposition algorithms, based on scenario decomposition, were proposed in [Carøe and Schultz, 1997] and [Lulli and Sen, 2004]. These algorithms essentially view the SMIP problem as a very large scale MIP using a deterministic equivalent formulation. The decomposition principles used by the above methods are dual to each other (price and resource directive decomposition respectively) and may be recommended for instances in which special structures associated with scenario subproblems can be exploited (as in unit-commitment models, lot-sizing models etc.). However, when the number of scenarios is very large, and special structures are either absent or difficult to exploit, such scenario decomposition methods are not particularly effective.

Subsequent to the work of [Carøe and Tind, 1998], most authors addressed some sub-class of the two-stage SMIP problem. For instance, the global optimization algorithm of [Ahmed et al., 2004] assumed fixed tenders (i.e. deterministic T matrix in (2) below) in the two-stage model. Others have addressed alternative sub-classes which either focus on mixed-binary recourse decisions, or pure integer recourse decisions. Decomposition-based cutting plane algorithms (e.g. [Sen and Higle, 2005], [Sherali and Zhu, 2006], [Ntaimo, 2010]), novel branch-and-bound methods ([Escudero et al., 2007]), decomposition based branch-and-cut ([Sen and Sherali, 2006]), as well as primal approaches using certain IP value function characterizations ([Kong et al., 2006], [Trapp et al., 2013]) now offer a range of algorithms for alternative model characteristics. As a consequence of the sharper focus, there has been significant progress with SMIP algorithms for specialized cases of SMIP. In contrast to some of the earlier multi-stage SMIP algorithms, these methods are based on time-staged decomposition, very much in the spirit of Benders' decomposition. We refer to surveys by ([Louveaux and Schultz, 2003]) and ([Sen, 2010]) for discussions related to these and related advances.

In this paper, we are interested in designing time-staged decomposition algorithms for solving two-stage SMIPs in which mixed-integer decisions appear in both stages. In other words, we return to the class of models addressed in [Carøe and Tind, 1998]. Fortunately, due to significant algorithmic advances in the interim, we are able to draw upon new approximations which not only ensure finite convergence, but also avoid intractable operations in each iteration. The first thought which comes to mind for addressing

problems with general integer variables is to replace the general integers by their binary expansion, thus, converting the general SMIP problem to one with mixed-binary variables. However, such transformations have already been discredited in the deterministic MIP literature where [Owen and Mehrotra, 2002] have shown, mathematically and computationally, that branch-and-bound methods examine many more nodes for the transformed problem, than for the original instance with general integer variables. Consequently, the issue of devising decomposition algorithms using branch and cut methods for SMIP remains an open question which we address in this paper. Perhaps, the most critical result that helps us resolve this open question is the recent constructive characterization due to [Chen et al., 2012] that the feasible set of a mixed-integer linear program can be convexified using a hierarchy of multi-term disjunctions. This result helps us design the ABC algorithm presented in this paper.

In order to state the problem, let $A \in \mathbb{R}^{m_1 \times n_1}, b \in \mathbb{R}^{m_1}$, and $X_L = \{x \in \mathbb{R}^{n_1} : Ax \leq b\}$. Then consider the SMIP as stated in (1) and (2).

$$\min_{x \in X \cap Q_1} c^{\mathsf{T}} x + \mathsf{E}[f(x, \tilde{\omega})] \tag{1}$$

where

$$\begin{aligned} X =& \{x \in X_L \mid x_i \text{ is integer}, \ \forall i \in I_2 \subseteq I_1 = \{1...n_1\}\} \subseteq \mathbb{R}^{n_1} \\ Q_1 =& \{x \mid l_1 \leq x \leq u_1\}. \end{aligned}$$

Also $\tilde{\omega}$ denotes a discrete random variable, and for each scenario (realization) ω of $\tilde{\omega}$, we define the recourse function by

$$\begin{aligned} f(x, \omega) = \min \quad & g(\omega)^{\top} y \\ \text{s.t.} \quad & W(\omega) y \geq r(\omega) - T(\omega) x \\ & y \in Y \cap Q_2 \end{aligned} \tag{2}$$

where

$$\begin{aligned} Y &= \{y \mid y_j \text{ is integer}, \ \forall j \in J_2 \subseteq J_1 = \{1...n_2\}\} \subseteq \mathbb{R}^{n_2} \\ Q_2 &= \{y \mid l_2 \leq y \leq u_2\}. \end{aligned}$$

For subproblem (2), we let the decision variable $y \in \mathbb{R}^{n_2}$, objective coefficient $g(\omega) \in \mathbb{R}^{n_2}$, constraint matrix $W(\omega) \in \mathbb{R}^{m_2 \times n_2}$, $r(\omega) \in \mathbb{R}^{m_2}$ and $T(\omega) \in \mathbb{R}^{m_2 \times n_1}$. In our notation, when $I_2(J_2)$ is a strict subset of $I_1(J_1)$, then stage 1 (stage 2) has both continuous as well as integer variables. We assume the random variable $\tilde{\omega}$ is discrete with each scenario ω having a non-zero probability $p(\omega)$, for all $\omega \in \Omega$.

The algorithms that we propose in this paper impose the following assumptions on the model.

A1 Both X and Y are assumed to be non-empty mixed-integer sets, and the integer variables in both stages are bounded.

A2 The random variable $\tilde{\omega}$ in the problem is discrete, with a finite number of scenarios, $\omega \in \Omega$, each with an associated non-zero probability of occurrence $p(\omega), \forall \omega \in \Omega$.

A3 For any $x \in X_L \cap Q_1$, the set defined by $\{y \mid W(\omega)y \geq r(\omega) - T(\omega)x, y \in Y \cap Q_2\}$ is feasible for all $\omega \in \Omega$.

Due to assumption **A2**, (1) can be rewritten as:

$$\min_{x \in X \cap Q_1} c^{\mathsf{T}} x + \sum_{\omega \in \Omega} p(\omega) f(x, \omega). \tag{3}$$

For the rest of the paper, we begin by first introducing the overall architecture in which a branch-and-bound (B&B) algorithm in the first stage controls how approximations are created and passed from one generation of nodes to another. Subsequently, we present two closely related convexification schemes for the second stage, one based on the cutting plane tree method ([Chen et al., 2012]), and the other referred to as a B&B -based convexification method. Both of these methods can be used to *approximate* the second stage value function $f(x, \omega)$. Finally, several variants of an SMIP example in the literature illustrate how our approach can be used under different assumptions (or structures) for SMIP models. In addition we present preliminary evidence that the decomposition framework promises to be more scalable than solving a deterministic equivalent problem (3) using a standard commercial MIP solver. We also present a new class of test instances which we refer to as Stochastic Server Location and Sizing. Computational experience with these instances demonstrates that under some restrictions on the growth of first stage decisions, the algorithm provides a realistic approach to solving SMIP models with many scenarios. Overall, our framework provides the most comprehensive time-staged decomposition approach to date, allowing randomness in all data elements, while also allowing general mixed-integer variables as decision variables in both stages.

2 An Algorithm with Ancestral Benders' Cuts

If we denote decision variables y under scenario ω as $y(\omega)$, the deterministic equivalent formulation (DEF) for (3) is

$$\min_{x,y(\omega)} \quad c^\top x + \sum_{\omega\in\Omega} p(\omega) g(\omega)^\top y(\omega) \tag{4a}$$

$$\text{s.t.} \quad T(\omega)x + W(\omega)y(\omega) \geq r(\omega),\ \forall \omega \in \Omega \tag{4b}$$

$$x \in X \cap Q_1, y(\omega) \in Y \cap Q_2,\ \forall \omega \in \Omega. \tag{4c}$$

The basic idea of the ABC algorithm is to use a branch-and-bound process to search for first stage decisions while carrying out a successive approximation scheme to update approximations of the expected recourse function $E[f(x,\tilde{\omega})]$ using manageable piecewise linear approximations as in Benders' decomposition. The process integrates the following ingredients which are also important for convergence.

a) The B&B search in the first stage divides the range of first stage integer variables into a disjoint subset covering the entire set Q_1. The partition is refined using a breadth-first search strategy. A partition of Q_1 in iteration k may consist of subsets $\{Q_1^t\}, t = 1, \ldots, N_k$

b) For any subset Q_1^t of the first stage decisions (x), we relax the integer restrictions and construct a polyhedral approximation of the convex hull of second stage mixed-integer points, together with the first stage decision variables (treated as continuous). The existence and construction of the polyhedral set follows the work of [Chen et al., 2011] on using multi-term disjunctions.

c) Given polyhedral approximations associated with every combination of subset (Q_1^t) and scenario (ω), we can construct an L-shaped structure in which both stages have continuous decision variables due to set convexification.

d) Using Benders' decomposition for each partition, we can then create lower bounding approximation for each partition. Using these bounds, one can proceed to partition the most promising node, and continue with the branch-and-bound search.

Clearly, the above ideas are fairly straightforward. As with most stochastic programming algorithms however, the effectiveness of any scheme depends on the ease with which approximations are updated and optimized, sequentially. In addition, implementing these processes requires us to assemble a variety of *concepts*, including disjunctive programming, L-shaped decomposition, and appropriate data structures. In this section, we provide a summary of these aspects of the ABC algorithm. In the following section,

we discuss multi-term disjunctions based on either the cutting plane tree algorithm [Chen et al., 2011] or a branch-and-bound algorithm.

In describing the algorithm, we use index k to denote an iteration index. A B&B tree for the first stage provides a partition of the bounding constraints denoted by Q_1. In iteration k, we let $t(k)$ denote a node of the first stage B&B tree such that the first stage solution $x^k \in Q_1^{t(k)}$, where $Q_1^{t(k)} \subseteq Q_1$ is the bounding constraints of node $t(k)$. Next define

$$\mathcal{Y}(Q_1^{t(k)}, \omega) \equiv \{(x, y) \mid T(\omega)x + W(\omega)y \geq r(\omega), x \in X_L \cap Q_1^{t(k)}, y \in Y \cap Q_2\}. \tag{5}$$

In addition to node $t(k)$'s bounds $Q_1^{t(k)}$, $\mathcal{Y}(Q_1^{t(k)}, \omega)$ covers all constraints for variables x and y. Note that the integrality restrictions on x have been removed in the set $\mathcal{Y}(Q_1^{t(k)}, \omega)$. However, we impose integrality of the second stage integer variables, and towards this end, we use the CPT algorithm ([Chen et al., 2011]) to construct a polyhedral approximation $\mathcal{Y}_L(Q_1^{t(k)}, \omega)$ such that

$$\mathcal{Y}_L(Q_1^{t(k)}, \omega) \supseteq \mathbf{conv}\{\mathcal{Y}(Q_1^{t(k)}, \omega)\}. \tag{6}$$

Since the approximations are in the space of variables $(x, y(\omega))$, the polyhedron $\mathcal{Y}_L$ has valid inequalities appended to the linear inequalities in (5). That is,

$$\begin{aligned}\mathcal{Y}_L(Q_1^{t(k)}, \omega) = \{(x, y) \mid & T(\omega)x + W(\omega)y \geq r(\omega), x \in X_L \cap Q_1^{t(k)}, y \in Q_2, \\ & \Pi_1^{t(k)}(\omega)x + \Pi_2^{t(k)}(\omega)y \geq \Pi_0^{t(k)}(\omega)\}\end{aligned} \tag{7}$$

Since these inequalities are valid for the entire set $Q_1^{t(k)}$, they can be re-used for any of its subsets. And moreover, we can define a lower bounding approximation of the second stage objective by letting x vary in the following approximation $f_L^k(Q_1^{t(k)}, \omega)$ as

$$f_L^k(Q_1^{t(k)}, \omega) = \min\{g(\omega)^\top y : \ (x, y) \in \mathcal{Y}_L(Q_1^{t(k)}, \omega)\}. \tag{8}$$

Now note that if one were to use Benders' decomposition to solve $\min\{c^\top x + \mathsf{E}[f_L^k(Q_1^{t(k)}, \tilde{\omega})] : x \in X_L \cap Q_1^{t(k)}\}$ then one would exploit the L-shaped structure of the SMIP problem. This is the basis of the ABC algorithm. Note that the strength of Benders' cuts depend on the strength of the polyhedral approximation, and the strongest Benders' cut would result from situations in which the valid inequalities are strong enough to generate second stage mixed-integer solutions for all outcomes.

To make this discussion more precise, let $W^k(\omega)$ denote a matrix consisting of the original matrix $W(\omega)$ together with the entire collection of cuts generated for scenario ω, and let $W^{t(k)}(\omega)$ denote the

subset of inequalities in the matrix $W^k(\omega)$ which are valid for $x \in Q_1^{t(k)}$. Thus, we put $W^{t(k)}(\omega) = \begin{bmatrix} W(\omega) \\ \Pi_2^{t(k)}(\omega) \end{bmatrix}$, and similarly, $r^{t(k)}(\omega) = \begin{bmatrix} r(\omega) \\ \Pi_0^{t(k)}(\omega) \end{bmatrix}$, $T^{t(k)}(\omega) = \begin{bmatrix} T(\omega) \\ \Pi_1^{t(k)}(\omega) \end{bmatrix}$. Then a relaxation of the integer recourse (or value) function can be obtained as in Benders' decomposition by using the dual multipliers of the following cut-enhanced LP relaxation, for any choice of $x \in Q_1^{t(k)}$.

$$f_L^k(x,\omega) = \min \quad g(\omega)^\top y \tag{9a}$$

$$\text{s.t.} \quad W^{t(k)}(\omega)y \geq r^{t(k)}(\omega) - T^{t(k)}(\omega)x \tag{9b}$$

$$y \geq 0. \tag{9c}$$

Since we are going to be deriving Benders' cuts, we have replaced the first argument of f_L^k by the first stage variable x, rather than the set $Q_1^{t(k)}$ in (8) because validity of Benders' cuts are implied by LP duality for all $x \in Q_1^{t(k)}$. Unlike standard Benders' decomposition (which is normally applied to the case where the second stage is a linear program), the subproblem stated above is a relaxation of the MIP subproblem. Let $y^k(\omega)$ denote the optimal solution for solving (9) and let $\Theta^{t(k)}(\omega)$ denote the optimal dual multipliers associated with (9). Then the Benders' approximation from (9) is given as

$$\mathsf{E}[f_L^k(x,\tilde{\omega})] \geq \xi_k - {\zeta_k}^\top x \text{ for } x \in Q_1^{t(k)} \tag{10}$$

where

$$\begin{aligned} \xi_k &= \sum_{\omega\in\Omega} p(\omega)\Theta^{t(k)}(\omega)^\top r^{t(k)}(\omega) \\ \zeta_k &= \sum_{\omega\in\Omega} p(\omega)\Theta^{t(k)}(\omega)^\top T^{t(k)}(\omega). \end{aligned} \tag{11}$$

Recall that Benders' cuts generated for Q_1^t can be used for all subsets $Q_1' \subseteq Q_1^t$. Similar inheritance also happens for second stage approximation (9b) or more specifically the cuts in (7). These cuts, generated for a particular first stage B&B node $t(k)$ can also be used for descendant nodes of $t(k)$. So we use a binary sequence denoted as $\mathcal{G}^t = [\mathcal{G}_x^t, \mathcal{G}_y^t]$ to encode valid cuts for node t where $\mathcal{G}_x^t$ is for Benders' cuts and $\mathcal{G}_y^t$ is for second stage approximation cuts (i.e. CPT cuts). Each bit of the sequence can only have value 1 or 0, where "1" indicates that the cut is valid for node t, and "0" indicates that it is not. The initial sequence for node t is the same as its parent node. When a new valid cut is derived for node t, one extends either $\mathcal{G}_x^t$ or $\mathcal{G}_y^t$ depending on whether it is a Benders' cut or a second stage approximation cut. The extended code is set to 1 for node t (and descendants), whereas all other nodes have an extended sequence with its corresponding bit set to 0.

We now proceed to discuss the B&B method for the first stage. Suppose the set of active (unfathomed) nodes for the first stage is denoted as $\mathcal{T}_1^k$, let Q_1^t denote the bounding constraints for $t \in \mathcal{T}_1$, and let $\mathcal{G}_x^t[s]$ denote the bit value for s-th Benders' cut. Then the lower bounding master problem for node t at iteration k is as follows,

$$\min \quad c^\top x + \eta_t \tag{12a}$$

$$\text{s.t.} \quad \eta_t \geq \xi_s - \zeta_s x, \text{for all } s \text{ such that } \mathcal{G}_x^t[s] = 1 \tag{12b}$$

$$\eta_t \geq -M, \tag{12c}$$

$$x \in X_L \cap Q_1^t, \tag{12d}$$

where $-M$ is a valid lower bound on second stage value function.

The entire algorithm starts from iteration $k = 1$ with $\mathcal{T}_1^k = \{o\}$ where the root node o has bounding constraint $Q_1^o \equiv Q_1$ and we initialize $\mathcal{G}^o$ to be NULL. Let the optimal objective value for (12) be denoted by v^t and let x^t denote the optimal solution. Thus, the global lower bound $v \equiv \min_{t \in \mathcal{T}_1^k}\{v^t\}$. Also denote the upperbound of optimal objective value for (1) by V. At iteration k, problem (12) is solved for node $t(k-1)$ with updated (12b) from iteration $k-1$ and for our breadth-first approach, we choose the node with the least lower bound as the node to branch on. Suppose this node is $\bar{t}$ and its optimal solution is $x^{\bar{t}}$. Following a variable selection rule such as choosing the fractional variable with smallest index or the most relative fractional variable as shown in (13) and (14),

$$\theta_i = \min\{x_i^{\bar{t}} - l_{1i}^{\bar{t}}, u_{1i}^{\bar{t}} - x_i^{\bar{t}}\} \tag{13}$$

$$p \in \text{argmax}_{i=1,\ldots,n}\{\frac{\theta_i}{u_{1i}^{\bar{t}} - l_{1i}^{\bar{t}}}\}, \tag{14}$$

we can select variable x_p and split its bounding constraint $[l_{1p}, u_{1p}]$ as shown in (15).

$$[l_{1p}, \lfloor x_p \rfloor] \text{ and } [\lceil x_p \rceil, u_{1p}], \text{ if } p \in I_2 \tag{15}$$

The sequence $\mathcal{G}^{\bar{t}}$ which contains Benders' cut (12b) for $\bar{t}$, is now used to initialize codes for two new nodes (denoted as $\bar{t}^1, \bar{t}^2$). With $\mathcal{T}_1^k$ updated,

$$\mathcal{T}_1^k \leftarrow \{\mathcal{T}_1^k \setminus \bar{t}\} \cup \{\bar{t}^1, \bar{t}^2\} \tag{16}$$

the node with least lower bound among $\mathcal{T}_1^k$ is selected again and this process repeats until the mixed-integer optimum is found.

When a solution (denoted as x^k) to the master problem is found, we identify the first stage node to which it belongs, and refer to it as $t(k)$ and its bounding constraint as $Q_1^{t(k)}$. Given x^k and $Q_1^{t(k)}$, we approximate the second stage value function for all ω as described in (10). With value function encoded in $\mathcal{G}^{t(k)}$ updated, the master problem continues to find new integer solutions and updating nodal value functions until the node is fathomed, or the algorithm stops. A summary of the prescribed process is shown in Algorithm 1, and we refer to it as the **ABC** (for Ancestral Bender's Cut) algorithm.

Algorithm 1 Ancestral Benders' Cut (ABC) Algorithm

Initialize: Iteration $k \leftarrow 1$, objective value upper bound $V \leftarrow \infty$, first stage active nodes $\mathcal{T}_1^k \leftarrow \{o\}$ with $Q_1^o \equiv \{x | l_1 \leq x \leq u_1\}$. $\mathcal{G}^t \leftarrow \emptyset$. Let ϵ denote the stopping tolerance and $(x^*, y^*(\omega))$ for $\omega \in \Omega$ the incumbent solutions. Solve problem (12) at node o and get its objective value v^o, and $\bar{t}(k) \leftarrow o$
while true **do**
  Update $v^{t(k)}$ by re-solving problem (12) for node $t(k)$ with updated $\mathcal{G}^{t(k)}$.
  while true **do**
    Update the global lower bound:

$$v \leftarrow \min_{t \in \mathcal{T}_1^k} \{v^t\}.$$

    Denote the node that has global lower bound as $\bar{t}$
    if $x_i^{\bar{t}}$ is integer for $\forall i \in I_2$ **then** $t(k) \leftarrow \bar{t}$, **break**.
    end if
    Choose variables to split and add two new child nodes $\bar{t}^1$,$\bar{t}^2$ of node $\bar{t}$ as in (15).
    Solve problem (12) for both new nodes and obtain $v^{\bar{t}^1}$ and $v^{\bar{t}^2}$.
  end while
  if $V - v \leq \epsilon$ **then STOP**.
  end if
  Derive (10) and update $\mathcal{G}^{t(k)}$.
  Update V and incumbent $(x^*, y^*(\omega))$ if $y^k(\omega)$ satisfies mixed-integer restrictions for all $\omega \in \Omega$.
  Fathom nodes for which $v^t \geq V$. Update $\mathcal{T}_1^k$: $\mathcal{T}_1^k \leftarrow \mathcal{T}_1^k \setminus \{t \mid t \in \mathcal{T}_1^k, V - v^t \geq \epsilon\}$.
  $k \leftarrow k + 1$
end while

Proposition 1. *Let assumptions **A1**-**A3** hold. Assume that the process of deriving (10) is finite, and let x^k denote a first stage solution such that $x^k \in \{\arg\min c^\top x + E[f_L^k(x, \omega)] : x \in X_L \cap Q_1^{t(k)}\}$. Let V^k denote an upper bound on the optimum at iteration k. Suppose that for any t, there exists a finite iteration $K(t)$ such that for $k \geq K(t)$ we either have $v^{t(k)} \geq V^k$ or $f_L^k(x^k, \omega) = f(x^k, \omega)$. Then, the B&B procedure produces an optimal solution x^*, as well as its optimal value V^* in finitely many iterations.*

Proof. Since the range of first stage variables is finite, there can only be finitely many partitions. In addition, those sets that are fathomed (either because of feasibility or because the lower bound exceeds an upper bound) satisfy $c^\top x + E[f_L^k(x, \omega)] \geq V^k \geq V^*$, for all x belonging to fathomed subsets of Q_1. Consider $k \geq K(t)$ as defined in the statement of the Proposition. Then for such k, any partition can only have $f_L^k(x^t, \omega) < f(x^t, \omega)$ for finitely many iterations. Consequently after finitely many iterations, $v^t = c^\top x^t + E[f_L^k(x^t, \omega)] = c^\top x^t + E[f(x^t, \omega)]$. Because the B&B process must ultimately satisfy the MIP restrictions of the first stage, the previous equation actually implies that $v^t = c^\top x^t + E[f_L^k(x^t, \omega)] =$

$c^\top x^t + E[f(x^t, \omega)] \geq V^*$ for all those t for which x^t satisfy the first stage MIP restrictions. Hence, we must have at least one index t for which $v^t = V^*$, implying that $x^t \equiv x^*$, which implies that the **ABC algorithm** terminates in finitely many steps. □

In the following section, we show how one satisfies the requirement in the above proposition that objective function evaluations should become more accurate as the search descends into the leaf nodes of the B&B tree.

3 Approximations using Multi-term Disjunctions

This section presents two approaches for approximating the second stage value function. Both methods convexify the set of feasible solutions of the second stage. However, one of these is based on a pure cutting plane approach, while the other performs a convexification prompted by a B&B tree.

3.1 Convexification using Pure Cutting Planes

Until recently, pure cutting plane algorithms were not known to provide finitely convergent algorithms for MIP problems with general mixed-integer variables. For example, ([Balas, 1979], [Owen and Mehrotra, 2001]) have shown that for general MIP, traditional two-term disjunctive cuts (e.g. [Balas et al., 1993]) are inadequate to the task of satisfying the conditions of Proposition 1. In a recent paper, [Chen et al., 2011] present two algorithms for constructing polyhedral approximations which provide the same solution as the original MIP in finitely many iterations. One of the algorithms, the convex hull tree algorithm, essentially provides a proof technique, and is not recommended as a solution methodology. The other algorithm, the cutting plane tree (CPT) algorithm, is a sequential cutting plane scheme which generates a finite sequence of cuts which, in the worst case, verifies that the polyhedral approximation provides a solution which is the same as the MIP solution. This method may be summarized as one that generates cutting planes from a hierarchy of multi-term disjunctions which are recorded in the form of nodes of a search tree. So long as each visit to a node (i.e. a multi-term disjunction) generates one facet of the disjunction, and nodes can only be visited finitely many times, finiteness of the number of multi-term disjunctions implies that the CPT algorithm converges in finitely many iterations. We refer the reader to [Chen et al., 2011] for the details, and [Chen et al., 2012] for computational results for deterministic MIP problems. Another result of similar type has been proposed by [Jörg, 2007], although its computability is not clear as of this writing. In any event, the method of [Chen et al., 2011] is able to deliver the property required by Proposition 1. Because these multi-term disjunctions are intimately tied to the CPT algorithm, we refer to the resulting cuts as "CPT cuts".

Suppose that at the k-th iteration of **ABC** algorithm, we have a fixed $x^k \in X_L \cap Q_1^{t(k)}$ and are given an initial approximation $f_L^k(x,\omega)$ from previous iterations. We seek to approximate $f(x,\omega)$ further using x^k and $f_L^k(x,\omega)$ by executing the CPT algorithm for a few iterations. Note that this approximation is only valid for $x \in X \cap Q_1^{t(k)}$. To initialize a sequence of subproblem approximations of the CPT algorithm, we start by solving (9). Let d denote the iteration counter of the CPT algorithm (for the second stage), and let $y^d(\omega)$ denote a solution with some fractional variable(s) at iteration d. Let $\mathcal{T}_2^d$ denote an index set of the sets that constitute a partition $\{Q_2^t, t \in \mathcal{T}_2^d\}$ of Q_2. For each $t \in \mathcal{T}_2^d$, let Q_2^t denote bounding constraints for the vectors y as shown in (17).

$$Q_2^t = \{y \mid l_2^t \leq y \leq u_2^t\}, \quad \forall t \in \mathcal{T}_2^d \tag{17}$$

The tree $\mathcal{T}_2^d$ embodies a disjunctive relaxation of $Y \cap Q_2$ because we require the following condition to hold,

$$\bigcup_{t \in \mathcal{T}_2^d} (Y_L \bigcap Q_2^t) \supseteq Y \cap Q_2. \tag{18}$$

Since $y^d(\omega)$ is presumed to have some fractional values for integer variables, we delete the point $(x^k, y^d(\omega))$ from the higher dimensional polyhedron. Thus, we construct a disjunctive set which satisfies the following,

$$(x^k, y^d(\omega)) \notin \bigcup_{t \in \mathcal{T}_2^d} \{(x, y(\omega)) \mid x \in X_L \cap Q_1^{t(k)}, y(\omega) \in Y_L \cap Q_2^t\}. \tag{19}$$

We use the same rules as in the CPT algorithm [Chen et al., 2012] to construct the disjunctive sets. Moreover, the hierarchy of disjunctions represented in $\mathcal{T}_2^d$ is maintained by a tree structure (called cutting plane tree) and $\mathcal{T}_2^d$ contains all nodes that do not have children nodes. The tree itself is initialized with one global node defining the constraints for Q_2. If the solution $y^d(\omega)$ does not satisfy the mixed-integer restrictions, then the algorithm walks through the cutting plane tree to locate the deepest node from the root node that contains $y^d(\omega)$. Let us refer to this node as t_2^d. If $t_2^d \in \mathcal{T}_2^d$ (meaning it does not have any children node), two nodes are created as its children nodes and t_2^d is removed from $\mathcal{T}_2^d$. On the other hand, if $t_2^d \notin \mathcal{T}_2^d$, no new node is created. In this way, $\mathcal{T}_2^d$ is updated such that conditions (18) and (19) are satisfied (see also Algorithm 2).

Assuming that (19) is satisfied, we use a cut generation linear program (CGLP) to derive a valid inequality. Using the partition (17), the CGLP shown in (20) can be formulated to derive multi-term

disjunctive cuts.

$$\max \quad \pi_0(\omega) - \pi_1(\omega)^\top x^k - \pi_2(\omega)^\top y^d(\omega) \tag{20a}$$

$$\text{s.t.} \quad \pi_{2j}(\omega) = {W_j^{t(k)}(\omega)}^\top \lambda_{2t} + \mu_{2jt} - \nu_{2jt} \quad \forall j \in J, t \in \mathcal{T}_2^d \tag{20b}$$

$$\pi_{1i}(\omega) = {T_i^{t(k)}(\omega)}^\top \lambda_{2t} + {A_i}^\top \lambda_1 + \mu_{1i} - \nu_{1i} \quad \forall i \in I, t \in \mathcal{T}_2^d \tag{20c}$$

$$r^{t(k)}(\omega)^\top \lambda_{2t} + b^\top \lambda_1 + {l_2^t}^\top \mu_{2t} + {l_1^{t(k)}}^\top \mu_1 - {u_2^t}^\top \nu_{2t} - {u_1^{t(k)}}^\top \nu_1 \geq \pi_0(\omega) \quad t \in \mathcal{T}_2^d \tag{20d}$$

$$-1 \leq \pi_{1i}(\omega) \leq 1, \quad \forall i \in I, \quad -1 \leq \pi_{2j}(\omega) \leq 1, \quad \forall j \in J \tag{20e}$$

$$-1 \leq \pi_0(\omega) \leq 1 \tag{20f}$$

$$\lambda_1, \lambda_{2t} \geq 0, \nu_1, \nu_{2t} \geq 0, \mu_1, \mu_{2t} \geq 0, \text{for } \forall\ t \in \mathcal{T}_2^d. \tag{20g}$$

The solution to the above CGLP provides the cut shown in (21).

$$\pi_1(\omega)^\top x + \pi_2(\omega)^\top y(\omega) \geq \pi_0(\omega). \tag{21}$$

Note that (21) is associated with scenario ω, and we derive separate cuts for each $\omega \in \Omega$. Since (19) includes inequalities that are valid for $X \cap Q_1^{t(k)}$, this CGLP combines the convexification in the space $(x, y(\omega))$ where $x \in Q_1^{t(k)}$. As a result, the CGLP suggested in this paper is larger than that used for the original D^2 algorithm ([Sen and Higle, 2005]). However, the **ABC** algorithm does not require the convexification using the epi-reverse polar used in ([Sen and Higle, 2005]). In this sense, the cuts used in the **ABC** algorithm are different from those in the D^2 algorithm.

Proposition 2. *Cutting plane (21) is valid for feasible set* $\{(x, y(\omega)) \mid T^{t(k)}(\omega)x + W^{t(k)}(\omega)y(\omega) \geq r^{t(k)}(\omega), x \in X \cap Q_1^{t(k)}, y(\omega) \in Y \cap Q_2\}$.

Proof. For any $\{(x, y(\omega)) \mid T^{t(k)}(\omega)x + W^{t(k)}(\omega)y(\omega) \geq r^{t(k)}(\omega), x \in X \cap Q_1^{t(k)}, y(\omega) \in Y \cap Q_2\}$, (20b,c,d) imply that

$$\pi_1(\omega)^\top x + \pi_2(\omega)^\top y(\omega) = (\lambda_{2t}^\top T^{t(k)}(\omega)x - \lambda_1^\top Ax + \mu_1^\top x - \nu_1^\top x) + (\lambda_{2t}^\top W^{t(k)}(\omega)y(\omega) + \mu_{2t}^\top y(\omega) - \nu_{2t}^\top y(\omega)) \tag{22a}$$

$$\geq r^{t(k)}(\omega)^\top \lambda_{2t} - b^\top \lambda_1 + {l_2^t}^\top \mu_{2t} + {l_1^{t(k)}}^\top \mu_1 - {u_2^t}^\top \nu_{2t} - {u_1^{t(k)}}^\top \nu_1 \tag{22b}$$

$$\geq \pi_0(\omega), \tag{22c}$$

as required. □

The cut (21) is included in the second stage formulation, leading to a stronger approximation of the second stage polyhedron, and consequently, a stronger approximation of the value function, which is denoted as $f_L^{k,d}(x,\omega)$. In general, after d iterations of the CPT algorithm for the subproblem, we have

$$f_L^{k,d}(x,\omega) = \min \quad g(\omega)^{\mathsf{T}} y \tag{23a}$$

$$\text{s.t.} \quad W^{t(k)^-}(\omega) y \geq r^{t(k)^-}(\omega) - T^{t(k)^-}(\omega) x \tag{23b}$$

$$\Pi_2^d(\omega) y \geq \Pi_0^d(\omega) - \Pi_1^d(\omega) x \tag{23c}$$

$$y \in Y_L \cap Q_2 \tag{23d}$$

$$Y_L = \{y \geq 0, y \in \mathbb{R}^{n_2}\} \tag{23e}$$

where constraint (23b) denotes the approximation of subproblem for $x \in Q_1^{t(k)}$ before starting iteration k. The superscript $t(k)-$ denotes the most recent update of any particular data element (e.g. W,r etc.). Note that this may not be the same as the index $t(k-1)$ because the latter may point to a completely different subset. In addition, (23c) denotes all the cuts generated during this round of iterations for solving/approximating the subproblem. There is another algorithmic point to be noticed here; (23) is essentially the same as (9), but the left hand side of (23a) includes two superscripts (k,d) while the function in (9a) has only one superscript. The reason for this distinction is to emphasize that there are "outer iterations" (first stage) indexed by k, whereas, the "inner iterations" (second stage) are indexed by d. Depending on the course of the CPT algorithm, (23c) needs to be included in the CGLP to ensure convergence of the algorithm ([Chen et al., 2011]). After the approximation $f_L^{k,d}(x,\omega)$ is obtained, the cut-enhanced LP is re-solved and new cuts are generated as long as the inner iteration generates y^d that are fractional. At any outer iteration k, we allow at most D cuts to be added for the second stage CPT (approximation) algorithm. Once the process of solving the subproblem stops, we form a Benders' cut as in (10) and return to the master problem. During the implementation, we use the binary sequence $\mathcal{G}_y^{t(k),\omega}$ to keep track of all CPT cuts generated for scenario ω with the first stage solutions from $Q_1^{t(k)}$ as well as cuts inherited from $t(k)$'s ancestor nodes. Cuts generated from first stage solutions x that do not belong to $Q_1^{t(k)}$ or one of its ancestors or for other scenarios are not included in (23b) by setting $\mathcal{G}_y^{t(k),\omega}[s] = 0$. Given $x^k \in X_L \cap Q_1^{t(k)}$, the algorithm to solve subproblems ω with at most D cuts added is shown as Algorithm 2.

Proposition 3. *Assume that the second stage problems are all bounded, and moreover, suppose that the cuts coefficients (21) correspond to extreme point solutions of the CGLP (20). Then for fixed $t(k)$ there exists a finite integer $D < \infty$, such that algorithm **CPT-D** solves all subproblems (2) indexed by ω to mixed-integer optimality; that is, with $x = x^k$ we have $f_L^{k,D}(x^k,\omega) = f(x^k,\omega)$ for $\forall\omega$.*

Algorithm 2 CPT-D

Initialize $d \leftarrow 1$, set CPT tree leaves set $\mathcal{T}_2^d(\omega) \leftarrow \{o\}$ where o is the root node with bounding constraint $Q_2^o \leftarrow Q_2$. Populate $W^{t(k)^-}(\omega)$, $r^{t(k)^-}(\omega)$, $T^{t(k)^-}(\omega)$ with valid cuts based on $\mathcal{G}_y^{t(k),\omega}$.
while $d \leq D$ **do**
  Evaluate $f_L^{k,d}(x,\omega)$ and get $y^d(\omega)$.
  if $y^d(\omega)$ satisfies mixed-integer restrictions **then**, STOP and $y^k(\omega) \leftarrow y^d(\omega)$
  else
    Find the deepest node σ that contains $y^d(\omega)$ in $\mathcal{T}_2^d$.
    if σ is a leaf node **then**,
      Make splits on σ and use updated $\mathcal{T}_2^d$ and first stage node bounds $Q_1^{t(k)}$ to formulate and solve (20) and obtain cut (21).
    else
      No splits are needed. Use σ and leaf nodes that in the subtree of σ and $Q_1^{t(k)}$ to formulate and solve (20) and obtain cut (21).
    end if
    Update $\Pi_1^d(\omega)$, $\Pi_2^d(\omega)$, $\Pi_0^d(\omega)$ with the new cut.
  end if
  $d \leftarrow d+1$
end while
Extend $\mathcal{G}_y^{t(k),\omega}$ to encode newly generated cuts.

Proof. When D is large enough, algorithm **CPT-D** is the same as using CPT algorithm to solve each subproblem. Due to the finiteness of the CPT algorithm proved in [Chen et al., 2011], all we need to prove is that for fixed $\mathcal{T}_2^k$ and $Q_1^{t(k)}$, only finitely many constraints can be generated from the CGLP (20) as $(x^k, y^d(\omega))$ changes. To see this, note that the constraints in (20) depend on the disjunction that is violated, but not the specific values of $(x^k, y^d(\omega))$. Moreover, there are only finitely many disjunctions that can be violated. Finally, with the generation of each cut, there is one less extreme point of the CGLP that can be generated. Hence, the finiteness of the number of disjunctions, and the finiteness of the number of extreme points of (22) implies that in the worst case, all extreme points are generated. This completes the proof. □

In their study, [Chen et al., 2012] shows that a 1-norm formulation (24) provides better numerical stability than the CGLP in (20). Therefore, our computational experiments use (24) as the CGLP to

generate disjunctive cuts.

$$\min \quad \sum_i |\pi_{1i}(\omega)| + \sum_j |\pi_{2j}(\omega)| \tag{24a}$$

$$\text{s.t.} \quad \pi_{1i}(\omega) = {T_i^{t(k)}(\omega)}^\top \lambda_{2t} - {A_i}^\top \lambda_1 + \mu_{1i} - \nu_{1i} \quad \forall i \in I, t \in \mathcal{T}_2^d \tag{24b}$$

$$\pi_{2j}(\omega) = {W_j^{t(k)}(\omega)}^\top \lambda_{2t} + \mu_{2jt} - \nu_{2jt} \quad \forall j \in J, t \in \mathcal{T}_2^d \tag{24c}$$

$$r^{t(k)}(\omega)^\top \lambda_{2t} - b^\top \lambda_1 + {l_2^t}^\top \mu_{2t} + {l_1^{t(k)}}^\top \mu_1 - {u_2^t}^\top \nu_{2t} - {u_1^{t(k)}}^\top \nu_1$$

$$\geq \pi_1(\omega)^\top x^k + \pi_2(\omega)^\top y^d(\omega) + 1 \quad t \in \mathcal{T}_2^d \tag{24d}$$

$$\lambda_1, \lambda_{2t} \geq 0, \nu_1, \nu_{2t} \geq 0, \mu_1, \mu_{2t} \geq 0, \text{for } \forall\ t \in \mathcal{T}_2^d. \tag{24e}$$

If one wishes to implement (24) and still ensure finiteness of the algorithm, one should observe that every feasible solution in (24) can mapped to a point in (20). If the mapping reveals an extreme point of (20), then, finiteness is ensured. However, if the solution from (24) does not map to an extreme point of (20), then, one should identify the lowest dimensional face of (20) which contains the mapped point, and then one should optimize (20) by restricting the LP to that face. Such a modification would also ensure finiteness of the cut generation scheme.

In addition, we gradually increase the number of cuts we generate during any outer iteration k. To do so, we initialize an integer $D \leftarrow 2$ and at each outer iteration, we increment $D \leftarrow D + 2$. This style of implementation is motivated by our prior experience (e.g. [Yuan and Sen, 2009]) that seeking very accurate objective function estimates requires much more computational resources than can be justified in early (outer) iterations of the algorithm; in later iterations however, seeking greater accuracy tends to pay off.

3.2 Convexification implied by Branch-and-Bound

Most deterministic MIP algorithms combine valid inequalities in the context of Branch-and-Bound (B&B) methods, and as a result, Branch-and-Cut methods form the backbone for most state-of-the-art commercial solvers. The CPT algorithm of the previous section is a pure cutting plane method. The fact that it also utilizes a tree structure to manage the disjunctive sets inspires a way that transforms the B&B tree obtained from an MILP solver to help create a polyhedral approximation that gives the same IP optimal value as obtained from the B&B method. For the remainder of this section, we describe such an algorithm and prove that this approximation can also be obtained in finitely many steps. It turns out that this combination (of B&B with valid inequalities) extends the branch-and-cut approaches of [Sen and Sherali, 2006], and [Yuan and Sen, 2009].

Suppose for fixed $x^k \in X_L \cap Q_1^{t(k)}$, subproblem $f(x^k, \omega)$ is either solved to optimality by a B&B method, or a truncated B&B process is carried out until a node limit or a time limit is reached. Let the optimal/incumbent solution be denoted $y^k(\omega)$. Due to the B&B (or truncated B&B) process, it is reasonable to assume that we have a set of leaf nodes. Let $\mathcal{T}_2^{\text{remain}}$ denote the remaining leaf nodes in the B&B tree and $\mathcal{T}_2^{\text{fathom}}$ denote the leaf nodes that have been fathomed. Then, we define $\mathcal{T}_2 = \mathcal{T}_2^{\text{remain}} \cup \mathcal{T}_2^{\text{fathom}}$. Suppose the constraint set used in the calculation of $f_L(Q_1^{t(k)}, \omega)$ is given by (see also (23))

$$\mathcal{Y}_L(Q_1^{t(k)}, \omega) = \{(x, y(\omega)) | T^{t(k)-}(\omega)x + W^{t(k)-}(\omega)y(\omega) \geq r^{t(k)-}(\omega), x \in X_L \cap Q_1^{t(k)}, y(\omega) \in Q_2\} \quad (25)$$

and define

$$\mathcal{Y}(Q_1^{t(k)}, \omega) = \mathcal{Y}_L(Q_1^{t(k)}, \omega) \cap Y \text{ and } \mathcal{Y}_D(Q_1^{t(k)}, \omega) = \bigcup_{t \in \mathcal{T}_2} (\mathcal{Y}_L(Q_1^{t(k)}, \omega) \bigcap Q_2^t). \quad (26)$$

Since $Q_2^t, \ \forall \ t \in \mathcal{T}_2$ are disjoint from each other, $\mathcal{T}_2$ provides us a disjunctive relaxation in the space of $(x, y(\omega))$

$$X \times \mathcal{Y}_D(x^k, \omega) = X \times \bigcup_{t \in \mathcal{T}_2} (\mathcal{Y}_L(x^k, \omega) \bigcap Q_2^t). \quad (27)$$

Thus, the same form of CGLP as in (20) can be used to derive multi-term disjunctive cuts with $\mathcal{T}_2$ replacing $\mathcal{T}_2^k$ in the formulation. The rest of algorithm is similar to **CPT-D**. There are two phases: the first phase is to use a B&B method to either solve the second stage MIP, or a truncated B&B process discussed earlier. In either case, we have $\mathcal{T}_2$ which is used to obtain a disjunctive approximation. The second phase starts by seeking the value $f_L^{k,d}(x^k, \omega)$ with $d = 1$. At iteration d, if the optimal solution of $f_L^{k,d}(x^k, \omega)$ is fractional (denoted as $y^d(\omega)$), (20) is formulated based on $\mathcal{T}_2$ and $Q_1^{t(k)}$ to cut off $y^d(\omega)$. The new cut is encoded into $\mathcal{G}_y^{t(k)}$. Then $f_L^{k,d}(x^k, \omega)$ is re-evaluated and this process continues until $y^d(\omega)$ belongs to $\mathcal{Y}_D$. The method is shown in Algorithm 3. While the form of the **BB-D** process is similar to the **CPT-D** process presented in the previous section, the inclusion of B&B, especially its truncated version, makes this decomposition approach much more realistic for practical instances of MIP in the second stage. Nevertheless, the proof of convergence derives from the same concept that one can obtain a polyhedral approximation of a disjunctive set embodied by a B&B tree. This is summarized in the following proposition.

Proposition 4. *Under the same assumptions as in Proposition 3, Algorithm 3 terminates in finitely many steps. When D is sufficiently large so that the B&B process provides an optimal second stage solution, there exists $d < \infty$ such that for fixed $t(k)$, $f(x^k, \omega) = f_L^{k,d}(x^k, \omega)$. Here k is the iteration*

Algorithm 3 BB-D

Initialize iteration $d \leftarrow 1$.
Phase 1:
Solve subproblem: Evaluate $f^k(x, \omega)$ by using a B&B method with $x = x^k$ for D iterations and get leaf nodes set $\mathcal{T}_2$ and solution $y^*(\omega)$.
Phase 2:
while true **do**
  Evaluate $f_L^{k,d}(x, \omega)$ and get $y^d(\omega)$.
  if $y^d(\omega) \in \mathcal{Y}_D$ **then**, STOP and $y^k(\omega) \leftarrow y^d(\omega)$
  else
    Use $\mathcal{T}_2$ and Q_1^t to formulate and solve (20) and obtain cut (21).
    Update $\Pi_1^d(\omega)$, $\Pi_2^d(\omega)$, $\Pi_0^d(\omega)$ with the new cut.
  end if
  $d \leftarrow d + 1$
end while
Extend $\mathcal{G}_y^{t(k),\omega}$ to encode newly generated cuts.

counter for the first stage, and d is the iteration counter for the second stage.

Proof. The proof here is similar to Proposition 3. □

Algorithm 2 and Algorithm 3 both use multi-term disjunctive cuts to obtain value function approximation. The difference between the two algorithms is the way to construct disjunctive sets. Algorithm 3 uses the B&B nodes to construct the disjunctive set, whereas, Algorithm 2 iteratively builds up the disjunctive set.

4 Illustrative Examples

This section illustrates the workings of the algorithms of this paper by starting from an example which only includes binary variables in both stages, and serves as a benchmark for its further extensions solved later. Figures illustrating the progress of the first stage search are provided in Appendix.

Example 1.0. Consider the following example with binary recourse variables from [Sen et al., 2003] which is a variation of an example that first appeared in ([Schultz et al., 1998]) and has been used by several authors, under alternative problem structures (such as fixed recourse ([Carøe and Schultz, 1998]) and fixed tenders ([Ahmed et al., 2004])).

$$\min \quad -1.5x_1 - 4x_2 + \sum_{\omega \in \Omega} p(\omega) f(x, \omega) \tag{28a}$$

$$\text{s.t.} \quad x_1, x_2 \text{ binary} \tag{28b}$$

where

$$f(x,\omega) = \min \quad -16y_1 - 19y_2 - 23y_3 - 28y_4 + 100R \tag{29a}$$

$$\text{s.t.} \quad \begin{bmatrix} 2y_1 + 3y_2 + 4y_3 + 5y_4 - R \\ 6y_1 + 1y_2 + 3y_3 + 2y_4 - R \end{bmatrix} \leq r(\omega) - T(\omega)x \tag{29b}$$

$$y_i \text{ binary } i = 1, ..., 4, R \geq 0, \tag{29c}$$

$\Omega = \{\omega_1, \omega_2\}$, $p(\omega_1) = p(\omega_2) = 0.5$, $r(\omega_1) = \begin{bmatrix} 10 \\ 3 \end{bmatrix}$, $T(\omega_1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $r(\omega_2) = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$, $T(\omega_2) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$. We first apply the **ABC** algorithm with **CPT-D** to solve this example.

At iteration $k = 1$, the algorithm starts from solving the LP relaxed master problem with η bounded.

$$\min \quad -1.5x_1 - 4x_2 + \eta \tag{30a}$$

$$\text{s.t.} \quad 0 \leq x_1, x_2 \leq 1 \tag{30b}$$

$$\eta \geq -M \tag{30c}$$

We get a solution $(x_1, x_2, \eta) = (1, 1, -M)$ with objective $v = -M - 5.5$. Here only the root node is in the $B\&B$ tree. $x = (1, 1)$ with $Q_1^o = \{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}$ is inserted into subproblems. Then **CPT-D** is called for each $\omega \in \Omega$. For ω_1, $f_L(x, \omega_1)$ is solved and we get $y(\omega_1) = (0, 1, 0, 0.5, 0)$. y_4 is fractional and partitions are formed for integer variables: $\{y_4 \leq 0\} \cap Q_2$ or $\{y_4 \geq 1\} \cap Q_2$. The cut derived from CGLP for $x \in Q_1^o$ is

$$-2y_2 - 2y_4 + 2R \geq -4 + 2x_2. \tag{31}$$

After adding the cut, $f_L(x, \omega_1)$ is re-optimized and the solution is $y(\omega_1) = (0, 0, 0, 1, 0)$. It satisfies integer constraints. So no more cuts are generated in this iteration. For ω_2, $f_L(x, \omega_2)$ is solved and we get $y(\omega_2) = (0, 1, 0, 0, 0)$. Again, this solution satisfies integer constraints, and hence, no cuts are needed. All scenarios have integer solution, so V is updated. $V = -29$ and, the value function cut for $x \in Q_1^o$ is

$$-16.5x_2 + \eta \geq -40. \tag{32}$$

At iteration $k = 2$, the master problem continues to be solved by B&B method. We get solution $(x_1, x_2, \eta) = (1, 0, -40)$ with objective $v = -41.5$. Still only the root node is in the B&B tree. $x = (1, 0)$

with $Q_1^o = \{0 \le x_1 \le 1, 0 \le x_2 \le 1\}$ is inserted into subproblems. **CPT-D** is called for each $\omega \in \Omega$. For ω_1, $f_L(x, \omega)$ is initialized as follows:

$$f_L(x, \omega_1) = \min \quad -16y_1 - 19y_2 - 23y_3 - 28y_4 + 100R \tag{33a}$$

$$\text{s.t.} \quad 2y_1 + 3y_2 + 4y_3 + 5y_4 - R \le 10 - x_1 \tag{33b}$$

$$6y_1 + 1y_2 + 3y_3 + 2y_4 - R \le 3 - x_2 \tag{33c}$$

$$-2y_2 - 2y_4 + 2R \ge -4 + 2x_2 \tag{33d}$$

$$0 \le y_i \le 1 \; i = 1, ..., 4, R \ge 0 \tag{33e}$$

where constraint (33d) is from cut (31) generated in iteration 1. (33) is solved and we get $y(\omega_1) = (0, 1, 0, 1, 0)$. The solution satisfies integer constraints and hence, no cuts are needed. For ω_2, $f_L(x, \omega_2)$ is solved and we get $y(\omega_2) = (0.1154, 1, 0, 0.1538, 0)$. y_1 is the variable we choose to split. The partitions we form are: $\{y_1 \le 0\} \cap Q_2$ or $\{y_1 \ge 1\} \cap Q_2$. The cut derived from CGLP for $x \in Q_1^o$ is

$$-4.875y_2 - 6.5y_4 + 1.625R \ge -6.5 + 1.625x_1. \tag{34}$$

After adding the cut, $f_L(x, \omega_2)$ is re-optimized and the solution is

$$y(\omega_2) = (0.056, 1, 0.222, 0, 0). \tag{35}$$

Since $y(\omega_2)$ is located on the root node, no more splits are needed. The same partition is used: $\{y_1 \le 0\} \cap Q_2$ or $\{y_1 \ge 1\} \cap Q_2$. The cut derived from CGLP for $x \in Q_1^o$ is

$$-2.25y_2 - 4.5y_3 + 2.25R \ge -2.25. \tag{36}$$

After adding the cut, $f_L(x, \omega_2)$ is re-optimized and the new solution is $y(\omega_2) = (0.06, 0.68, 0.16, 0.24, 0)$. Again, $y(\omega_2)$ is located on the root node, and no more splits are needed. The same partition: $\{y_1 \le 0\} \cap Q_2$ or $\{y_1 \ge 1\} \cap Q_2$ is used to formulate CGLP. With only $y(\omega_2)$ changed, another cut derived from CGLP for $x \in Q_1^o$ is

$$-2.5y_3 - 2.5y_4 + 2.5R \ge -2.5 + 2.5x_1. \tag{37}$$

After adding the cut, $f_L(x, \omega_2)$ is re-optimized and the solution is $y(\omega_2) = (0.1667, 1, 0, 0, 0)$. $y(\omega_2)$ is located still on the root node. No more splits are needed. The same partition: $\{y_1 \le 0\} \cap Q_2$ or

$\{y_1 \geq 1\} \cap Q_2$ is used to formulate CGLP. The cut derived from CGLP for $x \in Q_1^o$ is

$$-6y_1 + 1.5R \geq 0. \tag{38}$$

After adding the cut, $f_L(x, \omega_2)$ is re-optimized and the solution is $y(\omega_2) = (0, 1, 0, 0, 0)$. The solution satisfies integer constraints, and no more cuts are needed. All scenarios have integer solution, so V is updated. $V = -34.5$ and, the value function cut for $x \in Q_1^o$ is

$$-7.55x_1 - 3.8333x_2 + \eta \geq -40.55. \tag{39}$$

At iteration $k = 3$, the master problem continues to be solved by the B&B method. We get solution $(x_1, x_2, \eta) = (0, 0, -40)$ with objective $v = -40$. The solution is on node 1 with $Q_1^1 = \{0 \leq x_1 \leq 0, 0 \leq x_2 \leq 1\}$. Hence, $x = (0, 0)$ and Q_1^1 are treated as input parameters for **CPT-D** for each $\omega \in \Omega$. For ω_1, 0 cuts are needed. The solution is $y(\omega_1) = (0, 1, 0, 1, 0)$. For ω_2, 1 cut is needed (shown below). The solution is $y(\omega_2) = (0, 0, 0, 1, 0)$.

$$-3.6923y_2 - 2.4615y_3 - 3.6923y_4 + 1.2308R \geq -3.6923. \tag{40}$$

V is updated. $V = -37.5$ and, the value function cut for Q_1^1 is

$$-8.3333x_2 + \eta \geq -37.5. \tag{41}$$

At iteration $k = 4$, with updated value function cut for node 1, the master problem continues to be solve by B&B method. We get solution $(x_1, x_2, \eta) = (0, 0, -37.5)$ with objective $v = -37.5$. $V - v \leq \epsilon$. The algorithm stops. A short summary of using **ABC** algorithm with **CPT-D** to solve this problem is shown in Table 1. Each row shows the information for one iteration. Column "Num Nodes" denotes the

Iter	v	V	x	Num Nodes	$f(x, \omega_1)$	Num Cuts	$f(x, \omega_2)$	Num Cuts	Value function cut for Node
1	-M-5.5	inf	(1,1)	1	-28	1	-19	0	$\eta \geq -40 + 16.5x_2$
2	-41.5	-29	(1,0)	1	-47	0	-19	4	$\eta \geq -40.55 + 7.55x_1 + 3.8333x_2$
3	-40	-34.5	(0,0)	2	-47	0	-28	1	$\eta \geq -37.5 + 8.3333x_2$
4	-37.5	-37.5	(0,0)	3					

Table 1: **ABC** Algorithm with **CPT-D** for Example 1.0

number of active nodes in the B&B tree. "Num Cuts" means the number of multi-term disjunctive cuts generated for that scenario.

We also apply the same **ABC** algorithm but with **BB-D** to solve this example. The algorithm starts by solving the LP relaxed master problem with η bounded. At iteration $k = 1$, we get a solution $(x_1, x_2, \eta) = (1, 1, -M)$ with objective $v = -M - 5.5$ from the B&B method. Using $x = (1, 1)$ with

$Q_1^o = \{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}$ is inserted into **BB-D** for each $\omega \in \Omega$. For ω_1, $f(x, \omega_1)$ is solved by the B&B method and we get 2 nodes in $\mathcal{T}_2$. With one cut derived from CGLP for $x \in Q_1^o$, we get a new constraint

$$-2y_2 - 2y_4 + 2R \geq -4 + 2x_2, \tag{42}$$

and upon solving the updated formulation, we get $y(\omega_1) = (0, 0, 0, 1, 0)$, and no more cuts are necessary. For ω_2, $f(x, \omega_2)$ is solved by B&B method and we get one nodes in $\mathcal{T}_2$. No cuts are needed, and $y(\omega_2) = (0, 1, 0, 0, 0)$. All scenarios have integer solution, so V is updated. $V = -29$ and, the value function cut for $x \in Q_1^o$ is

$$-16.5x_2 + \eta \geq -40. \tag{43}$$

At iteration $k = 2$, the master problem continues to be solved by B&B method. We get a solution $(x_1, x_2, \eta) = (1, 0, -40)$ with objective $v = -41.5$. Using $x = (1, 0)$ with $Q_1^o = \{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}$ we now derive the subproblems. **BB-D** is called for each $\omega \in \Omega$. For ω_1, $f(x, \omega_1)$ is initialized as follows:

$$\begin{aligned}
f(x, \omega_1) = \min \quad & -16y_1 - 19y_2 - 23y_3 - 28y_4 + 100R && \text{(44a)}\\
\text{s.t.} \quad & 2y_1 + 3y_2 + 4y_3 + 5y_4 - R \leq 10 - x_1 && \text{(44b)}\\
& 6y_1 + 1y_2 + 3y_3 + 2y_4 - R \leq 3 - x_2 && \text{(44c)}\\
& -2y_2 - 2y_4 + 2R \geq -4 + 2x_2 && \text{(44d)}\\
& y_i \text{ binary } i = 1, ..., 4, R \geq 0 && \text{(44e)}
\end{aligned}$$

where constraint (44d) is from cut (42) generated in iteration 1. We solve (44) by B&B method and we get $\mathcal{T}_2$ with one node. Hence, no cuts are needed and we have $y(\omega_1) = (0, 1, 0, 1, 0)$. For ω_2, $f(x, \omega_2)$ is solved by B&B method and we get $\mathcal{T}_2$ with 4 nodes. Here 4 cuts are derived from CGLP for $x \in Q_1^o$:

$$\begin{aligned}
-8.6667y_1 + 2.1667R &\geq 0\\
-4y_3 + 4R &\geq 0\\
-3.75y_2 - 5y_4 + 1.25R &\geq -5\\
-x_1 - y_4 + R &\geq -1.
\end{aligned} \tag{45}$$

With these 4 cuts added, the solution $y(\omega_2) = (0, 1, 0, 1, 0)$. All scenarios have integer solution, so V is updated. $V = -34.5$ and, the value function cut for $x \in Q_1^o$ is

$$-4.5x_1 - 3.8333x_2 + \eta \geq -37.5. \tag{46}$$

At iteration $k = 3$, the master problem continues to be solve by B&B method. We get a solution $(x_1, x_2, \eta) = (0, 0, -37.5)$ with objective $v = -37.5$. The solution is on node 2 with $Q_1^1 = \{0 \leq x_1 \leq 1, 0 \leq x_2 \leq 0\}$. Here $x = (0, 0)$ and Q_1^2 are treated as input parameters for **BB-D** for each $\omega \in \Omega$. For ω_1, 0 cuts are needed. The solution is $y(\omega_1) = (0, 1, 0, 1, 0)$. For ω_2, 0 cuts are needed. The solution is $y(\omega_2) = (0, 0, 0, 1, 0)$. $V = -37.5$ and, the value function cut for Q_1^2 is

$$-2.8333x_1 - 5.1667x_2 + \eta \geq -37.5. \tag{47}$$

At iteration $k = 4$, with updated value function cut for node 2, the master problem continues to be solved by B&B method. We obtain $(x_1, x_2, \eta) = (0, 0, -37.5)$ with objective $v = -37.5$. $V - v \leq \epsilon$ and, the algorithm stops. A short summary of the **ABC** algorithm with **BB-D** is shown in Table 2. As one

Iter	v	V	x	Num Nodes	$f(x, \omega_1)$	Num Cuts	$f(x, \omega_2)$	Num Cuts	Value function cut for Node
1	-M-5.5	inf	(1,1)	1	-28	1	-19	0	$\eta \geq -40 + 16.5x_2$
2	-41.5	-29	(1,0)	1	-47	0	-19	4	$\eta \geq -37.5 + 4.5x_1 + 3.8333x_2$
3	-37.5	-34.5	(0,0)	2	-47	0	-28	0	$\eta \geq -37.5 + 2.8333x_1 + 5.1667x_2$
4	-37.5	-37.5	(0,0)	2					

Table 2: **ABC** Algorithm with **BB-D** for Example 1.0

might notice, there is only small difference between **BB-D** and **CPT-D** for **Example 1.0**. At iteration 2, because **BB-D** uses partitions with 4 terms to generate cuts, the quality of the Benders' cut is better than **CPT-D**.

Example 1.1 is an extension of **Example 1.0**, and is intended to illustrate the workings of the algorithm when we include general integer variables in the second stage.

$$\min \quad -1.5x_1 - 4x_2 + \sum_{\omega \in \Omega} p(\omega) f(x, \omega) \tag{48a}$$

$$\text{s.t.} \quad x_1, x_2 \text{ binary} \tag{48b}$$

where

$$f(x, \omega) = \min \quad -16y_1 - 19y_2 - 23y_3 - 28y_4 + 100R \tag{49a}$$

$$\text{s.t.} \quad \begin{bmatrix} 2y_1 + 3y_2 + 4y_3 + 5y_4 - R \\ 6y_1 + 1y_2 + 3y_3 + 2y_4 - R \end{bmatrix} \leq r(\omega) - T(\omega)x \tag{49b}$$

$$y_i \in \{0, 1...5\}, i = 1, ..., 4; R \geq 0, \tag{49c}$$

$\Omega = \{\omega_1, \omega_2\}$, $p(\omega_1) = p(\omega_2) = 0.5$, $r(\omega_1) = \begin{bmatrix} 10 \\ 3 \end{bmatrix}$, $T(\omega_1) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $r(\omega_2) = \begin{bmatrix} 5 \\ 2 \end{bmatrix}$, $T(\omega_2) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

The summaries of applying **ABC** algorithm with **CPT-D** and **BB-D** on **Example 1.1** are shown in Table 3 and Table 4.

Iter	v	V	x	Num Nodes	$f(x,\omega_1)$	Num Cuts	$f(x,\omega_2)$	Num Cuts	Value function cut for Node
1	-M-5.5	Inf	(1,1)	1	-57	0	-76	1	$\eta \geq -73.7560 + 6.2292x_1 + 1.0268x_2$
2	-76.7292	-72	(0,1)	1	-57	0	-80.625	4	$\eta \geq -80.0714 + 0.8929x_1 + 11.2589x_2$
3	-73.7560	-72.8125	(0,0)	2	-60.2979	6	-80	5	$\eta \geq -70.1489 + 2.4734x_1 + 0.9468x_2$
4	-72.8125	-72	(0,1)	3	-57	0	-80	2	$\eta \geq -79 + 1.4583x_1 + 10.5x_2$
5	-72.5	-72.5	(0,1)	3					

Table 3: **ABC** Algorithm with **CPT-D** for Example 1.1

Iter	v	V	x	Num Nodes	$f(x,\omega_1)$	Num Cuts	$f(x,\omega_2)$	Num Cuts	Value function cut for Node
1	-M-5.5	Inf	(1,1)	1	-57	0	-76	1	$\eta \geq -73.7560 + 6.2292x_1 + 1.0268x_2$
2	-76.7292	-72	(0,1)	1	-57	0	-80.1250	6	$\eta \geq -79.7232 + 1.5089x_1 + 11.1607x_2$
3	-73.7560	-72	(0,0)	2	-57	6	-80	4	$\eta \geq -68.5 + 2x_1$
4	-72.5625	-72	(0,1)	2	-57	0	-80	2	$\eta \geq -78.8571 + 1.4643x_1 + 10.3571x_2$
5	-72.5	-72.5	(0,1)	3					

Table 4: **ABC** Algorithm with **BB-D** for Example 1.1

In Table 3 and Table 4, the rows show the information generated in each successive iteration of the algorithm. The column header "Num Nodes" indicates the number of active nodes in the B&B tree and "Num Cuts" indicates the number of multi-term disjunctive cuts generated for that scenario. In the first iteration, $v = -M - 5.5$ where $-M$ denotes the lower bound for η_t (see (12)) and in our experiment we set $M = 1000000$. From the table, we can observe that both algorithms require one more iteration and generate more cuts in the subproblem than in **Example 1.0** (Table 1 and Table 2), but differences between the execution of **BB-D** and **CPT-D** for this example are minimal.

To better illustrate the algorithm, we also include two sets of figures (Figure 1 and Figure 2 in Appendix) which show the master problem B&B tree at each iteration of the algorithm. The node with bold circle contains the solution x^k and gets value function updated in the iteration. Similar to the results shown in the tables, there are only minor differences for the master problem B&B tree between the two algorithms.

In connection with this illustration, we also present two figures (Figure 3 and Figure 4 in Appendix) which show the encoded sequence for each master problem B&B tree node. With all generated Benders' cuts as one pool and all generated subproblem' cuts as another pool, the sequence encodes the validity of the cuts from each pool for any given node of the master problem B&B tree. Again, there are only minor differences in the execution of the two algorithms for this example.

Example 1.2 This example extends **Example 1.1** by requiring general integer variables in the

first stage as well. So instead of having x_1, x_2 to be binary, they are now allowed to be integers and bounded by $0 \leq x_1, x_2 \leq 5$. The summaries of applying two algorithms are shown in Table 5 and Table 6. Compared to the previous example, Tables 5 and 6 demonstrate that due to a larger number of

Iter	v	V	x	Num Nodes	$f(x,\omega_1)$	Num Cuts	$f(x,\omega_2)$	Num Cuts	Value function cut for Node
1	-M-27.5	Inf	(5,5)	1	100	0	-47	2	$\eta \geq -261 + 5x_1 + 52.5x_2$
2	-261	-1	(0,0)	1	-60.9545	4	-81.9048	4	$\eta \geq -71.4297 + 3.7564x_1 + 1.0352x_2$
3	-80.3241	-1	(0,3)	2	-19	0	-77.9165	6	$\eta \geq -83.2386 + 11.5934x_2$
4	-74.3945	-1	(0,1)	3	-57	0	-80	4	$\eta \geq -72.0974 + 1.1915x_1 + 3.5974x_2$
5	-72.5	-72.5	(0,1)	4					

Table 5: **ABC** Algorithm with **CPT-D** for Example 1.2

Iter	v	V	x	Num Nodes	$f(x,\omega_1)$	Num Cuts	$f(x,\omega_2)$	Num Cuts	Value function cut for Node
1	-M-27.5	Inf	(5,5)	1	100	0	-47	2	$\eta \geq -261 + 5x_1 + 52.5x_2$
2	-261	-1	(0,0)	1	-59.6667	5	-80	5	$\eta \geq -69.8333 + 2x_1 + 1.3333x_2$
3	-77.8333	-1	(0,3)	2	-19	0	-76	3	$\eta \geq -90.25 + 14.25x_2$
4	-73.6667	-59.5	(3,2)	5	-38	0	-61	5	$\eta \geq -74.1 + 1.8667x_1 + 9.5x_2$
5	-72.75	-62	(2,2)	5	-38	0	-66	6	$\eta \geq -81 + 5x_1 + 9.5x_2$
6	-72.5	-63	(0,1)	5	-57	0	-80	0	$\eta \geq -70.5 + 2x_1 + 2x_2$
7	-72.5	-72.5	(0,1)	5					

Table 6: **ABC** Algorithm with **BB-D** for Example 1.2

choices resulting from general integer variables in the first stage, the algorithms require more iterations to solve the problem. Note also that using **BB-D** requires more iterations than **CPT-D** for this instance, although the average number of cuts for **BB-D** is fewer than that used by **CPT-D**.

Example 1.3 As with **Example 1.2**, we maintain the same range of integers in the range $[0, 5]$, although we allow randomness in the T matrix, stated as follows.

$$T(\omega_1) = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.5 \end{bmatrix}, T(\omega_2) = \begin{bmatrix} 0.3 & 0.2 \\ 0 & 0.2 \end{bmatrix}.$$

The summary of applying the algorithms are shown in Table 7 and Table 8. From Table 7 and

Iter	v	V	x	Num Nodes	$f(x,\omega_1)$	Num Cuts	$f(x,\omega_2)$	Num Cuts	Value function cut for Node
1	-M-27.5	Inf	(5,5)	1	-22.8333	2	-66.5	2	$\eta \geq -91.1061 + 0.95x_1 + 8.3379x_2$
2	-93.8561	Inf	(5,0)	1	-59.3929	4	-71.3750	4	$\eta \geq -74.6060 + 1.8444x_1 + 0.4550x_2$
3	-81.6960	Inf	(0,2)	2	-57	0	-79.36	6	$\eta \geq -79.2 + 0.84x_1 + 5.51x_2$
4	-80.8424	Inf	(5,3)	2	-38	4	-66.925	8	$\eta \geq -65.1930 + 2.3312x_1 + 0.3581x_2$
5	-79.48	Inf	(5,2)	3	-57	0	-67.7086	10	$\eta \geq -84.1132 + 2.3273x_1 + 5.0612x_2$
6	-78.6816	Inf	(4,2)	4	-57	0	-69.2180	12	$\eta \geq -75.4437 + 0.5315x_1 + 5.1043x_2$
7	-78.16	Inf	(3,2)	4	-57	0	-70.5438	14	$\eta \geq -82.267 + 2.7281x_1 + 5.1554x_2$
8	-77.8543	Inf	(5,2)	6	-57	0	-66.4930	16	$\eta \geq -71.2465 + 4.75x_2$
9	-77.8065	Inf	(1,1)	7	-57	3	-76	13	$\eta \geq -72.3252 + 5.8252x_2$
10	-77.69	-72	(0,1)	7	-57	1	-77.3333	20	$\eta \geq -67.1667$
11	-77.5	-72	(2,2)	7	-57	0	-76	0	$\eta \geq -79.2524 + 1.1355x_1 + 5.2407x_2$
12	-77.5	-77.5	(2,2)	7					

Table 7: **ABC** Algorithm with **CPT-D** for Example 1.3

8 we can conclude that **BB-D** requires fewer disjunctive cuts on average for solving subproblems than **CPT-D**. In addition, the total number of iterations and Benders' cuts in the first stage are also fewer. Comparing **Example 1.3** with **Example 1.2**, we also observe it takes more iterations to solve Example 1.3 (with random T) than to solve Example 1.2 (with fixed T). This is due the fact that T in Example 1.3 has higher density than in Example 1.2.

Iter	v	V	x	Num Nodes	$f(x,\omega_1)$	Num Cuts	$f(x,\omega_2)$	Num Cuts	Value function cut for Node
1	-M-27.5	Inf	(5,5)	1	-22.8333	2	-65.6250	3	$\eta \geq -91.2652 + 1.0375x_1 + 8.3697x_2$
2	-93.5777	Inf	(5,0)	1	-57	4	-70.5	4	$\eta \geq -73.6806 + 1.9861x_1$
3	-81.6806	Inf	(0,2)	2	-57	0	-76	4	$\eta \geq -77.2950 + 5.3975x_2$
4	-80.4686	-74.5	(5,3)	2	-38	6	-64.6375	5	$\eta \geq -65.8580 + 2.7049x_1 + 0.3382x_2$
5	-79.7361	-74.5	(4,2)	3	-57	0	-67.5333	7	$\eta \geq -73.1165 + 0.2569x_1 + 4.9111x_2$
6	-79	-74.5	(3,2)	4	-57	0	-68.4810	10	$\eta \geq -81.0190 + 3.7595x_1 + 3.5x_2$
7	-77.5098	-74.5	(5,2)	5	-57	0	-66	4	$\eta \geq -69.8442 + 1.0174x_1 + 1.6287x_2$
8	-77.5	-77	(2,2)	5	-57	0	-76	0	$\eta \geq -76.0454 + 1.1008x_1 + 3.6719x_2$
9	-77.5	-77.5	(2,2)	5					

Table 8: **ABC** Algorithm with **BB-D** for Example 1.3

5 Computational Experiments

This section presents three computational experiments. In Section 5.1, we present computations using larger versions of the examples of Section 4. These examples are solved using a combination of Matlab and CPLEX. The Matlab code implements the first stage B&B, whereas, the rest of the algorithm was coded in C++. The purpose of this combination was to study whether a deterministic equivalent formulation (DEF), solved using a commercial state-of-the-art solver provides any advantage over our decomposition methodology. One would expect that the DEF approach would win this race because of the advances in commercial solvers for deterministic MIPs. The results of Section 5.1 reveal that despite the strides made by commercial MIP solvers for DEF, they are not competitive with the decomposition approach for SMIP models.

There are two further sub-sections of computations in this section. In Section 5.2 we upgrade the first stage branch-and-bound implementation using C++, and report whether the algorithm scales well with first stage decisions. And finally in Section 5.3, we present computations with a new class of test instances which are generalizations of the SSLP instances ([Ntaimo and Sen, 2005]). The new test instances, which we refer to as Stochastic Server Location and Sizing (SSLS), allow general integer variables in both stages.

5.1 Comparisons between ABC and DEF

In this subsection we compare how the decomposition approach of ABC compares with solving a deterministic equivalent formulation (DEF) using state-of-the-art MIP solvers. Our experiments show that even with the handicap of using a scripting language (Matlab) to code the first stage B&B, the decomposition-based ABC approach is able to provide solutions with reasonable computational resources, while CPLEX, arguably the fastest deterministic MIP solver, fails. We design a Matlab implementation of the **ABC** algorithm which calls the CPLEX LP solver whenever an LP solution is required. For all other purposes (e.g. managing the B&B tree) the Matlab script operates in the Matlab environment.

Three sets of instances are generated based on **Example 1.1-1.3**. For each example, we create 4,9,36,121,441,1681,10201 scenarios by generating the right hand sides $r(\omega)$ from equidistant lattice points in $[5, 15] \times [5, 15]$ with equal probability assigned to each point. This methodology was borrowed

from [Schultz et al., 1998] (see also [Ahmed et al., 2004]). For the seven instances based on Example 1.3, we use the same random right hand side $r(\omega) = \begin{bmatrix} r_1(\omega) \\ r_2(\omega) \end{bmatrix}$ as in Example 1.2, but in addition $T(\omega)$ is also random. The entries for these matrices were 0 or 1 with equal probability.

Table 9 compares the performance of three algorithms: two based on using the **ABC** algorithm with **CPT-D** and **BB-D** whereas, the third algorithm used CPLEX 12.3 (with default setting) for the MILP formulation of the deterministic equivalent formulation (**DEF**). All approaches were run on a Windows 7 PC operating with Intel i7-3770K 3.5GHz processors and 8 GB memory. Instances 1-3 in the table correspond to the variations based on Example 1.1-1.3. The column heading **Obj** denotes the optimal objective value of the SMIP, **Var** and **Constr** denote the number of variables and constraints in the **DEF**. The entries in column **ABC(CPT-D)** and **ABC(BB-D)** denote Iterations (Master Nodes, Second stage Leaf Nodes, Running Time) which correspond to the total number of iterations, the total number of nodes in the B&B tree in the master problem, the maximum leaf nodes encountered during solving the subproblem and the total running time. The **DEF** column shows the cpu time (in secs) required to solve **DEF** using the default version CPLEX 12.3 MIP solver. The maximum cpu time allowed was 60 minutes.

	Scenarios	**Obj**	**Var**	**Constr**	**ABC(CPT-D)**	**ABC(BB-D)**	**DEF**
Instance 1	4	-63.50	26	17	7 (4, 5, 0.234)	5 (4,5,0.14)	0.23
	9	-66.56	56	37	6 (2, 6, 0.29)	18 (15, 10, 0.98)	0.02
	36	-66.83	218	145	7 (2, 7, 1.36)	6 (2, 7, 1.01)	0.02
	121	-67.17	728	485	7 (1, 8, 4.43)	6 (1, 8, 2.96)	0.16
	441	-65.58	2648	1765	16 (3, 15, 46.63)	10 (2, 7, 15.27)	1.58*
	1681	-64.72	10088	6725	16 (3, 17, 262.44)	12 (3, 23, 85.29)	Timed out(>3600)
	10201	-64.19	61208	40805	Numerical Issue	13 (3, 23, 583.08)	Timed out(>3600)
Instance 2	4	-63.50	26	17	12 (10, 10, 0.37)	12 (14, 10, 0.3)	0.02
	9	-66.56	56	37	20 (15, 10, 1.92)	18 (15, 10, 0.94)	0.02
	36	-69.86	218	145	19 (16, 12, 7.64)	18 (16, 10, 4.54)	0.03
	121	-71.12	728	485	18 (16, 11, 25.18)	17 (16, 11, 13.51)	4.09
	441	-69.64	2648	1765	20 (16, 22, 168.43)	18 (15, 21, 58.97)	Timed out(>3600)
	1681	-68.85	10088	6725	Numerical Issue	20 (16, 27, 312.36)	Timed out(>3600)
	10201	-68.45	61208	40805	Numerical Issue	21 (18, 31, 2333.307)	Timed out(>3600)
Instance 3	4	-63.50	26	17	10 (6, 7, 0.55)	19 (18,8,0.41)	0.00
	9	-64.22	56	37	19 (15, 10, 2.09)	19 (15, 10, 0.83)	0.13
	36	-66.42	218	145	25 (21, 12, 7.85)	25 (19, 13, 3.62)	4.18
	121	-64.78	728	485	25 (20, 12, 26.94)	25 (19, 13, 12.04)	Timed out(>3600)
	441	-63.33	2648	1765	28 (21, 24, 222.82)	27 (20, 24, 82.60)	Timed out(>3600)
	1681	-62.19	10088	6725	31 (22, 28, 1210.94)	28 (23, 33, 419.42)	Timed out(>3600)
	10201	-61.83	61208	40805	Numerical Issue	31 (24, 34, 3243.82)	Timed out(>3600)

Table 9: Comparison of **ABC** with DEF

The results reported in Table 9 clearly demonstrate that the approach of solving a **DEF** with a commercial solver does not scale well, failing to solve 9 instances for which the number of scenarios were somewhat large. In comparing the performance of **ABC(CPT-D)** and **ABC(BB-D)**, we observe that the former also runs into numerical difficulties for 4 of the larger instances. In contrast, **ABC(BB-D)**

*Obj=65.576 and the solution has numerical issues

produces optimal solutions for all the instances within very reasonable computational times. From the log-log plot in Figure 5 (see Appendix), as number of scenarios increases, the running time also increases polynomially for **ABC(BB-D)**. The slopes of each of the three graphs are slightly larger than one (with values 1.0048, 1.1245, 1.1677) which suggests that as the number of scenarios increases, the running time increases at a rate that is only slightly worse than "linear". To sum up, based on the instances tested, algorithm **ABC** shows very stable results and scales quite well with the number of scenarios.

5.2 Computations with Larger First Stage Instances

In this subsection we study how the ABC algorithm performs when the number of first stage (search) variables increase. The larger instances are generated by including two more variables at a time to **Example 1.2** and **1.3**. For the i-th time, the newly added pair of variables have objective coefficients given by $-1.5 * 2^i$ and $-4 * 2^i$. For the larger instance of **Example 1.2**, matrix $T(\omega)$ is fixed. The fixed matrix is extended by a 2×2 identity matrix with the newly pair of variables added. In the case of **Example 1.3**, $T(\omega)$'s elements are uniformly generated from set $\{0.1, 0.2, 0.3....0.9\}$. The rest of the parameters are generated the same way as in instance 2 and instance 3.

We remind the reader that the objective function contours of this instance have been depicted in the case of 2 variables by [Schultz et al., 1998] and [Ahmed et al., 2004]. These papers also captured the challenges posed by the two variable instance. In comparison, the total number of first stage feasible solutions in our extension increases the number of feasible first stage solutions exponentially according to the sequence 36^i, $i = 1, 2$, etc. The largest instances we attempt have a total number of feasible first stage solutions given by 36^6, implying a total of more than 2.1 billion feasible solutions. Of course, using Matlab to search for optimal solutions of such instances would be untenable. So, the entire ABC algorithm is implemented in C++ including the first stage B&B and second stage decomposition. All linear programs were solved using CPLEX. Finally, we also note that our experiments were conducted for the most promising scheme among those compared in the previous section. Thus, the following reports our results using the **BB-D** version of the ABC algorithm.

The computational result of extending **Example 1.2** is shown in Table 10. The column heading **First Stage Var Num** denotes the number of first stage variables, and the remaining information follows the same style as reported in Table 9. Comparing **First Stage Var Num** = 2 with instance 2 from Table 9, we can see the speed up of C++ implementation compared to using Matlab. And as **First Stage Var Num** grows each time by two, Iterations and Master Nodes required to solve the problem grow exponentially. We encounter some numerical difficulties for solving instances of 8 variables with 121 and 441 scenarios and 10 variables with 441 scenarios. Nevertheless, the algorithm did solve instances

of 12 variables in reasonable time. Thus, as the number of first stage variables grow, instances may not become harder, and such anomalies are also known to exist for deterministic MIPs as well.

The computational result of extending **Example 1.3** (random T) is shown in Table 11. We can see the same speed up of C++ implementation compared to Matlab as well as the exponential growth of Iterations and Master Nodes as **First Stage Var Num** are added by two each time. There are numerical difficulties for solving instances of 8 and 10 variables with 121 and 441 scenarios and 12 variables with 441 scenarios. Because T is random and relatively dense, these instances becomes harder to solve.

First Stage Var Num	Scenarios	Obj	Var	Constr	ABC(BB-D)
2	4	-63.5	26	17	11 (11, 8, 0.04)
	9	-66.56	56	37	10 (7, 10, 0.07)
	36	-69.86	218	145	12 (9, 10, 0.35)
	121	-71.12	728	485	12 (11, 10, 1.24)
	441	-70.23	2648	1765	14 (17, 21, 9.97)
4	4	-74	28	17	21 (47, 9, 0.09)
	9	-81.89	58	37	26 (51, 10, 0.17)
	36	-87.08	220	145	21 (35, 10, 0.63)
	121	-89.07	730	485	31 (61, 10, 2.71)
	441	-87.39	2650	1765	30 (69, 19, 20.90)
6	4	-125.75	30	17	181 (833, 8, 0.49)
	9	-130.78	60	37	135 (635, 10, 0.76)
	36	-134.17	222	145	145 (603, 10, 3.73)
	121	-135.3	732	485	152 (591, 10, 17.99)
	441	-134.07	2652	1765	177 (677, 17, 158.55)
8	4	-235.75	32	17	56 (185, 10, 0.18)
	9	-240.78	62	37	1810 (6681, 9, 18.82)
	36	-244.17	224	145	2255 (8031, 10, 193.42)
10	4	-455.95	34	17	24 (107, 8, 0.07)
	9	-460.87	64	37	19 (63, 8, 0.11)
	36	-489.45	226	145	78 (229, 9, 1.70)
	121	-500.53	736	485	442 (1325, 9, 73.96)
12	4	-965.25	36	17	14611 (41993, 9, 184.77)
	9	-1045.33	66	37	217 (905, 9, 1.14)
	36	-1089.56	228	145	52 (231, 8, 0.92)
	121	-1108.23	738	485	65 (245, 8, 4.53)
	441	-1115.72	2658	1765	58 (215, 11, 20.97)

Table 10: Computational results with **ABC/BB-D** as first stage variables increase (fixed T)

5.3 Computational Results with Stochastic Server Location and Sizing

One application of SMIP is in the network design and planning where servers need to be located in a least cost manner before demand from customers is known. This problem arises in a variety of domains such as telecommunication, electricity power, water distribution, internet service and anti-terrorism. One of the instances available in the literature, attributed to [Ntaimo and Sen, 2005] is a stochastic server location problem (SSLP) in which demand uncertainty is limited to whether or not demand will occur at nodes of a graph. The goal of SSLP is to place servers at discrete locations in such a way that the expected cost of serving demand is minimized. That collection of instances have 0-1 variables in both

First Stage Var Num	Scenarios	Obj	Var	Constr	ABC(BB-D)
2	4	-63.5	26	17	13 (31, 16, 0.10)
	9	-64.22	56	37	24 (63, 23, 0.44)
	36	-66.46	218	145	22 (47, 21, 1.40)
	121	-64.79	728	485	27 (55, 32, 7.07)
	441	-63.58	2648	1765	27 (55, 28, 28.23)
4	4	-79.7	28	17	13 (31, 16, 0.10)
	9	-82.78	58	37	24 (63, 23, 0.44)
	36	-83.02	220	145	22 (47, 21, 1.40)
	121	-83.14	730	485	27 (55, 32, 7.07)
	441	-82.41	2650	1765	27 (55, 28, 28.23)
6	4	-127.75	30	17	135 (371, 20, 0.54)
	9	-124.49	60	37	139 (407, 21, 1.49)
	36	-126.56	222	145	163 (435, 22, 7.28)
	121	-128.61	732	485	183 (475, 24, 39.77)
	441	-128.87	2652	1765	186 (463, 23, 246.74)
8	4	-232.5	32	17	1127 (4105, 20, 8.16)
	9	-242.78	62	37	1657 (5627, 22, 26.94)
	36	-229.61	224	145	2324 (7243, 23, 293.37)
10	4	-444.75	34	17	27505 (87493, 20, 111.41)
	9	-488.11	64	37	144 (325, 15, 1.03)
	36	-517.42	226	145	2098 (6297, 21, 205.03)
12	4	-886.4	36	17	5 (1, 17, 0.02)
	9	-1016.78	66	37	8 (3, 17, 0.04)
	36	-1070.31	228	145	158 (367, 17, 3.27)
	121	-1094.80	738	485	146 (355, 16, 13.23)

Table 11: Computational results with **ABC/BB-D** as first stage variables increase (random T)

stages. In order to test the ABC algorithm, we extend the above instances to a class of problems where in addition to location, we are interested in sizing the servers, where the size is measured in discrete units $[0, 1, 2, .., 5]$. We refer to these test instances as the Stochastic Server Location and Sizing (SSLS) instances.

To describe the model, let I be the index set for client locations and J be the index sets for possible server locations. For $i \in I$ and $j \in J$, we define the following.

Parameters

c_j Cost of locating a server at location j.

q_{ij} Revenue from a client at location i being served by servers at location j.

q_{j0} Loss of revenue because of overflow at server j.

d_{ij} Resource requirement of client i for server at location j.

r Upper bound on the total number of servers that can be located.

u Upper bound of number of server located at one location.

w Server capacity.

v Upper bound of possible number of clients.

$h^i(\omega)$ The number of clients at location i in scenario ω

$h(\omega)$ The vector made of elements $h^i(\omega)$.

$p(\omega)$ Probability of occurrence for scenario $\omega \in \Omega$.

Decisions variables

x_j number of servers located at site j.

y_{ij} number of clients at location i served by servers at location j.

The SSLS can be formulated as follows.

$$\min \quad \sum_{j \in J} c_j x_j - \mathsf{E}[f(x, \tilde{\omega})] \tag{50a}$$

$$\textbf{s.t.} \quad \sum_{j \in J} x_j \leq r, \tag{50b}$$

$$x_j \in \{0, 1, ...u\}, \quad \forall j \in J, \tag{50c}$$

where

$$\mathsf{E}[f(x, \tilde{\omega})] = \sum_{\omega \in \Omega} p(\omega) f(x, \omega) \tag{51}$$

For any x satisfying (50b,50c) and $\omega \in \Omega$,

$$f(x, \omega) = \min \quad -\sum_{i \in I} \sum_{j \in J} q_{ij} y_{ij} + \sum_{j \in J} q_{j0} y_{j0} \tag{52a}$$

$$\textbf{s.t.} \quad \sum_{i \in I} d_{ij} y_{ij} - y_{j0} \leq w x_j, \quad \forall j \in J, \tag{52b}$$

$$\sum_{j \in J} y_{ij} = h^i(\omega), \quad \forall i \in I, \tag{52c}$$

$$y_{ij} \in \{0, 1, 2...v\}, \forall i \in I, j \in J, \tag{52d}$$

$$y_{j0} \geq 0, \quad \forall i \in I, j \in J. \tag{52e}$$

We generated a set of SSLS instances following the method in [Ntaimo and Sen, 2005]. For problem data, the server location costs c_j were generated randomly from the uniform distribution in the interval $[40, 80]$ and the client demands d_{ij} were generated in the interval [0,25]. The client-server revenue were set at one unit per unit of client demand. The overflow costs q_{j0} were fixed at 1000. For scenario data, the number of clients available in each scenario $h^i(\omega)$ were generated from a binomial distribution with $p = 0.5$ and v trials.

The size of the model is captured by the name "SSLS-$(m \times u)$-$(n \times v)$-S", where m is the number of potential server locations, u is the maximum number of servers allowed for each location, n is the number of potential client locations, v is the maximum number of possible clients for each location and S is the number of scenarios. We report results on problem instances with $m \in \{2, 3, 4\}$, $n \in \{5, 10, 15\}$, $S \in \{50, 100, 500\}$, $u = 5$ and $v = 5$. A summary of the notations is shown in Table 12. We should note that while the number of potential server locations appear to be small, the fact that there are alternative sizes possible makes the combinatorial content significant, and more-or-less on par with the SSLP instances.

Notation	Interpretation	Value
m	number of potential server locations	2,3,4
u	maximum number of servers allowed for each location	5
n	number of potential client locations	5,10,15
v	maximum number of clients for each location	5
S	number of scenarios	50,100,500

Table 12: SSLS-$(m \times u)$-$(n \times v)$-S

The computational environment is the same as in Section 5.2 and we used the C++ implementation for this experiment. Table 13 shows the results which are reported in a manner similar to Table 9, 10 etc. Again the number of variables (**Var**) and constraints (**Constr**) refer to numbers in the DEF formulation. The metrics reported in the column labeled **ABC(BB-D)** refer to the number of iterations and in parentheses, the numerical quantities refer to the number of master nodes, the number of second stage leaf nodes, and the running time. The algorithm can handle most of the instances except the last two where the task of generating cuts for all outcomes slowed down the process. One should expect that problems with fixed recourse would be easier to handle. Among the runs completed, we can see that as m gets larger, Iterations and Master Nodes required to solve the problem grow dramatically as expected. On the other hand, as n gets larger while m,u,v and S stay the same, we find the Iterations and Master Nodes do not change in general, while the Second stage Leaf Nodes and Running Time grow exponentially. Given the presence of mixed-integer variables in both stages, such computational results are not surprising.

6 Conclusion

As stated at the outset, our paper returns to the general class of two stage SMIP problems that was the focus of the paper by [Carøe and Tind, 1998]. This class of problems involves mixed-integer variables in both stages, and randomness is also allowed in all data elements of the second stage MIP. Despite the elegance of the work of [Carøe and Tind, 1998], the chasm between first and second stage strategies

Instance	Scenarios	Obj	Var	Constr	ABC(BB-D)
$SSLS\_(2\times5)\_(5\times5)\_50$	50	-128.05	602	601	11 (13, 16, 0.30)
$SSLS\_(2\times5)\_(5\times5)\_100$	100	-40.45	1202	1201	9 (13, 2, 0.38)
$SSLS\_(2\times5)\_(5\times5)\_500$	500	-151.2	6002	6001	12 (13, 22, 3.58)
$SSLS\_(2\times5)\_(10\times5)\_50$	50	-232.63	1102	1101	14 (15, 50, 0.52)
$SSLS\_(2\times5)\_(10\times5)\_100$	100	-237.67	2202	2201	13 (13, 515, 4.84)
$SSLS\_(2\times5)\_(10\times5)\_500$	500	-256.88	11002	11001	13 (17, 258, 4.43)
$SSLS\_(2\times5)\_(15\times5)\_50$	50	-457.24	1602	1601	20 (23, 344, 4.26)
$SSLS\_(2\times5)\_(15\times5)\_100$	100	-420.35	3202	3201	16 (21, 100, 1.64)
$SSLS\_(2\times5)\_(15\times5)\_500$	500	-399.47	16002	16001	15 (17, 411, 51.28)
$SSLS\_(3\times5)\_(10\times5)\_50$	50	-110.32	903	651	23 (33, 2, 0.59)
$SSLS\_(3\times5)\_(10\times5)\_100$	100	-107.63	1803	1301	23 (33, 2, 1.23)
$SSLS\_(3\times5)\_(10\times5)\_500$	500	-86.67	9003	6501	23 (35, 5, 7.09)
$SSLS\_(3\times5)\_(10\times5)\_50$	50	-362.43	1653	1151	36 (59, 343, 4.68)
$SSLS\_(3\times5)\_(10\times5)\_100$	100	-339.12	3303	2301	32 (45, 8632, 158.61)
$SSLS\_(3\times5)\_(10\times5)\_500$	500	-364.58	16503	11501	28 (41, 566, 27.15)
$SSLS\_(3\times5)\_(15\times5)\_50$	50	-436.42	2403	1651	37 (55, 9435, 191.92)
$SSLS\_(3\times5)\_(15\times5)\_100$	100	-371.24	4803	3301	29 (33, 1290, 19.56)
$SSLS\_(3\times5)\_(15\times5)\_500$	500	-426.54	24003	16501	40 (57, 8984, 1069.92)
$SSLS\_(4\times5)\_(5\times5)\_50$	50	-131.05	1204	701	58 (87, 8, 1.59)
$SSLS\_(4\times5)\_(5\times5)\_100$	100	-119.18	2404	1401	59 (91, 9, 3.36)
$SSLS\_(4\times5)\_(5\times5)\_500$	500	-73.55	12004	7001	57 (87, 2, 20.99)
$SSLS\_(4\times5)\_(10\times5)\_50$	50	-308.69	2204	1201	63 (91, 130, 3.60)
$SSLS\_(4\times5)\_(10\times5)\_100$	100	-350.08	4404	2401	68 (111, 6419, 261.89)
$SSLS\_(4\times5)\_(10\times5)\_500$	500	-311.3	22004	12001	64 (97, 8683, 745.61)
$SSLS\_(4\times5)\_(15\times5)\_50$	50	-521.07	3204	1701	104 (161, 5849, 1653.67)
$SSLS\_(4\times5)\_(15\times5)\_100$	100	N/A	6404	3401	Timed out(>3600)
$SSLS\_(4\times5)\_(15\times5)\_500$	500	N/A	32004	17001	Timed out(>3600)

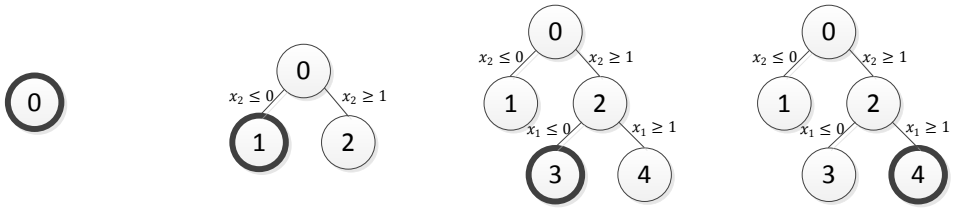
Table 13: Computational results with **ABC/BB-D** on SSLS

has persisted over the past 15 years. Using several building blocks that have been effective in the interim, we have developed a time-staged decomposition algorithm for very general SMIP models. Other effective ideas, such as allowing the second stage problem to be solved inexactly are also permitted within the overall strategy. The key feature of this algorithm is a first stage B&B process (i.e. the ABC algorithm) which simultaneously guides both the construction of approximations as well as the search for optimal first stage decisions. Furthermore, the value function approximations remain piecewise linear and convex for each first stage B&B node, and similarly, the second stage relaxations (built using multi-term disjunctions) are also polyhedral. While these elements maintain convex building blocks, the overall search is facilitated by an encoding strategy (to record approximations) that allows us to approximate severe non-convexities typical of general SMIP models.

We have also presented the most comprehensive computational experiment to date for problems in which mixed-integer variables appear in both stages. Our computations reveal that as the number of scenarios grow, our Matlab-guided implementation was faster and more stable than the commercial solver for an extensive form SMIP. We also reported computational results for larger instances which required a more sophisticated implementation than the Matlab prototype. Both of those experiments (one with an extended test instance, and another with a new test problem) reveal that the ABC framework, especially

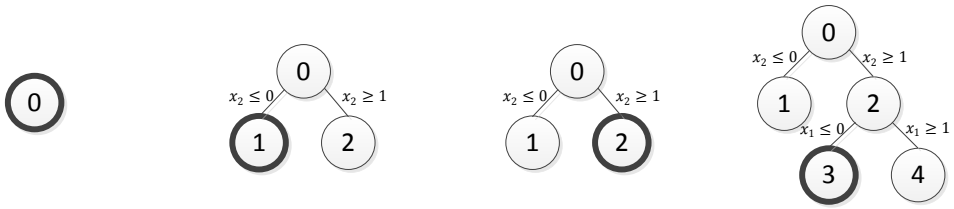
in the BB-D setting, leads to a viable approach for realistic instances. We recommend at least two avenues in which the ABC/BB-D approach may be promising: a) implementing these ideas on parallel machines should be fruitful, and b) these concepts should be extended to multi-stage models.

Appendix A Figures for Examples



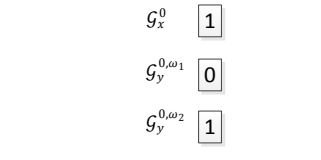
(a) Iteration 1,2 (b) Iteration 3 (c) Iteration 4 (d) Iteration 5

Figure 1: Master Problem B&B Tree for **ABC** Algorithm with **CPT-D** on Example 1.1

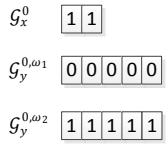


(a) Iteration 1,2 (b) Iteration 3 (c) Iteration 4 (d) Iteration 5

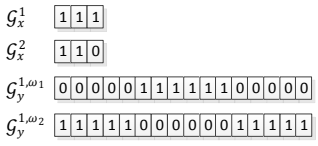
Figure 2: Master Problem B&B Tree for **ABC** Algorithm with **BB-D** on Example 1.1



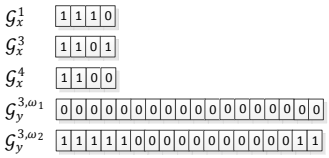
(a) Iteration 1



(b) Iteration 2

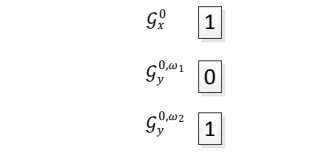


(c) Iteration 3

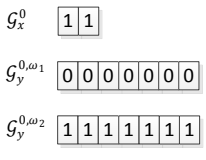


(d) Iteration 4

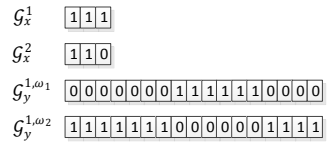
Figure 3: $\mathcal{G}_x, \mathcal{G}_y$ for **ABC** Algorithm with **CPT-D** on Example 1.1



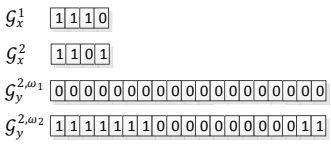
(a) Iteration 1



(b) Iteration 2



(c) Iteration 3



(d) Iteration 4

Figure 4: $\mathcal{G}_x, \mathcal{G}_y$ for **ABC** Algorithm with **BB-D** on Example 1.1

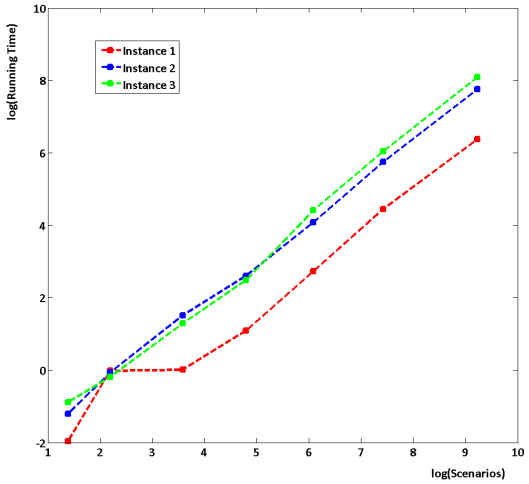


Figure 5: Log-log Plot of Running Time and Scenarios for **ABC/BB-D** for instances in Table 9

References

[Ahmed et al., 2004] Ahmed, S., Tawarmalani, M., and Sahinidis, N. V. (2004). A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100(2):355–377.

[Balas, 1979] Balas, E. (1979). Disjunctive programming. *Annals of Discrete Mathematics*, 5:3–51.

[Balas et al., 1993] Balas, E., Ceria, S., and Cornuéjols, G. (1993). A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324.

[Carøe and Tind, 1998] Carøe, C. and Tind, J. (1998). L-shaped decomposition of two-stage stochastic programs with integer recourse. *Mathematical Programming*, 83:451–464.

[Carøe and Schultz, 1997] Carøe, C. C. and Schultz, R. (1997). Dual decomposition in stochastic integer programming. *Operations Research Letters*, 24:37–45.

[Carøe and Schultz, 1998] Carøe, C. C. and Schultz, R. (1998). A two-stage stochastic program for unit commitment under uncertainty in a hydro-thermal power system. In *Preprint SC 98-11, Konrad-Zuse-Zentrum fur Informationstechnik*, pages 98–13.

[Chen et al., 2011] Chen, B., Küçükyavuz, S., and Sen, S. (2011). Finite disjunctive programming characterizations for general mixed integer linear programs. *Operations Research*, 59(1):202–210.

[Chen et al., 2012] Chen, B., Küçükyavuz, S., and Sen, S. (2012). A computational study of the cutting plane tree algorithm for general mixed-integer linear programs. *Operations Research Letters*, 40(1):15 – 19.

[Escudero et al., 2007] Escudero, L., Garn, A., Merino, M., and Pérez, G. (2007). A two-stage stochastic integer programming approach as a mixture of branch-and-fix coordination and benders decomposition schemes. *Annals of Operations Research*, 152:395–420. 10.1007/s10479-006-0138-0.

[Jörg, 2007] Jörg, M. (2007). k-disjunctive cuts and a finite cutting plane algorithm for general mixed integer linear programs. *arXiv preprint arXiv:0707.3945*.

[Kong et al., 2006] Kong, N., Schaefer, A. J., and Hunsaker, B. (2006). Two-stage integer programs with stochastic right-hand sides. *Mathematical Programming*, 108(24):275–296.

[Louveaux and Schultz, 2003] Louveaux, F. and Schultz, R. (2003). Stochastic integer programming. *Handbooks in Operations Research and Management Science*, 10:213–266.

[Lulli and Sen, 2004] Lulli, G. and Sen, S. (2004). A branch-and-price algorithm for multistage stochastic integer programming with application to stochastic batch-sizing problems. *Management Science*, 50(6):786–796.

[Ntaimo, 2010] Ntaimo, L. (2010). Disjunctive decomposition for two-stage stochastic mixed-binary programs with random recourse. *Operations Research*, 58(1):229–243.

[Ntaimo and Sen, 2005] Ntaimo, L. and Sen, S. (2005). The million-variable "march" for stochastic combinatorial optimization. *Journal of Global Optimization*, 32(3):385–400.

[Owen and Mehrotra, 2001] Owen, J. H. and Mehrotra, S. (2001). A disjunctive cutting plane procedure for general mixed-integer linear programs. *Mathematical Programming*, 89(3):437–448.

[Owen and Mehrotra, 2002] Owen, J. H. and Mehrotra, S. (2002). On the value of binary expansions for general mixed-integer linear programs. *Operations Research*, 50(5):810–819.

[Schultz et al., 1998] Schultz, R., Stougie, L., and van der Vlerk, M. H. (1998). Solving stochastic programs with integer recourse by enumeration: A framework using Gröbner basis. *Mathematical Programming*, 83(1-3):229–252.

[Sen, 2010] Sen, S. (2010). *Stochastic Mixed-Integer Programming Algorithms: Beyond Benders' Decomposition*. John Wiley & Sons, Inc.

[Sen and Higle, 2005] Sen, S. and Higle, J. L. (2005). The C3 Theorem and a D2 Algorithm for Large Scale Stochastic Mixed-Integer Programming: Set Convexification. *Mathematical Programming*, 104(1):1–20.

[Sen et al., 2003] Sen, S., Higle, J. L., and Ntaimo, L. (2003). A summary and illustration of disjunctive decomposition with set convexification. *Woodruff, D. (ed.) Network Interdiction and Stochastic Integer Programming*, pages 105–125.

[Sen and Sherali, 2006] Sen, S. and Sherali, H. D. (2006). Decomposition with branch-and-cut approaches for two-stage stochastic mixed-integer programming. *Mathematical Programming*, 106(2):203–223.

[Sherali and Zhu, 2006] Sherali, H. D. and Zhu, X. (2006). On solving discrete two-stage stochastic programs having mixed-integer first- and second-stage variables. *Mathematical Programming*, 108(2-3):597–616.

[Trapp et al., 2013] Trapp, A. C., Prokopyev, O. A., and Schaefer, A. J. (2013). On a level-set characterization of the integer programming value function and its application to stochastic programming. *Operations Research*, 61(2):498–511.

[Wollmer, 1980] Wollmer, R. D. (1980). Two-stage linear programming under uncertainty with $0-1$ integer first stage variables. *Mathematical Programming*, 19(3):279–288.

[Yuan and Sen, 2009] Yuan, Y. and Sen, S. (2009). Enhanced cut generation methods for decomposition-based branch and cut for two-stage stochastic mixed-integer programs. *INFORMS Journal on Computing*, 21(3):480–487.